(2001)				
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	7 May, 2001	8 September, 20153	GEN-CXXXXX-REVA	1 of 16
Author:	Laurent Lefebvre			
Issue To:		Convillo		
issue to.		Copy No:		
	R400 S	equencer Spe	ecification	
		SEQ		
		Version 0. <u>3</u> 2		
req		cted uses of the block. It a	cer block (SEQ). It provides an ov lso describes the block interfaces,	
Document Lo		erforce\r400\arch\doc\gfx\RE D Sequencer Specification APPROVALS	\R400_Sequencer.doc	
	Name/Dept		Signature/Date	
Remarks:				
	NTIALLY DETRIME	NTAL TO THE INTE	NFORMATION THAT CO REST OF ATI TECHNOL E OR DISCLOSURE.	
work created in unpublished we confidential, pro	a 2001. The use of this cop ork. The copyright notice oprietary information and tra	yright notice is intended to is not an admission that ade secrets of ATI. No part	rial in this document constitutes an provide notice that ATI owns a cop publication has occurred. This w of this document may be used, re sion of ATI Technologies Inc."	oyright in this ork contains
Exhibit 2009.docR400_S	Sequences.doc 16178 Bytes*** ⓒ AT	l Confidential. Reference C	Copyright Notice on Cover Page ©) *** <u>09/08/15 01:03</u>

ATI 2009 LG v. ATI IPR2015-00325

AMD1044_0256673

ATI Ex. 2105 IPR2023-00922 Page 1 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
7 May, 2001	8 September, 20153		2 of 16

Table Of Contents

1. OVERVIEW
1.1 Top Level Block Diagram
1.2 Data Flow graph
1.3 Control Graph
2. INTERPOLATED DATA BUS 1110
2. INTERPOLATED DATA BUS1140 3. INSTRUCTION STORE1140
4. CONSTANT STORE1244
4.CONSTANT STORE12115.LOOPING AND BRANCHES1211
6. REGISTER FILE ALLOCATION 1211
7. TEXTURE ARBITRATION
8. ALU ARBITRATION 1312
9. HANDLING STALLS 1413
10. CONTENT OF THE RESERVATION
STATION FIFOS1413
11. THE OUTPUT FILE (RB FIFO AND
PARAMETER CACHE)1413
12. INTERFACES
12.1 External Interfaces 1413
12.1.1 Sequencer to Shader
Engine Bus1413
12.1.2 Shader Engine to Output
File 1413
12.1.3 Shader Engine to Texture
Unit Bus (Fast Bus)
12.1.4 Sequencer to Texture Unit bus
(Slow Bus) 1514

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

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.

Exhibit 2009.docR400_Sequences.doc 16178 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** <u>PM08/13/01-03:17 PM</u>07/13/01-02:10 PM

AMD1044_0256674

ATI Ex. 2105 IPR2023-00922 Page 2 of 239

12.1.5 Shader Engine to RE/PA Bus
<u> </u>
12.1.6 PA to sequencer
<u>13. OPEN ISSUES1614</u>
1. OVERVIEW
1.1 Top Level Block Diagram
2. TEXTURE ARBITRATION
3. ALU ARBITRATION
4. HANDLING STALLS
5. CONTENT OF THE RESERVATION
STATION FIFOS
6. INTERFACES 10
6.1 External Interfaces 10
6.1.1 Sequencer to Shader Engine
Bus10
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 Bus11
7. OPEN-ISSUES

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE]
7 May, 2001	8 September, 20153	GEN-CXXXXX-REVA	3 of 16	

1. Overview

The sequencer first arbitrates between vectors of 16-(maybe 32) vertices that arrive directly from primitive assembly and vectors of 84-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 ag clause before looking for a new clause of the same type. <u>Two ALU clauses are executed interleaved to hide the ALU latency</u>. 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 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. <u>There are two separate sets of reservation stations, one for pixel vectors</u> 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.

Exhibit 2009.docR400_Sequencer.doc 16178 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/08/15/01/03

AMD1044_0256675

ATI Ex. 2105 IPR2023-00922 Page 3 of 239

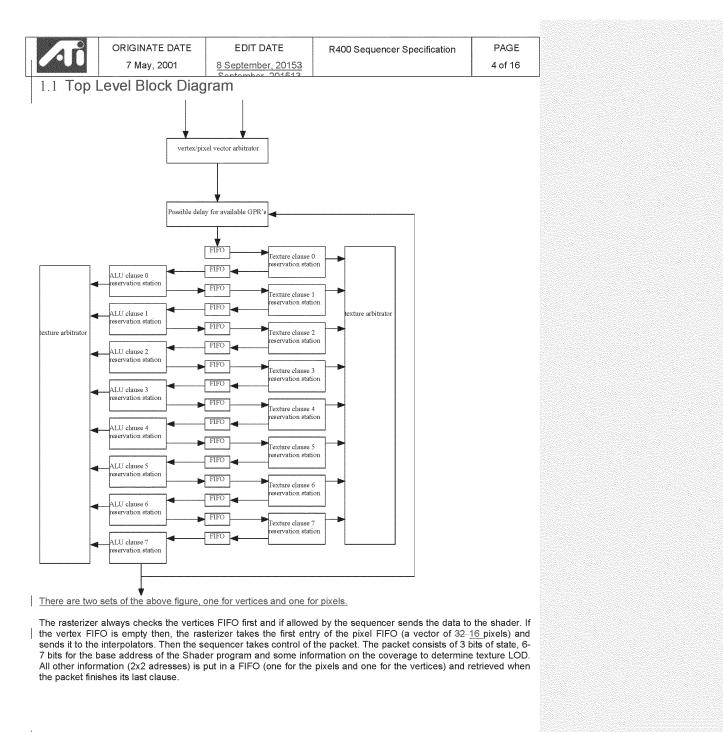


Exhibit 2008.doc R400_Sequences.doc 16178 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/08/15 01:03 PM08/13/01-03:17-PM/07/13/01-02:10-PM

AMD1044_0256676

ATI Ex. 2105 IPR2023-00922 Page 4 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
7 May, 2001	8 September, 20153 September, 201513	GEN-CXXXXX-REVA	5 of 16

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.

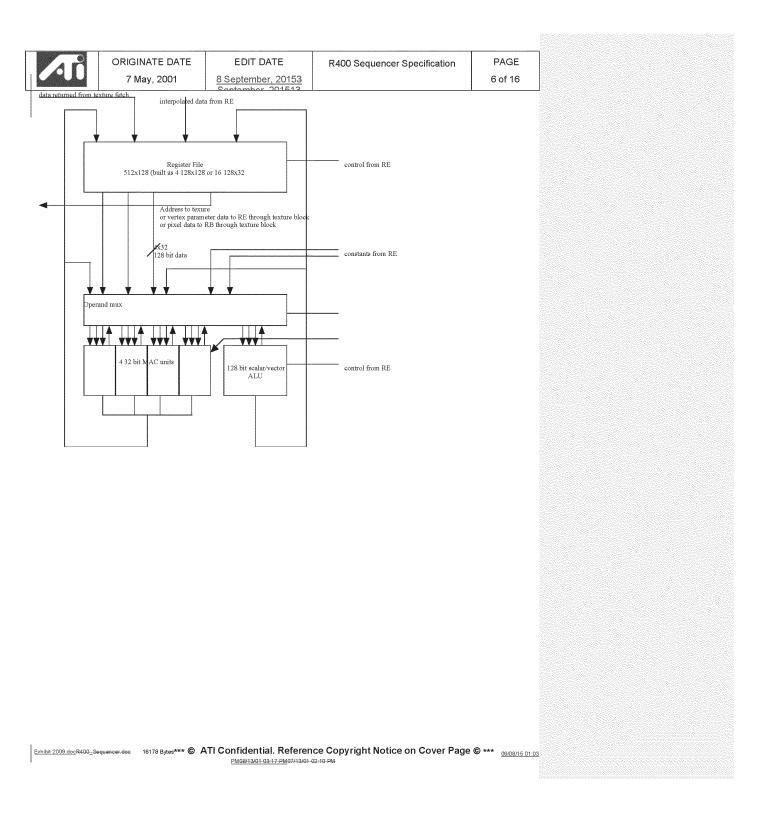
Only one-two ALU state machine may have access to the SRAMregister file address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the SRAMregister file address bus at one time. Arbitration is performed by two-three arbitrer blocks (one-two for 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 SRAMregister Sfiles.

Each state machine maintains an address pointer specifying where the 16-(or 32) entries vector is located in the SRAMregister 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 SRAMregister file memory is available

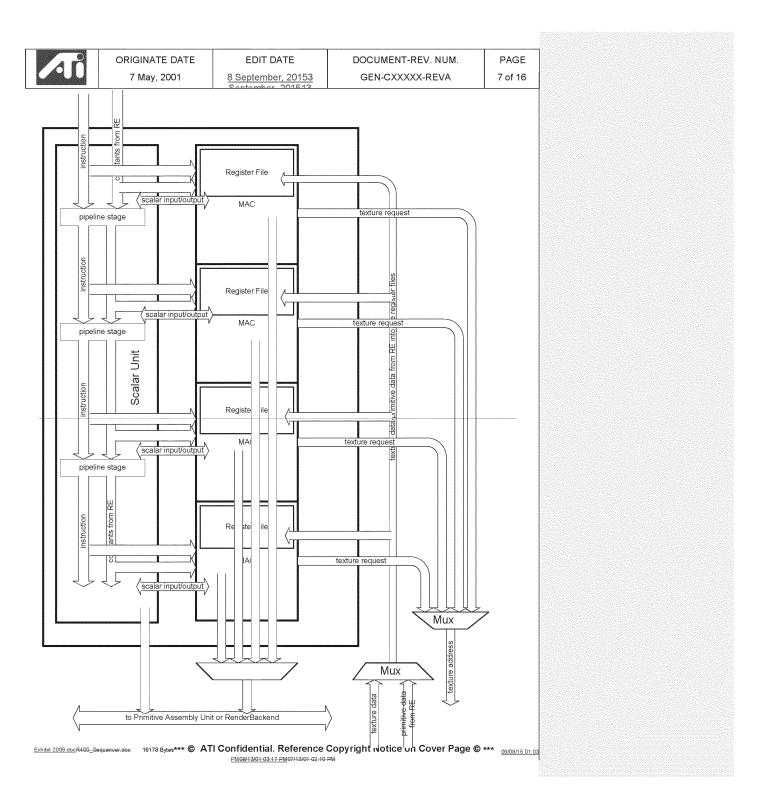
Exhibit 2009.doc P400_Sequences.doc 16178 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** PM08/13/01 03:17 PM07/13/01 02:10 PM

AMD1044_0256677

ATI Ex. 2105 IPR2023-00922 Page 5 of 239



ATI Ex. 2105 IPR2023-00922 Page 6 of 239



ATI Ex. 2105 IPR2023-00922 Page 7 of 239

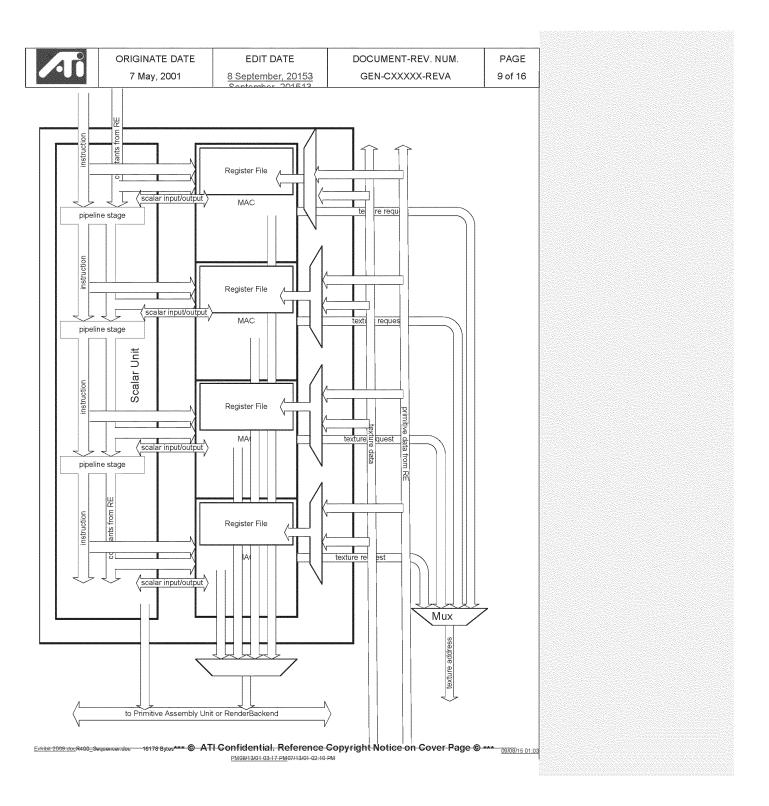
		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
		7 May, 2001	8 September, 20153		8 of 16	
_	10 Data	Elaur araab				

1.2 Data Flow graph

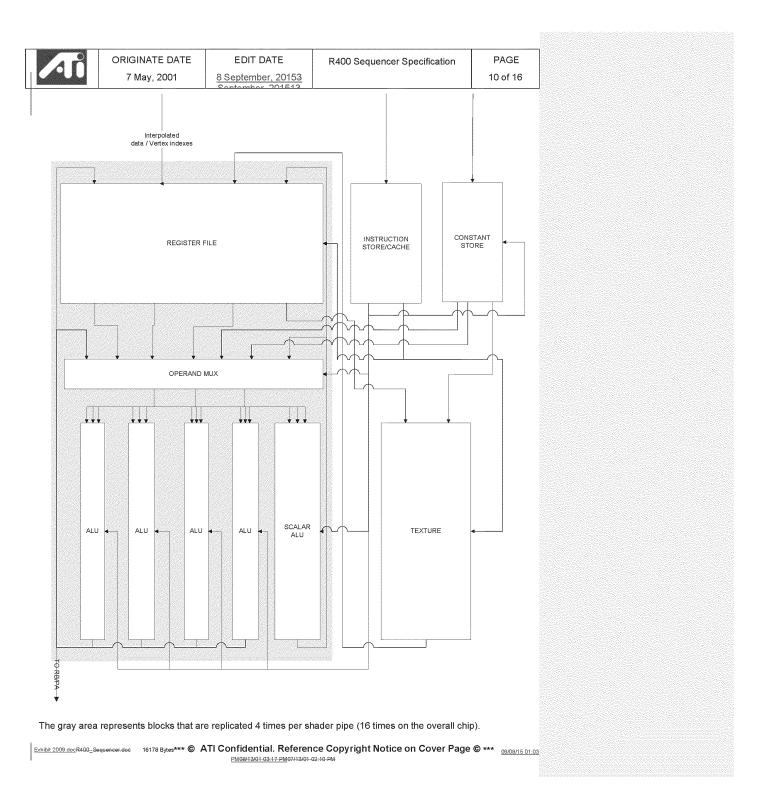
Exhibit 2009.doc.R400_Sequencer.doc 16178 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** <u>PM08/1301-03:17-PM07/13/01-02:10-PM</u>

AMD1044_0256680

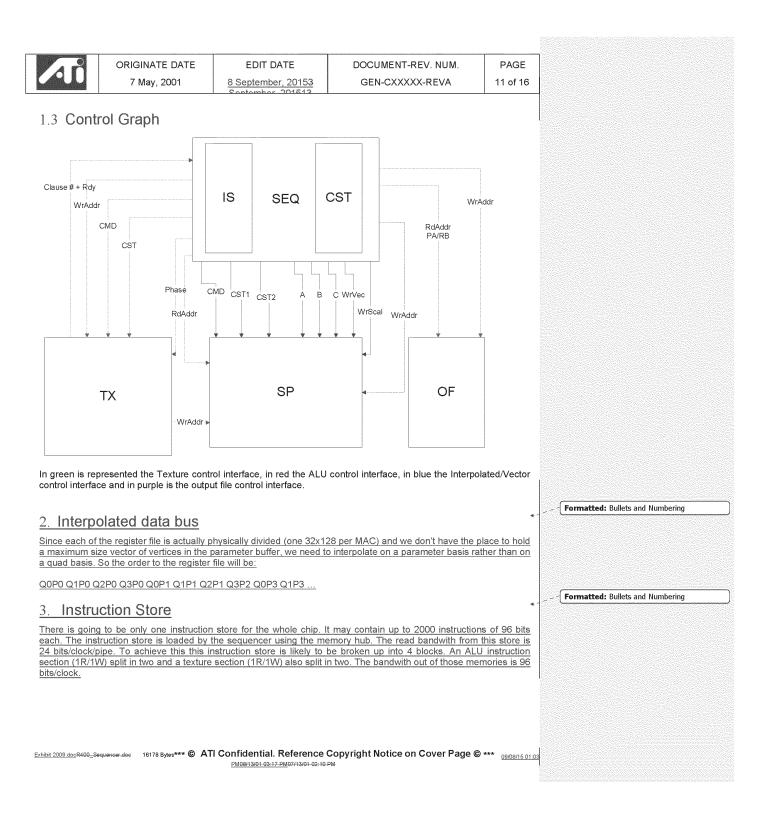
ATI Ex. 2105 IPR2023-00922 Page 8 of 239



ATI Ex. 2105 IPR2023-00922 Page 9 of 239



ATI Ex. 2105 IPR2023-00922 Page 10 of 239



ATI Ex. 2105 IPR2023-00922 Page 11 of 239

_						
		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
		7 May, 2001	8 September, 20153		12 of 16	Formatted: Bullets and Numbering
	4 0				*	

4. 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 512/4 bits/clock/pipe and the write bandwith is 32/4 bits/clock.

5. Looping and Branches

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

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.

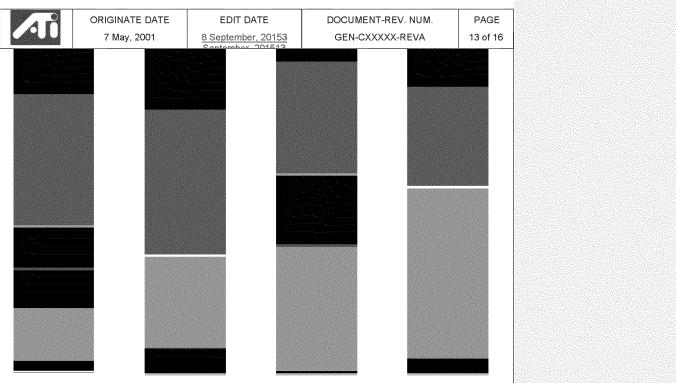
Exhibit 2009.doc R400_Sequencer.doc 16178 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

- - Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044_0256684

ATI Ex. 2105 IPR2023-00922 Page 12 of 239



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.

2.7. Texture Arbitration

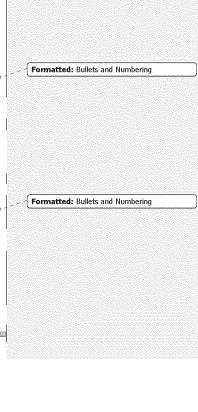
The texture arbitration logic chooses one of the 8 potentially pending texture clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. Once chosen, the clause state machine will send one 2x2 texture fetch per clock (or 4 fetches in one 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 $100 \times (?)$ in flight texture fetches and thus there can be a fair number of active clauses waiting for their texture return data.

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 <u>execute-secute</u>. 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

Exhibit 2009 docR400_Sequencer.doc 16178 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/08/15 01:03 PM08/13/01-03:17-PM07/13/01-02:10-PM



AMD1044_0256685

ATI Ex. 2105 IPR2023-00922 Page 13 of 239

		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE		
		7 May, 2001	8 September, 20153		14 of 16		
Т	one for the old clocks. For exemple, here is the sequencing of two interleaved AI U clauses (F 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

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<u>the packet is a vertex packet and the position buffer is full (POS_FULL)</u> then the sequencer also prevents a thread to enter the exporting clause (<u>4?</u>). 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

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)

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 EIFO. 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

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

7.112.1 External Interfaces

7.1.112.1.1 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 firstblue last
Constant 1	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha firstblue last
Instruction	SEQ-> SP	30	120 bits transferred over 4 cycles (order TBD) ?

7.1.212.1.2 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

Exhibit 2009 docR400_Sequencer.doc 16178 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

PM08/13/01-03:17 PM07/13/01-02:10 PM

AMD1044_0256686

ATI Ex. 2105 IPR2023-00922 Page 14 of 239

		ORIGINATE 7 May, 20		EDIT <u>8 Septem</u>		0 <u>153</u>	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	PAGE 15 of 16
	UL Scalar	Out	SP-> OF		32		Data out	1
	UR_Vector_Out SP-> OF		SP-> OF		128 Vector Data out		Data out	
	UR_Scalar_Out SP-> OF			32	Vector Data out			
	UR_Scalar_Out SP-> OF							

Name	Direction	Bits	Description	
LL_Vector_Out	SP-> OF	128	Vector Data out	100
LL_Scalar_Out	SP-> OF	32	Vector Data out	1996
LR_Vector_Out	SP-> OF	128	Vector Data out	
LR_Scalar_Out	SP-> OF	32	Vector Data out	146

7.1.312.1.3 Shader Engine to Texture Unit Bus (Fast Bus)

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 fileregister 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 FileRegister file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register fileregister 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 <u>Register file</u> s for reading Texture Address
Tex_RegFile_Read_Data	SP->TEX	512	4 Texture Addresses read from the Register FileRegister file
Tex_Write_Register_Index	SEQ->TEX	8	Index into Register fileRegister file for write of returned Texture Data

7.1.4<u>12.1.4</u> Sequencer to Texture Unit bus (Slow Bus)

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 fileregister file where to write the texture return data.

Name	Direction	Bits	Description
Tex_Ready	$TEX \rightarrow SEQ$	1	Data ready
Tex_Clause_Num	$TEX \rightarrow SEQ$	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

7.1.512.1.5 Shader Engine to RE/PA Bus

Name	Direction	Bits	Description
Interpolator_Register_Index	SEQ->SP	8	Index into Register FileRegister 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

ſ	Name	Direction	Bits	Description	1
	Adress	<u>PA</u> →SEQ	?	Dealocation adress sent by the PA telling the Sequencer	4
				that it is now possible to free this space in the parameter	
				buffer. This token is a pointer in the parameter cache and	1000
				4 bits to tell the size wich is to be freed up,	100

Exhibit 2009.docR400_Sequencer.doc 16178 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

PM08/13/01-03:17 PM07/13/01-02:10 PM

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering
Formatted
Formatted
Formatted
Formatted

Formatted: Bullets and Numbering

AMD1044_0256687

ATI Ex. 2105 IPR2023-00922 Page 15 of 239

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	7 May, 2001	8 September, 20153		16 of 16	Formatted: Bullets and Numbering
0.10 0	*			*	

8-1	13.	0	р	e	n	S	S	u	е	S

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:

- 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.

Exhibit 2009 doc R400_Sequencer.doc 16178 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/08/15 01:03 PM08/13/01 02:17 PM07/13/01 02:19 PM

AMD1044_0256688

ATI Ex. 2105 IPR2023-00922 Page 16 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE			
	14 August, 200114	4 September, 201524	GEN-CXXXXX-REVA	1 of 20			
Author:	Laurent Lefebvre						
Issue To:		Copy No:					
R400 Sequencer Specification							
	R400 S	equencer Spe	ecification				
		SEQ					
		Version 0. <u>4</u> 2					
req		cted uses of the block. It a	ncer block (SEQ). It provides an ov also describes the block interfaces,				
Document L		erforce\r400\arch\doc\gfx\RE D Sequencer Specification APPROVALS	E\R400_Sequencer.doc Signature/Date				
Remarks:							
	NTIALLY DETRIME	INTAL TO THE INTE	NFORMATION THAT CO EREST OF ATI TECHNOL SE OR DISCLOSURE.				
work created in unpublished w confidential, pr	n 2001. The use of this cop ork. The copyright notice oprietary information and tr	yright notice is intended to is not an admission that ade secrets of ATI. No parl	erial in this document constitutes ar provide notice that ATI owns a co publication has occurred. This w t of this document may be used, re ssion of ATI Technologies Inc."	pyright in this /ork contains			
Exhibit 2010.docR400_	Sequencer.doc 25584 Bytes*** ⓒ A1	1 Confidential. Reference (PM08/13/01 03:17 PM07/13/01 02:10	Copyright Notice on Cover Page @	•*** <u>09/04/15 12:48</u>			

ATI 2010 LG v. ATI IPR2015-00325

AMD1044_0256689

ATI Ex. 2105 IPR2023-00922 Page 17 of 239



Table Of Contents

1. OVERVIEW
1.1 Top Level Block Diagram
1.2 Data Flow graph
1.3 Control Graph
2. INTERPOLATED DATA BUS 1210
3. INSTRUCTION STORE
4. CONSTANT STORE
5. LOOPING AND BRANCHES 1311
6. REGISTER FILE ALLOCATION 1311
7. TEXTURE ARBITRATION
8. ALU ARBITRATION
9. HANDLING STALLS
10. CONTENT OF THE RESERVATION
STATION FIFOS
11. THE OUTPUT FILE (RB FIFO AND
PARAMETER CACHE)1513
12. INTERFACES
12.1 External Interfaces
12.1.1 Sequencer to Shader
Engine Bus
12.1.2 Shader Engine to Output
File 1543
12.1.3 Shader Engine to Texture
Unit Bus (Fast Bus)1614
12.1.4 Sequencer to Texture Unit bus
(Slow Bus) 1614
12.1.5 Shader Engine to RE/PA Bus
1614
12.1.6 PA? to sequencer
13. EXAMPLES OF PROGRAM
EXECUTIONS
13.1.1 Sequencer Control of a Vector
of Vertices 1744
13.1.2 Sequencer Control of a Vector
of Pixels 1816
<u>13.1.3</u> Notes
<u>14. OPEN ISSUES1947</u>
1. OVERVIEW
1.1 Top Level Block Diagram
1.2 Data Flow graph
1.3 Control Graph

2. INTERPOLATED DATA BUS
3. INSTRUCTION STORE
4. CONSTANT STORE
5. LOOPING AND BRANCHES11
6. REGISTER FILE ALLOCATION11
7. TEXTURE ARBITRATION 12
8. ALU ARBITRATION
9. HANDLING STALLS
10. CONTENT OF THE RESERVATION
STATION FIFOS13 11. THE OUTPUT FILE (RB FIFO AND
PARAMETER CACHE)13
12. INTERFACES
12.1 External Interfaces
12.1.1 Sequencer to Shader
Engine Bus13
12.1.2 Shader Engine to Output
File13
12.1.3 Shader Engine to Texture
Unit Bus (Fast Bus)
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
13. OPEN ISSUES
1. OVERVIEW
1.1 Top Level Block Diagram
2. TEXTURE ARBITRATION
3. ALU ARBITRATION
4. HANDLING STALLS
5. CONTENT OF THE RESERVATION
STATION FIFOS10
6. INTERFACES10
6.1 External Interfaces 10
6.1.1 Sequencer to Shader Engine
Bus 10
6.1.2 Shader Engine to Output File
<u>10</u>
6.1.3 Shader Engine to Texture Unit
Bus (Fast Bus)10
· /

Exhibit 2010.doc/R400_Sequencer.doc 25584 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***
<u>PM08//3/01-03:47-PM</u>07/13/01-02:10-PM

AMD1044_0256690

ATI Ex. 2105 IPR2023-00922 Page 18 of 239

onionionionionionionionionionionionionio			1	7
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	14 August, 200144	4 September, 201524	GEN-CXXXXX-REVA	3 of 20
6 <u>14</u> s	Sequencer to Texture L		OPEN ISSUES	
)			
· · ·	, Shader Engine to RE/P			
0.x.0 0				
D	0			
Revision	Changes:			
Rev 0.1 (Laure		First dra	aft.	
Date: May 7, 2	001			
Rev 0.2 (Laure	nt Lefebvre)	Change	ed the interfaces to reflect the change	s in the
Date : July 9, 2	2001	SP. Add	ded some details in the arbitration sec	tion.
Rev 0.3 (Laure Date : August 6			ed the Sequencer spec after the mee 3, 2001.	eting on
Rev 0.4 (Laure	nt Lefebvre)	Added	the dynamic allocation method for	
Date : August 2	24, 2001		an example (written in part by Vic) pixels/vertices in the sequencer.	of the
		Autoral Subject Subject	lanta katu katu katu katu katu katu katu ka	

Exhibit 2010 doc/R400_Sequencer.doc 25584 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 009/04/15 12:48

AMD1044_0256691

ATI Ex. 2105 IPR2023-00922 Page 19 of 239

		EDIT DATE	R400 Sequencer Specification	PAGE	
22/12/02/02/02/02/02/02/02/02/02/02/02/02/02	4 August, 200114	4 September, 201524		4 of 20	

1. Overview

The sequencer first arbitrates between vectors of 16-(maybe 32) vertices that arrive directly from primitive assembly and vectors of 84-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 aa clause before looking for a new clause of the same type. <u>Two ALU clauses are executed interleaved to hide the ALU latency</u>. 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 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. <u>There are two separate sets of reservation stations, one for pixel vectors</u> 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.

Exhibit 2010 doc R400_Sequencer.doc 25584 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:48 PM08/43/01-02:17-PM07/13/01-02:10-PM

AMD1044_0256692

ATI Ex. 2105 IPR2023-00922 Page 20 of 239

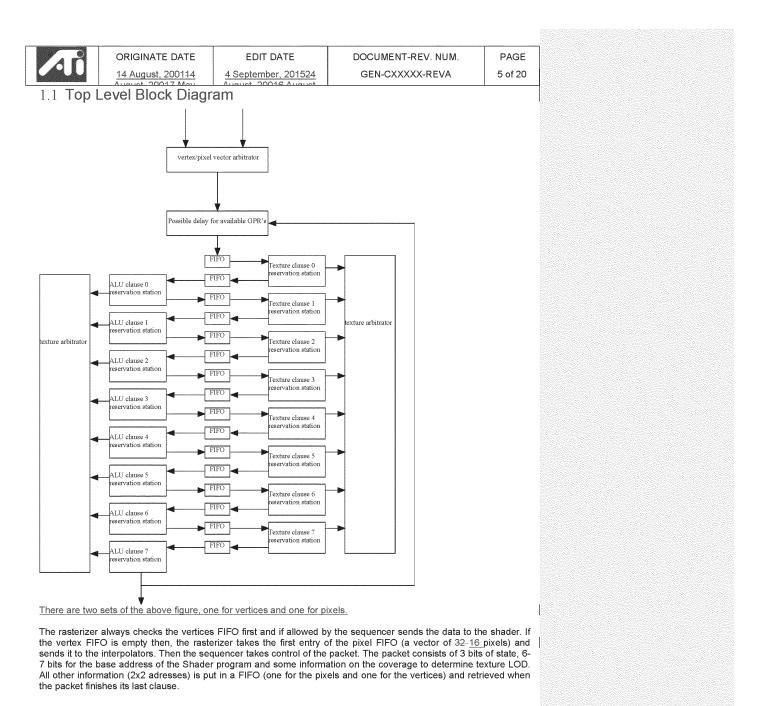


Exhibit 2010 doc R400_Sequencer.doc 25684 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:48

AMD1044_0256693

ATI Ex. 2105 IPR2023-00922 Page 21 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
14 August, 200114	4 September, 201524		6 of 20

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 (tomd) is passed to the texture system identifying the current level number (0) as well as the register set being usedwrite 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 tend 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 ene-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 <u>one-two</u> ALU state machine may have access to the <u>SRAMregister file</u> address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the <u>SRAMregister file</u> address bus at one time. Arbitration is performed by <u>two-three</u> arbitrer blocks (<u>one-two</u> 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 <u>SRAMregister Sfiles</u>.

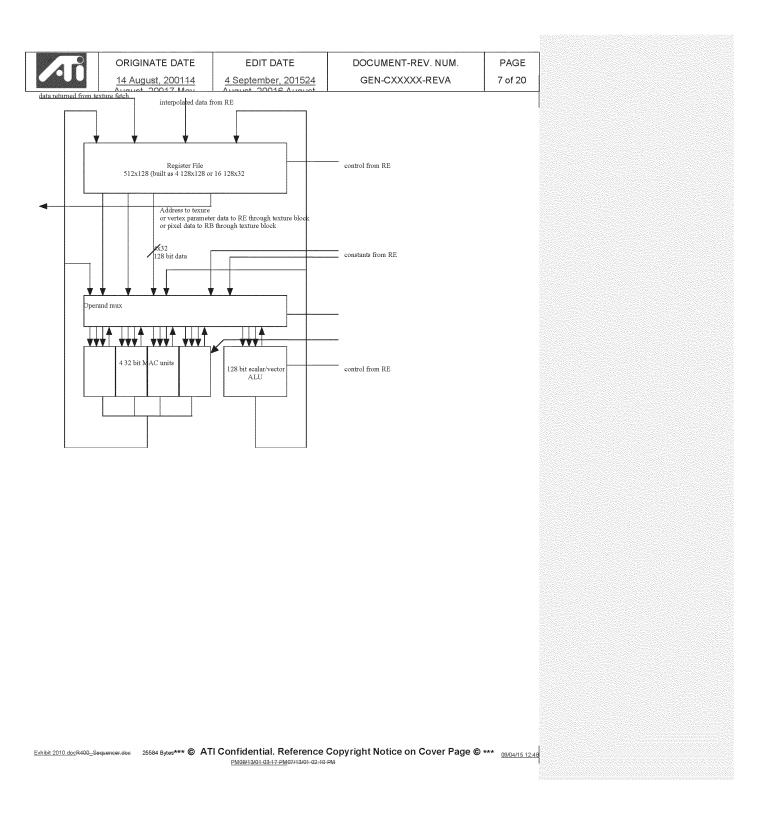
Each state machine maintains an address pointer specifying where the 16-(or-32) entries vector is located in the <u>SRAMregister file</u> (the texture machine has two pointers one for the read address and one for the write). Upon completion of its job, the address pointer is incremented by a predefined 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 SRAMregister file memory is available

Exhibit 2010.docR400_Sequencer.doc 25584 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 00/04/15 12:48

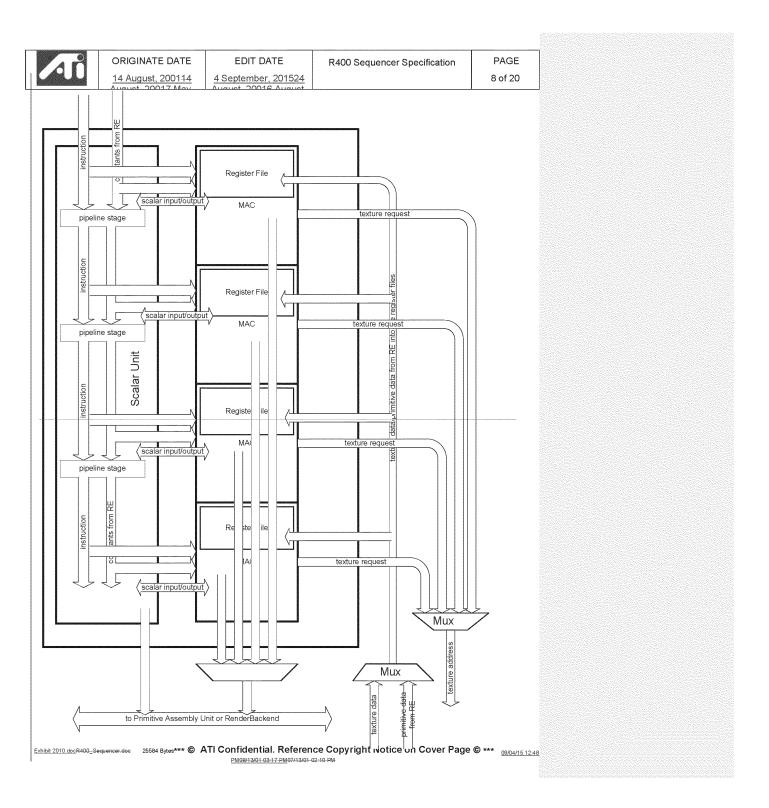
Formatted

AMD1044_0256694

ATI Ex. 2105 IPR2023-00922 Page 22 of 239



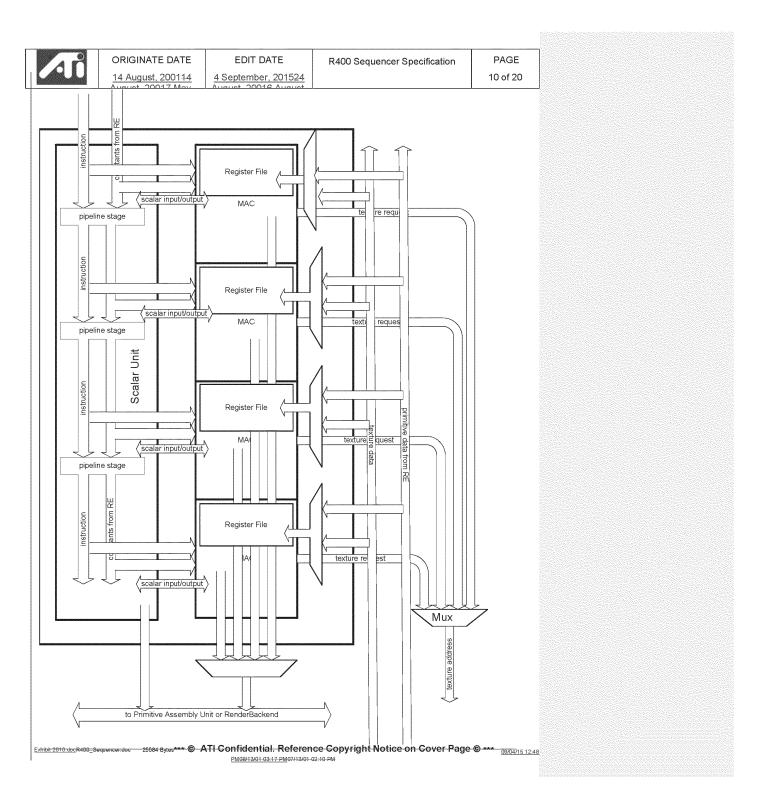
ATI Ex. 2105 IPR2023-00922 Page 23 of 239



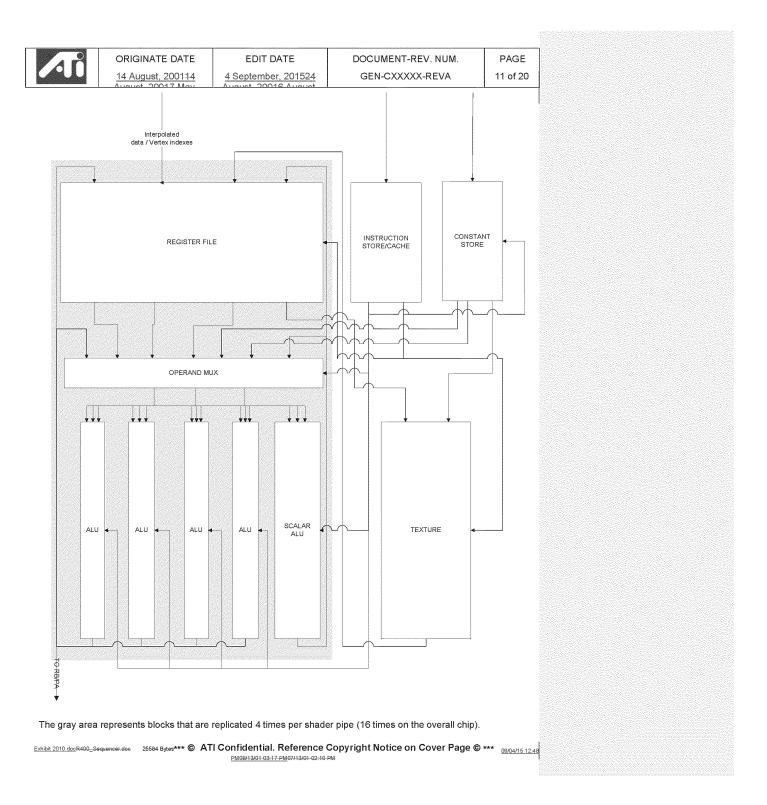
ATI Ex. 2105 IPR2023-00922 Page 24 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE]
	14 August, 200114	4 September, 201524	GEN-CXXXXX-REVA	9 of 20	
1.2 Data I	Flow graph				
	~		opyright Notice on Cover Page ©		
bit 2010.docR400_Seq	uencer.doc 25584 Bytes*** 🛈 🗛 🕇	I Confidential. Reference Confidential. Reference Confidential. Reference Confidential. PM08/13/01-02:10 PM	opyright Notice on Cover Page ©	*** 00/04/15 12:4	a

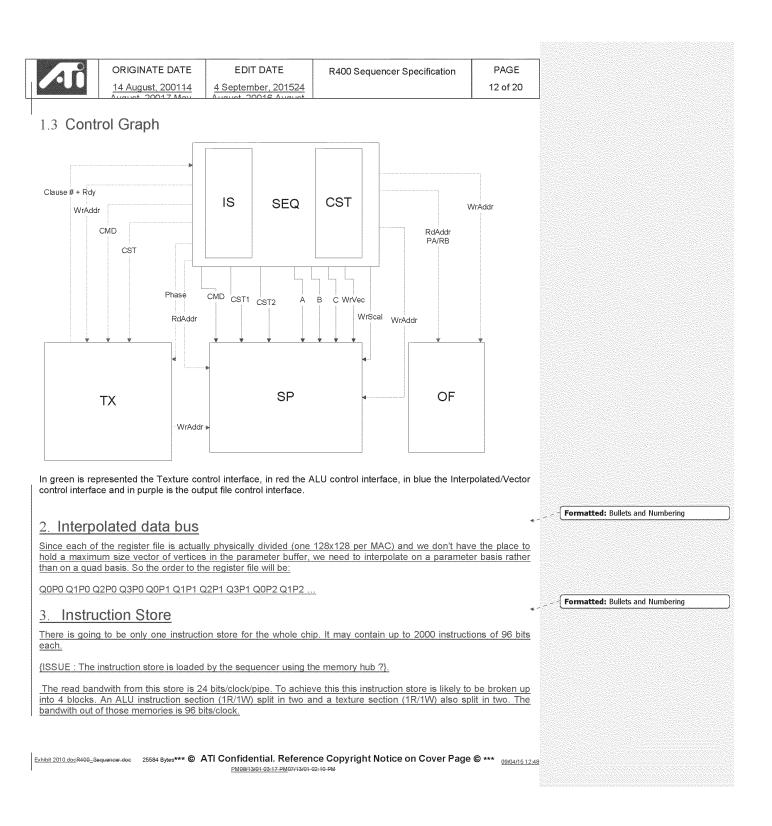
ATI Ex. 2105 IPR2023-00922 Page 25 of 239



ATI Ex. 2105 IPR2023-00922 Page 26 of 239



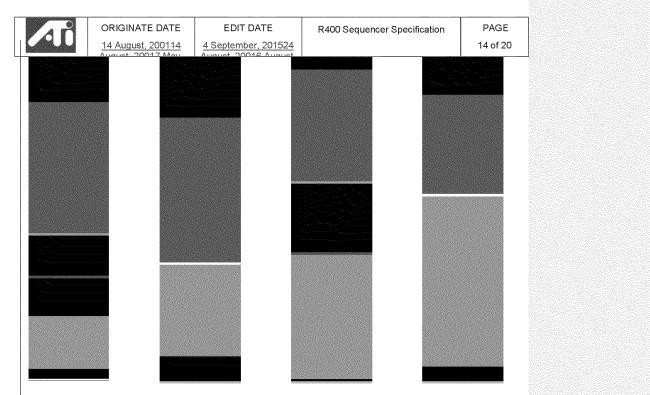
ATI Ex. 2105 IPR2023-00922 Page 27 of 239



ATI Ex. 2105 IPR2023-00922 Page 28 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	14 August, 200144	4 September, 201524	GEN-CXXXXX-REVA	13 of 20	Formatted: Bullets and Numbering
4. Consta	ant Store			*	
			e of where the constants are using		
			512x128 bits. The constant store is a /pipe and the write bandwith is 32/4 b		
" Landi	a and Descelars			*	
	ng and Branches				
		upported and will have to be dependent branches or not.	e dealt with at the sequencer level. He	owever, it is	
6 Decict	or file allocation			*	Formatted: Bullets and Numbering
	er file allocation	ad afaala aana afabaan ka adad		uinten film im	
			ic or dynamic. In both cases, the re- es). In the dynamic case the bounda		
	tices is allowed to move, G_SIZE for pixels.	in the static case it is fixed	I to VERTEX REG_SIZE for vertice	s and 256-	
		Confidential Deferences	Conversion to Notice on Course Door to	anta E	
Exhibit 2010.docR400_Se	iquencer.doc 25584 Bytes*** 𝔤 AI	PM08/13/01_03:17_PM07/13/01_02:10-	Copyright Notice on Cover Page ©	09/04/15 12:48	

ATI Ex. 2105 IPR2023-00922 Page 29 of 239



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.

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 $400 \underline{X}(?)$ in flight texture fetches and thus there can be a fair number of active clauses waiting for their texture return data.

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 <u>executeexecute. If the packet chosen is a packet of vertices, the state machine issues</u> 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

Exhibit 2010 doc R400_Sequencer.doc 25584 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:48
PM08/13/01-02:17-PM07/13/01-02:17-PM

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044_0256702

ATI Ex. 2105 IPR2023-00922 Page 30 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	14 August, 200114	4 September, 201524	GEN-CXXXXX-REVA	15 of 20
one for the odd			terleaved ALLI clauses (F and O stand	de 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

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

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)

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



7.112.1 External Interfaces

7.1.112.1.1 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 firstblue last
Constant 1	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha firstblue last
Instruction	SEQ-> SP	30	120 bits transferred over 4 cycles (order TBD) ?

7.1.212.1.2 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	100	
UL_Vector_Out	SP-> OF	128	Vector Data out] []	

Exhibit 2010 docR400_Sequencer.doc 25584 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

PM08/13/01-03:17 PM07/13/01-02:10 PM

Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering

AMD1044_0256703

ATI Ex. 2105 IPR2023-00922 Page 31 of 239

	ORIGINATE D	0114	EDIT D	r, 2015		R400 Sequencer Specification	PAGE 16 of 20
UL_Scalar	_Out	SP-> C	١F	32	Vec	tor Data out	
UR_Vector	_Out	SP-> C	١F	128	Vec	tor Data out	
UR_Scalar	_Out	SP-> C	١F	32	Vec	tor 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

7.1.312.1.3 Shader Engine to Texture Unit Bus (Fast Bus)

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 fileregister 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 FileRegister file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register fileregister 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 FileRegister files for reading Texture Address
Tex_RegFile_Read_Data	SP->TEX	512	4 Texture Addresses read from the Register FileRegister
Tex_Write_Register_Index	SEQ->TEX	8	Index into Register fileRegister file for write of returned Texture Data

7.1.412.1.4 Sequencer to Texture Unit bus (Slow Bus)

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 fileregister file where to write the texture return data.

Name	Direction	Bits	Description
Tex_Ready	$TEX \rightarrow SEQ$	1	Data ready
Tex_Clause_Num	$TEX \rightarrow SEQ$	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

7.1.512.1.5 Shader Engine to RE/PA Bus

Name	Direction	Bits	Description
Interpolator_Register_Index	SEQ->SP	8	Index into Register FileRegister 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

Name	Direction	Bits	Description
Adress	<u>PA→SEQ</u>	?.	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,

Exhibit 2010 doc P400_Sequencer.doc 25584 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12.48

PM08/13/01-03:17-PM07/13/01-02:10 PM

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted Formatted Formatted

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted

AMD1044_0256704

ATI Ex. 2105 IPR2023-00922 Page 32 of 239

	T	r		1				
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE				
CVUU	14 August, 200114	4 September, 201524	GEN-CXXXXX-REVA	17 of 20	Formatted: Bullets and Numbering			
<u>13. Exar</u>	nples of program	executions		*				
12.1.1.0								
13.1.1 50	equencer Control of	a vector of vertices						
 state space also shad The v the state 	a vector of 16 vertices (actu pointer as well as tag into po- e was allocated in the positic before the vector is sent to er program (using the MH' ertex program is assumed to be SEQ then accesses the IS equencers by the RBBM who							
 at this the at 	 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). 							
**********			PRs used by the program					
state	 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 							
the 1 the 1	Is the vector to the SP regis 5 vertex indices are sent to t F0 of SU0, SU1, SU2, and s F1 of SU0, SU1, SU2, and s F2 of SU0, SU1, SU2, and s F3 of SU0, SU1, SU2, and s dex is written to the least sig							
			<u>g data bits are set to zero (x, y, z)</u> e first reservation station (the FIFO in f	front of				
texture st • the co								
	cepts the control packet and) was first selected by the TS							
7. all instruc	tions of texture clause 0 are	issued by TSM0						
8. the contro FIFO)	I packet is passed to the ne	ext reservation station (the F	IFO in front of ALU state machine 0, c	or ASMO				
TSM0 for th once 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 							
	cepts the control packet (after rom the global instruction st		M arbiter) and gets the instructions for	ALU				
	tions of ALU clause 0 are is a FIFO in front of texture sta		ntrol packet is passed to the next rese O)	rvation				
 positi share A par going 	 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 							
Exhibit 2010.docR400_S	equences.doc 25584 Bytes*** © AT	1 Confidential. Reference PM08/13/01-03:17-PM07/13/01-02:10	Copyright Notice on Cover Page ©	*** <u>09/04/15 12:48</u>				

ATI Ex. 2105 IPR2023-00922 Page 33 of 239

ORIGINATE DATE EDIT DATE R400 Sequencer Specification PAGE 14 August, 200114 4 September, 201524 18 of 20 • the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full • parameter data is sent to the Parameter Cache over a decleaded bus • the SQ allocates storage in the Parameter Cache, and the SC deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token). • the SQ allocates storage in the Parameter Cache over a decleaded bus 14 as and exported in clause fit the parameter cache (or the position buffer if position is being exported) is full 12 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 13.1.2 Sequencer Control of a Vector of Pixels 14 4.5 the R1 is point it is assumed that the pixel program is loaded into the global instruction store by the CP • At this point it is assumed that the pixel program is loaded into the global instruction store by the CP • At this point it is assumed that the pixel program is loaded into the parameter data to the vector • after the HZ culing stage a request is made by the RE to send parameter data to the Parameter buffer • the R1 is valid stage a request will free up the parameter store locations not used anymore using the token provided by the PA 3. the R1's Pizel FIFO is loaded with the barycentricic coordinates for	10000000000000000000000000000000000000	1	1			
Advantable applied the second se		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
 the ASM arbiter will prevent a packet from starting an exporting clause if the position case if it was not exported inclause? (as well as position data if was not exported earlier) parameter data is sent to 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 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 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 global instruction store by the CP At this point it is assumed that the pixel program is loaded into the global instruction store by the CP at the the Aculting stage a request is made by the RE to send parameter data to the Parameter buffer the RE's Parameter Buffer is loaded from the Parameter Cache before the SEQ takes control of the vector at this moment the right sequencer will free up the parameter store locations not used anymore using the token provided by the PA. the RE's Parameter Putifer is loaded with the barcentric coordinates for pixel guads by the detailed walker the state pointer and the LOD correction bits are also placed in the Pixel FIFO. the state pointer and the LOD correction bits are also placed in the Pixel RE' the RE's Parameter Pixel FIFO and Vertex FIFO – when there are no vertices pending OR there is no space left in the register files for a data to be sent to the shader program the number of GPR's required by the program is stored in a local state registe	M				18 of 20	
 the state pointer and the LOD correction bits are also placed in the Pixel FIFO the Pixel FIFO is wide enough to source one quad's worth of barycentrics per cycle 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 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 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 POi, POi, and POk (the value of PO at each vertex) are loaded into the interpolator from the Parameter buffer QO i, j, and k are loaded into the interpolator from the Pixel FIFO The interpolator then generates the parameter value for each pixel in QO (QOPO) POi, POi, and POk 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 POi, POi, and POk 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 value for each pixel in Q1 (Q1PO) 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 (Q1PO) Q2PO and Q3PO are generated in a similar manner The interpolator then generates the parameter value for each pixel in Q1	paran p p p p t t p t	A Market 2004.7 May be ASM arbiter will preven neter data is exported in c arameter data is sent to th the SEQ allocates storage unger a need for the paran per a need for the paran position is being exported shader program has comp ogram equencer Control c rertex shader programs, s point it is assumed that the Parameter Buffer is loaded the HZ culling stage a require arameter buffer is wide er s moment the right sequire of the PA	t a packet from starting an lause 7 (as well as positior ie Parameter Cache, an neters (it is told by the PA v t a packet from starting on) is full leted, the SEQ will free up of a Vector of Pixels pixel shaders are loaded he pixel program is loaded from the Parameter Cach uest is made by the RE to so ough to source 3 vertices v lencer will free up the para	a data if it was not exported earlier) a dedicated bus nd the SEQ deallocates that space why when using a token). ASM7 if the parameter cache (or the p the GPRs so that they can be used by the GPRs so that they can be used by into the global instruction store by into the instruction store and thus rea be before the SEQ takes control of the send parameter data to the Parameter worth of a particular parameter in one rameter store locations not used an	FIFO is full en there is no position buffer another the CP dy to be read. vector buffer cycle ymore using	
 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 POi, POi, and POk (the value of PO at each vertex) are loaded into the interpolator from the Parameter buffer Q0 i, j, and k are loaded into the interpolator for Mthe Pixel FIFO The interpolator then generates the parameter value for each pixel in Q0 (QOPO) POi, POj, and POk 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 POj, POj, and POk 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 value for each pixel FIFO The interpolator then generates the parameter value for each pixel in Q1 (Q1PO) Q2PO and Q3PO are generated in a similar manner The next set of parameter data - P1i, P1i, 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 	the state pointer and the LOD correction bits are also placed in the Pixel FIF0					
 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 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, P0i, 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 value for each 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, P1i, 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 						
 coordinates from the Pixel FIFO and the parameters from the Parameter Buffer to the interpolator P0i, P0i, 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, P1i, 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 	• the nu state	umber of GPRs required b pointer	y the program is stored in	a local state register, which is accesse		
7. SEQ controls the transfer of interpolated data to the SP register file over the RE SP interface (which has a	coordinate P0i, F Q0i, The ii P0i, F from interr Q1i, Q1i, Q1i, Q1i, Q1i, Q1i, Q1i, Q1i, Q1i, Q2PC The ii Q2PC The right Q0i, throw causi so the	es from the Pixel FIFO an POj, and POk (the value of i, and k are loaded into the heterpolator then generates POj, and POk are sent to t the same primitive as Q bolator and reused for Q "different prim" contro nat indicates whether ne i, and k are then loaded in heterpolator then generates and Q3PO are generated ext set of parameter data i, and k now must be re-re- gh the top four entries on ing the top four sets of bar e order of parameter info g	d the parameters from the PO at each vertex) are load a interpolator from the Pixe the parameter value for ex- the interpolator for Q1 on 0, then the POi, POj, and F 1 bit is passed with the bi- w parameter data needs to the interpolator from the the parameter value fore in a similar manner - P1i, P1j, and P1k - is the each read command until a generated is QOPO, Q1PO,	Parameter Buffer to the interpolator led into the interpolator from the Parar I FIFO ach pixel in Q0 (QOPO) Ily if Q1 is from a different primitive; POk values loaded for Q0 are held by arycentric data for each quad in the to be loaded into the interpolator Pixel FIFO ach pixel in Q1 (Q1PO) n loaded into the interpolator his means that the output of the Pixel f at the end a final "block pop" signal is Ily be removed Q2PO, Q3PO, QOP1, Q1P1, etc.	if Q1 is the Pixel FIFO	
	Exhibit 2010.docR400_S	equencer.doc 25584 Bytes*** © /	ATI Confidential. Referent	ce Copyright Notice on Cover Page	© *** <u>09/04/15 12:48</u>	

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
14 August, 200114	4 September, 201524	GEN-CXXXXX-REVA	19 of 20	
8. SEQ constructs a control packet for th		first reservation station (the FIFO in fi	ront of	-
	f reservation stations/arbite ate pointer, the register file t	rs/state machines for vertices and for pase pointer, and the LOD correction t vels in a separate FIFO		
9. TSM0 accepts the control packet and • TSM0 was first selected by the TS			ction store	
10. all instructions of texture clause 0 are	issued by TSM0			
11. the control packet is passed to the ne FIFO)	xt reservation station (the F	FO in front of ALU state machine 0, o	r ASMO	
TSM0 does not wait for texture re index for the texture data to the T once the TU has written all the data	U, which will write the data t ta for a particular clause to a count greater than zero inc	o the RF as it is received the register files, it increments a count dicates that the ALU state machine ca	ter that is	
12. ASM0 accepts the control packet (after clause 0 from the global instruction struction struc		A arbiter) and gets the instructions for	ALU	
13. all instructions of ALU clause 0 are iss station (the FIFO in front of texture sta			vation	
14. the control packet continues to travel • pixel data is exported in the last A • it is sent to an output FIFO w • the ASM arbiter will prevent a	LU clause (clause 7) here it will be picked up by t	he render backend	executed	
<u>15. after the shader program has complet</u> shader program	ed, the SEQ will free up the	GPRs so that they can be used by ar	nother	
<u>13.1.3 Notes</u>			4	Formatted: Bullets and Numbering
16. the state machines and arbitrers will c threads or stall.	perate ahead of time so tha	t they will be able to immediately star	the real	
<u>17.</u> the register file base pointer for a vect instruction store base pointer does no pointer is only different for each state	t – this is because the RF p	ointer is different for all threads, but th		
18. Waterfalling, parameter buffer alloca be specked out.	tion, loops and branches ar	d parameter cache de-allocation still i	needs to	
8.14. Open issues				
bandwith there to operate. Th	o feed the ALUs. Two soluti case and put those instru is requires a significant amo ing only at a given time the	ons exists for this problem: ictions in a texture clause so we ca bunt of temporary storage in the regist vertices having the same constants to	an use the er store.	
Need to do some testing on the size of the method (dynamic VS static).	e register file<u>register</u> file as	well as on the register file <u>register file</u>	allocation	
Ability to export at any clause?				
Saving power?				
Are we working on 32 vertices at a time of	- 16?			
Exhibit 2010.docR400_Sequencer.doc 25584 Bytes*** ⓒ AT	Confidential. Reference	Copyright Notice on Cover Page © - ™	*** <u>09/04/15 12:48</u>	

ATI Ex. 2105 IPR2023-00922 Page 35 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE]
14 August, 200114	4 September, 201524		20 of 20	

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.

Exhibit 2010.docR400_Sequencer.doc 25584 Bytes***
ATI Confidential. Reference Copyright Notice on Cover Page

Og/04/15 12:48
PM08/13/01-02:10-PM

AMD1044_0256708

ATI Ex. 2105 IPR2023-00922 Page 36 of 239

		1	
ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
14 August, 200144	4 September, 20157	GEN-CXXXXX-REVA	1 of 26
Author: Laurent Lefebvre			
	,		
Issue To:	Copy No	:	
R400 S	equencer Sp	ecification	
	adament ab.	alle, alle, x x x alle, aller all x alle, x x	
	SEQ		
	SEQ		
	14		
	Version 0. <u>5</u> 2		
Overview: This is an architectural specifi	cation for the R400 Sequer	ncer block (SEQ). It provides an ove	erview of the
required capabilities and expendious blocks, and provides internal s		also describes the block interfaces,	internal sub-
AUTOMATICALLY UPDATED FIELDS:			
Document Location: C:\p	erforce\r400\arch\doc\gfx\RE	E\R400_Sequencer.doc	
Current Intranet Search Title: R40	0 Sequencer Specification APPROVALS		
Name/Dept		Signature/Date	
Remarks:	I		
THIS DOCUMENT CONTAIN			
SUBSTANTIALLY DETRIME			OGIES
INC. THROUGH	UNAUTHORIZED U	SE OR DISCLOSURE.	
"Conversion 2004 ATL Technologies to a	Il vishte vecenced The moto	vial in this desument and the	unnuh Kaba -
"Copyright 2001, ATI Technologies Inc. A work created in 2001. The use of this co	pyright notice is intended to	provide notice that ATI owns a cop	pyright in this
unpublished work. The copyright notice confidential, proprietary information and to	rade secrets of ATI. No par	t of this document may be used, re	
transmitted in any form or by any means w	ithout the prior written permi	ssion of ATI Technologies Inc."	
Exhibit 2011.docR400_Sequencer.doc 31302 Bytes*** @ A	TI Confidential. Reference	Copyright Notice on Cover Page ©) *** <u>09/04/15 12:48</u>
	PM08/13/01-03:17 PM07/13/01-02:10	PM	I

ATI 2011 LG v. ATI IPR2015-00325

AMD1044_0256709

ATI Ex. 2105 IPR2023-00922 Page 37 of 239



Table Of Contents

1. OVERVIEW43
1.1 Top Level Block Diagram
1.2 Data Flow graph
1.3 Control Graph
2. INTERPOLATED DATA BUS 1210
3. INSTRUCTION STORE 1210
4. CONSTANT STORE 1311
5. LOOPING AND BRANCHES 1311
6. REGISTER FILE ALLOCATION 1311
7. TEXTURE ARBITRATION 1412
8. ALU ARBITRATION
9. HANDLING STALLS
10. CONTENT OF THE RESERVATION
STATION FIFOS1513
11. THE OUTPUT FILE (RB FIFO AND
PARAMETER CACHE)1513
12. INTERFACES
12.1 External Interfaces
12.1.1 Sequencer to Shader
Engine Bus
12.1.2 Shader Engine to Output
File 1513
12.1.3 Shader Engine to Texture
Unit Bus (Fast Bus)
<u>12.1.4</u> Sequencer to Texture Unit bus
<u>(Slow Bus) 1614</u>
12.1.5 Shader Engine to RE/PA Bus
1614
12.1.6 PA? to sequencer
13. EXAMPLES OF PROGRAM
EXECUTIONS
13.1.1 Sequencer Control of a Vector
of Vertices 1744
13.1.2 Sequencer Control of a Vector
of Pixels 18 16
<u>13.1.3</u> Notes
14. OPEN ISSUES
1. OVERVIEW
1.1 Top Level Block Diagram
1.2 Data Flow graph7
1.3 Control Graph10

2. INTERPOLATED DATA BUS
3. INSTRUCTION STORE
4. CONSTANT STORE
5. LOOPING AND BRANCHES11
6. REGISTER FILE ALLOCATION11
7. TEXTURE ARBITRATION
8. ALU ARBITRATION
9. HANDLING STALLS
10. CONTENT OF THE RESERVATION
STATION FIFOS13 11. THE OUTPUT FILE (RB FIFO AND
PARAMETER CACHE)13
12. INTERFACES
12.1 External Interfaces
12.1.1 Sequencer to Shader
Engine Bus13
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
13. OPEN ISSUES 14
1. OVERVIEW
1.1 Top Level Block Diagram
2. TEXTURE ARBITRATION
3. ALU ARBITRATION
4. HANDLING STALLS
5. CONTENT OF THE RESERVATION
STATION FIFOS
6. INTERFACES
6.1 External Interfaces
6.1.1 Sequencer to Shader Engine
Bus 10
6.1.2 Shader Engine to Output File
6.1.3 Shader Engine to Texture Unit
Bus (Fast Bus)

Exhibit 2011.docR400_Sequencer.doc 31302 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***
<u>PM08/13/01-02-10-PM</u>

AMD1044_0256710

ATI Ex. 2105 IPR2023-00922 Page 38 of 239

		<u>z</u>	r	
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	14 August, 200114	4 September, 20157	GEN-CXXXXX-REVA	3 of 26
6.1.4 0	Sequencer to Texture L		OPEN ISSUES	
)			
0.1.3-8	Shader Engine to RE/P	A-BUS+		
Revision	Changes:			
Rev 0.1 (Laure		First dra	aft.	
Date: May 7, 2	2001			
Rev 0.2 (Laure	ant Lefebyre)	Change	d the interfaces to reflect the change	es in the
Date : July 9, 2			led some details in the arbitration se	
Rev 0.3 (Laure			ed the Sequencer spec after the me	<u>eting on</u>
Date : August (3, 2001.	
Rev 0.4 (Laure Date : August 2			the dynamic allocation method for an example (written in part by Vic	
Date . August .	24, 2001		pixels/vertices in the sequencer.	.) of the
Rev 0.4 (Laure	ent Lefebvre)		iming diagrams (Vic)	
Date : Septeml	<u>ber 7, 2001</u>			

Exhibit 2011 docR400_Sequencer.doc 31302 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 00/04/15/12/48 PM08/13/01-02:17-PM07/13/01-02:10-PM

AMD1044_0256711

ATI Ex. 2105 IPR2023-00922 Page 39 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
14 August, 200114	4 September, 20157		4 of 26	

1. Overview

The sequencer first arbitrates between vectors of 16 (maybe 32) vertices that arrive directly from primitive assembly and vectors of 84-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 aa clause before looking for a new clause of the same type. <u>Two ALU clauses are executed interleaved to hide the ALU latency</u>. 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 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. <u>There are two separate sets of reservation stations, one for pixel vectors</u> 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.

Exhibit 2011.docR400_Sequencer.doc 31302 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:48 PM08/13/01-02:10-PM

AMD1044_0256712

ATI Ex. 2105 IPR2023-00922 Page 40 of 239

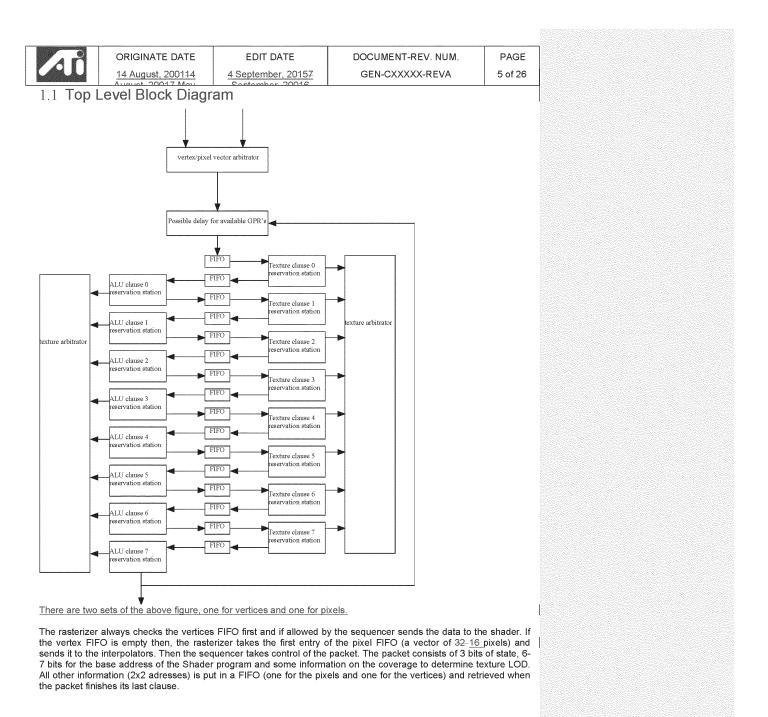


Exhibit 2011 doc R400_Sequencer.doc 31302 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15 12:48

AMD1044_0256713

ATI Ex. 2105 IPR2023-00922 Page 41 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
14 August, 200114	4 September, 20157		6 of 26

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 usedwrite 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 tord 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 ene_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 <u>one-two</u> ALU state machine may have access to the <u>SRAMregister file</u> address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the <u>SRAMregister file</u> address bus at one time. Arbitration is performed by <u>two-three</u> arbitrer blocks (<u>one-two</u> 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 <u>SRAMregister Sfiles</u>.

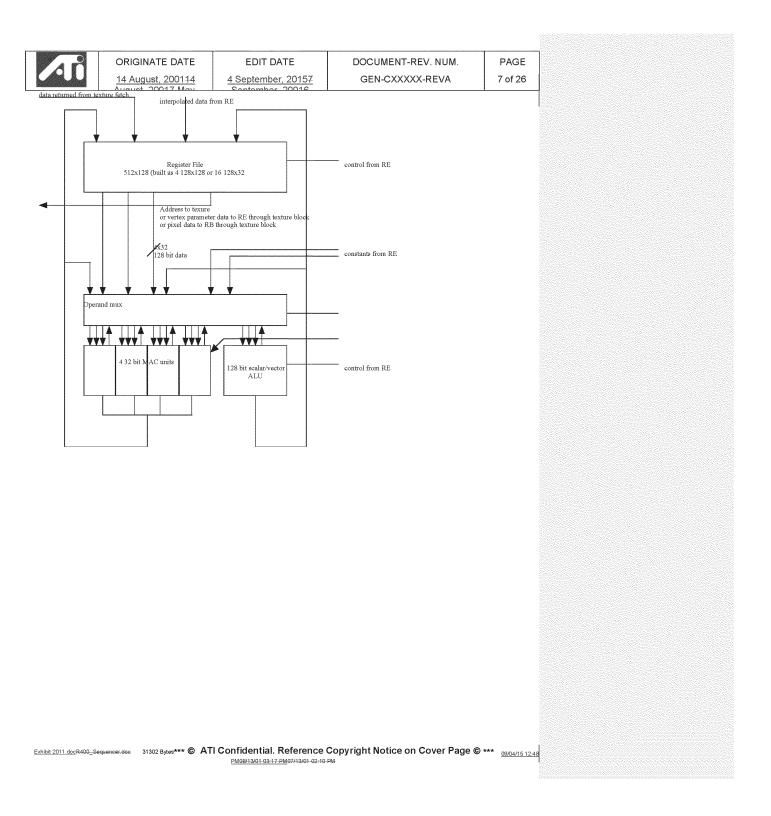
Each state machine maintains an address pointer specifying where the 16-(or-32) entries vector is located in the <u>SRAMregister file</u> (the texture machine has two pointers one for the read address and one for the write). Upon completion of its job, the address pointer is incremented by a predefined 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 SRAMregister file memory is available

Exhibit 2011.docR400_Sequencer.doc 31302 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 00/04/15 12:48

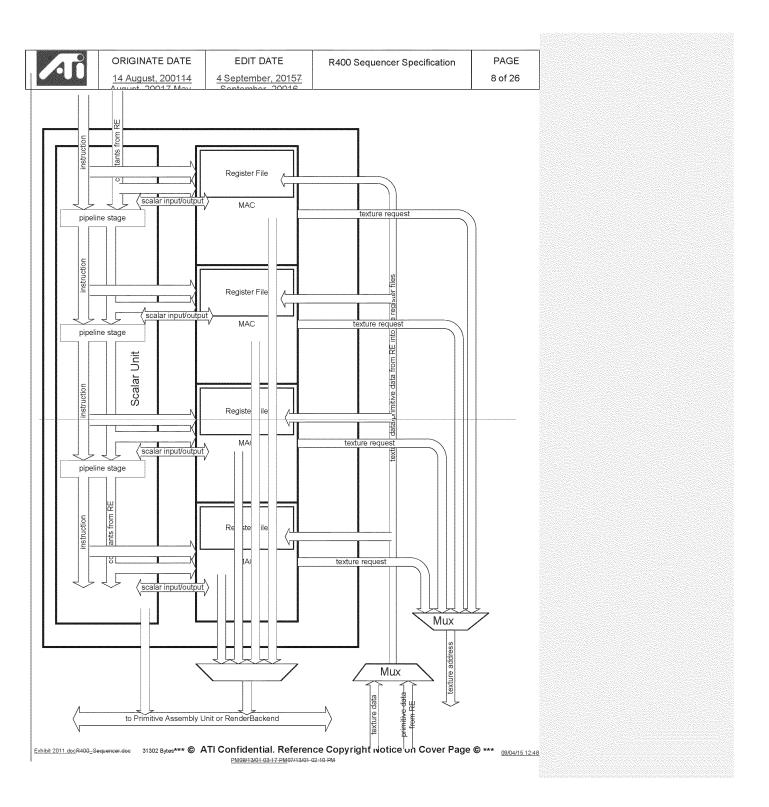
Formatted

AMD1044_0256714

ATI Ex. 2105 IPR2023-00922 Page 42 of 239



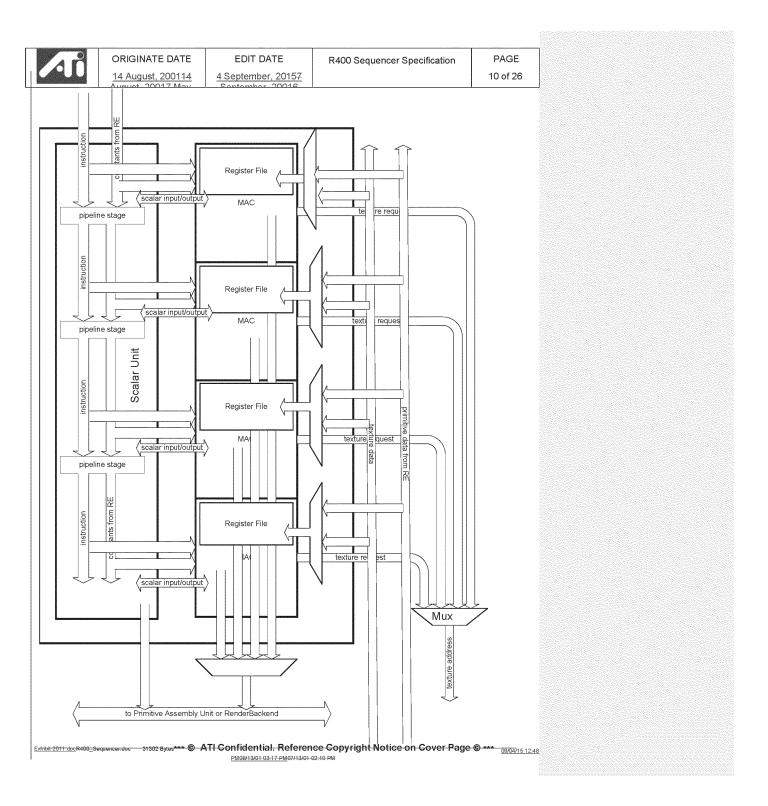
ATI Ex. 2105 IPR2023-00922 Page 43 of 239



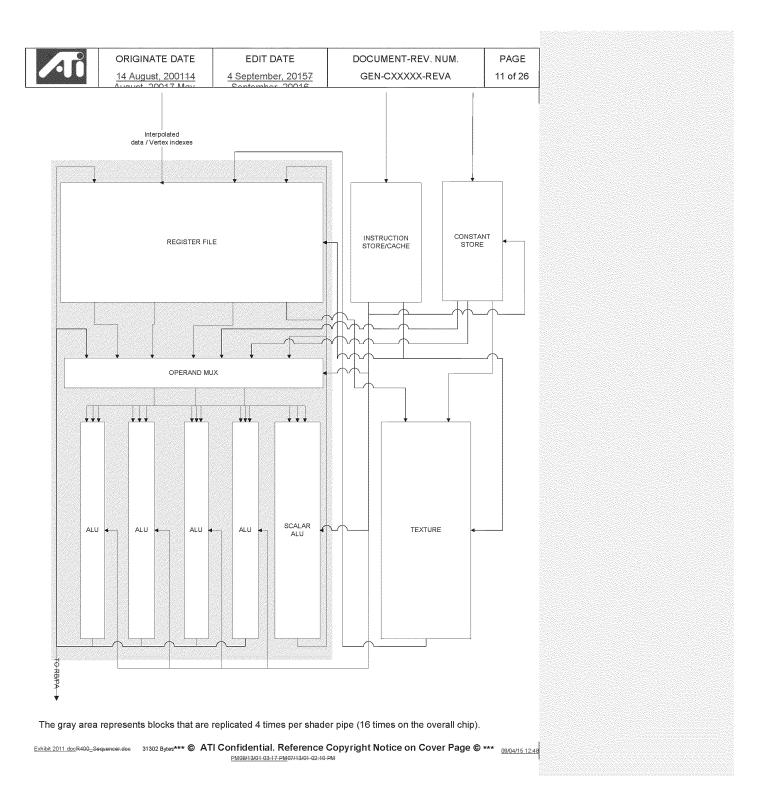
ATI Ex. 2105 IPR2023-00922 Page 44 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	14 August, 200114	4 September, 20157	GEN-CXXXXX-REVA	9 of 26
1.2 Data	Flow graph	Soptombor 20016		
	0			
Exhibit 2011.docR400_Seq	uencer.doc 31302 Bytes*** © A1	1 Confidential. Reference C PM08/13/01-03:17-PM07/13/01-02:10 PM	opyright Notice on Cover Page @	•*** <u>09/04/15 12:</u>
		PM08/13/01-03:17 PM	4	

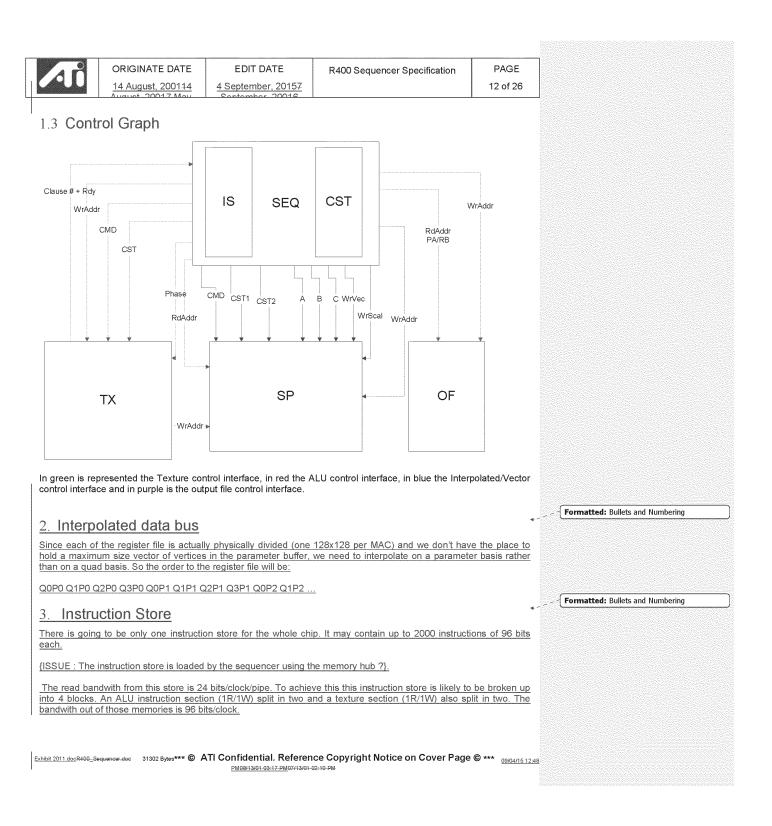
ATI Ex. 2105 IPR2023-00922 Page 45 of 239



ATI Ex. 2105 IPR2023-00922 Page 46 of 239



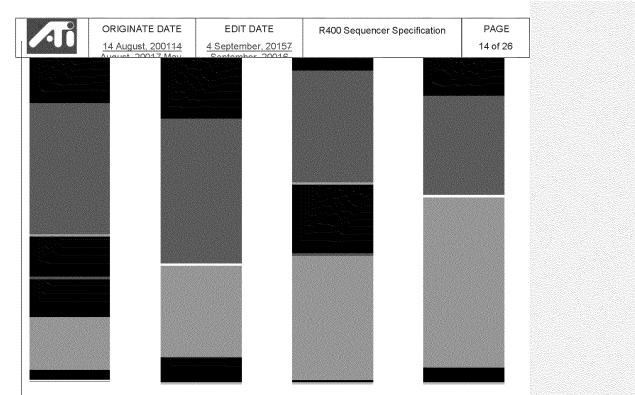
ATI Ex. 2105 IPR2023-00922 Page 47 of 239



ATI Ex. 2105 IPR2023-00922 Page 48 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	14 August, 200114	4 September, 20157	GEN-CXXXXX-REVA	13 of 26	Formatted: Bullets and Numbering
4. Consta	ant Store	Sontomnor Julie		*	
			e of where the constants are using		
			512x128 bits. The constant store is a /pipe and the write bandwith is 32/4 I		
			- F . X	*	Formatted: Bullets and Numbering
	ng and Branches				
		pported and will have to be dependent branches or not.	e dealt with at the sequencer level. He	owever, it is	
				*	Formatted: Bullets and Numbering
6. Regist	er file allocation				
			ic or dynamic. In both cases, the re- es). In the dynamic case the bounda		
pixels and ver	tices is allowed to move,		to VERTEX REG SIZE for vertice		
VERIEX REC	<u>G_SIZE for pixels.</u>				
Exhibit 2011.docR400_Se	equencer.doc 31302 Bytes*** ⓒ AT	Confidential. Reference	Copyright Notice on Cover Page ©	*** 09/04/15 12:48	
		PM08/13/01-03:17-PM07/13/01-02:10-	HM.		

ATI Ex. 2105 IPR2023-00922 Page 49 of 239



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.

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 $400 \underline{X}(?)$ in flight texture fetches and thus there can be a fair number of active clauses waiting for their texture return data.

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 <u>execute</u>xecute. 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

Exhibit 2011.docR400_Sequencer.doc 31302 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:48
PM08/13/01-02:10-PM

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044_0256722

ATI Ex. 2105 IPR2023-00922 Page 50 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1			
	14 August, 200114	4 September, 20157	GEN-CXXXXX-REVA	15 of 26				
Designed and the party of the second	August 20017 May	Sontombor 20016		1	1000			
one for the odd clocks. For exemple, here is the sequencing of two interleaved ALU clauses (F 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

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.

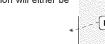
5-10. Content of the reservation station FIFOs

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)

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



7.112.1 External Interfaces

7.1.112.1.1 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 firstblue last
Constant 1	SEQ-> SP	32	128 bits transferred over 4 cycles, alpha firstblue last
Instruction	SEQ-> SP	30	120 bits transferred over 4 cycles (order TBD) ?

7.1.212.1.2 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	100
UL_Vector_Out	SP-> OF	128	Vector Data out	

Exhibit 2011.docR400_Sequencer.doc 31302 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

PM08/13/01-03:17-PM07/13/01-02:10-PM

Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering

AMD1044_0256723

∕ Ĵĵ	ORIGINATE D	0114	EDIT D	er, 201		R400 Sequencer Specification	PAGE 16 of 26
UL_Scalar	_Out	SP-> OF	-	32	Vec	tor Data out	
UR_Vector	_Out	SP-> OF	=	128	Vec	tor Data out	
UR_Scalar	Out	SP-> OF	=	32	Vec	tor 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

7.1.312.1.3 Shader Engine to Texture Unit Bus (Fast Bus)

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 FileRegister file every clock. These are directed to a different pixel of the sub-engines repeating every 4 clocks. The register fileregister 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 FileRegister files for reading Texture Address
Tex_RegFile_Read_Data	SP->TEX	512	4 Texture Addresses read from the Register FileRegister file
Tex_Write_Register_Index	SEQ->TEX	8	Index into Register fileRegister file for write of returned Texture Data

7.1.412.1.4 Sequencer to Texture Unit bus (Slow Bus)

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 fileregister file where to write the texture return data.

Name	Direction	Bits	Description
Tex_Ready	$TEX \rightarrow SEQ$	1	Data ready
Tex_Clause_Num	$TEX \rightarrow SEQ$	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

7.1.512.1.5 Shader Engine to RE/PA Bus

Name	Direction	Bits	Description
Interpolator_Register_Index	SEQ->SP	8	Index into Register FileRegister 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

Name	Direction	Bits	Description
Adress	PA→SEQ	2	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,

Echibit 2011 doc 84400_Sequences.do 31302 Bytes***
C ATI Confidential. Reference Copyright Notice on Cover Page

9904/1512.48

PM08/13/01-03:17-PM07/13/01-02:10-PM

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering
Formatted: Bullets and Numbering
Formatted
Formatted
Formatted
Formatted

Formatted

AMD1044_0256724

ATI Ex. 2105 IPR2023-00922 Page 52 of 239

	r	r			
An	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
~ 0	14 August, 200114	4 September, 20157	GEN-CXXXXX-REVA	17 of 26	Formatted: Bullets and Numbering
<u>13. Exar</u>	nples of program	executions		*	
10.1.1 0					
13.1.1 56	equencer Control of	a vector of vertices			
	a vector of 16 vertices (acti pointer as well as tag into po		/index for 512 bit total) to the RE's V	ertex FIFO	
 space 	was allocated in the position	on cache for transformed po	sition before the vector was sent		
200000000000000000000000000000000000000	pefore the vector is sent to er program (using the MH	00000000000000000000000000000000000000	ed the global instruction store with	the vertex	
	ertex program is assumed to		<u>ve the vertex vector.</u> the local state pointer (provided to a	11	
	equencers by the RBBM wh				
			asically the Vertex FIFO always has p	priority	
	point the vector is removed bitrer is not going to select a select and select and sel		f the parameter cache is full unless th	ne pipe as	
	ng else to do (ie no pixels ar				
	ates space in the SP register umber of GPRs required by		PRs used by the program ocal state register, which is accessed	using the	
	pointer that came down with will not send vertex data unt	***************************************	as been allocated		
			rface (which has a bandwidth of 512	bits/cvcle)	
************************************	overtex indices are sent to t		Alexandra and a second and		
	F0 of SU0, SU1, SU2, and S F1 of SU0, SU1, SU2, and S				
	F2 of SU0, SU1, SU2, and S F3 of SU0, SU1, SU2, and S				
• the in	dex is written to the least sig	gnificant 32 bits (floating po	pint format?) (what about compour	nd indices)	
			ng data bits are set to zero (x, y, z) e first reservation station (the FIFO in	front of	
texture st	ate machine 0, or TSM0 FIF	<u>(O)</u>			
			osition cache and a register file base texture clause 0 from the global instr		
) was first selected by the TS				
7. all instruc	tions of texture clause 0 are	issued by TSM0			
8. the contro FIFO)	I packet is passed to the ne	xt reservation station (the F	IFO in front of ALU state machine 0,	or ASM0	
 TSMC 			complete; it passes the register file v	write index	
• once		ata to the register files, it inc	rements a counter that is associated		
FIFO: claus		ndicates that the ALU state	machine can go ahead start to execu	Ite the ALU	
9. ASMO ac	cepts the control packet (aft		M arbiter) and gets the instructions fo	or ALU	
	rom the global instruction st		ntrol packet is passed to the next res	onvotion	
***************************************	e FIFO in front of texture st			ervation	
			n stations until all clauses have been nd the tag) is sent over a position bus		
share	d with all four shader pipes)	back to the PA's position c	ache		
	ameter cache pointer is also to be in the parameter cach		n data. This tells to the PA where the	data is	
fotocererererererererererererererererererer	······		ition data before it gets sent back to t	he PA	
Exhibit 2011.docR400_S	equencer.doc 31302 Bytes*** ⓒ AT	I Confidential. Reference	Copyright Notice on Cover Page ©	*** 09/04/15 12:48	
		PM08/13/01_03:17 PM07/13/01_02:10-	PM]	

ATI Ex. 2105 IPR2023-00922 Page 53 of 239

				PAGE
	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	14 August, 200114	4 September, 20157 September, 20016		18 of 26
			n exporting clause if the position export	FIFO is full
	 parameter data is exported if parameter data is sent to 		n data if it was not exported earlier) a dedicated bus	
			and the SEQ deallocates that space wh	en there is
		parameters (it is told by the		
	 the ASM arbiter will prev buffer if position is being 		ASM7 if the parameter cache (or the	position
17			o the GPRs so that they can be used b	anothar
12	shader program	inpleted, the OLQ will hee up	o the Gritts so that they can be used b	y another
1	3.1.2 Sequencer Control	of a Vector of Pixel	S	
<u>1.</u>	As with vertex shader program	s, pixel shaders are loade	d into the global instruction store by	the CP
	 At this point it is assumed that 	it the pixel program is loade	d into the instruction store and thus rea	idy to be
2.			he before the SEQ takes control of the	
	A		send parameter data to the Parameter worth of a particular parameter in one	*****************************
			arameter store locations not used ar	
	using the token provided b			
3.	the RE's Pixel FIFO is loaded with	h the barycentric coordinate	s for pixel quads by the detailed walke	r
	the state pointer and the LOI			
	the Pixel FIFO is wide enoug			
4.	SEQ arbitrates between Pixel FII left in the register files for vertice		there are no vertices pending OR there	e is no space
5.	SEQ allocates space in the SP re	gister file for all the GPRs u	ised by the program	
		I by the program is stored in	a local state register, which is accesse	ed using the
	 state pointer SEQ will not allow interpolate 	d data to be sent to the sha	der until space in the register file has b	een allocated
6.	SEQ control starts with the interp	olation of parameters (up to	16 per thread) by sending the barycen	tric
			Parameter Buffer to the interpolator	
	 PUI, PUI, and PUK (the value) Q0 i, j, and k are loaded into 		el EIEO	meter putter
	 The interpolator then general 			
			nly if Q1 is from a different primitive	
	from the same primitive as interpolator and reused for		P0k values loaded for Q0 are held b	<u>y the</u>
	***************************************		parycentric data for each quad in the	Pixel FIFO
			to be loaded into the interpolator	
	 Q1 i, j, and k are then loaded The interpolator then general 			
	 Q2P0 and Q3P0 are generat 		each pixel in QT (QTPO)	
			en loaded into the interpolator	
			this means that the output of the Pixel	
	causing the top four entries of b		at the end a final "block pop" signal is ally be removed	asserted,
			, Q2P0, Q3P0, Q0P1, Q1P1, etc.	
<u>7.</u>	SEQ controls the transfer of inter bandwidth of 512 bits/cvcle)	polated data to the SP regis	ter file over the RE_SP interface (whic	<u>h has a</u>
	······································	d parameter data is sent to	the 16 register files over 4 cycles	
		and SU3 is written with Q0P		
		and SU3 is written with Q1P and SU3 is written with Q2P		
		and SU3 is written with Q3P		
1				~
Exhib	2011.docR400_Sequencer.doc 31302 Bytes*** C	ATI Confidential. Referen PM08/13/01_03:17_PM07/13/01	nce Copyright Notice on Cover Page	© *** 09/04/15 12:48

ATI Ex. 2105 IPR2023-00922 Page 54 of 239

		- <u>.</u>		,
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
(AU)	14 August, 200114	4 September, 20157	GEN-CXXXXX-REVA	19 of 26
texture s note the c all ot 9. TSM0 ac stor#SM 10. all instruct 11. the contr FIFO)	tate machine 0, or TSM0 F that there is a separate set ontrol packet contains the s her informations (such as o cepts the control packet an 0 was first selected by the ctions of texture clause 0 an ol packet is passed to the r	FO) of reservation stations/arbite state pointer, the register file juad address for example) tra- id fetches the instructions for TSM arbiter before it could sta- re issued by TSMO next reservation station (the F	texture clause 0 from the global instr art IFO in front of ALU state machine 0,	r pixels bits uction or ASM0
inde) ● once asso ahea	(for the texture data to the the TU has written all the optimized ciated with the ASM0 FIFO d and pop the FIFO and state	TU, which will write the data data for a particular clause to ; a count greater than zero in art to execute the ALU clause	the register files, it increments a cour dicates that the ALU state machine c	nter that is an go
clause 0	from the global instruction	store	ntrol packet is passed to the next rese	
		state machine 1, or TSM1 FIF		
• pixel	data is exported in the last it is sent to an output FIFO			executed
<u>15. after the</u> shader p		eted, the SEQ will free up the	e GPRs so that they can be used by a	nother
<u>13.1.3 N</u>	otes			4
16. the state threads of		l operate ahead of time so tha	at they will be able to immediately sta	rt the real
instructio	n store base pointer does r		vector through the reservation station pointer is different for all threads, but t via the state pointer	
<u>18.</u> Waterfa be speck		cation, loops and branches a	nd parameter cache de-allocation still	needs to
Exhibit 2011.docR400_	Sequencer.doc 31302 Bytes*** © A	TI Confidential. Reference	Copyright Notice on Cover Page ©	*** 09/04/15 12:48

ATI Ex. 2105 IPR2023-00922 Page 55 of 239

PAGE	20 of 26
R400 Sequencer Specification	
EDIT DATE	4 September, 20157
ORIGINATE DATE	14 August, 200114
	Ę

14. Timing Diagrams

14.1 MAC 0

Exhibit 2011 doc R400_Sequences doc 31302 Bytes ***
Carried ATI Confidential. Reference Copyright Notice on Cover Page

S004/15 12:45 PM084300+03:47 PM074301-02:40-PM

AMD1044_0256728

ATI Ex. 2105 IPR2023-00922 Page 56 of 239

	17	-	srcA	Ĺ	r			e				S S	
	16	2	9					5		р	PS	Ъ	
ЭЕ 26	15	2	C S					-		D	Ş	Ω Γ	
PAGE 21 of 26	11	ţ	srcB	_				0		L.	ı	٥	
	43	2	srcA		ē			m		m	₽	g	
NUM.	10	71	9					2			PS	<u>≧</u>	
REV. I XX-RE	11	-	C S					-			Ş	P	
DOCUMENT-REV. NUM. GEN-CXXXX-REVA	10	2	SICB	~			l	۰ ۲	2		ı	٥	
GEN-	σ	D	srcA		0	10_3		m	srcC		≙	м С	0
	α	0	60_3 1_3 1C			10_2		0	srcB		PS	Ş	0, MAC
1510		-	C0_2 C1_2 srcC			IO_1	ſ	-	srcA		ЪЧ	₽	ler Unit
ATE 1, 201	9	þ	c0_1 c1_1 srcB			0_01	Ĺ	2			,	٥	e 0, Shao
EDIT DATE 4 September, 20157	A 5 6 6	5	c0_0 c1_0 srcA	Ĺ	₽			e			₽	S	Timing Diagram 1: Sequencer to Shader Pipe 0, Shader Unit 0, MAC 0
4 Sep	N N	t	1					7			PS	₹	to Sha
	٣	°	SrcO					-			ЪЧ	β	duencei
DATE 201144	100m (7	8 20 3				Ĺ	2			ı	٥	m 1: Se
NATE lust, 20		-	srcA	Ĺ							₽		j Diagra
ORIGINATE DATE													Timinç
R	0		SEQ_SP_constant0 SEQ_SP_constant1 SEQ_SP_renstant1	SEQ_SP_phase	RE_SP_data[511:384]	SEQ_SP_instruction SEQ_SP_instr_start	mac0 phase	mac0_cycle_count	RF0_read_data	mac0_vector_result	SEQ_SP_write_addr	RF0 write cycle	

14.2 Sequencer to Shader Pipe

Formatted: Bullets and Numbering

*

AMD1044_0256729

Exhibit 2011 docR400_Sequencent doc 31302 Bytes ++++
ATI Confidential. Reference Copyright Notice on Cover Page
++++
G004/15 12:48 PM08/13/01 03:17.PM07/13/01 02:10.PM

ATI Ex. 2105 IPR2023-00922 Page 57 of 239

	17		×		×	P1"	53#		00P1 '	Sd		Ι]
	16		Q3"		×	P1"	°2"		Q3P0" Q0P1'	Ъ			
	15		Q2 "		×	"Iđ	Q1 "		Q2 P0 "	Ê	ΙL		
	14		۵1″		= Г. А	, 1ª	- 0Õ		20P0 ' 21P0" 22P0"	≙			
Е 26	13		õ		L T		Q3"			S S S			
PAGE 22 of 26	12		Q3"		, Lđ	#0đ	82		Q3P1	Ş		,	
	-1-		g2 "		Ĩď	F0."	Q1"		Q2P1	þ			
R400 Sequencer Specification	10		Q1"		= 04	- 0đ	ŏ		01P1	₽	,		
r Speci	თ		, og	_	- Ođ	P1	0 3	L □ L	00F1	S		ΙL	
anence	ω		Q3	L	о Од	11	0 2		Q3P0	Z		1	sfer
too Sec	2		Q2		р Од	F1	<u>0</u> 1		Q2 P0	P	ļ		ata Tran
Å	G		01 01		р Са	Id	õõ		0410	≙			Pipe Da
0157	ۍ		õ		Гd	Da	Š 3	L □ L	0400	S			Timing Diagram 2: RE Interpolator to Shader Pipe Data Transfer
EDIT DATE	4		õ3		ст д	ΡŪ	<u>0</u> 2			Ş			lator to
EDI Septer	e		<u>0</u> 2		P1	οđ	01			P	ΙL		Interpo
4	5		61		Tđ	Ođ	50			۵ (m 2: RE
DATE 001144	-		00	5	PO	×	×	Ĺ					Diagra
ORIGINATE DATE 14 August, 2001144	0		õõ		PO	×	×						Timing
CRIGIT 14 Aug		PXF_SEQ_rts- PXF_SEQ_new_prim-	PXF_INT_data	SEQ_PXF_vector_pop	PMB_INT_data	SEQ_INT_pm_load_ INT_param_reg	SEQ_INT_px_load INT_quad_reg	SEQ_SP_phase SEQ_SP_write_addr RE_SP_valid	RE_SP_data	RF0 write cycle mac0_phase	mac1_phase mac2_phase	mac3_phase	_

Exhibit 2011 doc/4400_Sequencer-doc 31302 Bytes****
Carl Confidential. Reference Copyright Notice on Cover Page

Sequence 12:48 PM0843004 03:17 PM0743004 02:10 PM

AMD1044_0256730

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
3	14 August, 200144	4 September, 20157	GEN-CXXXX-REVA	23 of 26
No. of Concession, Name	Allowed DODA7 Mani	Contombor 20016		

Pipe	-
Ω	1
5 669866	
Ω.	-
500000000	
0	
<u> </u>	-
	~
- upond	-
\sim	
(1)	-
Texture	
r	
	-
	~
- against	
- Anne	-
Ŵ	-
0	-
(****	
n	-
~	-
Sequencer to	
\circ	-
(1)	-
ñ	
(\mathbf{v})	1
	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
5	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
14.3	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1	-
énnen	-

	17		TC2			1	T1_3	Ĺ	0 8 1	
	16	TC srcB	TC1		L		T1_2		S V	
-	15	srcA	TCO				T1_1		₹ þ	
alum manananananana	14	10	TC3			12	11 0 0 0		· <u>0</u>	
	13	srcC	TC2			、	10 130	t	ର ସ	
aditationalistationa	12	TC srcB	TC1				T0_2		S V V	 ransfer
	11	srcA	100				TOL		2 P	e Data 1
	6	10	TC3			<u>4</u>	10 10		· <u>0</u>	Timing Diagram 3: Sequencer - Texture Unit Interface and Texture Unit - Shader Pipe Data Transfer
	თ	srcC	TC2	11 13				l	0 X	nit - She
	∞	TC srcB	TC1	11_2					а ч	 xture U
	7	srcA	TC0	II_I					Z ₽	 and Te
datationation	9	TC	TC3	11 0 0 22		J			· <u>0</u>	nterface
	ம	srcC	TC2	IO_3		````			0 X	re Unit I
	4	TC srcB	TC1	IO_2					S Z	- Textu
2	e	srcA	TC0	I OI					₹ P	luencer
	7	TC		10 0 74					· <u>0</u>	n 3: Sec
	-	srcO				<u> </u>		Ţ	<u>⊖</u> 8	Diagrar
j į	0	TC srcB			L				R	Timing
	8	SEQ_SP_read_addr RF0_read_data	SP_TX_tc	SEQ_TX_instr_start SEQ_TX_instruction SEQ_TX_clause SEQ_TX_write_addr SEQ_TX_last	SEQ_TX_phase tx_phase	TX_SP_write_addr TX_SP_valid	TX_SP_data TX_SEQ_clause TX_SEQ_done	SEQ_SP_phase	SEQ_SP_write_addr RF0 write cycle	

Exhibit 2011 doc/R400_Sequences:doc 31302 Bytes 444 
Confidential. Reference Copyright Notice on Cover Page 
++++ 
G004/15 12:48 PM08/1301-03:417 PM07/4301-02:40-PM

# AMD1044_0256731

¥

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	<u>14 August, 200114</u>	4 September, 20157		24 of 26	
And and a second second second second	August 20017 May	Sontombor 20016			
14.4 Tim	ing diagrams ex	nlanations		4-	
<u>1-TT 11111</u>	ing alagramo ex	planatorio			
The numberin	g of the four shader pipes	s, the four shader units, an	d the four MACs is from left to right ar	id from 0 to 3.	
So for exampl	e the most significant 512	2 bits of a SP goes to SU	) and the least significant 512 bits go		
	assumptions are made:	MACU and the least signif	icant 128 bits go to MAC3.		
1. all blo	ck to block signals are reg				Formatted: Bullets and Numbering
			the MAC one clock after a RF reared data is valid out of the RF two cl		
	ss is asserted on the SEC				
1441 <i>Tir</i>	mina Diaaram 1: S	equencer to Shade	er Pipe 0, Shader Unit 0, M	AC 0	
		***************************************	interface. For simplicity only the tim		
			ACO, MAC2 one clock later than MA		
means that mo		be delayed in the SP by o	one cycle for MAC1, two cycles for MA	C2, and three	
		oits over 4 cycles). Pipelin	ed in SP for other MACs.		
		oits over 4 cycles). Pipelin			
			ned in SP for other MACs. om the RE, as well as defining the ord	er of all writes	
into the RF. If	t is asserted during the cy		(ID) is valid on the RE_SP_ID bus. P		
for other MAC RE SP IDI51		ionificant 128 bits of the R	E_SP_data interface (meaning that th	is MAC0 is in	
<u>SU0).</u>					
			<ul> <li>Pipelined in SP for other MACs. nstruction transfer. Pipelined in SP for</li> </ul>	r other MACs	
			0 (this may not be he actual signal na		
		the MAC that keeps track is may not be he actual sig	of the RF write cycles; 0 here corre-	sponds to the	
			may not be he actual signal name).		
	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 writ	e addr: Register File Wr	ite Address (8 bits). Note	that the SEQ does not send the Text	ure Data write	
	his bus. Pipelined in SP			<b>D</b> ( <b>D</b> )(	
			es (ID = Interpolated Data, TD = Textu ence point on the diagram).	re Data, PV =	
14.4.2 Tir	ming Diagram 2: R	E Interpolator to S	hader Pipe Data Transfer	4-	
			and k) is sent from the Pixel FIFO to the	ne interpolator	
under SEQ co	ntrol, and how parameter	data (for each vertex) is a	llso sent to the interpolator under SEC		
	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	being sent over the RE_S t of the pixel FIFO is valid.	P interface.		
PXF_SEQ_ne	w prim: The current out	out of the Pixel FIFO is fro	n a different primitive that the previous		
		<u>be fetched (if its not from a</u> I FIFO – goes to the Interp	new prim, then new parameter data is	not needed).	
SEQ PXF rtr	: Indicates that the curre	nt Pixel FIFO output will I	be taken by the Interpolator (driven b	y SEQ). Then	
	ata will be driven the next		or of pixels (otherwise RTRs cause t	ha data ta ha	
	n the four quads).	Fixer The O to pop a veci	or or pixels (otherwise it it's cause i	ne data to be	
PMR INT dat	ta: Nata from the Parame	ter Ruffer to the Internola	tor. (Note that the control of the parar	neter huffer is	
TBD).	a, Data nom me r'didilit	ater Durier to the milerpold	tor. more that the control of the paral	neter puller 18	
SEQ_INT_pm	load: controls the loadir	ng of parameter data into th	ne Interpolator.		
Exhibit 2011.docR400_Se	equencer.doc 31302 Bytes*** © 🖌		ce Copyright Notice on Cover Page	© *** 09/04/15 12:48	
		PM08/13/01-03:17-PM07/13/01-0			

ATI Ex. 2105 IPR2023-00922 Page 60 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	14 August, 200114	4 September, 20157	GEN-CXXXXX-REVA	25 of 26	
	g: register in the Interpola		x parameter data while the per-pixel	parameters	
		y not be the actual signal na of pixel data into the Interpo			
INT quad reg name).	: : register in the Interpola	tor that holds one quad's v	vorth of pixel data(may not be the a	ctual signal	
Accession					
	<u>se: see above under TD1.</u> e_addr: see above under T				
	Interpolator Data Valid – in Data from the RE interpolat		write the ID on the appropriate cycle.		
RF0 write cyc	le: see above under TD1.				
	see above under TD1. Th Shader Unit has a set of the	ACs. Note			
1443 Tir	ning Diagram 3: Se	equencer - Tevture	Unit Interface and Textur	⊳ I Init _*	Formatted: Bullets and Numbering
	pe Data Transfer				
Activity of the second s		linate read from the registe	r file and its transfer to the TX. The	instruction	
		ture data transfer to the sha	ader pipe. ure coordinate read address is assert	od	
RF0 read add	dr: see above.			<u>-u.</u>	
		from the shader pipe to the t st cycle of a SEQ to TX instr			
		nstruction transferred over 4 ociated with this instruction.			
SEQ TX write	e addr: RF write index use	d by TX for returned texture	data.		
		st texture instruction of a cla write. Note that it is asserte	<u>uuse.</u> ed early enough to be registered into '	TX and still	
allow TX to sou	urce the texture data to the	SP on the correct cycle.	สมองแมงการและเหมือสมองแมงการและให้แสมองแมงการและการแส่หรือสมองการและการและเหมงการและการและเหมงการและการและเหมงก		
	phase signal after being re				
	addr: RF write index for te indicates that valid texture	<u>xture data.</u> data is being driven to the S	SP.		
TX_SP_data: 1	he texture data.	ociated with the texture data			
TX_SEQ_don	e: indicates to the SEQ that		is complete for the clause number the	at is on the	
TX_SEQ_clau	se bus.				
	se: see above under TD1 -	shown here for reference. D1- shown here for referen	00		
	le: see above under TD1- s		<u></u>		
Exhibit 2011.docR400_Se	quencer.doc 31302 Bytes*** © AT	I Confidential. Reference <u>PM08/13/01-03:17-PM</u> 07/13/01-02:10	Copyright Notice on Cover Page ©	*** <u>09/04/15 12:48</u>	

ATI Ex. 2105 IPR2023-00922 Page 61 of 239

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	14 August, 200114	4 September, 20157 September, 20016		26 of 26	
					Formatted: Bullets and Numbering
<u>8-15. Op</u>	<u>en issues</u>			* -	
have the band 1) Le ba 2) W	lwith from the texture store et the compiler handle th andwith there to operate. T /aterfall down the pipe allo	e to feed the ALUs. Two so ne case and put those in This requires a significant	It the same for the whole vector of vert olutions exists for this problem: istructions in a texture clause so we amount of temporary storage in the reg the vertices having the same constants y a factor of 16.	can use the gister store.	
	ome testing on the size of mic VS static).	the register fileregister file	e as well as on the register fileregister	file allocation	
Ability to expo	rt at any clause?				
Saving power	?				
Are we workin	ig on 32 vertices at a time	or 16?			
Size of the fifestations.	o containing the informati	ion of a vector of pixels/v	ertices. And size of the fifos before th	ne reservation	
Sequencer Ins	struction memory, and cor	nstant memory.			
Arbitration pol	icy for the output file.				
Loops and bra	anches.				
The paramete	r cache may end up in the	PA rather than in the RS	. Parameter cache management thus r	nay change.	
Exhibit 2011.docR400_S	equencer.doc 31302 Bytes*** ⓒ 🗸	ATI Confidential. Referen	ce Copyright Notice on Cover Page	© *** 09/04/15 12:48	
1		PM08/13/01-03:17-PM07/13/01-0	02:10 PM		

ATI Ex. 2105 IPR2023-00922 Page 62 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 200124	4 September, 20153	GEN-CXXXXX-REVA	1 of 31
Author:	Laurent Lefebvre	Cotobor 200124		1
Issue To:		Сору No		
		t		
	R400 Se	equencer Sp	ecification	
		8 8		
		SEQ		
		which, provide which the		
		Version 0.65		
			ncer block (SEQ). It provides an over also describes the block interfaces,	
	cks, and provides internal sta		also describes the block intenaces,	Internal Sub-
AUTOMATIC Document L	ALLY UPDATED FIELDS:	rforce\r400\arch\doc\gfx\RE	-\R400 Sequencer doc	
	1	Sequencer Specification		
	Name/Dept	APPROVALS	Signature/Date	
	namo, Dopt		oignataio.sato	
Remarks:				
i tomanio.				
THIS DO	CUMENT CONTAIN	S CONFIDENTIAL I	NFORMATION THAT CO	ULD BE
	NTIALLY DETRIME	NTAL TO THE INTE	EREST OF ATI TECHNOL	
	INC. THROUGH U	INAUTHORIZED U	SE OR DISCLOSURE.	
			· · · · · · · · · · · · · · · · · · ·	
work created in	n 2001. The use of this copy	right notice is intended to	rial in this document constitutes an provide notice that ATI owns a cop	yright in this
			publication has occurred. This w t of this document may be used, re	
transmitted in a	ny form or by any means wit	hout the prior written permi	ssion of ATI Technologies Inc."	
Exhibit 2012.docR400_Seg	uencer.docR400_Sequencer.doc 32166 Byt	es*** © ATI Confidential. I	Reference Copyright Notice on Cov	
			<u>uanatio 12:aa kweetaana</u>	2-17-MW

ATI 2012 LG v. ATI IPR2015-00325

AMD1044_0256735

ATI Ex. 2105 IPR2023-00922 Page 63 of 239



PAGE 2 of 31

Table Of Contents

	VIEW 43
<u>1.1 Top L</u>	evel Block Diagram
	Flow graph97
<u>1.3 Contr</u>	ol Graph118
	RPOLATED DATA BUS
<u>3. INSTI</u>	RUCTION STORE1411
<u>4. SEQU</u>	JENCER INSTRUCTIONS 1411
	STANT STORE1411
	PING AND BRANCHES 1411
	dependant predicate instructions
1611	
	ter file indexing17 12
	STER FILE ALLOCATION 1712
	URE ARBITRATION1813
	ARBITRATION
	DLING STALLS
	TENT OF THE RESERVATION
STATION	FIFOS
	OUTPUT FILE (RB FIFO AND
	ER CACHE)
<u>13. REGI</u>	STERS
<u>14. INTE</u>	ERFACES
14.1 Ext	tornal Interfaces 2014
<u>14.1</u>	ternal Interfaces2014
<u>14.1</u> L	Sequencer to Shader
14.1.1	
<u>14.1.1</u> Engine E	<u>Sequencer to Shader</u> 3us2014
<u>14.1.1</u> Engine E 14.1.2	<u>Sequencer to Shader</u> 3us2014 Shader Engine to Output
<u>14.1.1</u> Engine E <u>14.1.2</u> File	Sequencer to Shader 3us2014 Shader Engine to Output 2015
14.1.1 Engine E 14.1.2 File 14.1.3	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4 (Slow Bus	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4 (Slow Bus	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4 (Slow Bus 14.1.5	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4 (Slow Bus 14.1.5 14.1.6	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4 (Slow Bus) 14.1.5 14.1.6 15. EXAM	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4 (Slow Bus) 14.1.5 14.1.6 15. EXAM EXECUTIO	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4 (Slow Bus 14.1.5 14.1.6 15. EXAM EXECUTIO 15.1.1	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4 (Slow Bus 14.1.5 14.1.6 15. EXAM EXECUTIO 15.1.1 of Vertice	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4 (Slow Bus 14.1.5 14.1.6 15. EXAM EXECUTIO 15.1.1 of Vertice 15.1.2	Sequencer to Shader 3us
14.1.1 Engine E 14.1.2 File 14.1.3 Unit Bus 14.1.4 (Slow Bus 14.1.5 14.1.6 15. EXAN EXECUTIO 15.1.1 of Vertice 15.1.2 of Pixels	Sequencer to Shader 3us

16.	OPEN ISSUES
1	OVERVIEW
	Top Level Block Diagram
	Data Flow graph7
1.3	Control Graph10
	INTERPOLATED DATA BUS
	INSTRUCTION STORE
	CONSTANT STORE
	LOOPING AND BRANCHES13
	REGISTER FILE ALLOCATION 13
	TEXTURE ARBITRATION
	ALU ARBITRATION14
	HANDLING STALLS
	CONTENT OF THE RESERVATION
STAT	FION FIFOS
	THE OUTPUT FILE (RB FIFO AND
	AMETER CACHE)15
	INTERFACES15
	External Interfaces 15
$\frac{12}{12}$	1.1 Sequencer to Shader
En	gine Bus
	1.2 Shader Engine to Output
	1.20nador_Engine to Odtpat
12	1.3 Shader Engine to Texture
	it Bus (Fast Bus)
12	1.4 Sequencer to Texture Unit bus
	ow Bus) 16
$\frac{12}{12}$	1.5 Shader Engine to RE/PA Bus 16
$\frac{12}{12}$	1.6 PA? to sequencer
13	EXAMPLES OF PROGRAM
EXEC	CUTIONS
	1.1 Sequencer Control of a Vector
of \	/ertices 16
13	1.2 Sequencer Control of a Vector
ofF	Pixels 18
$\frac{13}{13}$	<u>1.3</u> Notes
	OPEN ISSUES
	OVERVIEW
	Top Level Block Diagram
	Data Flow graph
	Control Graph 10

Ehbit 2012 docR400_Sequencer.docR400_Sequencer.doc 32166 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

AMD1044_0256736

ATI Ex. 2105 IPR2023-00922 Page 64 of 239

	ORIGINATE DATE	EDIT DATE		OCUMENT-REV. NUM.	PAGE
	24 September, 200124 September, 200114	4 September, 20153		GEN-CXXXXX-REVA	3 of 31
2. INTER	POLATED DATA BU	s10	12.1.2	Shader Engine to Ou	tput
3. INSTR	UCTION STORE		-ile		,
	TANT STORE		1012	Shadar Engine to To	vturo
5. LOOP	ING AND BRANCHES	**********		—Shader Engine to Te	1
6. REGIS	STER FILE ALLOCATI	ON11	Jnit Bus	(Fast Bus)	
7. TEXTI	URE ARBITRATION		12.1.4-	-Sequencer to Texture L	Jnit-bus
8. ALU A	RBITRATION		Slow But	s) 1 4	
	LING STALLS		12.1.5	Shader Engine to RE/P	A Bus 14
10. CONT	ENT OF THE RESER				
STATION F	IFOS	***********		PA? to sequencer	
11. THE C	UTPUT FILE (RB FIF	JAND		APLES OF PROGRAM	
PARAMETE	ER CACHE)	<u>1</u> 3 E	(ECUTIC	NS	
12INTE	RFACES	***********		Sequencer Control of a	Vector
12.1 Ext	ernal Interfaces		of Vertice		
1211	Sequencer to Shad	or	13.1.2-	-Sequencer Control of a	Vector
			of Pixels		
Engine B	us		1212		4-7
			13.1.3-	- Notes	······································

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 0.4-5 (Laurent Lefebvre) Date : September 7, 2001 Rev 0.6 (Laurent Lefebvre) Date : September 24, 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.

Exhibit 2012.docR400_Sequencer.docR400_Sequencer.doc 09/04/15 12:49 PM08/13/01 03:17 PM

AMD1044_0256737

ATI Ex. 2105 IPR2023-00922 Page 65 of 239

AP	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	7
	24 September, 200124	4 September, 20153		4 of 31	

1. Overview

The sequencer first arbitrates between vectors of $46-\underline{64}$ vertices that arrive directly from primitive assembly and vectors of $4-\underline{16}$ quads ($46-\underline{64}$ 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 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 and all 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 instruction thus there is only one sequencer for the whole chip.

Ehibit 2012 doc R400_Sequencer.doc R400_Sequencer.doc 32166 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

AMD1044_0256738

ATI Ex. 2105 IPR2023-00922 Page 66 of 239

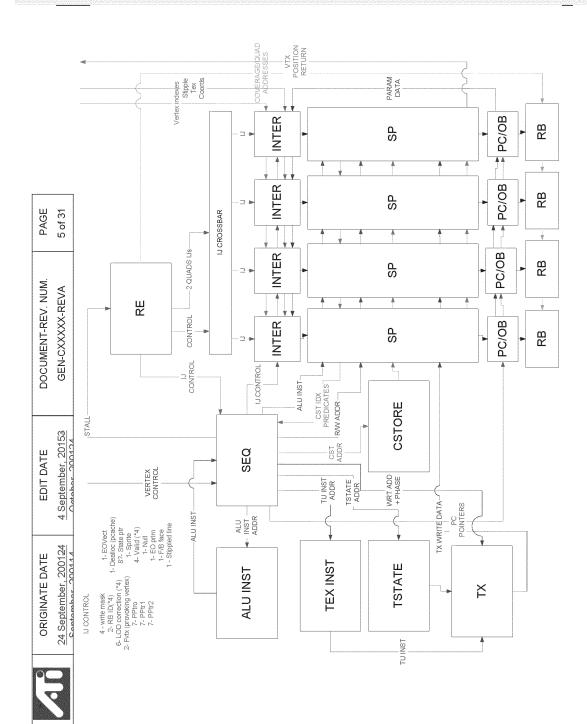
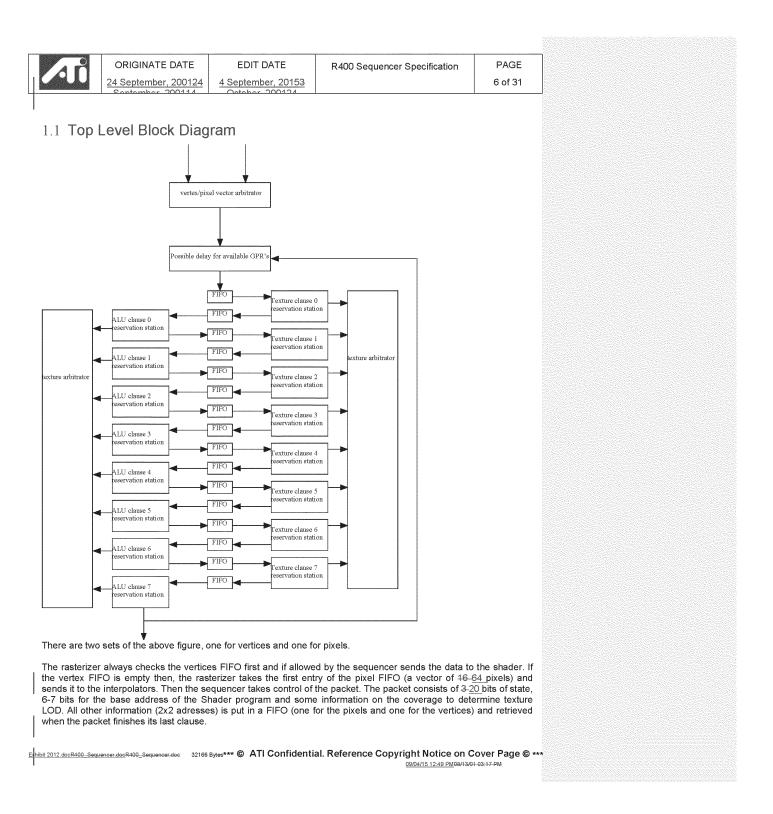


Exhibit 2012 docR400_Sequencer.dooR400_Sequencer.doo 32168 Bytes***
 ATI Confidential. Reference Copyright Notice on Cover Page

 3004/15.12.49 FW08433014034174W

AMD1044_0256739

ATI Ex. 2105 IPR2023-00922 Page 67 of 239



ATI Ex. 2105 IPR2023-00922 Page 68 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 200124	4 September, 20153	GEN-CXXXXX-REVA	7 of 31

{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.

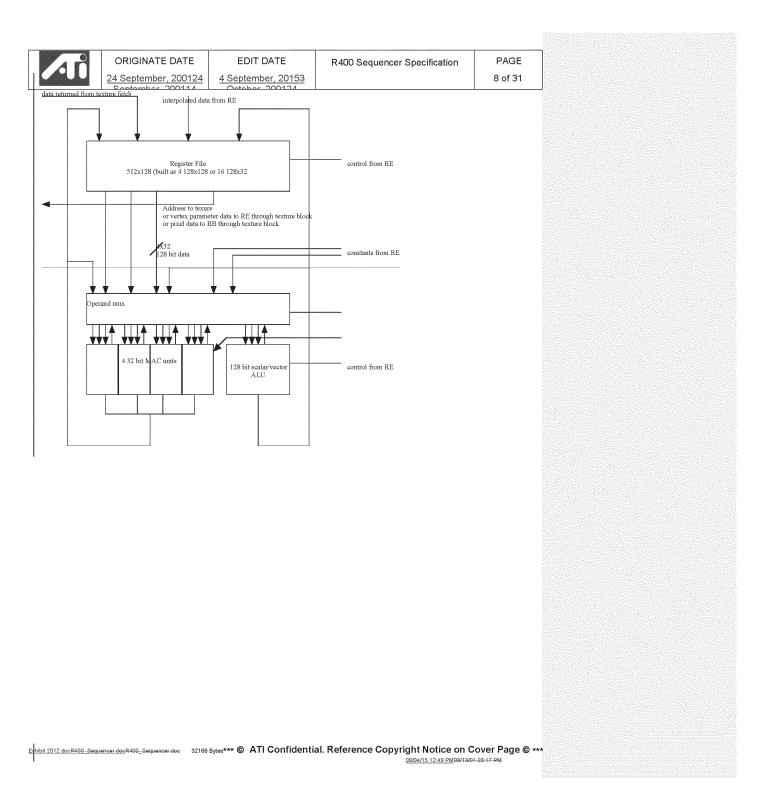
Only two ALU state machine may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the register file address bus at one time. Arbitration is performed by three arbitrer blocks (two for the ALU state machines 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

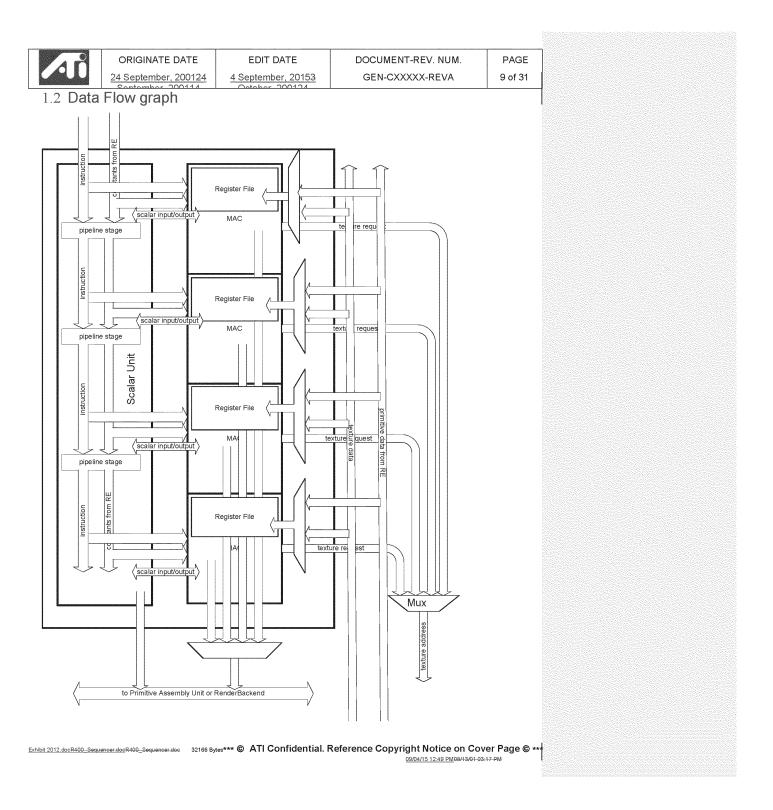
Exhibit 2012.docR400_Sequencer.docR400_Sequencer.doc 32166 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

AMD1044_0256741

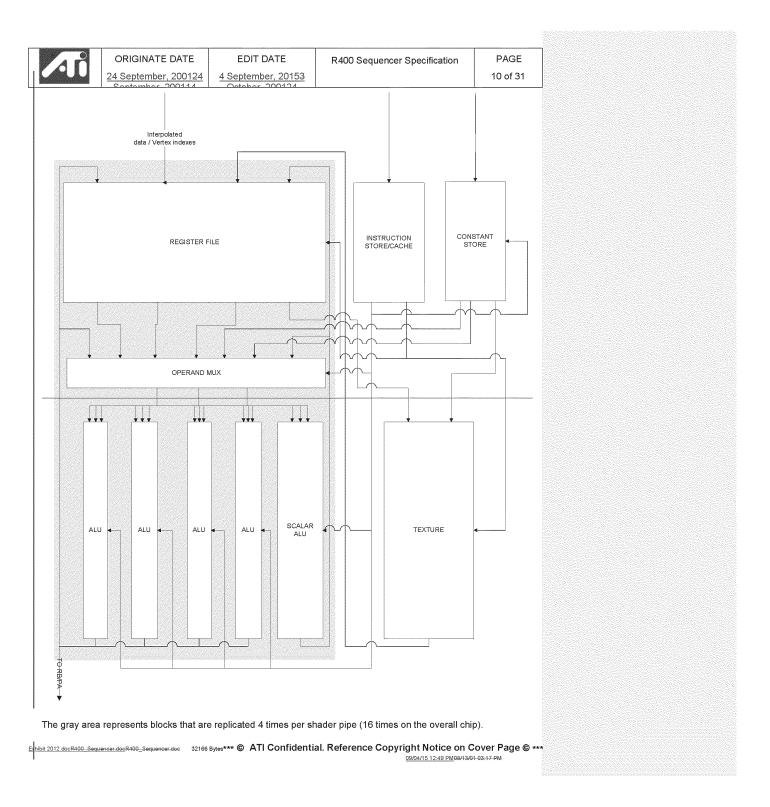
ATI Ex. 2105 IPR2023-00922 Page 69 of 239



ATI Ex. 2105 IPR2023-00922 Page 70 of 239



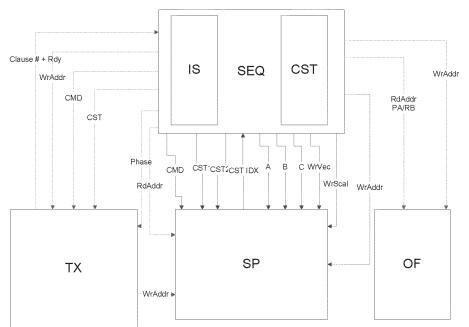
ATI Ex. 2105 IPR2023-00922 Page 71 of 239



ATI Ex. 2105 IPR2023-00922 Page 72 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 200124 September, 200114	4 September, 20153	GEN-CXXXXX-REVA	11 of 31

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.

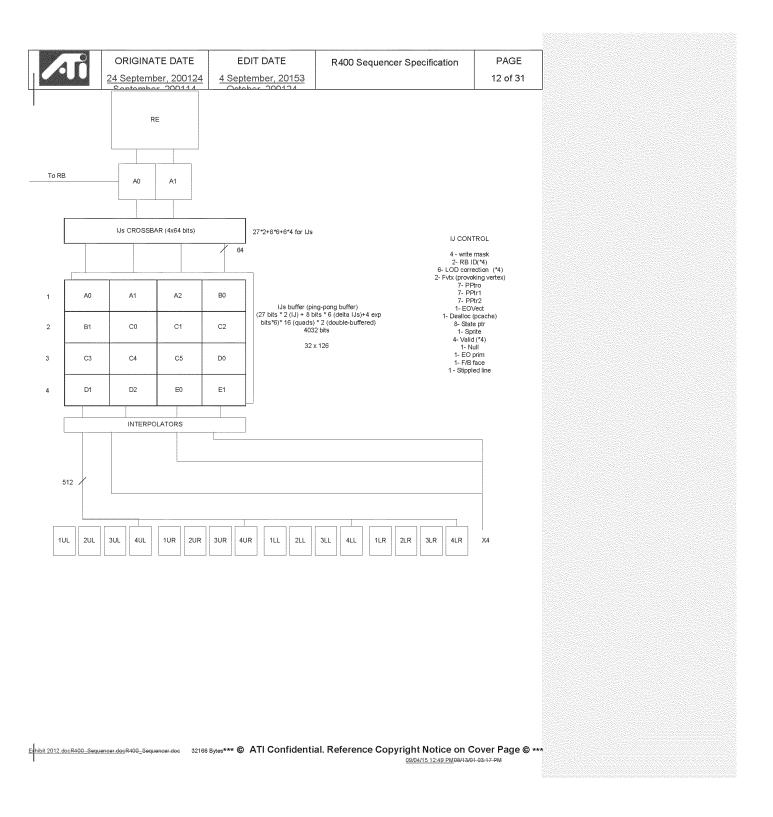
2. Interpolated data bus

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

Exhibit 2012.docR400_Sequencer.docR400_Sequencer.doc 32166 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C ***

AMD1044_0256745

ATI Ex. 2105 IPR2023-00922 Page 73 of 239



ATI Ex. 2105 IPR2023-00922 Page 74 of 239 PROTECTIVE ORDER MATERIAL

	T19	51 - 51 -	52- 55	56- 56-	63 - 63 <	
	T18	32- 35	39 3 8 <	5 40 < <	> 4 4 < 47	
	T17	> ¹ 6 − 2	53 g <	24- 27	31 ²⁸ <	
	T16	> °-0	> ⁴ -7	8-11 8-1	15 ⁻ 7 5	
	T15			ОШ	μ	
ЗЕ 131	T14				DO	
PAGE 13 of 31	T13		S	5	3	
	T12				BO	<u> </u>
DOCUMENT-REV. NUM. GEN-CXXXX-REVA	T11	Б	D2			LL.
OCUMENT-REV. NUN GEN-CXXXX-REVA	T10	ü	C4	C5		
CUMEI	Т9	B1				
O O	T8	AO	A1	A2		
(D)	1			Ш	ш	
ATE r, 2015 00134	T6				DO	
EDIT DATE 4 September, 20153	T5		C	5	62	
4 Ser	Т4				BO	0
н 124 Г	13	5	6			0
ORIGINATE DATE 24 September, 200124 Sentember, 200144	12	ទ	5 2	C5		
GINAT ptemb∉	È	B.				
ORI 24 Se	TO	AO	A1	A2		
		SP0	SP1	SP2	SP3	

AMD1044_0256747

Exhibit 2012. docR400_Sequencer.docR400_Sequencer.doc

32168 Bytes***
 ATI Confidential. Reference Copyright Notice on Cover Page

 <u>B004/15 1249 FW0043301 03477 FW</u>

ATI Ex. 2105 IPR2023-00922 Page 75 of 239

ORIGIN	ATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE		
	<u>ber, 200124</u>	4 September, 20153		14 of 31		
Above is an example of a	i tile we might	receive. The IJ informati	on is packed in the IJ buffer 2 quads			
same primitive. Then the register file is actually ph	sequencer con ysically divided the parameter	trols the write mask to th d (one 128x128 per MAC buffer, we need to inter	rpolate a parameter. They all have to e register to write the valid data in Sin C) and we don't have the place to hol polate on a parameter basis rather th	ce each of the d a maximum		
Q0P0 Q1P0 Q2P0 Q3P0	Q0P1 Q1P1 Q	2P1 Q3P1 Q0P2 Q1P2				
3. Instruction St	tore					
There is going to be only bits each. There is also go			It may <u>will contain up to 2004096</u> 0 ins 56x32.	tructions of 96		
{ISSUE : The instruction s	tore is loaded	by the sequencer using t	ne memory hub ?}.			
likely to be broken up inte	o 4 blocks. An dwith out of th	ALU instruction section ose memories is 96-48 b	ts/clock)/pipe. To achieve this this instr (1R/1W) split in two and a texture se its/clock. <u>It is likely to be a 1R/1W por</u> rexture instruction.	ction (1R/1W)	ſ	Formatted: Bullets and Numbering
4. Sequencer In	structions	2		*	1	rormatted: Bullets and Numbering
All control flow instruction during this time (MOV PV)		structions are handled b	y the sequencer only. The ALUs will p	perform NOPs	ſ	Formatted: Bullets and Numbering
4.5. Constant St	ore			*	-<'(romatted: builds and Numbering
table also managed by the	e CP. A likely s	size for the constant store	ware of where the constants are usine a is 512x128 bits. The constant store is -2/4 bits/clock/pipe and the write bar	also planned		
GPRs). There are 144 windata must pass thru the	res from the ex Shader pipe fo	kit of the SP to the seque or the float to fixed conv	loaded first with the indexes (that c encer (9 bits pointers x 16 vertexes/clo ertion, there is a latency of 4 clocks index into the constant store. The asso	ck). Since the (1 instruction)		
NOP //	latency of the	float to fixed conversion	of R2.X, also copies the content of R2.7 dd R4 to C0[R2.X] into R3	<u>K into R1.X</u>		
Note that we don't really R2.X is just written again			cause we use the state from the MOV	/A instruction.		
The storage needed in the	sequencer in	order to support this feat	ure is 2*64*9 bits = 1152 bits.		1	Formatted: Bullets and Numbering
5.6. Looping and	d Branche	<u>es</u>		*	2- L	romatten, puicts and NUHIDEINIY
			to be dealt with at the sequencer leve The control program has 4 instructions			
6.1 The controlli	<u>ng state.</u>			•-	{	Formatted: Bullets and Numbering
As per Dx9 the following s	tate is availab	le for control flow:				
Boolean[15:0]						
hibit 2012.docR400_Sequencer.docR400_Se	quencer.doc 32166	Bytes*** © ATI Confidenti	al. Reference Copyright Notice on C 09/04/15 12:49 PM0843404			

ATI Ex. 2105 IPR2023-00922 Page 76 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 200124	4 September, 20153	GEN-CXXXXX-REVA	15 of 31	
loop_count[7: In add loop_start [7:0 loop_step [7:0	dition: 0] [7:0] 0] [7:0]			1	
EXIST	to give more control to the c	ontrolling program.			
We will extend Boolean[31:0 Loop_count[7 Loop_Start[7: Loop_End[7:0	:0][15:0] 0] [15:0]				(Fouriette de Dollate and Novelación
6.2 The	Control Flow Prog	iram		4	
compared wit	h addresses (or address?) i	n a control table. The "ever	recuted, and at every instruction its nt" in the control table can redirect or lexity when the program size is increa	perations in	
	prefer is a "control program" rogram has four basic instruc xecute				
Loop_end					
Conditional e store to be ex Loop_start re	ecuted. sets the corresponding loop rements (decrements?) the	first, and if true, causes the counter to the start value.	ore to be executed. e specified number of instructions in the specified number of instructions		
if we to, and f	it the control flow instruction	s into 32 hit words, the follow	ving instructions are possible choices	e e	
in we ay and i			wing instructions are possible endices	<u></u>	
	8 27 26 25 24 23 22 2 0 instruction	mus manue manue anne manue manue	14 13 12 11 10 9 8 7 6 5 4 eerved Address	3210	
Execute up to	4K instructions at the speci	fied address in the instruction	on memory.		
	1 <u>Boolean</u> 0 1	Conditional Execute 21 20 19 18 17 16 15 x = 0 Instruction_cc x = 1 3:NA	14 13 12 11 10 9 8 7 6 5 4	3210	
	d boolean (6 bits can addre ip to 64 instructions)	ess 64 booleans) meets the	e specified condition then execute th	e specified	
<u>31 30 29 2</u> <u>0</u> <u>1</u> <u>0</u> 2	1 <u>Reserved =0</u> 0 1	Conditional Execute Pred 21 20 19 18 17 16 15 := 0 Instruction_co := 1 Instruction_co	14 13 12 11 10 9 8 7 6 5 4	3210	
Check the OF	t of all current predicate bits.	If OR matches the conditio	n execute the specified number of ins	structions.	
31 30 29 2	8 27 26 25 24 23 22 2	Loop_Start 21 20 19 18 17 16 15	14 13 12 11 10 9 8 7 6 5 4	3210	
Exhibit 2012.docR400_Sequ	iencer.docR400_Sequences.doc 32166 By	tes*** © ATI Confidential. F	Reference Copyright Notice on Cov 09/04/15 12:49 PM08/43/01-03:		

ATI Ex. 2105 IPR2023-00922 Page 77 of 239

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 200124 September, 200114	4 September, 20153		16 of 31	
0010				Loop ID	
Initialize the sp	pecified loop				
04 00 00 00		Loop End			
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$		(must = 0)	15 14 13 12 11 10 9 8 7 6 5 Start Address Reserved (must = 0)	4 3 2 1 0 Loop ID	
			h back to the specified address in th inite loop, the jump should be to loop		
the way this is	described does not preve	ent nested loops, and the i	inclusion of the loop id make this easy	to do.	
The basic mod	tel is as follows:				
The render sta	te defined the clause bou	indaries:			
	r_fetch[7:0][7:0] // eight	8 bit pointers to the location	on where each clauses control progran on where each clauses control progran		
Pixel_shader	fetch[7:0][7:0] // eight	8 bit pointers to the locati	on where each clauses control program	n is located	
Pixel shader	*********	***************************************	on where each clauses control progran		
			e is an offset added to the address fro for multiple programs resident at the sa		
			letion before moving to another clau	use, (with the	
exception of th	e pick two nature of the a	lu execution)			
The addresses same time.	s from the control progra	m are added to another	offset to allow for multiple programs r	esident at the	
Under this mo	del, all subroutine calls m	ust be inlined into the con	trol program.		
6.3 Data	dependant pred	icate instruction	S	ج- ا	Formatted: Bullets and Numbering
Data dependa		oported in the R400. The	only way we plan to support those is	by supporting	
	PRED_SETGT - similar	to SETGT except that the	esult is 'exported' to the sequencer. e result is 'exported' to the sequencer		
			the result is 'exported' to the sequence	er	
For the scalar	operations only we will all PRED_SETE0 – SETE0 PRED_SETE1 – SETE	2	ng instructions:		
maintain the 6	······································	use it to control the write	path as the MOVA instruction. The s masking (two sets for interleaved oper		
	two conditional execute b 1 or 0. For exemple, the		tional execute "on" bit and the second	bit tells us if	
PO_A	DD R0,R1,R2				
only write the I		e predicate bit is set. The	e predicate bit is 0. Alternatively, P1_A use of the P0 or P1 without prechargin		
{lssue: do we	have to have a NOP betw	een PRED and the first in	struction that uses a predicate?}		
			struction that uses a predicate?} al. Reference Copyright Notice on C 09/04/15 12:49 PM08/13/02		*

ATI Ex. 2105 IPR2023-00922 Page 78 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 200124 September, 200114	4 September, 20153	GEN-CXXXXX-REVA	17 of 31	

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.

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.

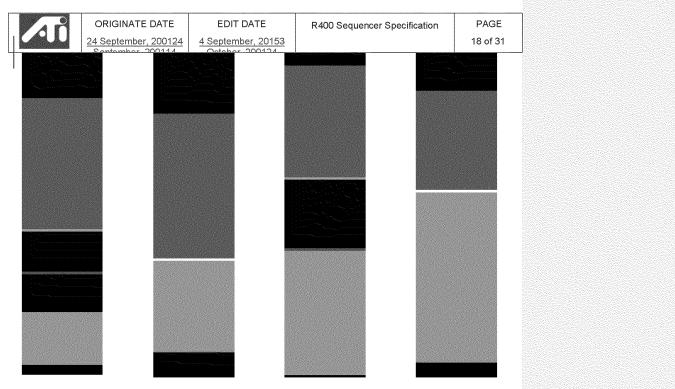
Exhibit 2012.docR400_Sequencer.docR400_Sequencer.doc R400_Sequencer.doc 32166 Bytes***
ATI Confidential. Reference Copyright Notice on Cover Page ***

AMD1044_0256751

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

ATI Ex. 2105 IPR2023-00922 Page 79 of 239



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 movinge again.

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.

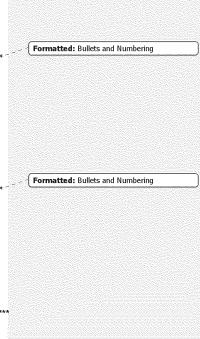
The arbitrator will not wait for the texture fetches to return prior to selecting another clause for execution. The texture pipe will be able to handle up to X(?) in flight texture fetches and thus there can be a fair number of active clauses waiting for their texture return data.

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.

Ehihbit 2012 docR400_Sequencer.docR400_Sequencer.doc 32166 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***



AMD1044_0256752

ATI Ex. 2105 IPR2023-00922 Page 80 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 200124	4 September, 20153	GEN-CXXXXX-REVA	19 of 31	Formatted: Bullets and Numbering
 2.5.1 dTb - 1.5			*	

9.10. 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.

10.11. Content of the reservation station FIFOs

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)

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

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[016]	parameter mask to specify wich parameters the pixel shader
FLAT_GOUR[016]	wich parameters are to be gouraud shaded
GEN TEX[016]	for wich parameters do we need to generate tex coords.
CYL WRAP[064]	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

Exhibit 2012.docR400_Sequencer.docR400_Sequencer.doc R400_Sequencer.doc 32166 Bytes***
ATI Confidential. Reference Copyright Notice on Cover Page

08/04/15/12/49 PM/08/13/04-03:17-PM

Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering

AMD1044_0256753

ATI Ex. 2105 IPR2023-00922 Page 81 of 239

	24 September, 20012	4 4 Septem	her 2014	R400 Sequencer Specification	20 of 31	
	Sontombor 200114		- 200124	<u> </u>	200101	Formatted: Bullets and Numbering
1 <u>2.14</u> . Int	terfaces				4. ·	
1 <u>2.114.1</u>	External Interfa	ices				
1 <u>2.1.1</u> 14.	1.1_Sequencer.to	Shader El	ngine E	BusPA/SC to RE : IJ bus		
This is a b	us that sends the instruc	ion and consta	nt data to	all 4 Sub-Engines of the Shader. Because a	new instruction	
is needed	only every 4 clocks, the	width of the b	us is divi	ded by 4 and both constants and instruction	n are sent over	
	ocks <u>This is a bus that</u> Introl information goes to		ormation	to the IJ fifos on the top of each shader pipe	e. At the same	
unic uic oc	sition anonination goes to	the sequencer				
Vame		ction	Bits	Description		
nstruction-Sta	rt <u>IJs</u> SEC	<u>→ SPPA→RE</u>	4 <u>64</u>	High on first cycle of transfer[J informatic clocks	on sent over 2	
			1	00083		
12.1.2-					4	
1 <u>2.1.2</u> Sha	der Engine to Ou	put File				
Even, cloc	k each Sub-Engine can	output 128 hits	of 'vecto	r' data and 32 bits of 'scalar' data to an outp	out file (?) This	
	compressed into 128 b					
	,		Q			
1410 -	z zana za za zana zana za				ada ser	,
14.1.2 PA	VSC to SEQ : IJ (Control bus			đe 194	
			r in order	to control the IJ fifos and all other informal	tion needed to	
This is the cor		the sequence	r in order	to control the IJ fifos and all other informat	tion needed to	
This is the cor execute a sha	ntrol information sent to der program on the ser	the sequence t pixels.			.tion needed to	
This is the cor execute a sha Name	ntrol information sent to der program on the ser Dire	the sequence t pixels. :tion	Bits	Description	tion needed to	
This is the cor execute a sha <u>Name</u> <u>Nrite Mask</u>	ntrol information sent to der program on the ser Dire PA-	the sequence t pixels.			tion needed to	
This is the cor execute a sha <u>Name</u> <u>Write Mask</u> R <u>B_ID</u>	trol information sent to der program on the ser Dire PA- PA- PA-	the sequences t pixels. :tion :SEQ(RE)	Bits 4	Description Quad Write mask left to right		
This is the cor execute a sha <u>Name</u> Write Mask RB ID LOD CORRE	trol information sent to der program on the ser Dire PA- CT PA- PA-	the sequences t pixels. tion SEQ(RE) SEQ(RE)	Bits 4 8	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad		Formatted: Bullets and Numbering
This is the con execute a sha Name Write Mask RB ID OD CORRE TVTX PPTR0	trol information sent to der program on the ser PA- CT PA- CT PA- PA- PA- PA- PA-	the sequences t pixels. SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE)	Bits 4 8 24 2 11	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0		Formatted: Bullets and Numbering
This is the con execute a sha Write Mask RB ID LOD CORRE EVTX PPTR0 PPRT1	trol information sent to der program on the ser PA- PA- CT PA- PA- PA- PA- PA- PA- PA- PA-	the sequences t pixels. SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE)	Bits 4 8 24 2 11 11	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1		• Formatted: Bullets and Numbering
This is the con execute a sha Write Mask RB ID LOD CORRE EVTX PPTR0 PPRT1 PPTR2	trol information sent to der program on the ser PA- CT PA- CT PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequences t pixels. SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE)	Bits 4 8 24 11 11 11	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2		• Formatted: Bullets and Numbering
This is the con execute a sha <u>Nrite Mask</u> RB ID OD CORRE FVTX PPTR0 PPR1 PPTR2 E OFF VEC1	trol information sent to der program on the ser PA- PA- CT PA- PA- PA- PA- PA- PA- PA- OR PA-	the sequences t pixels. SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE)	Bits 4 8 24 11 11 11 11	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector		• Formatted: Bullets and Numbering
This is the col execute a sha Write Mask RB_ID OD_CORRE EVTX PPTR0 PPTR0 PPTR1 PPTR2 E_OFF_VEC1 DEALLOC	trol information sent to der program on the ser PA- CT PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequences t pixels. SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE)	Bits 4 8 24 11 11 11 11 11	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store		• Formatted: Bullets and Numbering
This is the cor execute a sha Write Mask RB ID OD CORRE VTX PPTR0 PPTR0 PPTR1 PPTR2 E OFF VEC1 DEALLOC STATE	trol information sent to der program on the ser PA- PA- CT PA- PA- PA- PA- OR PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequence t pixels. seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE)	Bits 4 8 24 11 11 11 11	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3)		Formatted: Bullets and Numbering
This is the col execute a sha Nrite Mask RB ID OD CORRE OTX PPTR0 PPTR0 PPTR1 PPTR2 OFF VEC1 DEALLOC STATE SPRITE	trol information sent to der program on the ser PA- PA- CT PA- CT PA- PA- PA- OR PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequences t pixels. SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE)	Bits 4 8 24 11 11 11 11 12 13 14	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store		Formatted: Bullets and Numbering
This is the col execute a sha <u>Name</u> <u>Nrite Mask</u> RB ID <u>OD CORRE</u> <u>OD CORRE</u> <u>OD CORRE</u> <u>OD CORRE</u> <u>OD CORRE</u> <u>OD FORE</u> <u>OD F</u>	trol information sent to der program on the ser PA- PA- CT PA- CT PA- PA- PA- OR PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequence t pixels. :tion SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE) SEQ(RE)	Bits 4 8 24 2 11 11 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords		Formatted: Bullets and Numbering
This is the cor execute a sha Write Mask RB ID LOD CORRE FVTX PPTR0 PPR1 PPTR2 E OFF VECT DEALLOC STATE SPRITE VALID NULL E OFF PRIM	trol information sent to der program on the ser PA- PA- CT PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequence t pixels. seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE)	Bits 4 8 24 2 11 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords Valid bits for all pixels Null Primitive (for deallocation purposes) End Of the primitive		Formatted: Bullets and Numbering
This is the cor execute a sha Write Mask RB ID LOD CORRE FVTX PPTR0 PPTR0 PPTR0 PPTR2 E OFF VECT DEALLOC STATE SPRITE VALID NULL E OFF PRIM FBFACE	trol information sent to der program on the ser PA- PA- CT PA- PA- PA- PA- OR PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequence t pixels. seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE)	Bits 4 8 24 2 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords Valid bits for all pixels Null Primitive (for deallocation purposes) End of the primitive		Formatted: Bullets and Numbering
This is the cor	trol information sent to der program on the ser PA- PA- CT PA- PA- PA- PA- OR PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequence t pixels. seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE) seq(RE)	Bits 4 8 24 2 11 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords Valid bits for all pixels Null Primitive (for deallocation purposes) End of the primitive Front face = 1, back face = 0 Stippled line need to load tex cords for		Formatted: Bullets and Numbering
This is the cor execute a sha Write Mask RB_ID_ LOD_CORRE FVTX PPTR0 PPTR0 PPTR2 E OFF_VEC1 DEALLOC STATE SPRITE VALID NULL E OFF_PRIM FBFACE STIPPLE_LIN	trol information sent to der program on the ser PA- PA- CT PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequences t pixels. :tion SEQ(RE)	Bits 4 8 24 1 11 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Povoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords Valid bits for all pixels Null Primitive (for deallocation purposes) End Of the primitive Front face = 1, back face = 0 Stippled line need to load tex cords f		Formatted: Bullets and Numbering
This is the cor execute a sha Write Mask RB_ID OD_CORRE VTX PPTR0 PPTR0 PPTR2 E_OFF_VEC1 DEALLOC STATE SPRITE VALID NULL E_OFF_PRIM FBFACE STIPPLE_LIN RTRn	trol information sent to der program on the ser PA- PA- PA- CT PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequence t pixels. :tion SEQ(RE) SEQ(Bits 4 8 24 2 11 11 11 11 11 11 11 11 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords Valid bits for all pixels Null Primitive (for deallocation purposes) End Of the primitive Front face = 1, back face = 0 Stippled line need to load tex cords f buffer Stalls the PA in n clocks		Formatted: Bullets and Numbering
This is the col execute a sha Nrite Mask RB ID OD CORRE OT X PPTR0 PPTR0 PPTR0 PPTR2	trol information sent to der program on the ser PA- PA- PA- CT PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequences t pixels. :tion SEQ(RE)	Bits 4 8 24 1 11 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Povoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords Valid bits for all pixels Null Primitive (for deallocation purposes) End Of the primitive Front face = 1, back face = 0 Stippled line need to load tex cords f		Formatted: Bullets and Numbering
This is the cor execute a sha Write Mask RB_ID_ OD_CORRE VTX PPTR0 PPRT1 PPTR2 EOFF_VEC1 DEALLOC STATE SPRITE VALID NULL EOFF_PRIM EBFACE STIPPLE_LIN RTRn RTS	trol information sent to der program on the ser PA- PA- PA- CT PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequence t pixels. :tion SEQ(RE)	Bits 4 8 24 2 11 11 11 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords Valid bits for all pixels Null Primitive (for deallocation purposes) End Of the primitive Front face = 1, back face = 0 Stippled line need to load tex cords f buffer Stalls the PA in n clocks		
This is the cor execute a sha Write Mask RB_ID_ OD_CORRE VTX PPTR0 PPRT1 PPTR2 EOFF_VEC1 DEALLOC STATE SPRITE VALID NULL EOFF_PRIM EBFACE STIPPLE_LIN RTRn RTS	trol information sent to der program on the ser PA- PA- PA- CT PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequence t pixels. :tion SEQ(RE)	Bits 4 8 24 2 11 11 11 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords Valid bits for all pixels Null Primitive (for deallocation purposes) End Of the primitive Front face = 1, back face = 0 Stippled line need to load tex cords f buffer Stalls the PA in n clocks		• Formatted: Bullets and Numbering
This is the col execute a sha Name Write Mask RB ID LOD CORRE FVTX PPTR0 PPTR0 PPTR2 E OFF VECT DEALLOC STATE SPRITE VALID NULL E OFF PRIM FBFACE STIPPLE LIN RTRN RTS 14.1.3 PA	trol information sent to der program on the ser PA- CT PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequence t pixels. tion SEQ(RE)	Bits 4 8 24 2 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords Valid bits for all pixels Null Primitive (for deallocation purposes) End Of the primitive Front face = 1, back face = 0 Statis the PA in n clocks PA ready to send data		
This is the cor execute a sha Write Mask RB_ID_ OD_CORRE VTX PPTR0 PPRT1 PPTR2 EOFF_VEC1 DEALLOC STATE SPRITE VALID NULL EOFF_PRIM EBFACE STIPPLE_LIN RTRn RTS	trol information sent to der program on the ser PA- CT PA- PA- PA- PA- PA- PA- PA- PA- PA- PA-	the sequence t pixels. :tion SEQ(RE)	Bits 4 8 24 2 11 11 11 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad LOD correction per quad (6 bits per quad Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Need to generate tex cords Valid bits for all pixels Null Primitive (for deallocation purposes) End Of the primitive Front face = 1, back face = 0 Stippled line need to load tex cords f buffer Stalls the PA in n clocks		

ATI Ex. 2105 IPR2023-00922 Page 82 of 239

200124 4 Sept 200124 \bigcirc Sept Direction \bigcirc SEQ Direction \bigcirc CP \rightarrow SEQ CP \rightarrow SEQ $ \bigcirc$ CP \rightarrow SEQ	Bits Descr 21 Rende 0ad Bits Descr 8 Addre 512 Data s 10 Rema 8 Rema 0re Ioad Bits Descr 8 Rema 8 Rema 0re Ioad Bits Descr 8 Rema Addre 512 Data s 10 Rema 8 Rema 0re Ioad 8 Rema 8 Rema 8 Rema	iption ss of the block of 4 constants ent over X clocks oing address write address oing pointer iption ss of the block of 4 state constants ent over X clocks oing address write address oing address write address	21 of 31	 Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering
: Vertex Contr Direction $PA \rightarrow SEQ$ Direction $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ CP	Bits Descr 21 Rende 0ad Bits Bits Descr 8 Addre 512 Datas 10 Rema 8 Addre 512 Datas 10 Rema 8 Addre 512 Datas 8 Addre 512 Datas 10 Rema 8 Addre 512 Datas 10 Rema 8 Rema 0re Ioad	iption ss of the block of 4 constants ent over X clocks oing address write address oing pointer iption ss of the block of 4 state constants ent over X clocks oing address write address oing address write address		
$\begin{array}{c} \hline \text{Direction} \\ \hline \text{PA} \rightarrow \text{SEQ} \\ \hline \text{Onstant store log} \\ \hline \text{Onstant store log} \\ \hline \text{Or-SEQ} \\ \hline \text{CP} \rightarrow \text{SEQ} \\ \hline \text{Or-SEQ} \\ \hline \text{CP} \rightarrow \text{SEQ} \\ \hline \text{Or-SEQ} \\ \hline \text{CP} \rightarrow \text{SEQ} \hline \hline \text{CP} \rightarrow \text{SEQ} \\ \hline \text{CP} \rightarrow \text{SEQ} \hline \hline \text{CP} \rightarrow \text{SEQ} \\ \hline \text{CP} \rightarrow \text{SEQ} \hline \hline \ \text{CP} \rightarrow \text{SEQ} \hline \hline \ \text{CP}$	Bits Descr 21 Rende 0ad Bits Descr 8 Addre Addre 512 Datas Bits Descr 10 Rema Rema Bits Descr 8 Addre Addre Bits Datas S 10 Rema Rema Bits Descr Batas Addre 512 Datas 10 Rema Rema Batas Rema	iption ss of the block of 4 constants ent over X clocks oing address write address oing pointer iption ss of the block of 4 state constants ent over X clocks oing address write address oing address write address		
$PA \rightarrow SEQ$ Direction $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ Direction $CP \rightarrow SEQ$ $CP \rightarrow SEQ$	21 Rende 0ad Bits Description 8 Addressing Addressing 512 Data s 10 8 Rema 0re load 8 Addressing 9 Rema 0re load 10 Rema 10 Rema 10 Rema 8 Rema 10 Rema 8 Rema	iption ss of the block of 4 constants ent over X clocks oing address write address oing pointer iption ss of the block of 4 state constants ent over X clocks oing address write address oing address write address		
Direction $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ exture State state Direction $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$	Description Bits Description 8 Addre 512 Data s 10 Rema 0re load Bits Description 8 Addre 512 Data s 10 Rema 8 Addre 512 Data s 10 Rema 8 Rema 0 Rema	iption ss of the block of 4 constants ent over X clocks oing address write address oing pointer iption ss of the block of 4 state constants ent over X clocks oing address write address oing pointer		
Direction $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ exture State state Direction $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$	Bits Descr 8 Addre 512 Data s 10 Rema 8 Rema ore load 8 Addre 9 Bits Descr 8 Addre Addre 512 Data s 10 8 Addre 512 512 Data s 10 8 Rema 8 9 Rema 8 9 Rema 8	ss of the block of 4 constants ent over X clocks bing address write address bing pointer iption ss of the block of 4 state constants ent over X clocks bing address write address bing pointer		
Direction $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ exture State state Direction $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$	Bits Descr 8 Addre 512 Data s 10 Rema 8 Rema ore load 8 Addre 9 Bits Descr 8 Addre Addre 512 Data s 10 8 Addre 512 512 Data s 10 8 Rema 8 9 Rema 8 9 Rema 8	ss of the block of 4 constants ent over X clocks bing address write address bing pointer iption ss of the block of 4 state constants ent over X clocks bing address write address bing pointer		Formatted: Bullets and Numbering
$\begin{array}{c} \hline CP \rightarrow SEQ \\ \hline exture State state state \\ \hline Direction \\ \hline CP \rightarrow SEQ \hline CP \rightarrow SEQ \\ \hline CP \rightarrow SEQ \hline CP \rightarrow SEQ \\ \hline CP \rightarrow SEQ \hline$	8 Addre 512 Data s 10 Rema 8 Rema ore load	ss of the block of 4 constants ent over X clocks bing address write address bing pointer iption ss of the block of 4 state constants ent over X clocks bing address write address bing pointer		Formatted: Bullets and Numbering
CP→SEQ CP→SEQ Direction CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ	10 Rema 8 Rema ore load Bits Descr 8 Addre 512 Data s 10 Rema 8 Rema 9 Rema 9 Rema	ping address write address ping pointer iption ss of the block of 4 state constants ent over X clocks ping address write address ping pointer		Formatted: Bullets and Numbering
CP→SEQ Direction CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ	8 Rema ore load Bits Descr 8 Addre 512 Data s 10 Rema 8 Rema 0 Rema	iption ss of the block of 4 state constants ent over X clocks oing address write address oing pointer	· · ·	Formatted: Bullets and Numbering
exture State sto Direction CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ	Bits Descr 8 Addre 512 Datas 10 Rema 8 Rema 000000000000000000000000000000000000	iption ss of the block of 4 state constants ent over X clocks ping address write address ping pointer		Formatted: Bullets and Numbering
Direction $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ $CP \rightarrow SEQ$ Difference of the state	Bits Descr 8 Addre 512 Data s 10 Rema 8 Rema	ss of the block of 4 state constants ent over X clocks ping address write address ping pointer	<pre>> + </pre>	 Formattee, buncts and wumbering
$\begin{array}{c} \underline{CP \rightarrow SEQ} \\ \underline{CP \rightarrow SEQ} \\ \underline{CP \rightarrow SEQ} \\ \underline{CP \rightarrow SEQ} \\ \underline{CP \rightarrow SEQ} \end{array}$	8 Addre 512 Data s 10 Rema 8 Rema 0 Rema	ss of the block of 4 state constants ent over X clocks ping address write address ping pointer		
CP→SEQ CP→SEQ CP→SEQ CP→SEQ	512 Data s 10 Rema 8 Rema	ent over X clocks ping address write address ping pointer		
<u>CP→SEQ</u> <u>CP→SEQ</u> ontrol State sto	10 Rema 8 Rema	oing address write address oing pointer	•	
<u>CP→SEQ</u> ontrol State sto	8 Rema	ping pointer		
ontrol State sto	ore load		*-	
			4-	
				Formatted: Bullets and Numbering
Direction	Bits Descr			
		iption]	
Struction store	Bits Descr 12 Instruc	tion address		
ALALIA DE LA CALLER DE LA CALLE	ALASIAN ALASIA			
Analyziniana and an	Construction of the local division of the lo			
el read from Ri	<u>Bs</u>		4-	- (Formatted: Bullets and Numbering
Direction	Bits Descr	iption		
OB→RB				
<u>OR→KR</u>	20 XY ad	aress 10 bits per		
· Docition rotur	m hue		*-	- Formatted: Bullets and Numbering
Direction	AND	adorementarian		
	And the second s			
SP→PA	KONTRON EDGODIAGOGINOTATIA		iche	
	$\begin{array}{l} \underline{MH} \rightarrow \underline{SEQ} \\ \underline{OB} \rightarrow \underline{RB} \\ \underline{SP} \rightarrow \underline{PA} \\ \underline{SP} = \underline{SP} \\ \underline{SP} = \underline{SP} \\ \underline{SP} = \underline{SP} \\ \underline{SP} \\ \underline{SP} = \underline{SP} \\ \underline{SP} \\$	MH→SEQ 96 Instruct MH→SEQ 8 Pointe MH→SEQ 32 Control MH→SEQ 32 Control MH→SEQ 32 Control Direction Bits Descr OB→RB 128 2 pixel OB→RB 20 XY add Position return bus Direction Bits Descr SP→PA 128 Positic SP→PA SP→PA 72 Pointe SP→PA 11 Pointe Engine to Texture Unit Bus Engine to Texture Unit Bus	MH \rightarrow SEQ 96 Instruction X times MH \rightarrow SEQ 8 Pointer to the instruction store MH \rightarrow SEQ 32 Control Instruction X times Direction Bits Description OB \rightarrow RB 128 2 pixels (or ½ quad) OB \rightarrow RB 20 XY address 10 bits per Position return bus Direction Bits Direction Bits Description SP \rightarrow PA 128 Position data or sprite size SP \rightarrow PA 72 Pointer to the position cache SP \rightarrow PA 11 Pointer where the data will be in the parameter ca Engine to Texture Unit Bus (Fast Bus) Engine to Texture Unit Bus (Fast Bus)	MH-SEQ 96 Instruction X times MH-SEQ 8 Pointer to the instruction store MH-SEQ 32 Control Instruction X times MH-SEQ 32 Control Instruction X times Pirection Bits Description OB-RB 128 2 pixels (or ½ quad) OB-RB 20 XY address 10 bits per Position return bus ************************************

ATI Ex. 2105 IPR2023-00922 Page 83 of 239

A î	ORIGINATE DATE		ATE	R400 Sequencer Specification	PAGE 22 of 31
Name	Dire	tion	Bits	Description	
Tay Dood	Pagistar Inday SEA	~CD	Q	Index into Degister files for reading Texts	Iro Addross

Tex_Read_Register_Index	SEQ->SP	8	Index into Register files for reading Texture Address
Tex_RegFile_Read_Data	SP->TEX	5122048	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

<u>12.1.414.1.12</u> Sequencer to Texture Unit bus (Slow Bus)

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	$TEX \rightarrow SEQ$	1	Data ready
Tex_Clause_Num	$TEX \rightarrow SEQ$	3	Clause number
Tex_cst	SEQ→TEX	? <u>10</u>	Texture constants Xstate address 10 bits sent over 4 clocks
Tex_Inst	SEQ→TEX	? <u>12</u>	Texture fetch instruction Xaddress 12 bits sent over 4 clocks

12.1.5 Shader Engine to RE/PA Bus

12.1.6 PA? to sequencer

13.15. Examples of program executions

<u>13.1.115.1.1</u> Sequencer Control of a Vector of Vertices

- 1. PA sends a vector of 16-<u>64</u> vertices (actually vertex indices 32 bits/index for 512-<u>2048</u> 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
- 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

Ehibit 2012 docR400_Sequencer.docR400_Sequencer.doc 32166 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:49 PM08/H3/0-103:17 PM

Formatted: Bullets and Numbering

AMD1044_0256756

ATI Ex. 2105 IPR2023-00922 Page 84 of 239

Original Energy End of the provide state of the set of the							
 It is a control packet control to the set of the set			ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
 RF3 of SUO, SU1, SU2, and SU3 is written the fourth cycle the index is written to the least significant 32 bits (floating point formar?) (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) SED constructs a control packet forth we 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. TSM0 accepts the control packet and fetches the instructions for texture clause 0 from the global instruction store TSM0 does not wait for requests made to the Texture Unit to complete; it passes the register file write index for the toture 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; ocunt grader than zero indicates that the ALU state machine can go shead start to execute the 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) di instructions and the appet down the path of reservation stations until all clauses have been executed is shared with all four shader pipes) back to the PA is position cache. A parameter cache point is allow smithing and guidting parameter cache. the SED allocates storage in the Parameter Cache over a declared bus the SED allocates storage in the Parameter Cache over a declared bus the SED allocates exported in study of the Parameter Cache over a declared bus the SED allocates exported in study of the parameter Cache over a declared bus the SED allocates exported in study of the Parameter Cache over a declared bus to the PA where the	ſ	<u>AU</u>		4 September, 20153	GEN-CXXXXX-REVA	23 of 31	
 the control packet contains the state pointer, the tag to the position cache and a register file base pointer. 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 statt all instructions of texture clause 0 are issued by TSM0 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 writen 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 0 form the global instruction store. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 form the global instruction store. 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). the control packet control packet to instands to the PA where the data is given with all parameter cache, onither is also sent along with the position data. This tells to the PA where the data is shared with all four stater thom stating an exporting duale if the position export FIFO is full parameter data is sent to the Parameter Cache, and the SEQ hailow export FIFO is full parameter data is sent to the Parameter Cache, and the SEQ hailow export of the parameter cache is ontrol of the parameter cache, and the SEQ hailow export of the parameter cache is ontrol of the parameter cache bused to the PA when using a token). there is a position export FIFO in the SEQ will fr	odži	 the in- 	F3 of SU0, SU1, SU2, and s dex is written to the least sig	SU3 is written the fourth cycl gnificant 32 bits (floating po	int format?) (what about compour	nd indices)	
 6. TSM0 vacepts the control packet and fetches the instructions for texture clause 0 from the global instruction store TSM0 vacepts the control packet and fetches the instructions for texture clause 0 from the global instruction store TSM0 vacepts the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO, a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause 0 from the global instruction store 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 ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for 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) 10. the control packet control backet control store with the position data. This tells to the PA where the data is good with all of the reservation station glause if the position with path of position can be exported in ALU clause 0 (or 47); the data (and the tag) is sent over a position bus (which is shared with all four stader pipes) backet to the PA so position cance. 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, and the SEQ deallocates that space when there is no inorger a need for the parameter (ache to aver a dealcated bus the rest a position is being exported) in Aluz Clause 7 (as well as position data. This tells to the PA where the data is good to the PA store with prevent packet from starting on a SMM rith the position cance to restrict with increase the space when there is no inorger a need for the parameter Cache, and the SEQ deallocates that space when there is no inorger a need for the parameter Cache and the SEQ deallocat	Ę	texture sta	ate machine 0, or TSM0 FIF	O)			
 TSM0 was first selected by the TSM arbiter before it could start T all instructions of texture clause 0 are issued by TSM0 The control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO). T SM0 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 register file, it increments a counter that is associated with ASM0 FIFO, a count greater than zero indicates that the ALU state machine can go alread start to execute the ALU clause A SM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for 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) If the control packet continues to travel down the path of reservation stations until all clauses have been executed explored in ALU clause 3 (or 47); 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 sent along with the position data. This tells to the PA where the data is going to be in the parameter cache over a decideaded bus the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) in 51U parameter data is esent othe Parameter Cache, and the SEQ datalocates that space when there is no longer a need for the parameter and set as position data. This tells to the Parameter and the set of the parameter set is foll. 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. after the shader program has completed, the SEQ	c						
 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. a. 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. A. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause. 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 47); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA sposition 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 cache storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is toid 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. after the shader program has completed, the SEQ will fee up the GPRs so that they can be used by another shader program. As with vertex shader program has completer. Cache before the SEQ takes control of the vector effect the Parameter Juffer is du	,				•	uction store	
 FFO) • TSM 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 ASMO FIFO, a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause 0 from the global instruction store 9. ASMO accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 are issued by ASMO, then the control packet is passed to the next reservation station (the FIFO) in fort 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 47); the data (and the tag) is sent over a position bus (which is shared with all Boy shader pipes) back to the PA se 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 to the parameter bek to the PA • the ASM arbiter will prevent a packet from stating on exporting clause if the position packet FIFO is full • parameter data is sent of the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameter data is lood by the PA when using a token). • the ASM arbiter will prevent a packet from stating on ASM7 if the parameter cache (or the position buffer if position is being exported) is full 12. after the shader program, pixel shaders are loaded into the global instruction store by the CP • A this point it is assumed that the pixel program is loaded into the global instruction store by the CP •	7	. all instruct	ions of texture clause 0 are	issued by TSM0			
 clause 0 from the global instruction store 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 47); 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. 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 there is a position export FIFO in the SP that buffers position data before it gets when the tag to exported earlier) parameter data is sent to the Parameter Cache over a decloated bus the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameter (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 the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program. 13.1.215.1.2 Sequencer Control of a Vector of Pixels 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 global instruction store one cycle at the moment the right sequencer will free up the parameter store locations not used anymore using the token provid	Ę	FIFO) • TSMC for the • once f FIFO;	does not wait for requests texture data to the TU, whi the TU has written all the da a count greater than zero in	made to the Texture Unit to ich will write the data to the I ata to the register files, it incr	complete; it passes the register file v RF as it is received ements a counter that is associated	vrite index with ASM0	
 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 47); 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 stating an exporting clause if the position export FIFO is full 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 stating 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. 13.1.215.1.2 Sequencer Control of a Vector of Pixels 4. At this point it is assumed that the pixel program is loaded into the global instruction store by the CP 6. At this point it is assumed that the pixel program is loaded into the sEQ takes control of the vector "after the H2 culling dtage a request is made by the RE to send parameter and to the Parameter buffer is wide enough to source -3 vertices worth of a 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 Parameter Buffer is loaded with the barycentric coordinates for pixel quads by the detailed walk	ç				۸ arbiter) and gets the instructions fo	or ALU	
 position can be exported in ALU clause 3 (or 47); 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 there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA there is a position export FIFO in the SP that buffers position data if it was not exported earlier) parameter data is exported in clause 7 (as well as position data if it was not exported earlier) parameter data is exported 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. 13.1.215.1.2 Sequencer Control of a Vector of Pixels 1. As with vertex shader programs, pixel shaders are loaded into the 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 lakes control of the vector * after the H2 culling stage a request is made by the RE to eend parameter in one cycle ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** ** **<						ervation	
shader program 13.1.2 <u>15.1.2</u> 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 • after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer • after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer • after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer • after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer • after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer • after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer • after 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 FIFO • the Pixel FIFO is wide enough to source one four quad's worth of barycentrics per cycle 4. Fo bift in the register files for vertices, the Pixel FIFO – when there are no vertices pending OR there is no space 5.4. SEQ allocates space in the SP register file for all the GPRs used by the program Exhibit 2012 docRH00_Sequenced.com/COS_Sequenced.com/COS_Sequenced.com/COS_Sequenced.com/COS_Sequenced.com/COS_Sequenced.com/COS_Sequenced.com/COS_Sequen		 positic share A para going th th paran paran paran th lo th 	on can be exported in ALU of d with all four shader pipes) ameter cache pointer is also to be in the parameter cach ere is a position export FIFO e ASM arbiter will prevent a teter data is exported in clar arameter data is sent to the e SEQ allocates storage in nger a need for the parame e ASM arbiter will prevent a position is being exported) i	clause 3 (or 4?); the data (ar back to the PA's position ca sent along with the position packet from starting an exp use 7 (as well as position da Parameter Cache over a de the Parameter Cache, and t ters (it is told by the PA whe packet from starting on ASI s full	nd the tag) is sent over a position bus ache n data. This tells to the PA where the tion data before it gets sent back to t iorting clause if the position export FI ta if it was not exported earlier) dicated bus he SEQ deallocates that space wher n using a token). M7 if the parameter cache (or the pos	s (which is data is he PA FO is full n there is no sition buffer	
 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. Che 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. 3.2. 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 FIFO 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 for 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 		shader pr	ogram			Inother	1fi
 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 after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer after the HZ culling stage a request is made by the RE to send parameter data to the Parameter buffer after the HZ culling stage a request 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 FIFO 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 a 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 	-	<u>-3.1.215.</u>	1.2_Sequencer Con	ntrol of a Vector of Pl	ixels	**	
 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. 3.2. 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 FIFO 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 5.4. SEQ allocates space in the SP register file for all the GPRs used by the program 		. As with v	ertex shader programs, p	ixel shaders are loaded int	o the global instruction store by t	he CP	
 2. The KE C Farameter buffer is vide on the Parameter is not used on the Vector of the		At this	point it is assumed that the	e pixel program is loaded inte	o the instruction store and thus ready	to be read.	
the state pointer and the LOD correction bits are also placed in the Pixel FIF0 the Pixel FIFO is wide enough to source one <u>four</u> 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 5.4. SEQ allocates space in the SP register file for all the GPRs used by the program Exhibit 2012 docR400_Sequencer.docR400_Sequencer.doc 32166 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***	ź	∙after the ∘the Para ∘at this m	HZ culling stage a request i meter buffer is wide enough ioment the right sequence	s made by the RE to send p to source 3 vertices worth c	arameter data to the Parameter buffe of a particular parameter in one cycle	eer ef	{[
4-5_SEQ albitrates between Fixel FIFO and vertex FIFO - when there are no vertices pending OK 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 Exhibit 2012 docR400_Sequencer.doc R400_Sequencer.doc 32166 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***		 the state 	ate pointer and the LOD cor	rection bits are also placed i	in the Pixel FIF0		
Exhibit 2012. docR400_Sequencer.docR400_Sequencer.doc 32166 Bytes*** ATI Confidential. Reference Copyright Notice on Cover Page ***	4				e are no vertices pending OR there is	s no space 🔹	1 !
	ų	4.SEQ alloc	ates space in the SP registe	er file for all the GPRs used	by the program		
	<u>Exhibi</u>	2012.docR400_Sequ	encer.docR400_Sequencer.doc 32166 By	๙๙๙๙๙๙ © ATI Confidential. F			ł

ATI Ex. 2105 IPR2023-00922 Page 85 of 239

				DAOE	
	ORIGINATE DATE 24 September, 200124	EDIT DATE 4 September, 20153	R400 Sequencer Specification	PAGE 24 of 31	
the sum	Sontombor 200114	October 200124	a local state register, which is accesse		
I state po	ointer		-	-	
1			er until space in the register file has be		
	,	n of parameters (up to 16 ters from the Parameter E	per thread) by sending the barycentric Suffer to the interpolator	c-coordinates	
	· ·	it each vertex) are loaded erpolator from the Pixel F	into the interpolator from the Paramet	er buffer	
•The interp	olator then generates the	parameter value for each	pixel in Q0 (Q0P0)		
			if Q1 is from a different primitive; if alues loaded for Q0 are held by the i		
and-rei	used for Q1		-	-	
			centric data for each quad in the Pi e loaded into the interpolator	xerriru tilat	
		e interpolator from the Pi parameter value for each			
•Q2P0 and	Q3P0 are generated in a	similar manner			
1	,	· · · · · · · · · · · · · · · · · · ·	paded into the interpolator means that the output of the Pixel FIF	O-loops	
		ach read command until a centric coordinates to fina	at the end a final "block_pop" signal is a ally be removed	asserted,	
			P0, Q3P0, Q0P1, Q1P1, etc.		
		ated data to the SP registered interpolated data bus of	er file over the RE_SP interface (which	ו has a	
•16 pixels v	vorth of interpolated para	meter data is sent to the	16 register files over 4 cycles		
	1 1 1	J3 is written with Q0P0 the J3 is written with Q1P0 se	<i>,</i>		
•RF2-0	f SUO, SU1, SU2, and SU	J3 is written with Q2P0 thi	ird-cycle		
		J3 is written with Q3P0 for	*		
	e machine 0, or TSM0 FI		the first reservation station (the FIFO	in front of	
			piters/state machines for vertices and t ile base pointer, and the LOD correction		
			travels in a separate FIFO	511 5113	
			for texture clause 0 from the global ins	struction store *	
1	was first selected by the i uctions of texture clause	SM arbiter before it could	Istart	4	Formatted: Bullets and Numbering
			ו (the FIFO in front of ALU state machi	ine 0, or	
ASM0 FIFC)				
index fo	or the texture data to the	TU, which will write the da	ure Unit to complete; it passes the reg tta to the RF as it is received		
			to the register files, it increments a co indicates that the ALU state machine		
ahead a	and pop the FIFO and sta	art to execute the ALU cla	use	-	Example and Dulleto and Numbering
	accepts the control packe om the global instruction s		the ASM arbiter) and gets the instructi	ions for ALU	– – Formatted: Bullets and Numbering
		are issued by ASM0, then tate machine 1, or TSM1	the control packet is passed to the net FIFO)	xt reservation	
44. <u>12.</u> the con executed	trol packet continues to t	ravel down the path of res	ervation stations until all clauses have	been	
 pixel da 	ata is exported in the last				
		where it will be picked up a packet from starting on	by the render backend ASM7 if the output FIFO is full		
Ehibit 2012.docR400_Sequen	cer.docR400_Sequencer.doc 32166	Bytes*** © ATI Confidentia	al. Reference Copyright Notice on C		
I			<u>09/04/15 12:49 PM</u> 08/13/01	L03:17 PM	

ATI Ex. 2105 IPR2023-00922 Page 86 of 239

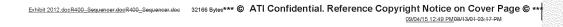
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE		
	24 September, 200124	4 September, 20153	GEN-CXXXXX-REVA	25 of 31		
15.12 offer t		ploted the SEO will free up	the CDPs on that they can be used	hy another *	Formatted: Bullets and Numbering	癌

45.13._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

13.1.315.1.3 Notes

- 46.<u>14.</u> the state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall.
- 47.15. 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
- <u>16.</u> Waterfalling, parameter buffer allocation, loops and branches and parameter cache de-allocation still needs to be specked out.



AMD1044_0256759

ATI Ex. 2105 IPR2023-00922 Page 87 of 239

		NATE		4 4		IT DAT			R400	Seque	ncer S	pecific	ation		PAG 26 of 3	_		
4. <u>Timing</u> 4.1 MAC (gram	<u>15</u>															
		1 4			1 .	1 -		-			10		40	1 40		45	1.40	47
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
SEQ_SP_constant0 SEQ_SP_constant1 SEQ_SP_read_addl SEQ_SP_phase RE_SP_data[511:384] SEQ_SP_instructior SEQ_SP_instr_star	-	srcA	srcB	srcC	тс	C0_0 C1_0 srcA	C0_1 C1_1 srcB	C0_2 C1_2 srcC	C0_3 C1_3 TC I0_2	srcA	srcB	srcC	тс	srcA ID	srcB	srcC	TC	srcA
							<u>۲</u>	Γ										
mac0_phase			/	∱—			<u>├</u> ──	<u>^</u>			ļ/	<u>`</u>				<u> </u>		
mac0_cycle_count	t		0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
RF0_read_data	ı							srcA	srcB	srcC	тс							
mac0_vector_result	t													а	r	g	b	
SEQ_SP_write_addu RF0 write cycle	1	ID	- ID	PV TD	PS PV	ID PS	- ID	PV TD	PS PV	ID PS	- ID	PV TD	PS PV	ID PS	- ID	PV TD	PS PV	PS
				1	1	1												

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

14.2 Sequencer to Shader Pipe

Ehibit 2012 docR400_Sequencer.docR400_Sequencer.docR400_Sequencer.doc 32166 Bytes*** S ATI Confidential. Reference Copyright Notice on Cover Page S ***

AMD1044_0256760

Formatted: Bullets and Numbering

ATI Ex. 2105 IPR2023-00922 Page 88 of 239

A î	24 Sep	GINATI	r, 2001	24	4 Sep	DIT D	r, 2015:	3			NT-RE		1.	PA 27 o					
]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
_	_SEQ_rts																		
PXF_SEQ_n				ケ			/	┝			ļ/—		h		ļ		٦		
	INT_data	QO	Q0	Q1	Q2	Q3	Q0	Q1	Q2	Q3	Q0'	Q1"	Q2 "	Q3"	Q0'	Q1"	Q2 "	Q3"	x
-	_PXF_rtr		ļ/																
SEQ_PXF_ve	ector_pop		ļ		ļ					/	$h_{}$				ļ			Ļ	
PMB_I	INT_data	PO	PO	P1	P1	P1	P1	P0 '	P0 '	P0'	P0'	P0"	P1'	P1'	P1'	P1"	x	x	x
SEQ_INT_			↓/	∱				┝					┝				▶		
INT_pa	aram_reg	х	х	PO	PO	PO	PO	P1	P1	₽1	P1	P0'	P0"	P0"	P0"	P1'	P1"	P1"	P1"
SEQ_INT			/	1															
INT_C	quad_reg	х	x	QQ	Q1	Q2	Q3	QO	Q1	Q2	Q3	Q0'	Q1"	Q2 "	Q3"	Q0'	Q1"	Q2 "	Q3"
-	P_phase		/	∱			/	╲			/	┝				┪			<u></u>
SEQ_SP_w	_						ID				ID				ID				ID
	SP_valid						ļ/	┝			/	ケ				T			
RE_	_SP_data						Q0P0	Q1P0	Q2P0	Q3P0	Q0P1	Q1P1	Q2P1	Q3P1	Q0P0	Q1P0"	Q2P0"	Q3P0"	Q0P1'
	rite cycle			ID	TD	PV	PS	ID	TD	PV	PS	D	TD	PV	PS	ID	TD	PV	PS
	:0_phase				∱			ļ/	<u>\</u>		ļ	/	∱			1	٦		
	c1_phase				\downarrow —	∖			\sim	<u>\</u>			Ļ	∖	ļ		ļ/	\	
	2_phase				<u> </u>	ļ/	┝				\uparrow			L/	ケ				
mac	p3_phase		└──	ļ			ļ/	\uparrow			ļ/	∱				╄╲			

Timing Diagram 2: RE Interpolator to Shader Pipe Data Transfer

Exhibit 2012.docR400_Sequencer.docR400_Sequencer.doc 32166 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page ***

AMD1044_0256761

ATI Ex. 2105 IPR2023-00922 Page 89 of 239

	ptembo	E DAT	124	4 Sept				R400 S	equenc	er Spe	cificatio	on		GE of 31	_ (Format	ted: Bul	lets and Numb
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
SEQ_SP_read_addr RF0_read_data	TC srcB	srcC	тс	srcA	TC srcB	srcC	тс	srcA	TC srcB	srcC	тс	srcA	TC srcB	srcC	тс	srcA	TC srcB	
SP_TX_tc				тсо	TC1	TC2	тсз	тсо	TC1	TC2	тсз	тсо	TC1	TC2	тсз	TCO	TC1	TC2
SEQ_TX_instr_start			/	h_{-}				h										
SEQ_TX_instruction			10_0	10_1	10_2	I0_3	11_0	11_1	11_2	I1_3								
SEQ_TX_clause			0				0											
SEQ_TX_write_addr SEQ_TX_last			r4				r5	L										
SEQ_TX_hase		<u> </u>				 ~	/	<u>\</u>		<u> </u>				Ļ				L.
oza_m_phace	4	\			٢	\			٢	<u> </u>			ť	<u> </u>			ť	
tx_phase		/	4				5			/	h			/	\mathbf{H}			
TX_SP_write_addr						-				F	r4			T	r5			
TX_SP_valid												<u>\</u>				<u> </u>		
TX_SP_data											т0_0	T0_1	T0_2	т0_3	T1_0	T1_1	T1_2	T1_3
TX_SEQ_clause															0			
TX_SEQ_done																↑		
SEQ_SP_phase																		
SEQ_SP_write_addr	PS		<u> </u>	PV	PS	/ ID	<u> </u>	PV	PS		<u> </u>	PV	PS		\square	PV	PS	
RF0 write cycle		PS	ID		PV	PS	- ID		PV PV	PS			PV	PS	ID.	TD	PV	PS
					1.0	10				10			1.			10		

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

Ehibit 2012. docR400_Sequencer.docR400_Sequencer.doc 32166 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page ***

AMD1044_0256762

ATI Ex. 2105 IPR2023-00922 Page 90 of 239

ORIGINATE	DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September	<u>, 200124</u>	4 September, 20153	GEN-CXXXXX-REVA	29 of 31	
14.4 Timing diagram	ms expla	anations		4	
So for example the most signif SUO, the most significant 128 I The following assumptions are 1.all block to block signals 2.for register file reads, the	ficant 512 bil bits go to MA made: are register e RF read da	ts of a SP goes to SUO an CO and the least significan to register ata is available in the MAC	one clock after a RF read address	-SU3; within 	Formatted: Bullets and Numbering
into the MAC (this is t asserted on the SEQ t			lid out of the RF two clocks after th	e-address-is	
14.4.1 Timing Diagra	<u>т 1: Sө</u> q	uencer to Shader F	Pipe 0, Shader Unit 0, M/	10-0	
MACO is shown. The timing f means that most of the signal	for MAC1 is	one clock later than MACC	erface. For simplicity only the timir), MAC2 one clock later than MAC cycle for MAC1, two cycles for MAC	1, etc. This	
into the RF. It is asserted duri for other MACs.	at 1 (128 bits r File Read A syncs the dat ing the cycle	over 4 cycles). Pipelined in address (8 bits). Pipelined ta transfer to the RF from t that interpolated data (ID)	n SP for other MACs. in SP for other MACs. he RE, as well as defining the order is valid on the RE_SP_ID bus. Pip	elined in SP	
RE_SP_ID[511:384]: This is the SUO). SEQ_SP_instruction: 96 bits			P_data interface (meaning that this pelined in SP for other MACs.	MACO_is_in	
SEQ_SP_instr_start: control I mac0_phase: registered versi mac0_cycle_count: a counte cycle RE interpolated data is w RF0_read_data: data that is re mac0_vector_result: the 32-b	bit that signa on of SEQ_S r inside the vritten (this m ead out of M/	Is the first cycle of the instr SP_phase used in MACO (tl MAC that keeps track of t hay not be he actual signal ACO's register file (this may	uction transfer. Pipelined in SP for his may not be he actual signal nam the RF write cycles; 0 here corres	ie). ponds to the	
address over this bus. Pipeline	ed in SP for (llocated to th	other MACs. e different write sources (II	t the SEQ does not send the Textur D = Interpolated Data, TD = Texture a point on the diagram).		
14.4.2 Timing Diagra	<u>m 2: RE</u>	Interpolator to Shad	ler Pipe Data Transfer	*-	Formatted: Bullets and Numbering
This diagram shows how pixel	l data (baryo arameter da on shown bei	entric coordinates i, j, and ta (for each vertex) is also ng sent over the RE_SP ini	k) is sent from the Pixel FIFO to the sent to the interpolator under SEQ		
the SEQ that new parameter in PXF_INT_data: Data output of	nfo must be f f the Pixel FII	etched (if its not from a new FO – goes to the Interpolate		not needed).	
next quad of data will be driver	n the next cy tells the Pix	cle.	aken by the Interpolator (driven by f pixels (otherwise RTRs cause th	,	
PMB_INT_data: Data from the TBD).	e Parameter	Buffer to the Interpolator.	(Note that the control of the param	eter buffer is	
SEQ_INT_pm_load: controls t	the loading o	f parameter data into the Ir	terpolator.		
bit 2012.docR400_Sequencer.docR400_Sequencer	doc 32166 Byte	₅*** © ATI Confidential. R	eference Copyright Notice on Co 09/04/15 12:49 PM08/13/01-0		
			<u>00/04/10/12,40 FW106/10/10</u>		

ATI Ex. 2105 IPR2023-00922 Page 91 of 239

Verture Provide the set of the set						
Per generative and the second of the second seco		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
INT_prate_regis register in the interpolate that holds the per-volex parameter data while the per-poled parameters are generated for one or now equations the boarding of poled data much interpolate? INT_pole_registers the two interpolates that holds one quade south of - pole data (may not be the actual signal areas). SEQ_NT_pole_registers the two interpolates that holds one quade south of - pole data (may not be the actual signal areas). SEQ_NT_pole_registers the two interpolates that holds one quade south of - pole data (may not be the actual signal areas). SEQ_NT_pole_registers the two interpolates that holds one quade south of - pole data (may not be the actual signal areas). SEQ_NT_pole_registers the two interpolates that holds one quade south of - pole data (may not be the actual signal areas). Interpolate case above under TDI. The diagram stats with the two interpolate interpolate interpolate one of the south areas interpolate one of the south areas interpolate one of the actual signal areas and the south coordinate read from the register file and its transfer to the table one interpolate one of the south areas in the two interpolate interpolate one of the south areas interpolate interpolate interpolate one of the south areas and the south areas interpolate one of the south areas interpolate one of the south areas and the two interpolates into accurate interpolate and accurate interpolate and the interpolate register data and the interpolate register data with the south areas interpolate one of the south areas and the south are		24 September, 200124	4 September, 20153		30 of 31	
 SEQ_SP_mitte_addresses above under TD1. SEQ_SP_write_addresses above under TD1. SE_SP_entate. Data Yuid—Indicates that the SP aboutd write the ID on the appropriate cycle. RE_SP_entate. The above the RE interprise. RE_SP_entate. Data Yuid—Indicates that the SP aboutd write the ID on the appropriate cycle. RE_SP_entate. Data Yuid—Indicates that the SP aboutd write the ID on the appropriate cycle. RE_SP_entate. The interprise of the set of these eignate (all with the same timing). I.4.1.3. Turning_Diagram_3. Sequencer — Texture Unit Interface and Texture Unit — the instruction transfer is the notwork, followed by the focus out attransfer to the address is associate. RE_SP_entate. Texture does the interprise of the address is associate. RE_SP_entate. Texture does the interprise of the statue on interprise of the statue on interface is the notwork. RE_SP_entate. Texture does the interprise of the statue on interprise. RE_SP_entate. Texture is and the interprise of the statue on interprise. RE_SP_entate. Texture is an interprise of the statue on interprise. RE_SP_entate. Texture is an interprise of the statue on interprise. RE_SP_entate. Texture is an interprise of the statue on interprise. RE_SP_entate. Texture is an interprise of the statue on interprise. RE_SP_entate. Texture is an interprise of the statue on intor. RE_SP_entate. Texture is an interprise of the statue on intor. RE_SP_entate. Texture is an interprise of the statue on intor. RE_SP_entate. Texture is an interprise of the statue on intor. RE_SP_entate. Texture is an interprise of the statue on intor. RE_SP_entate. Texture is an interprise of the statue on intor. RE_SP_entate. Texture is an interprise of the statue on intor. RE_SP_entate. Texture is an interprise of the intore intor. RE_SP_entate. Texture is an interprise of the i	are generated SEQ_INT_px INT_quad_re	reg: register in the Interpol d for one or more quads (m _load: controls the loading	ator that holds the per-ve ay not be the actual signa of pixel data into the Inte	al name). rpolator.		
14.4.3 - Timing Diagram 3: Sequencer - Texture Unit Interface and Texture Unit-* Transfer This degram data with the texture data transfer to the shader pipe. SEQ.SP-read_addr: see above. Here chows the crycle hards the texture coordinate read addrese is asserted. RPD_read_addr: see above. Seq.SP-read_addr: see above. Seq.SP-read-addr: see above. Seq.SP-read	SEQ_SP_ph SEQ_SP_wr RE_SP_valic RE_SP_data RF0 write cy mac*_phase	ite_addr: see above under I: Interpolator Data Valid — : Data from the RE interpol cle: see above under TD1. : see above under TD1. T	TD1. indicates that the SP sho ator to the SP. hese phase signals help	to show the timing offset between the		
This diagram starts with the texture coordinate read from the register file and its transfer to the TX. — The instruction transfer is then shown, field subve, there yold is the ladder pipe. SEQ SP. read_addr: eee above. There is the late the texture coordinate read address is assented. RP - TK. (1: Takture coordinate data sent from the shader pipe to the texture unit. SEQ TX. Instr_start: Assented on the first cycle of a SEQ to TX instruction transfer. SEQ TX. Instr_start: Assented on the first cycle of a SEQ to TX instruction transfer. SEQ TX. Instr_start: Assented on the first cycle of a SEQ to TX instruction. SEQ TX. Units of texture coordinate data sent from the shader pipe to the texture unit. SEQ TX. Instruction: 60 bits of texture instruction for a clause. SEQ TX. Instruction: 60 bits of texture instruction of a clause. SEQ TX. Instruction: 60 bits of texture instruction of a clause. SEQ TX. Instreme index set to clause the output instruction of a clause. SEQ TX. Instreme index for toxture data. TX SEQrunite_addr: RF while index set to toxture data. TX SEQrunite_index for toxture data is being driven to the SP. TX SEQrunite_index to the toxture data is being driven to the SP. TX SEQfustase the clause number associated with the texture data. SEQ _SP_phase: see above under: TD1 = shown here for reference. SEQ _SP_write_addr: RF while index set to the SEQ that it instruction. SEQ _SP_write_addr: RF while index set for reference. SEQ _SP_write_addr: SEQ _SP_write_addr: SEQ _SP_write explose under: TD1 = shown here for reference. SEQ _SP_write_addr: RF while index set to the for texture data. SEQ _SP_write_addr: RF where the for texture data is the for reference. SEQ _SP_write_addr: SEQ _SP_write_ad		iming Diagram 3: S	Sequencer - Textu	ire Unit Interface and Text	ture Unit -**	Formatted: Bullets and Numbering
SED_TX_Instruction-95 bits of texture instruction transfer. SEQ_TX_Instruction-95 bits of texture instruction. SEQ_TX_Instruction-95 bits of texture instruction. SEQ_TX_Instruction-95 bits of texture instruction framework of a clause. SEQ_TX_Instruction-95 bits of texture instruction framework of a clause. SEQ_TX_Instruction-95 bits of texture instruction framework of a clause. SEQ_TX_Instruction-95 bits of texture instruction framework of a clause. SEQ_TX_Instruction-95 bits of texture data write-Note that it is asserted early enough to be registered into TX and still allow TX to eource the texture data to the SP on the correct cycle. IX_SP_write_add1r.RF write index for texture data. TX_SEQ_Clause that valid texture data. TX_SEQ_Clause the texture data is being driven to the SP. TX_SEQ_Clause the texture data. TX_SEQ_Clause the clause number secondated with the texture data. TX_SEQ_Clause the clause number to 1-shown here for reference. SEQ_SP_phase: sees above under TD1-shown here for reference. SEQ_SP_write_add1r. eas above under TD1-shown here for reference. SEQ_SP_write_add1r. eas above under TD1-shown here for refere	This-diagram transfer is the SEQ_SP_rea RF0_read_ad	en shown, followed by the to id_addr: see above. Here ddr: see above.	exture data transfer to the shows the cycle that the	shader pipe. texture coordinate read address is ass		
SEQ_TX_brace: synce the texture data wite — 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: RR-write index for texture data. TX_SP_valid: indicates that valid texture data. TX_SP_data: the texture data. TX_SP_data: the feature data. TX_SP_data: the feature data. TX_SP_data: the feature data. TX_SP_clause that chance number associated with the texture data. TX_SEQ_clause to the SEQ that the texture data transfer is complete for the clause number that is on the T_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. SEQ_SP_write_addr: see above under TD1 - shown here for reference. RF0 write cycle: see above under TD1 - shown here for reference. RF0 write cycle: 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. SEQ_SP_write_addr: see above under TD1 - shown here for reference. SEQ_SP_write_addr: see above under TD1 - shown here for reference. SEQ_SP_write_addr: see above under TD1 - shown here for reference. SEQ_SP_write_addr: see a	SEQ_TX_ins SEQ_TX_ins SEQ_TX_cla SEQ_TX_cla	tr_start: Asserted on the fi truction: 96 bits of texture use: the clause number as ite_addr: RF write index us	rst cycle of a SEQ to TX i instruction transferred ov sociated with this instruct ed by TX for returned tex	nstruction transfer. er 4 cycles. ion. ture data.		
T_SP_write_addr. RF write index for tofutre data. TX_SP_valid: indicates that valid texture data is being driven to the SP. TX_SEQ_clause: the clause number associated with the texture data. TX_SEQ_clause: the clause number associated with the texture data. 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. RF0 write cycle: 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. SEQ_SP_write_addr: see above under TD1- shown here for reference. RF0 write cycle: 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. SEQ_SP_write_addr: 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. SEQ_SP_write_addr: see above under TD1-	SEQ_TX_ph	ase: syncs the texture data	write. Note that it is ase	erted early enough to be registered in	to TX and still	
TV_SEQ_clause: the clause number associated with the texture data: TX_SEQ_clause indicates to the SEQ that the texture data transfer is complete for the clause number that is on the TX_SEQ_sheep bus. SEQ_SP_write_addr: see above under TD1- shown here for reference. SEQ_SP_write cycle: see above under TD1- shown here for reference. RF0 write cycle: see above under TD1- shown here for reference.	TX_SP_write TX_SP_valid	addr: RF write index for t indicates that valid texture	exture data.	ne SP.		
SEQ_SP_write_addr: see above under TD1- shown here for reference. RF0 write cycle: see above under TD1- shown here for reference.	TX_SEQ_cla TX_SEQ_do	use: the clause number as ne: indicates to the SEQ th			that is on the	
	SEQ_SP_wr	ite_addr: see above under	TD1- shown here for refe	ence.		
	Eshibit 2012.docR400Seg	uencer.docR400_Sequencer.doc 32166	_{Bytes} *** © ATI Confidenti			•

ATI Ex. 2105 IPR2023-00922 Page 92 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 200124	4 September, 20153	GEN-CXXXXX-REVA	31 of 31	
•				•	

15.16. 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.Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in
- 2) Waterfall down the pipe allowing only at a given time the vertices having the same constants to ope 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.

Exhibit 2012.docR400_Sequencer

AMD1044_0256765

Formatted: Bullets and Numbering

ATI Ex. 2105 IPR2023-00922 Page 93 of 239

ORIG	INATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE							
	ember, 200124	4 September, 20155	GEN-CXXXXX-REVA	1 of 33							
Author: Laurent L	efebvre										
		0									
Issue To:		Copy No:									
R400 Sequencer Specification											
		8 8									
		SEQ									
		Version 0.75									
		version v. <u>r</u> ə									
Overview: This is an arc	hitectural specific	ation for the R400 Sequen	cer block (SEQ). It provides an ov	verview of the							
	bilities and expe ovides internal st		lso describes the block interfaces,	internal sub-							
AUTOMATICALLY UPD	ATED FIELDS:										
Document Location: Current Intranet Search	C:\pe	erforce\r400\arch\doc\gfx\RE D Sequencer Specification	\R400_Sequencer.doc								
Guitent intranct ocard		APPROVALS									
Na	ame/Dept		Signature/Date								
Remarks:											
			NFORMATION THAT CO								
			REST OF ATI TECHNOL	OGIES							
INC.	HRUUGHU	JNAUTHORIZED US	SE OR DISCLOSURE.								
"Convright 2001 ATL Tool		I rights record. The meta	rial in this document constitutes ar								
work created in 2001. The	e use of this cop	yright notice is intended to	provide notice that ATI owns a co	pyright in this							
unpublished work. The copyright notice is not an admission that publication has occurred. This work conta confidential, proprietary information and trade secrets of ATI. No part of this document may be used, reproduced											
transmitted in any form or by any means without the prior written permission of ATI Technologies Inc."											
Exhibit 2013.docR400_Sequencer.docR400_S	Sequencer.doc 35011 By	ntes*** © ATI Confidential. F	Reference Copyright Notice on Co 09/04/15 12:50 PM10/05/01-10:28-AM08/13/01-0								
			03/04/10 12:00 PM/10/06/07-10:28-AM/08/13/01-0	J U. I / Y WI							

ATI 2013 LG v. ATI IPR2015-00325

AMD1044_0256766

ATI Ex. 2105 IPR2023-00922 Page 94 of 239



Table Of Contents

1. OVERVIEW
1.1 Top Level Block Diagram 76
1.2 Data Flow graph 108
1.3 Control Graph
2. INTERPOLATED DATA BUS 129
3. INSTRUCTION STORE1512
4. SEQUENCER INSTRUCTIONS 1512
5. CONSTANT STORE
6. LOOPING AND BRANCHES 1512
6.1 The controlling state
6.2 The Control Flow Program
6.3 Data dependant predicate instructions 1714
And the second
6.4Register file indexing
8. TEXTURE ARBITRATION
9. ALU ARBITRATION
10. HANDLING STALLS
11. CONTENT OF THE RESERVATION
STATION FIFOS
12. THE OUTPUT FILE
13. THE PARAMETER CACHE
14. VERTEX POSITION EXPORTING 2117
15. REGISTERS
15. REGISTERS
15. REGISTERS 2117 16. INTERFACES 2218
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1 PA/SC to RE : IJ bus 2248
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 2348
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 2348 16.1.4 PA/SC to SEQ : Vertex Control
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 2348 16.1.4 PA/SC to SEQ : Vertex Control Bus 2348
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 2348 16.1.4 PA/SC to SEQ : Vertex Control Bus 2348 16.1.5 CP to SEQ : Constant store
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 16.1.4 PA/SC to SEQ : Vertex Control Bus 2348 16.1.5 CP to SEQ : Constant store load 2349
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 2348 16.1.4 PA/SC to SEQ : Vertex Control Bus 2348 16.1.5 CP to SEQ : Constant store load 2349 16.1.6 CP to SEQ : Texture State store
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 2348 16.1.4 PA/SC to SEQ : Vertex Control Bus 2348 16.1.5 CP to SEQ : Constant store load 2349 16.1.6 CP to SEQ : Texture State store load 2349
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 16.1.4 PA/SC to SEQ : Vertex Control Bus 2348 16.1.5 CP to SEQ : Constant store load 2349 16.1.6 CP to SEQ : Texture State store load 2349 16.1.7 CP to SEQ : Control State store
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 2348 16.1.4 PA/SC to SEQ : Vertex Control Bus 2348 16.1.5 CP to SEQ : Constant store load 2349 16.1.6 CP to SEQ : Texture State store load 2349 16.1.7 CP to SEQ : Control State store load 2349
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 2348 16.1.4 PA/SC to SEQ : Vertex Control Bus 2348 16.1.5 CP to SEQ : Constant store load 2349 16.1.6 CP to SEQ : Texture State store load 2349 16.1.7 CP to SEQ : Control State store load 2349 16.1.8 MH to SEQ: Instruction store
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 2348 16.1.4 PA/SC to SEQ : Vertex Control Bus 2348 16.1.5 CP to SEQ : Constant store load 2349 16.1.7 CP to SEQ : Control State store load 2349 16.1.7 CP to SEQ : Control State store load 2349 16.1.8 MH to SEQ: Instruction store Load 2349
15. REGISTERS 2117 16. INTERFACES 2218 16.1 External Interfaces 2248 16.1.1 PA/SC to RE : IJ bus 2248 16.1.2 PA/SC to SEQ : IJ Control bus 2248 16.1.3 PA/SC to RE : Vertex Bus 2348 16.1.4 PA/SC to SEQ : Vertex Control Bus 2348 16.1.5 CP to SEQ : Constant store load 2349 16.1.6 CP to SEQ : Texture State store load 2349 16.1.7 CP to SEQ : Control State store load 2349 16.1.8 MH to SEQ: Instruction store

16110 CD to DA/CO + Desition action
16.1.10 SP to PA/SC : Position return bus 2419
16.1.11 Shader Engine to Texture
Unit Bus (Fast Bus)2419
16.1.12 Sequencer to Texture Unit bus
(Slow Bus) 2420
17. INTERNAL INTERFACES 2520
18. EXAMPLES OF PROGRAM
EXECUTIONS
18.1.1 Sequencer Control of a Vector
of Vertices 25 20
18.1.2 Sequencer Control of a Vector
of Pixels 2624
<u>18.1.3</u> Notes
19. OPEN ISSUES
1. OVERVIEW
1.1 Top Level Block Diagram
1.2 Data Flow graph7
1.3 Control Graph8
2. INTERPOLATED DATA BUS8
3. INSTRUCTION STORE 11
4. SEQUENCER INSTRUCTIONS 11
5. CONSTANT STORE
6. LOOPING AND BRANCHES
6.1 Data dependant predicate instructions
6.2 Register file indexing
7. REGISTER FILE ALLOCATION 12
8. TEXTURE ARBITRATION
9. ALU ARBITRATION
10. HANDLING STALLS
11 CONTENT OF THE RESERVATION
STATION FIFOS14
12. THE OUTPUT FILE (RB FIFO AND
PARAMETER CACHE)
13. REGISTERS
14. INTERFACES
14.1 External Interfaces
14.1.1 Sequencer to Shader
Engine Bus
14.1.2 Shader Engine to Output
File 15
EH© 13

Ehbit 2013. docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

AMD1044_0256767

ATI Ex. 2105 IPR2023-00922 Page 95 of 239

22	ORIGINATE DATE	EDIT DATE		DOCUMENT-REV. NUM.	PAGE
	24 September, 200124 September, 200114	4 September, 20	0155	GEN-CXXXXX-REVA	3 of 33
14.1.3	Shader Engine to T			.1.1 Sequencer Control of	a Vector
Unit Bus	(Fast Bus)			Vertices 16	
14.1.4	Sequencer to Texture	Unit bus	13	.1.2 Sequencer Control of	a Vector
(Slow But				Pixels 18	
14.1.5_	Shader Engine to RE	PA-Bus 15	13	.1.3 Notes	
	PA? to sequencer			OPEN ISSUES	
	APLES OF PROGRAM		1.	OVERVIEW	3
	NS.			Top Level Block Diagram	
	-Sequencer Control of			Data Flow graph	
of Vertice		4 100101		Control Graph	
	-Sequencer Control of	a Vector		-INSTRUCTION STORE	
of Pixels		-d-vootor		CONSTANT STORE	
	Notes	18		LOOPING AND BRANCHES	
	VISSUES			REGISTER FILE ALLOCATI	
	2VIEW		7	TEXTURE ARBITRATION	
	evel Block Diagram		8	ALU ARBITRATION	
	Flow graph			HANDLING STALLS	
	ol-Graph			CONTENT OF THE RESERV	
2. INTEI	RPOLATED DATA BUS	s10		TION FIFOS	
	RUCTION STORE			THE OUTPUT FILE (RB FIF	
	STANT STORE			AMETER CACHE)	
	PING AND BRANCHES			INTERFACES	
	STER FILE ALLOCATI		12.1	External Interfaces	
	URE ARBITRATION		12	.1.1 Sequencer to Shad	er
	ARBITRATION OLING STALLS		En	gine Bus	
	FENT OF THE RESERV			.1.2 Shader Engine to C	
	FIFOS			e	E.
	OUTPUT FILE (RB FIF			.1.3 Shader Engine to T	exture
PARAMET	ER CACHE)			iit Bus (Fast Bus)	
	ERFACES				
	ternal Interfaces			.1.4 Sequencer to Texture	Unit DUS
	Sequencer to Shad			ow Bus) 14	
	Bus			.1.5 Shader Engine to RE	
0			$\frac{12}{12}$.1.6 PA? to sequencer	
	-Shader Engine to C	Putput		EXAMPLES OF PROGRAM	
File				CUTIONS	
	Shader Engine to T			.1.1 Sequencer Control of	a Vector
Unit Bus	- (Fast Bus)			Vertices 14	
12.1.4	Sequencer to Texture	Unit-bus	13	.1.2 Sequencer Control of	a Vector
(Slow But			of-l	Pixels—16	
12.1.5-	Shader Engine to RE	PA Bus 16	13	.1.3 Notes	
	PA? to sequencer		14.—	OPEN ISSUES	
	APLES OF PROGRAM				
	NS				
			ontial	Reference Convright Notice on Co	wor Dago @ +
113.docH400_Sequ	uencer.docR400_Sequencer.doc 35011 By		ential. I	Reference Copyright Notice on Co 09/04/15 12:50 PM10/05/01-10:28-AM08/13/01	

ATI Ex. 2105 IPR2023-00922 Page 96 of 239

Γ	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE]
	24 September, 200124 September, 200114	4 September, 20155		4 of 33	
Т					

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 0.4-5 (Laurent Lefebvre) Date : September 7, 2001 Rev 0.6 (Laurent Lefebvre) Date : September 24, 2001 Rev 0.7 (Laurent Lefebvre) Date : October 5, 2001 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.

Ehibit 2013. docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C ***

AMD1044_0256769

ATI Ex. 2105 IPR2023-00922 Page 97 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1
24 September, 200124 September, 200114	4 September, 20155	GEN-CXXXXX-REVA	5 of 33	

1. Overview

The sequencer first arbitrates between vectors of $46-\underline{64}$ vertices that arrive directly from primitive assembly and vectors of $4-\underline{16}$ quads ($46-\underline{64}$ 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.

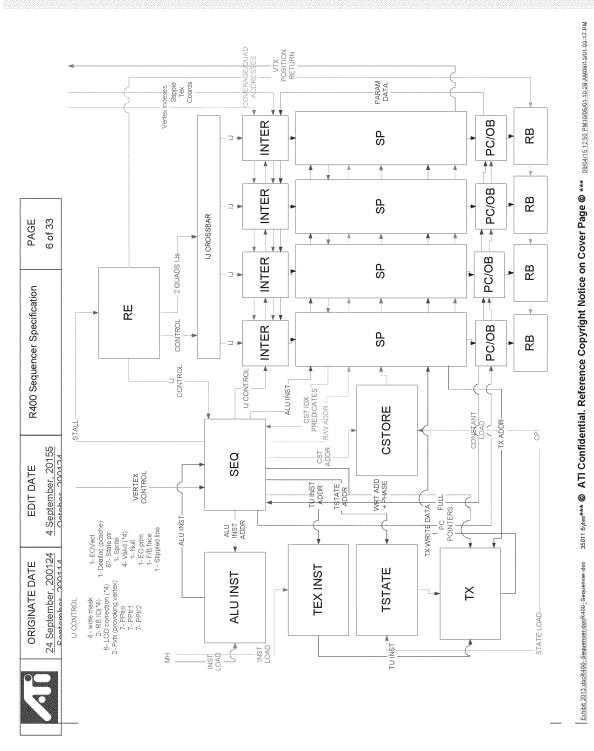
The sequencer is based on the R300 design. It chooses two ALU clauses and a texture clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight texture 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 all reservation station. A FIFO exists between each reservation station can be chosen to execute. The sequencer looks at all eight all reservation stations to choose an all 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 and all 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 instruction thus there is only one sequencer for the whole chip.

Exhibit 2013.docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page ***

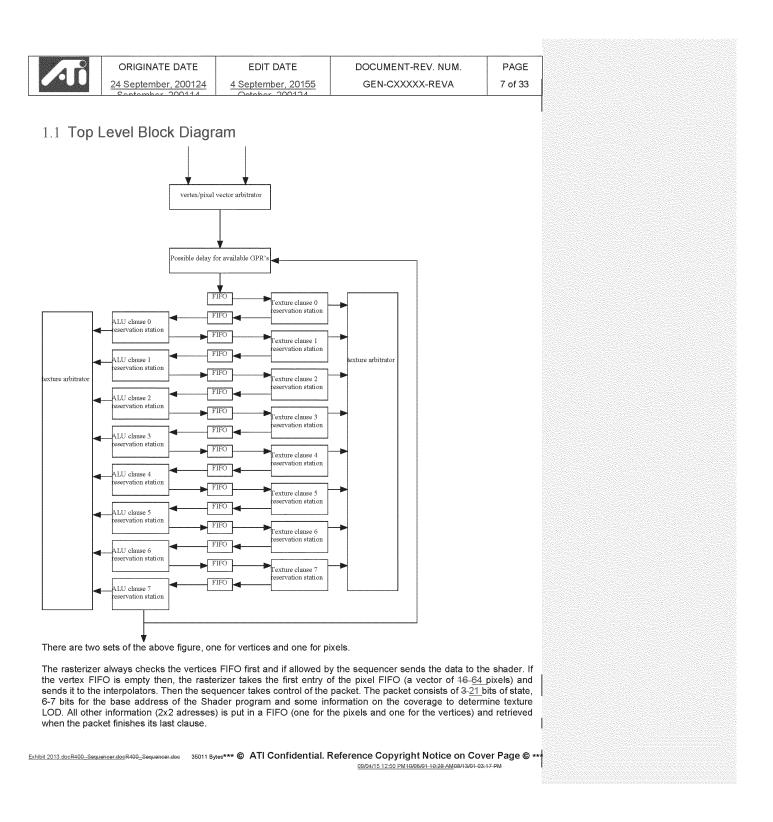
AMD1044_0256770

ATI Ex. 2105 IPR2023-00922 Page 98 of 239 PROTECTIVE ORDER MATERIAL



AMD1044_0256771

ATI Ex. 2105 IPR2023-00922 Page 99 of 239



ATI Ex. 2105 IPR2023-00922 Page 100 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 200124	4 September, 20155		8 of 33

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 (tomd) 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 <u>16-64</u> 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.

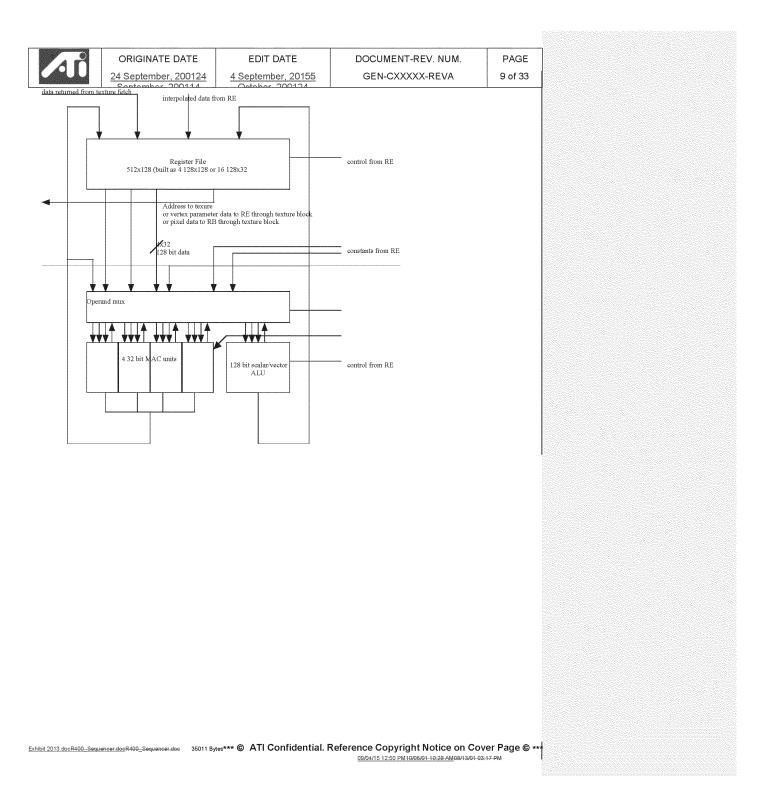
Only two ALU state machine may have access to the register file address bus or the instruction decode bus at one time. Similarly, only one texture state machine may have access to the register file address bus at one time. Arbitration is performed by three arbitrer blocks (two for the ALU state machines 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

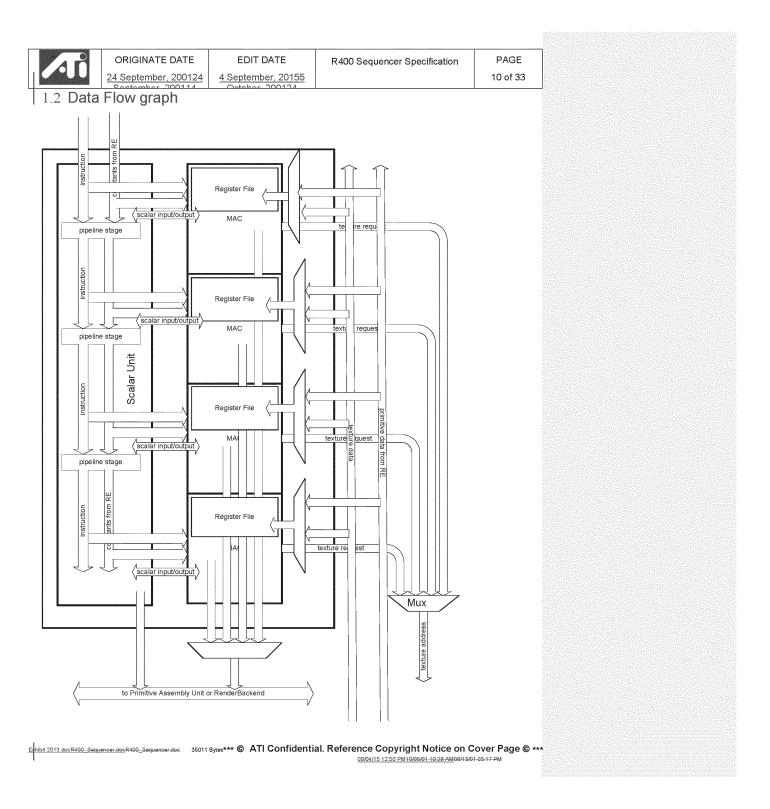
Ehihbit 2013. docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C ***

AMD1044_0256773

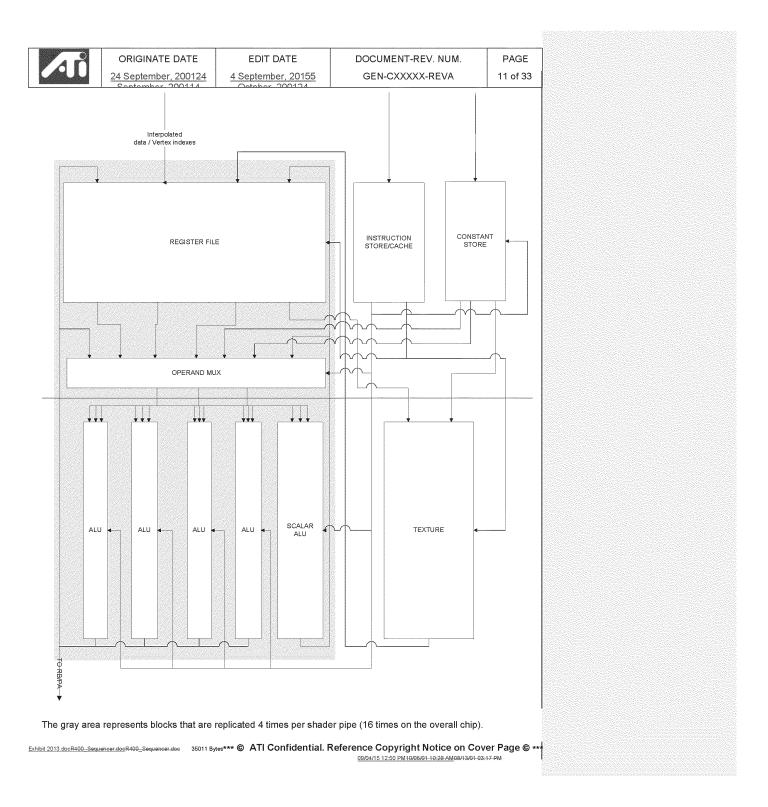
ATI Ex. 2105 IPR2023-00922 Page 101 of 239



ATI Ex. 2105 IPR2023-00922 Page 102 of 239



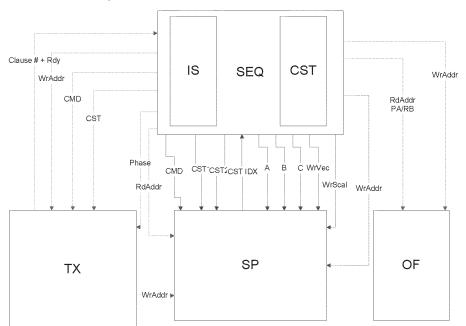
ATI Ex. 2105 IPR2023-00922 Page 103 of 239



ATI Ex. 2105 IPR2023-00922 Page 104 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 200124 September, 200114	4 September, 20155		12 of 33

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.

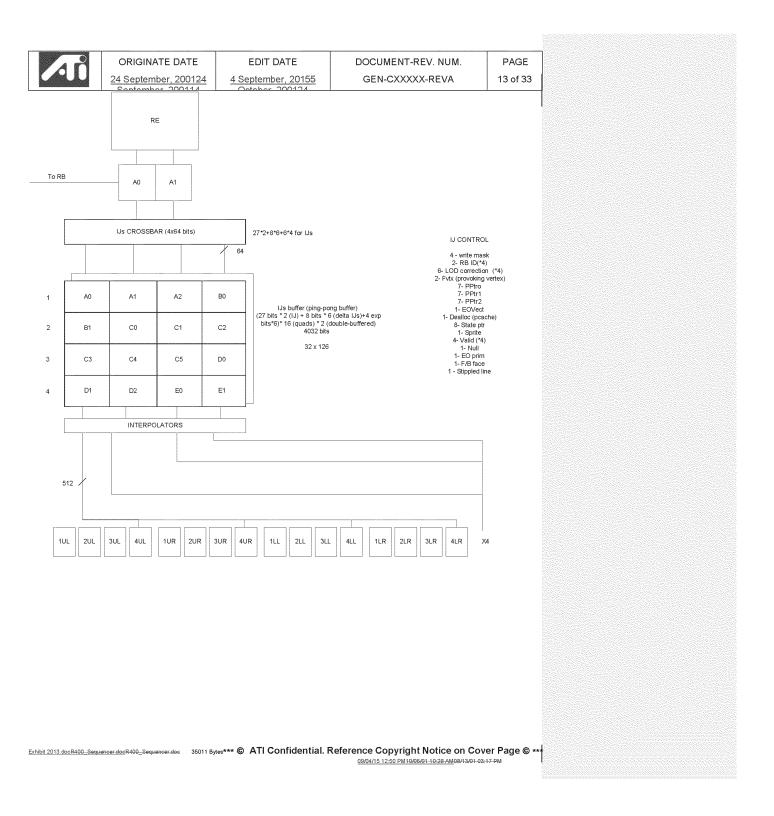
2. Interpolated data bus

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

Ehibit 2013. docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

AMD1044_0256777

ATI Ex. 2105 IPR2023-00922 Page 105 of 239



ATI Ex. 2105 IPR2023-00922 Page 106 of 239 PROTECTIVE ORDER MATERIAL

	ი						
	T19	51 51	52- 55	59 <	80 - C		
	T18	32- 35	39 - 36 <	> \$ 4 5 4	> 44 - 47		
	T17	> 0 0	23 ² 0 <	24- 27	31 ² 8 <		
	T16	> °-3	> ⁴ -7	> ⁶ -11	15 ⁻ 15 <		
	T15			Û	Ш		***
	T14				8		e e e e e e e e e e e e e e e e e e e
PAGE 14 of 33	T13		හි	5	62		over D.
	T12				BO	~	ann an Couer Dare (a ATI Confidential Deference Convirint Notice on Couer Dare () ***
cation	T11	5	6			۵.	+ Notic
Specifi	T10	ទ	5	C5			huridh
uencer	Т9	<u>8</u>					c o
R400 Sequencer Specification	Т8	A0	A1	A2			Jofara
R4	4			С	ω		ntion
<u>)155</u>	T6				8		Confide
EDIT DATE ptember, 20	T5		පි	5	ß		
EDIT DATE 4 September, 20155	T4				BO	0	9
	13	5	D2			Ъ Д	
DATE 20012	12	ů.	64 24	C5			
ORIGINATE DATE 4 September, 20012 Contambar, 200144	Ĕ	B 1					0070
ORIGINATE DATE 24 September, 200124 Contember, 200114	T0	AO	A1	A2			
		SPO	SP1	SP2	SP3		

AMD1044_0256779

ATI Ex. 2105 IPR2023-00922 Page 107 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1		
	24 September, 200124	4 September, 20155	GEN-CXXXXX-REVA	15 of 33			
sequencer allo same primitive register file is size vector of	ows at any given time as ma e. Then the sequencer contr actually physically divided	eceive. The IJ information is ny as four quads to interpol ols the write mask to the re (one 128x128 per MAC) a puffer, we need to interpola	s packed in the IJ buffer 2 quads at late a parameter. They all have to co gister to write the valid data in.Sine nd we don't have the place to hold ite on a parameter basis rather tha	ome from the each of the a maximum			
Q0P0 Q1P0 G	22P0 Q3P0 Q0P1 Q1P1 Q2	P1 Q3P1 Q0P2 Q1P2					
3. <u>Instru</u>	iction Store						
	g to be only one instruction s are is also going to be a conf		ay_will_contain_up_to_200<u>4096</u>0 instr 256(512?)x32.	uctions of 96			
{ISSUE : The	instruction store is loaded b	y the sequencer using the m	nemory hub ?}.				
likely to be br also split in tw	oken up into 4 blocks. An	LU instruction section (1R e memories is 96 bits/clock.	<u>ock)</u> /pipe. To achieve this this instru /1W) split in two and a texture sec It is likely to be a 1R/1W port memo struction.	tion (1R/1W)			and Banda day
4. Seque	encer Instructions			46-	11-	ormatted: Bullets	and Numbering
	w instructions and move ins (MOV PV, PV, PS, PS).	tructions are handled by the	e sequencer only. The ALUs will pe	erform NOPs		ormatted: Bullets	and Numboring
4 <u>-5.</u> Con	stant Store			*	1 m	matteu. Dunets	
table also mai	naged by the CP. A likely si	ze for the constant store is s	e of where the constants are using 512x128 bits. The constant store is bits/clock/pipe and the write band	also planned			
GPRs). There data must par	are 144 wires from the exits a start the start and the shader pipe for	of the SP to the sequence the float to fixed convertion	ided first with the indexes (that co r (9 bits pointers x 16 vertexes/cloc nn, there is a latency of 4 clocks (x into the constant store. The asser	k). Since the 1 instruction)			
MOVA_R1.X, NOP ADD_R3,R4		oat to fixed conversion	2.X, also copies the content of R2.X R4 to C0[R2.X] into R3	into R1.X			
	don't really care about wha ritten again for the sake of s		se we use the state from the MOV	<u>A</u> instruction.			
The storage n	eeded in the sequencer in c	rder to support this feature i	is 2*64*9 bits = 1152 bits.			ormatted: Bullets	
5. 6. Loop	oing and Branche	<u>S</u>		*		matteu: Duilets	
			e dealt with at the sequencer level. e control program has 4(5) instructio				
6.1 The	controlling state.			4	F	ormatted: Bullets	and Numbering
As per Dx9 th	e following state is available	for control flow:					
Boolean[15:0]	L						
Exhibit 2013.docR400Sequ	iencer.docR400_Sequencer.doc 35011 By	es*** © ATI Confidential. F	Reference Copyright Notice on Co 09/04/15 12:50 PM10/06/01-10:28 AM(08/13/01-0		*		

ATI Ex. 2105 IPR2023-00922 Page 108 of 239

ORIGINATE DA	TE EDIT DATE	R400 Sequencer Specification	PAGE]
24 September, 20	0124 4 September, 201	55	16 of 33	
L Contempor 200 loop_count[7:0][7:0] In addition: 1000 <td< td=""><th>to the controlling program.</th><td></td><td></td><td>J</td></td<>	to the controlling program.			J
		<u>.</u>		
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]	<u>×</u>			
{ISSUE: How is the controlling sta	te loaded and how many c	contexts do we have?}		
6.2 The Control Flow	Program		4	Formatted: Bullets and Numbering
		ader is executed, and at every instructi The "event" in the control table can redi		
The Method chosen for the R400	is a "control program". The	control program has four basic instruction	ons:	
Execute Conditional_execute (Conditional Loop_start Loop_end	Execute Predicates)			
store to be executed. Loop_start resets the correspond Loop_end increments (decrement end condition is not met.	ing loop counter to the star ts?) the loop counter and ju	auses the specified number of instruction t value. umps back the specified number of instruc- the following instructions are possible cl	uctions if the loop	
	<u>Exec</u> 23 22 21 20 19 18 17 struction_count	ute 16 15 14 13 12 11 10 9 8 7 6 <u>Reserved</u> <u>Addres</u>		
Execute up to 4K instructions at t	he specified address in the	instruction memory.		
31 30 29 28 27 26 25 24 0 0 0 1 Boolean		I Execute 16 15 14 13 12 11 10 9 8 7 6 uction_count Addres		
if the specified boolean (6 bits c instructions (up to 64 instructions		meets the specified condition then exec	cute the specified	
31 30 29 28 27 26 25 24 0 1 0 1 Reserved =	Contract Contraction Contrac	tute Predicates		
Check the OR of all current predi	cate bits. If OR matches the	e condition execute the specified number	r of instructions.	
ibit 2013.docR400_Sequencer.docR400_Sequencer.doc	35011 Bytes*** © ATI Confi	dential. Reference Copyright Notice o		•

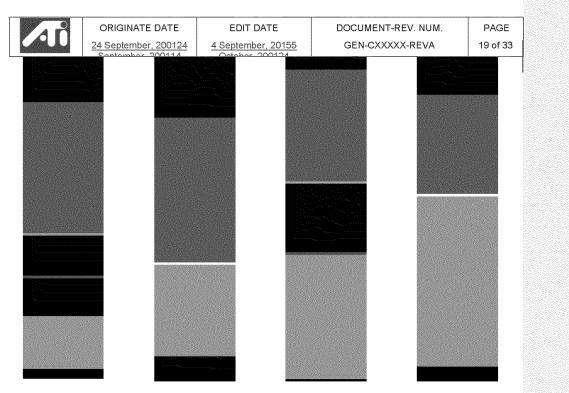
ATI Ex. 2105 IPR2023-00922 Page 109 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 200124	4 September, 20155	GEN-CXXXXX-REVA	17 of 33	
24 22 22 22		Loop Start			
<u>31 30 29 28</u> 0 0 1 0		1 20 19 18 17 16 15	<u>14 13 12 11 10 9 8 7 6 5 4</u>	<u>13 2 1 0</u> Loop ID	
Initialize the sp	ecified loop				
01 00 00 00					
<u>31 30 29 28</u> <u>0</u> <u>0</u> <u>1</u> <u>1</u>	27 26 25 24 23 22 2 Reserved (n		14 13 12 11 10 9 8 7 6 5 4 Start Address Reserved (must = 0) Reserved Reserv	3210 Loop ID	
			ack to the specified_address in the e loop, the jump should be to loop_sta		
The way this is	described does not preven	t nested loops, and the incl	usion of the loop id make this easy to	do.	
The basic mod	el is as follows:				
The render sta	te defined the clause bound	lariae.			
Vertex shader	fetch[7:0][7:0] // eight 8	bit pointers to the location v	vhere each clauses control program i		
Vertex shader Pixel shader f	etch[7:0][7:0] // eight 8	oit pointers to the location v	vhere each clauses control program is vhere each clauses control program is	s located	
Pixel shader a	alu[7:0][7:0] // eight 8	bit pointers to the location v	vhere each clauses control program is	s located	
			an offset added to the address from multiple programs resident at the sam		
			on before moving to another clause rogram is the only program aware of		
***********************************	from the control program	are added to another offe	et to allow for multiple programs res	ident at the	
same time.	norm the control program	are added to another ons	et to allow for multiple programs res		
Under this mod	lel, all subroutine calls mus	be inlined into the control	program.		
6.3 Data	dependant predic	ate instructions		*-	Formatted: Bullets and Numbering
Data dependar		orted in the R400. The onl	y way we plan to support those is by	supporting	
	PRED SETE - similar to	SETE excent that the resul	t is 'exported' to the sequencer.		
	PRED_SETGT - similar to	SETGT except that the re-	sult is 'exported' to the sequencer result is 'exported' to the sequencer		
For the scalar	operations only we will also	support the two following in	nstructions:		
	PRED_SETE0-SETE0 PRED_SETE1-SETE1				
The export is a		nt using the same data pat	h as the MOVA instruction. The seq	uencer will	
maintain the 64					
	two conditional execute bits 1 or 0. For exemple, the ins		al execute "on" bit and the second bit	<u>tells us if</u>	
PO_AC	DD R0,R1,R2				
Exhibit 2013.docR400_Seque	ncer.docR400_Sequencer.doc 35011 Byt	es*** © ATI Confidential.	Reference Copyright Notice on Cov		

ATI Ex. 2105 IPR2023-00922 Page 110 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
<u>24 September, 200124</u>	4 September, 20155		18 of 33	
Is only going to write the result of the AD only write the result of the GPRs whose sequencer with a PRED instruction is un	predicate bit is set. The			
{Issue: do we have to have a NOP betwe	een PRED and the first in	struction that uses a predicate?}		
6.4 Register file indexing			4	Formatted: Bullets and Numbering
Because we can have loops in texture c data created in a texture clause loop an register indexing and the instruction will	d use it into an ALU clau			
Bit7 Bit 6 0 0 0 1 1 0 1 1	'absolute register 'relative register' 'previous vector' 'previous scalar'	Ľ		
In the case of an absolute register we ju base address and we add to it the loc pipe.However, it is still unclear if we plan	p counter and this becc	omes our new address that we give		
6.7. Register file allocatio	n		*	Formatted: Bullets and Numbering
managed using two round robins (one pixels and vertices is allowed to move, VERTEX_REG_SIZE for pixels.				
hibit 2013.docR400_Sequencer.docR400_Sequencer.doc 35011	_{3ytes} *** © ATI Confidentia	al. Reference Copyright Notice on C		
		<u>040402.15:20 bW100090-10:58-VW</u> 084390	: ∪∂:17 ¤₩	

ATI Ex. 2105 IPR2023-00922 Page 111 of 239



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 movinge again.

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 first 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.

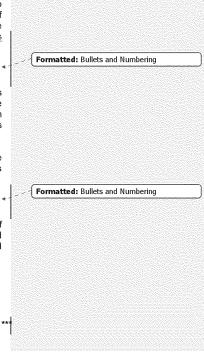
The arbitrator will not wait for the texture fetches to return prior to selecting another clause for execution. The texture pipe will be able to handle up to X(?) in flight texture fetches and thus there can be a fair number of active clauses waiting for their texture return data.

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.

Exhibit 2013 docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page ***



AMD1044_0256784

ATI Ex. 2105 IPR2023-00922 Page 112 of 239

Γ	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 200124 September, 200114	4 September, 20155		20 of 33	Formatted: Bullets and Numbering
	111 Oct 11			*	

9.10. 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.

10.11. Content of the reservation station FIFOs

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

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 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).

$\Delta 01I = I(1) - I(0)$			_			
$\Delta 01J = J(1) - J(0)$						
$\Delta 02I = I(2) - I(0)$	PO	<u>P1</u>				
$\Delta 02J = J(2) - J(0)$						
$\Delta 03I = I(3) - I(0)$						
$\Delta 03J = J(3) - J(0)$	<u>P2</u>	<u>P3</u>				
$P0 = C + I(0)^* (A - C) + J(0)^* (B - C)$						
$P1 = P0 + \Delta 01I^{*}(A - C) + \Delta 01J^{*}(B - C)$						
$P2 = P0 + \Delta 02I^{*}(A - C) + \Delta 02J^{*}(B - C)$						
$P3 = P0 + \Delta 03I^{*}(A - C) + \Delta 03J^{*}(B - C)$						
PO 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.						

Ehibit 2013. docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering

AMD1044_0256785

ATI Ex. 2105 IPR2023-00922 Page 113 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 200124 September, 200114	4 September, 20155	GEN-CXXXXX-REVA	21 of 33	
Multiplies (Re Subtracts 24x	<u>II Precision): 2</u> duced precision): 6 :24 yielding 8 bits (IJs): 6 :24 (Parameters): 2				
FORMAT OF	P0's IJ : Mantissa 23 Exp Mantissa 23 Exp				
FORMAT of E	Deltas (x3):Mantissa 8 Exp 4 Mantissa 8 Exp 4				
Total number	of bits : 23*2 + 8*6 + 4*8 = 1	26 (rounded up on the bus	to 128)		<u></u>
14 The	parameter cache			*1	Formatted: Bullets and Numbering
The parameter	er cache is where the verte ngine will make it so that a		a. It consists of 16 128x128 memorie nitive will hit different memories. (Rf		
					Formatted: Bullets and Numbering
pixel/ver that is st has 1 wi managin pipe_sho keeping vertices) allocation and the priority b each of t On clause 4 (is co at clause 3 using the text fifo drains 12 repeated 4 tir clause 5) bec exporting pro	tex shader finishe atically divided be rite port and 1 re- ing the addresses of build this output f the tail and hea and incrementi n policy. The sequ RB for the use of based or just inter the blocks). Vertex for 5) the vertex shader can b for the vertex shader can b for the vertex shader can b for the sequence must m re port in a clock to the PA a mes. The sequence must m ause the registers can be re	es. It constists of ptween pixels and ad port. The seq of this output file a ile fill up. The r id pointers of ea ng them using thencer must also the read port. This leaved evenly (1 position exportin export to the PA both the v export is done by putting manner, 16 positions are pu nd once empty is filled up ake sure that the program used at this point. The seg porting data. Along with the	arbitrate between the F is arbitration will either I read every 2 clocks	ell ile for by nd bin 2A be for can also do pPRs. Then right). This process is neter texture ealocate an	
16 Deci	atava			*	Formatted: Bullets and Numbering
PIXEL_M VERTEX Vshader Vshader	C REG Dynamic REG SIZE What por IN SIZE Minimal s MIN SIZE Minimal s fetch[7:0][7:0] eight 8 bi alu[7:0][7:0] eight 8 bi	ize of the register file's pixe ize of the register file's vert t pointers to the location wh t pointers to the location wh	served for vertices (static allocation or portion (dynamic only)	ocated ocated	

ATI Ex. 2105 IPR2023-00922 Page 114 of 239

	ORIGINATE DATE	EDIT D	ATE	R400 Sequencer Specification	PAGE	
	September, 200124	4 Septembe			22 of 33	
Pshader_alu[7	Contombor 200114 7:0][7:0] eight 8	bit pointers to t		where each clauses control program	n is located	J
PSHADER VSHADER		pinter for the pi				
PCNTLSHADI		pinter for the pi				
VCNTLSHAD		inter for the ve				
WRAP PWRAP				instruction store struction store		
REG_ALLOC				or pixel shader programs		
REG ALLOC				or vertex shader programs		
PARAM MAS		ter mask to sp rameters are to		parameters the pixel shader ud shaded		
GEN_TEX[0				to generate tex coords.		
CYL WRAPIC		vertices do we				
P_EXPORT V_EXPORT		of exports for		er der (also the number of interpolated	narameters for	
<u>v</u> 0111	pixel sh		, onen ona	as ano the number of interpolated	periorities to to	
V EXPORT L	LOC Vertex	shader exportir	ng to RB or	the PCACHE		
0.17	f				4	Formatted: Bullets and Numbering
2.17. Inter	Taces					
01171 5	to an all to to all					
<u>-2.+1/.1</u> EX	ternal Interfac	es				
	Socianos to	Chador Eng	nino Duo	DN/SC to DE . I hus		
211711	00000000000000000000000	mauer Eng	m o Duo	TVOU TO THE . IS DUO		
	nat sends the instruction			Sub-Engines of the Shader. Because a by 4 and both constants and instruction		
This is a bus the second of th	hat sends the instruction every 4 clocks, the w -This is a bus that ser information goes to th	idth of the bus ids the IJ inforr e sequencer	is divided mation to th	by 4 and both constants and instruction to IJ fifos on the top of each shader pi	in are sent over	
This is a bus the second only those 4 clocks time the contro	hat sends the instruction y every 4 clocks, the w - This is a bus that ser of information goes to th Directi	idth of the bus ids the IJ inforr e sequencer on	is divided mation to th Bits De	by 4 and both constants and instruction the LJ fifos on the top of each shader pi scription	en are sent over pe. At the same	
This is a bus th is needed only those 4 clocks time the contro lame	hat sends the instruction y every 4 clocks, the w - This is a bus that ser of information goes to th Directi	idth of the bus ids the IJ inforr e sequencer	is divided mation to th Bits De 164 Hig	by 4 and both constants and instruction to IJ fifos on the top of each shader pi	en are sent over pe. At the same	
This is a bus th is needed only those 4 clocks time the contro lame nstruction StartIJs	hat sends the instruction y every 4 clocks, the w - This is a bus that ser of information goes to th Directi	idth of the bus ids the IJ inforr e sequencer on	is divided mation to th Bits De 164 Hig	by 4 and both constants and instruction the IJ fifos on the top of each shader pi scription the on first cycle of transferl <u>J informat</u>	en are sent over pe. At the same	
This is a bus th is needed only those 4 clocks time the contro lame nstruction StartIJs	hat sends the instruction y every 4 clocks, the w - This is a bus that ser of information goes to th Directi	idth of the bus ids the IJ inforr e sequencer on	is divided mation to th Bits De 164 Hig	by 4 and both constants and instruction the IJ fifos on the top of each shader pi scription the on first cycle of transferl <u>J informat</u>	en are sent over pe. At the same	(Formatted: Bullets and Numbering
This is a bus the second only those 4 clocks time the controno time the controno transmission start second st	at sends the instruction vevery 4 clocks, the w This is a bus that ser of information goes to th Direction SEQ->	idth of the bus ids the IJ inforr e sequencer on SP <u>PA→RE</u>	is divided mation to th Bits De 164 Hig	by 4 and both constants and instruction the IJ fifos on the top of each shader pi scription the on first cycle of transferl <u>J informat</u>	en are sent over pe. At the same	
This is a bus th is needed only those 4 clocks time the contro Name nstruction StartIJs 2.1.2	r Engine to Outpe	idth of the bus ids the IJ inforr <u>e sequencer</u> on SP <u>PA→RE</u> Ut File	is divided mation to th Bits De 164 Hig clo	by 4 and both constants and instruction le IJ fifos on the top of each shader pi scription h on first cycle of transfer <u>IJ informat</u> cks	n are sent over pe. At the same ion sent over 2	
This is a bus the isneeded only those 4 clocks time the contronant of the contronant	r Engine to Outpe	idth of the bus ids the IJ inforr e sequencer on SPPA→RE ut File ut File	is divided mation to th Bits De 164 Hig clor	by 4 and both constants and instruction the IJ fifos on the top of each shader pi scription the on first cycle of transfer[<u>J informat</u> cks	n are sent over pe. At the same ion sent over 2	Formatted: Bullets and Numbering
This is a bus the is-needed only those 4 clocks time the contro lame -2.1.2- 12.1.2- Every clock-ea	r Engine to Output	idth of the bus ids the IJ inforr e sequencer on SPPA→RE ut File ut File	is divided mation to th Bits De 164 Hig clor	by 4 and both constants and instruction the IJ fifos on the top of each shader pi scription the on first cycle of transfer[<u>J informat</u> cks	n are sent over pe. At the same ion sent over 2	(Formatted: Bullets and Numbering
This is a bus the is-needed only those 4 clocks time the contro lame -2.1.2- 12.1.2- Every clock-ea	r Engine to Output	idth of the bus ids the IJ inforr e sequencer on SPPA→RE ut File ut File	is divided mation to th Bits De 164 Hig clor	by 4 and both constants and instruction the IJ fifos on the top of each shader pi scription the on first cycle of transfer[<u>J informat</u> cks	n are sent over pe. At the same ion sent over 2	(Formatted: Bullets and Numbering
This is a bus the is needed only those 4 clocks time the contro Name nstruction StartIJs 12.1.2 12.1.2 Eveny clock ea	r Engine to Output	idth of the bus ids the IJ inforr e sequencer on SPPA→RE ut File ut File	is divided mation to th Bits De 164 Hig clor	by 4 and both constants and instruction the IJ fifos on the top of each shader pi scription the on first cycle of transfer[<u>J informat</u> cks	n are sent over pe. At the same ion sent over 2	(Formatted: Bullets and Numbering
This is a bus th is needed only those 4 clocks time the contro lame astruction StartIJs 2.1.2 2.1.2 2.1.2 Every clock ea data will be con	r Engine to Output	idth of the bus ids the IJ inforr e sequencer on SP <u>PA→RE</u> ut File tput 128 bits of total prior to sto	is divided mation to th Bits De 164 Hig clor	by 4 and both constants and instruction the IJ fifos on the top of each shader pi scription the on first cycle of transfer[<u>J informat</u> cks	n are sent over pe. At the same ion sent over 2	Formatted: Bullets and Numbering
This is a bus th is needed only those 4 clocks time the contro Name nstruction Start <u>IJs</u> 12.1.2 12.1.2 Every clock ea data will be con	r Engine to Output	idth of the bus ids the IJ inforr e sequencer on SP <u>PA→RE</u> ut File tput 128 bits of total prior to sto	is divided mation to th Bits De 164 Hig clor	by 4 and both constants and instruction the IJ fifos on the top of each shader pi scription the on first cycle of transfer[<u>J informat</u> cks	n are sent over pe. At the same ion sent over 2	
This is a bus this needed only those 4 clocks time the control are astruction StartIJs	A sends the instruction y every 4 clocks, the w This is a bus that ser information goes to th Directi SEQ-> r Engine to Output ach Sub-Engine can our mpressed into 128 bits C to SEQ : IJ Co information sent to th	idth of the bus ids the IJ inforr e sequencer on SPPA→RE Ut File tput 128 bits of total prior to sto <u>ntrol bus</u> e sequencer in	is divided mation to th Bits De 164 Hig clo	by 4 and both constants and instruction the IJ fifos on the top of each shader pi scription the on first cycle of transfer[<u>J informat</u> cks	ion sent over 2	
This is a bus the is needed only those 4 clocks time the control are instruction Start[Js] 2.1.2 - 2.1.2 Shader Every clock ea data will be control for the control of the	A sends the instruction y every 4 clocks, the w This is a bus that ser information goes to th Directi SEQ-> r Engine to Output ach Sub-Engine can our mpressed into 128 bits C to SEQ : IJ Co	idth of the bus ids the IJ inforr e sequencer on SPPA→RE Ut File tput 128 bits of total prior to sto <u>ntrol bus</u> e sequencer in	is divided mation to th Bits De 164 Hig clo	by 4 and both constants and instruction in [J] fifos on the top of each shader pind scription wh on first cycle of transfer[<u>J</u> informatics the and 32 bits of 'scalar' data to an our put file.	ion sent over 2	
This is a bus the is needed only those 4 clocks time the control lame 1.1.2.1.2.1.2.1.2.1.2.1.2.1.2.1.2.1.2.1	A sends the instruction y every 4 clocks, the w This is a bus that ser information goes to th Directi SEQ-> r Engine to Output ach Sub-Engine can our mpressed into 128 bits C to SEQ : IJ Co information sent to th	idth of the bus ids the IJ inforr e sequencer on SPPA→RE ut File tput 128 bits of total prior to sto <u>ntrol bus</u> e sequencer in <u>ixels.</u>	is divided mation to the Bits De 164 Hig clor "vector' da trage in out;	by 4 and both constants and instruction in [J] fifos on the top of each shader pind scription wh on first cycle of transfer[<u>J</u> informatics the and 32 bits of 'scalar' data to an our put file.	ion sent over 2	
This is a bust is needed only those 4-clocks time the control arme 2.1.2- 2.1.2 PA/St Every clock ea data will be cor 7.1.2 PA/St his is the control xecute a shader arme /rite Mask	A sends the instruction A elocks, the w This is a bus that ser Information goes to th Directi SEQ-> C to SEQ : IJ Co Information sent to th program on the sent p Directi Directi	idth of the bus ids the IJ inforr e sequencer on SPPA→RE ut File tput 128 bits of total prior to sto <u>ntrol bus</u> e sequencer in ixels. on EQ(RE)	Bits De Bits De 164 Hig clor 'vector' da rage in outp order to c Bits De 4 Qu	by 4 and both constants and instruction is I fifos on the top of each shader ping scription wh on first cycle of transfer[<u>J informatics</u> ta and 32 bits of 'scalar' data to an our put file. ontrol the IJ fifos and all other inform scription ad Write mask left to right	ion sent over 2	
This is a bust the is needed only those 4 clocks time the control ame estruction StartIJs 2.1.2-2.1.2Shader Every clock ead data will be control to the cont	r Engine to Output sch Sub Engine can output control to SEQ : IJ Co information sent to th program on the sent p Directi C to SEQ : IJ Co information sent to th program on the sent p Directi PA→SI PA→SI	idth of the bus ids the IJ inforr e sequencer on SPPA→RE ut File tput 128 bits of total prior to sto ntrol bus e sequencer in ixels. on Q(RE) EQ(RE)	Bits De Bits De 164 Hig clor 'vector' datrage in output order to c Bits De 4 Qu 8 RB	by 4 and both constants and instruction is cription wh on first cycle of transfer[<u>J informatics</u> ta and 32 bits of 'scalar' data to an our put file. ontrol the IJ fifos and all other inform <u>scription</u> ad Write mask left to right i d for each guad sent 2 bits per guad	ion sent over 2 ion sent over 2 pe. At the same ion sent over 2 put file (?). This ation needed to	
This is a bust the is needed only those 4 clocks time the control are extended to the control of	A sends the instruction A clocks, the w This is a bus that ser Information goes to th Directi SEQ-> C to SEQ : IJ Co Information sent to th program on the sent p Directi Directi Directi Directi Direction	idth of the bus ids the IJ inforr e sequencer on SPPA→RE ut File tput 128 bits of total prior to sto ntrol bus e sequencer in ixels. on EQ(RE) EQ(RE) EQ(RE)	Bits De 164 Hig 'voctor' da 'vage in outp 'order to c Bits De 1 Qu 1 Qu 2 Qu 2 Qu 2 Qu 2 Qu	by 4 and both constants and instruction is cription scription ph on first cycle of transfer[J informatics ta and 32 bits of 'scalar' data to an our put file. ontrol the IJ fifos and all other inform scription ad Write mask left to right id for each guad sent 2 bits per guad D correction per guad (6 bits per guad	ion sent over 2 ion sent over 2 pe. At the same ion sent over 2 put file (?). This ation needed to	
This is a bus the is needed only those 4 clocks time the control lame astruction Startly 2.1.2- 2.1.2- 2.1.2 PA/Start Start	r Engine to Output sch Sub Engine can output control to SEQ : IJ Co information sent to th program on the sent p Directi C to SEQ : IJ Co information sent to th program on the sent p Directi PA→SI PA→SI	idth of the bus ids the IJ inforr e sequencer on SPPA \rightarrow RE ut File tput 128 bits of total prior to sto ntrol bus e sequencer in ixels. on EQ(RE) EQ(RE) EQ(RE) EQ(RE)	Bits De Bits De 164 Hig clor 'vector' da 'rage in out; order to c Bits De 4 Qu 8 RB 24 LO 2 Pro	by 4 and both constants and instruction is cription wh on first cycle of transfer[<u>J informatics</u> ta and 32 bits of 'scalar' data to an our put file. ontrol the IJ fifos and all other inform <u>scription</u> ad Write mask left to right i d for each guad sent 2 bits per guad	ion sent over 2 ion sent over 2 pe. At the same ion sent over 2 put file (?). This ation needed to	
This is a bus It is needed only those 4 clocks time the contro Name Instruction StartIJs I 2.1.2 I 2.1	A sonds the instruction / every 4 clocks, the w → This is a bus that ser of information goes to th Direction SEQ-> <i>C to SEQ : IJ Co</i> information sent to th program on the sent p Direction PA→SI PA→SI PA→SI PA→SI	idth of the bus ids the IJ inforr e sequencer on SPPA \rightarrow RE ut File total prior to sto introl bus e sequencer in ixels. on Q(RE) Q(RE) Q(RE) Q(RE) Q(RE)	Bits De Bits De 164 Hig clor 'vector' darage in out; 'order to c Bits De 4 Qu 8 RB 24 LO 2 Prc 11 P S	by 4 and both constants and instruction is I fifos on the top of each shader pind scription phon first cycle of transfer[J informatics ta and 32 bits of 'scalar' data to an our put file. ontrol the IJ fifos and all other inform scription ad Write mask left to right id for each guad sent 2 bits per guad D correction per guad (6 bits per guad poking vertex for flat shading	ion sent over 2 ion sent over 2 pe. At the same ion sent over 2 put file (?). This ation needed to	
This is a bus this needed only those 4 clocks time the control vame nstruction StartIJs	At sends the instruction y every 4 clocks, the w > This is a bus that ser Direction SEQ-> r Engine to Output ach Sub-Engine can output mpressed into 128 bits C to SEQ : IJ Collinformation sent to th program on the sent p Direction PA→SI PA→SI PA→SI PA→SI PA→SI PA→SI PA→SI PA→SI	idth of the busides the IJ informestion in the sequencer on the sequence of t	Bits De Bits De 164 Hig clor 'vector' da rage in outp order to c Bits De 4 Qu 8 RB 24 LOC 11 P S 11 P S	by 4 and both constants and instruction is I fifos on the top of each shader ping scription phon first cycle of transfer[J informatics ta and 32 bits of 'scalar' data to an our put file. ontrol the IJ fifos and all other inform scription ad Write mask left to right i d for each quad sent 2 bits per quad D correction per quad (6 bits per quad store pointer for vertex 0	ion sent over 2 ion sent over 2 pe. At the same ion sent over 2 put file (?). This ation needed to	

ATI Ex. 2105 IPR2023-00922 Page 115 of 239

	ORIGINATE DATE ED		EDI	T DATE		DOCUMENT-REV. NUM.	PAGE	
	24 September,		4 Septer	nber, 2		GEN-CXXXXX-REVA	23 of 33	
DEALLOC		PA→SEC	2(RE)	1	Deallo	cation token for the P Store		
STATE		PA→SEC	Q(RE)	21	State/c	constant pointer (6*3+3)		
SPRITE		PA→SE0	Q(RE)	1	Need to	o generate tex cords		
VALID		PA→SEG	Q(RE)	16	Valid b	bits for all pixels		
NULL		PA→SE0	Q(RE)	1	Null Pr	rimitive (for PC deallocation purposes)		
E OFF PRIM		PA→SE0	Q(RE)	1	End Of	Of the primitive		
FBFACE		PA→SEG	Q(RE)	1	Front fa	ace = 1, back face = 0		
STIPPLE_LINI	99 99	PA→SE0	Q(RE)	1	Stipple	d line need to load tex cords fron	n alternate	
	2003			-	buffer			
RTRn		SEQ→P/	Ą	1	Stalls t	he 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→SE0	Q(RE)	40	Quad)	Y address 10 bits per quad		

17.1.3 PA/SC to RE : Vertex Bus

Name	Direction	Rite	Description
19001110	1011 C C C C C C T I	8.256.3	Description
Vertex indexes	<u>PA→RE</u>	<u>32</u>	Pointers of indexes

17.1.4 PA/SC to SEQ : Vertex Control Bus

Name	Direction	Bits	Description	
STATE	<u>PA→SEQ</u>	21	Render State (6*3+3 for constants)	
Position Cache Pointer	<u>PA→SEQ</u>	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

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

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

1717	CP to	SEQ .	Control	State	store load	r -
1/.1./	UF IU	JEV.	CONTOR	Sidle	SIDIE IDau	

Name	Direction	Bits	Description	
{ISSUE: How,Who and what is t	he size of this bus?]			
17.1.8 MH to SEQ: Ins	struction store	Load		

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

Exhibit 2013 docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page ***

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

4

4

4

4

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044_0256788

ATI Ex. 2105 IPR2023-00922 Page 116 of 239

	E DATE	EDIT DA	ΑTE	R400 Sequencer Specification	PAGE	
24 September	er, 200124	4 September	r, 20155		24 of 33	
1.9 SP to RB : F	Pixel read	from RRs	00174		4	Formatted: Bullets and Numbering
			m:	No]	
ie I Data	Directio SP→RB			Description Pixels (or ½ quad)		
d Address	SP→RB			(Y address 10 bits per		
, and averaging alarma ("	7)					
one exporting clause (7) can be sele	ected at any gi	iven time	3.3 <u>~ .</u>		Formatted: Bullets and Numbering
1.10 SP to PA/S	<u>C : Positi</u>	ion return	<u>bus</u>		4	
	Directio	n	Bits C	Description		
tion return	<u>SP→PA</u>			Position data or sprite size		
tion Buffer pointer imeter cache pointer	SP→PA			Pointer to the position cache Pointer where the data will be in the parame	ter cache	
	0 11			ontor whole the data will be in the parame		
				interleaved on a 16 x 16 basis. We expo		
				ext ALU clause where they are going to exporting data before allowing the progra		
<u>clause.</u>		se that we ter 1		and the program of the second s	and the second country of	
1.3 17.1.11 Shac	lor Engine	to Taytur	o I Init	Rus (Fast Rus)	4	Formatted: Bullets and Numbering
				Unit every clock. These are sourced from a e register file index to read must precede		
				e register me index to read must precede int 1 clock after the Instruction Start signal		
data is read 3 clocks after	r the Instructio	on Start.				
OpeFour Quad's worth of	Texture Data	a may be writte	n to the	Register file every clock. These are directe	d to a different	
				er file index to write must accompany the		
					data. Data and	
				he Instruction Start signal is sent.	data. Data and	
ndex associated with the		be sent 3 cloc			data. Data and	
ndex associated with the Name Tex_Read_Register_Inde	Quad 0 must Directio	n E P 8	ks after t Bits 37	he Instruction Start signal is sent.	ure Address	
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data	Quad 0 must Directio X SEQ->S SP->TE	be sent 3 cloc n E P ह X ह	ks after t Bits 37 512 <u>2048</u>	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4-16 Texture Addresses read from the Re	ure Address egister file	
ndex associated with the Name Tex_Read_Register_Inde	Quad 0 must Directio X SEQ->S SP->TE	be sent 3 cloc n E P ह X ह	ks after t Bits 37	he Instruction Start signal is sent.	ure Address egister file	
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde	Quad 0 must Directio x SEQ->S SP->TEX x SEQ->T	be sent 3 cloc n E P & X & EX &	ks after t <u>37</u> <u>37</u> <u>37</u> <u>37</u>	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4.16 Texture Addresses read from the R Index into Register file for write of ref Data	ure Address egister file	Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data	Quad 0 must Directio x SEQ->S SP->TEX x SEQ->T	be sent 3 cloc n E P & X & EX &	ks after t <u>37</u> <u>37</u> <u>37</u> <u>37</u>	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4.16 Texture Addresses read from the R Index into Register file for write of ref Data	ure Address egister file	Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ	Quad 0 must Directio X SEQ->S SP->TEX X SEQ->T	ibe sent 3 cloc P & E X & E EX & E Texture U	ks after t 3its 37 5122048 37 Mnit bu	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4- <u>16</u> Texture Addresses read from the Re Index into Register file for write of ref Data S (Slow Bus)	ure Address egister file turned Texture	
ndex associated with the <u>Name</u> Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde <u>1.417.1.12</u> _Sequ e every four clock, the t	Quad 0 must Directio xx SEQ->S SP->TE2 x SEQ->T Jencer to exture unit se	be sent 3 cloc n E P & E X & E EX & E Texture U ends to the se	ks after t Bits 37 5122048 37 Init bu quencer	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4-16 Texture Addresses read from the R Index into Register file for write of ref Data S (Slow Bus) on wich clause it is now working and if t	ure Address egister file turned Texture *-	
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.4 <u>17.1.12</u> Sequ e every four clock, the t sters is ready or not. Thi	Quad 0 must Directio x SEQ->S SP->TEX x SEQ->T Jencer to exture unit se s way the se	be sent 3 cloc n EP 8 X 8 EX 8 Texture U ends to the se quencer can u	ks after t Bits 37 5122048 37 Init bu quencer update th	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4- <u>16</u> Texture Addresses read from the Re Index into Register file for write of ref Data S (Slow Bus)	ure Address egister file turned Texture *- the data in the ation fifos. The	Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the tr sters is ready or not. This uencer also provides the	Quad 0 must Directio x SEQ->S SP->TE2 x SEQ->T uencer to exture unit se s way the se e intruction al	be sent 3 cloc n E P & X & E EX & Texture U ends to the se quencer can u nd constants f	ks after t Bits 37 5122048 37 Init bu quencer update th	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4-16 Texture Addresses read from the R- Index into Register file for write of rei Data S (Slow Bus) on wich clause it is now working and if the texture counters for the reservation starts	ure Address egister file turned Texture *- the data in the ation fifos. The	Formatted: Bullets and Numbering
ndex associated with the <u>Name</u> <u>Fex_Read_Register_Inde</u> <u>Fex_RegFile_Read_Data</u> <u>Fex_Write_Register_Inde</u> <u>1.417.1.12_Sequ</u> <u>1.417.1.12_Sequ</u> e every four clock, the t iters is ready or not. This lencer also provides the where to write the texture	Quad 0 must Directio x SEQ->S SP->TE: x SEQ->T <i>uencer to</i> exture unit se s way the se intruction at return data.	be sent 3 cloc n E P E X E EX E Texture U ends to the secure can u nd constants f	ks after t Bits 37 5422048 37 <i>Init bu</i> quencer ipdate th for the te	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4-16 Texture Addresses read from the R Index into Register file for write of ref Data S (Slow Bus) on wich clause it is now working and if the texture counters for the reservation state exture fetch to execute and the address	ure Address egister file turned Texture *- the data in the ation fifos. The	Formatted: Bullets and Numbering
ndex associated with the Vame Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.4 <u>17.1.12_Seq</u> the every four clock, the tigters is ready or not. This tencer also provides the there to write the texture Vame	Quad 0 must Directio x SEQ->S SP->TE2 x SEQ->T uencer to exture unit se s way the se e intruction al	be sent 3 cloc n E P E X E Texture U ends to the se- quencer can u nd constants f	ks after t Bits 37 5422048 37 4 nit bu quencer pdate th for the te Bits [he Instruction Start signal is sent. Description Index into Register files for reading Textu 4-16 Texture Addresses read from the R- Index into Register file for write of rei Data S (Slow Bus) on wich clause it is now working and if the texture counters for the reservation starts	ure Address egister file turned Texture *- the data in the ation fifos. The	Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the t sters is ready or not. Thi uencer also provides the where to write the texture Name Tex_Ready Tex_Clause_Num	Quad 0 must Directio X SEQ->S SP->TE: X SEQ->T UENCER to exture unit se s way the se- e intruction al e return data. Directio TEX->S TEX->S	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the se quencer can u nd constants f SEQ 5 SEQ	ks after t Bits 37 5122048 37 4 11 5 37 4 11 5 3 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4-16 Texture Addresses read from the R Index into Register file for write of rel Data <i>S</i> (<i>Slow Bus</i>) on wich clause it is now working and if the texture counters for the reservation state exture fetch to execute and the address Description Data ready Clause number	turned Texture turned Texture the data in the tation fifos. The in the register	
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the t sters is ready or not. This incer also provides the vhere to write the texture Name Tex_Ready Tex_Clause_Num	Quad 0 must Directio X SEQ->S SP->TEX X SEQ->T <i>Jencer to</i> exture unit se s way the se intruction all return data. Directio TEX->S	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the se quencer can u nd constants f SEQ 5 SEQ	ks after t Bits 37 5422048 37 4 1011 bu quencer update th for the te Bits E 1 C 3 C 210 T	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4-16 Texture Addresses read from the R Index into Register file for write of rel Data S (Slow Bus) on wich clause it is now working and if the texture counters for the reservation state exture fetch to execute and the address Description Data ready Clause number Texture constants_Xstate address 10 bit	turned Texture turned Texture the data in the tation fifos. The in the register	Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.4 <u>17.1.12</u> _Sequ e every four clock, the t sters is ready or not. Thi uencer also provides the vhere to write the texture Name Tex_Ready Tex_Clause_Num Tex_cst	Quad 0 must Directio X SEQ->S SP->TE: X SEQ->T UENCER to exture unit se s way the se- e intruction al e return data. Directio TEX->S TEX->S	be sent 3 cloc P & EX & E	ks after t <u>Bits</u> <u>37</u> <u>5422048</u> <u>37</u> <i>1nit bu</i> quencer update the for the ter <u>Bits []</u> <u>1] []</u> <u>3] []</u> <u>3]</u> <u>4]</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>37</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>5422048</u> <u>542000000000000000000000000000000000000</u>	he Instruction Start signal is sent. Description Index into Register files for reading Textu 4-16 Texture Addresses read from the R Index into Register file for write of rel Data <i>S</i> (<i>Slow Bus</i>) on wich clause it is now working and if the texture counters for the reservation state exture fetch to execute and the address Description Data ready Clause number	the data in the the register	Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the t sters is ready or not. Thi uencer also provides the where to write the texture Name Tex_Ready Tex_Clause_Num Tex_Clause_Num Tex_Inst	Quad 0 must Directio × SEQ->S SP->TEX SEQ->T ////////////////////////////////////	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the se- quencer can u nd constants f SEQ SEQ EX EX	ks after t Bits 37 5422048 37 4nit bu quencer pdate th for the te Bits [1] 1] 2] 2] 2] 2] 2] 2] 2] 2] 2] 2	he Instruction Start signal is sent.	the data in the the register	Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the t sters is ready or not. Thi lencer also provides the vhere to write the texture Name Tex_Ready Tex_Clause_Num Tex_cst Tex_Inst EO_CLAUSE	Quad 0 must Directio x SEQ->S SP-TE2 x SEQ->T uencer to exture unit set s way the set e intruction at e return data. Directio TEX→S SEQ→T SEQ→T SEQ→T	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the sequencer can u nd constants f DEQ 1 DEQ 1 DEQ 1 EX EX	ks after t Bits 37 5122048 37 1/11 bu quencer apdate the ior the te Bits [1] 3] (210] 7 (212] 1] 1] 1] 1] 1] 1] 1] 1] 1] 1	he Instruction Start signal is sent.	the data in the the register	• Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the t sters is ready or not. Thi uencer also provides the where to write the texture Name Tex_Ready Tex_Clause_Num Tex_Clause_Num Tex_Inst	Quad 0 must Directio × SEQ->S SP->TEX SEQ->T ////////////////////////////////////	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the sequencer can u nd constants f DEQ 1 DEQ 1 DEQ 1 EX EX	ks after t Bits 37 5122048 37 1/11 bu quencer apdate the ior the te Bits [1] 3] (210] 7 (212] 1] 1] 1] 1] 1] 1] 1] 1] 1] 1	he Instruction Start signal is sent.	the data in the the register	Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the t sters is ready or not. Thi lencer also provides the where to write the texture Name Tex_Ready Tex_Clause_Num Tex_cst Tex_Inst EO_CLAUSE	Quad 0 must Directio x SEQ->S SP-TE2 x SEQ->T uencer to exture unit set s way the set e intruction at e return data. Directio TEX→S SEQ→T SEQ→T SEQ→T	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the sequencer can u nd constants f DEQ 1 DEQ 1 DEQ 1 EX EX	ks after t Bits 37 5122048 37 1/11 bu quencer apdate the ior the te Bits [1] 3] (210] 7 (212] 1] 1] 1] 1] 1] 1] 1] 1] 1] 1	he Instruction Start signal is sent.	the data in the the register	Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the t sters is ready or not. Thi lencer also provides the vhere to write the texture Name Tex_Ready Tex_Clause_Num Tex_cst Tex_Inst EO_CLAUSE	Quad 0 must Directio x SEQ->S SP-TE2 x SEQ->T uencer to exture unit set s way the set e intruction at e return data. Directio TEX→S SEQ→T SEQ→T SEQ→T	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the sequencer can u nd constants f DEQ 1 DEQ 1 DEQ 1 EX EX	ks after t Bits 37 5122048 37 1/11 bu quencer apdate the ior the te Bits [1] 3] (210] 7 (212] 1] 1] 1] 1] 1] 1] 1] 1] 1] 1	he Instruction Start signal is sent.	the data in the the register	• Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the t sters is ready or not. Thi lencer also provides the vhere to write the texture Name Tex_Ready Tex_Clause_Num Tex_cst Tex_Inst EO_CLAUSE	Quad 0 must Directio x SEQ->S SP-TE2 x SEQ->T uencer to exture unit set s way the set e intruction at e return data. Directio TEX→S SEQ→T SEQ→T SEQ→T	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the sequencer can u nd constants f DEQ 1 DEQ 1 DEQ 1 EX EX	ks after t Bits 37 5122048 37 1/11 bu quencer apdate the ior the te Bits [1] 3] (210] 7 (212] 1] 1] 1] 1] 1] 1] 1] 1] 1] 1	he Instruction Start signal is sent.	the data in the the register	• Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the t sters is ready or not. Thi lencer also provides the vhere to write the texture Name Tex_Ready Tex_Clause_Num Tex_cst Tex_Inst EO_CLAUSE	Quad 0 must Directio x SEQ->S SP-TE2 x SEQ->T uencer to exture unit set s way the set e intruction at e return data. Directio TEX→S SEQ→T SEQ→T SEQ→T	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the sequencer can u nd constants f DEQ 1 DEQ 1 DEQ 1 EX EX	ks after t Bits 37 5122048 37 1/11 bu quencer apdate the ior the te Bits [1] 3] (210] 7 (212] 1] 1] 1] 1] 1] 1] 1] 1] 1] 1	he Instruction Start signal is sent.	the data in the the register	• Formatted: Bullets and Numbering
ndex associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.417.1.12_Sequ e every four clock, the t sters is ready or not. Thi lencer also provides the vhere to write the texture Name Tex_Ready Tex_Clause_Num Tex_cst Tex_Inst EO_CLAUSE	Quad 0 must Directio x SEQ->S SP-TE2 x SEQ->T uencer to exture unit set s way the set e intruction at e return data. Directio TEX→S SEQ→T SEQ→T SEQ→T	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the sequencer can u nd constants f DEQ 1 DEQ 1 DEQ 1 EX EX	ks after t Bits 37 5122048 37 1/11 bu quencer apdate the ior the te Bits [1] 3] (210] 7 (212] 1] 1] 1] 1] 1] 1] 1] 1] 1] 1	he Instruction Start signal is sent.	the data in the the register	• Formatted: Bullets and Numbering
Index associated with the Name Tex_Read_Register_Inde Tex_RegFile_Read_Data Tex_Write_Register_Inde 1.4 <u>17.1.12</u> _Sequ e every four clock, the t sters is ready or not. Thi uencer also provides the where to write the texture Name Tex_Ready Tex_Clause_Num Tex_	Quad 0 must Directio × SEQ->S SP-TE: × SEQ->T <i>Jencer to</i> exture unit set s way the set intruction all return data. Directio TEX→S SEQ→T SEQ→T SEQ→T SEQ→T	be sent 3 cloc n EP 8 X EX 8 Texture U ends to the se- quencer can u nd constants f SEQ SEQ EX EX EX EX	ks after t Bits 37 5422048 37 4nit bu quencer pdate th for the te Bits [1] 2 210 T c 210 T c 212 C 1 L 1 V	he Instruction Start signal is sent.	the data in the ation fifos. The in the register	

ATI Ex. 2105 IPR2023-00922 Page 117 of 239

		ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	<u>au</u>	24 September, 200124	4 September, 20155	GEN-CXXXXX-REVA	25 of 33	Formatted: Bullets and Numbering
18	3. Interi	nal interfaces	Octobor 200124		*	
12	2 .1.5 _S#	ader Engine to REA	PA-Bus			
10	16 P/	\? to sequencer				
A. An	2.1.0 17	1. 10 0040011001				
13	3-19. Ex	kamples of progra	m executions			
	***************	**************************************				
13	<u>3.1.119.</u>	<u>1.1</u> _Sequencer Con	trol of a Vector of V	ertices		
1.			ctually vertex indices – 32 l	bits/index for 512-2048 bit total) to the	e RE's	
	 Vertex FIF state i 	-O pointer as well as tag into po	sition cache is sent along v	vith vertices		
	 space 	was allocated in the positio	n cache for transformed po	sition before the vector was sent		
		er program (using the MH?		d the global instruction store with	the vertex	
		ertex program is assumed to		e the vertex vector. the local state pointer (provided to al	1	
		equencers by the RBBM who			•	
2.				sically the Vertex FIFO always has p	riority	
		point the vector is removed bitrer is not going to select a		the parameter cache is full unless th	e pipe as	
		ng else to do (ie no pixels are				
3.		ates space in the SP registe umber of GPRs required by t		PRs used by the program cal state register, which is accessed	using the	
	state	pointer that came down with	the vertices	-	Ū	
А		will not send vertex data unti Is the vector to the SP regist		as been allocated face (which has a bandwidth of 512. ;	2048	
ч.	bits/cycle)		_			
		64 vertex indices are sent t F0 of SU0, SU1, SU2, and S		er 4 cycles	l	
		F1 of SU0, SU1, SU2, and S				
		F2 of SU0, SU1, SU2, and S F3 of SU0, SU1, SU2, and S				
				vint format?) (what about compoun g data bits are set to zero (x, y, z)	d indices)	
5.				first reservation station (the FIFO in	front of	
	texture sta	ate machine 0, or TSM0 FIF	D)	sition cache and a register file base		
6.				texture clause 0 from the global instru		
	• TSMO	was first selected by the TS	M arbiter before it could sta	art		
		tions of texture clause 0 are	•			
8.	the contro FIFO)	I packet is passed to the ne	t reservation station (the F	IFO in front of ALU state machine 0, o	or ASM0	
	• TSMO			complete; it passes the register file w	rite index	
	 once t 		ta to the register files, it inci	rements a counter that is associated		
	FIFO; clause		dicates that the ALU state i	machine can go ahead start to execu	te the ALU	
			ATL Confidential	Poforonce Convicted Nation on On		
Exhibit 20	13.docR400_Sequ	encer.docR400_Sequencer.doc 35011 By	es*** 🕲 All Confidential. F	Reference Copyright Notice on Cov 09/04/15 12:50 PM10/05/01-10:28 AM/08/13/01-03		

ATI Ex. 2105 IPR2023-00922 Page 118 of 239

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
2	4 September, 200124	4 September, 20155		26 of 33	
		fter being selected by the	ASM arbiter) and gets the instructions	for ALU	
		ssued by ASM0, then the state machine 1, or TSM1	control packet is passed to the next re FIFO)	servation	
 position shared w A param going to there the A parameter para the S long the A if position 	can be exported in ALU vith all four shader pipes eter cache pointer is als be in the parameter cac e is a position export FIH ASM arbiter will prevent er data is exported in cl meter data is sent to the SEQ allocates storage in er a need for the param ASM arbiter will prevent sition is being exported)	clause 3 (or 4?); the data s) back to the PA's positions so sent along with the posi- che. FO in the SP that buffers i a packet from starting an ause 7 (as well as position e Parameter Cache over a in the Parameter Cache, a eters (it is told by the PA a packet from starting on is full	ition data. This tells to the PA where th position data before it gets sent back to exporting clause if the position export n data if it was not exported earlier) a dedicated bus nd the SEQ deallocates that space wh	us (which is ne data is o the PA FIFO is full nen there is no position buffer	
shader progr		eted, the SEQ will free up	The GPRS so that they can be used by	/ another	
13.1.2 19.1.2	2_Sequencer Co	ntrol of a Vector o	f Pixels	*~	Formatted: Bullets and Numbering
1. As with vert	ex shader programs,	pixel shaders are loaded	d into the global instruction store by	the CP	
• At this po	pint it is assumed that th	ne pixel program is loaded	I into the instruction store and thus rea	dy to be read.	
 after the HZ the Parame at this mon 	culling stage a request ter buffer is wide enoug	t is made by the RE to ser ih to source 3 vertices wo cer will free up the parar	before the SEQ takes control of the ve nd parameter data to the Parameter bu rth of a particular parameter in one cyc neter store locations not used anym	iffer He	 Formatted: Bullets and Numbering
 the state 	pointer and the LOD co	prrection bits are also place	s for pixel quads by the detailed walker ced in the Pixel FIF0 worth of barycentrics per cycle	-	
		and Vertex FIFO – when he Pixel FIFO is selected	there are no vertices pending OR there	e is no space 🖜	(Formatted: Bullets and Numbering)
 the numl state poi 	ber of GPRs required by nter		sed by the program a local state register, which is accesse der until space in the register file has b		
from the Pixe •P0i, P0j, an •Q0 i, j, and •The interpo •P0i, P0j, an the sam and reus •a "diffe indik •Q1 i, j, and •The interpo •Q2P0 and C	el FIFO and the parame d P0k (the value of P0 a k are loaded into the int lator then generates the d P0k are sent to the i e primitive as Q0, ther sed for Q1 rrent_prim" control bit cates whether new pai k are then loaded into the lator then generates the Q3P0 are generated in a	ters from the Parameter I at each vertex) are loaded erpolator from the Pixel F o parameter value for eacl interpolator for Q1 only 1 the P0i, P0j, and P0k v is passed with the bary rameter data needs to b he interpolator from the P o parameter value for eacl a similar manner	I into the interpolator from the Paramet IFO 1 pixel in Q0 (Q0P0) If Q1 is from a different primitive; if alues loaded for Q0 are held by the i reentric data for each quad in the Pit e loaded into the interpolator ixel FIFO	ter buffer Q1 is from interpolator	(Formatted: Bullets and Numbering
Ehibit 2013.docR400Sequence	ar.docR400_Sequencer.doc 35011	Bytes*** © ATI Confidenti	al. Reference Copyright Notice on C 09/04/15 12:50 PM19/06/01-10:28 AM08/13/04		

ATI Ex. 2105 IPR2023-00922 Page 119 of 239

	r			······	1	
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE		
	24 September, 200124 September, 200114	4 September, 20155	GEN-CXXXXX-REVA	27 of 33		
throug	nd k now must be re-read fro	om the Pixel FIFO – this mea ch read command until at th entric coordinates to finally b				
bandwidth •16-pixels •RF0- •RF1- • R F2-	rols the transfer of interpolat n of 512-2048 bits/cycle). See worth of interpolated param of SU0, SU1, SU2, and SU3 of SU0, SU1, SU2, and SU3 of SU0, SU1, SU2, and SU3 of SU0, SU1, SU2, and SU3	e interpolated data bus diag eter data is sent to the 16 r His written with Q0P0 the fird His written with Q1P0 secon His written with Q2P0 third c	egister files over 4 cycles st cycle d cycle ycle	ias a		
texture sta note the co	ate machine 0, or TSM0 FIF0 hat there is a separate set o	D) f reservation stations/arbiter ite pointer, the register file b	first reservation station (the FIFO in s/state machines for vertices and for ase pointer, and the LOD correction /els in a separate FIFO	r pixels		
	epts the control packet and was first selected by the TS		exture clause 0 from the global instru rt	uction store *		Formatted: Bullets and Numbering
10. <u>8.</u> all ins	tructions of texture clause 0	are issued by TSM0		4	1	Formatted: Bullets and Numbering
ASM0 FIF • TSM0 index • once t assoc	O) does not wait for texture red for the texture data to the TI the TU has written all the dat	quests made to the Texture J, which will write the data to ta for a particular clause to t a count greater than zero inc	e FIFO in front of ALU state machine Unit to complete; it passes the regist o the RF as it is received he register files, it increments a cour licates that the ALU state machine c	ter file write hter that is		
	accepts the control packet from the global instruction sto		ASM arbiter) and gets the instructior	ns for ALU 🔦		Formatted: Bullets and Numbering
	tructions of ALU clause 0 are e FIFO in front of texture sta		control packet is passed to the next O)	reservation		
executed • pixel c • it	ntrol packet continues to tra data is exported in the last A is sent to an output FIFO wi e ASM arbiter will prevent a	LU clause (clause 7) here it will be picked up by t		een		
45. <u>13.</u> after t shader pro		pleted, the SEQ will free up	the GPRs so that they can be used	by another 🔦		Formatted: Bullets and Numbering
13.1.3<u>19.</u>	1.3_Notes			æ-	1	Formatted: Bullets and Numbering
16.<u>14.</u>the sta real thread		vill operate ahead of time so	that they will be able to immediately	start the		
the instruc		not – this is because the R	he vector through the reservation sta F pointer is different for all threads, k ia the state pointer			
<u>16.</u> Waterfall be specke		tion, loops and branches an	d parameter cache de-allocation still	needs to		
Exhibit 2013 docR400_Sequ	<u>encer.doc</u> R400_Sequencer.doc 35011 Byt	es*** © ATI Confidential. R	eference Copyright Notice on Cov		ł	
			09/04/15 12:50 PM10/05/01-10:28-AM08/13/01-03	:17 PM		

ATI Ex. 2105 IPR2023-00922 Page 120 of 239

ORIGI	NATE E		<u>4</u>		T DAT	E 20155		R400 \$	Seque	ncer S	pecific	ation		PAG 28 of				
I. Timing Dia	gram	S																
1.1 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 SEQ_SP_read_addr	srcA	srcB	srcC	тс	C1_0 srcA	C1_1 srcB	C1_2 srcC	C1_3 TC	srcA	srcB	srcC	тс	srcA	srcB	srcC	тс	srcA	
	310/1	3100	3100		310/1	3100	3100		316/1	3100	3100		SIGN	3100	3100	10	310/1	
SEQ_SP_phase E_SP_data[511:384]		┝			, пр	<u>\</u>			/ ID	<u>^</u>				∱				
3P_0ata[311.364]					ID				U									
SEQ_SP_instruction						10_0	10_1	10_2	10_3									
SEQ_SP_instr_start							<u>`</u>											
mac0_phase			h				<u>`</u>								<u> </u>			
man O avair accent					<u> </u>													
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	тс								
mac0_vector_result													а	r	g	b		
															3			
SEQ_SP_write_addr	ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS	ID	-	PV	PS		
RF0 write cycle		ID	TD	PV	PS	ID	ТD	PV	PS	ID	TD	PV	PS	ID	TD	PV	PS	
Timii	ng Diagra	m 1: Se	quencer	to Sha	der Pipe	e 0, Shad	ler Unit	0, MAC	; 0									
1.2 Sequence	-	<u> </u>		line											4	{I	Formatted:	Bullets and

Ehbit 2013. docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page ***

AMD1044_0256793

ATI Ex. 2105 IPR2023-00922 Page 121 of 239

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 PXF_SEQ_rts PXF_SEQ_rew.prim 00 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 02 03 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01	/ 1î	<u>24 Sep</u>	GINATI	r, 2001	24	4 Sep	DIT DA	r, 2015	5			NT-REV XXXX-		1.	PA 29 o					
PXF_SEQ_new_prim 00 00 01 02 03 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00'' 01'' 02'' 03'' 00'' 01'' 02'' 03'' 00'' 01'' 02'' 03'' 00'' 01'' 02''' 03'' 00'' 01''' 02''' 03''' 00''' 01''' 02'''' 03'''' 00''''' 01''''' 01''''''''' 01''''''''''''''''''''''''''''''''''''			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
PXF_SEQ_new_prim 00 00 01 02 03 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00' 01'' 02'' 03'' 00'' 01'' 02'' 03'' 00'' 01'' 02'' 03'' 00'' 01'' 02'' 03'' 00'' 01'' 02''' 03'' 00'' 01''' 02''' 03''' 00''' 01''' 02'''' 03'''' 00''''' 01''''' 01''''''''' 01''''''''''''''''''''''''''''''''''''																				
PXF_INT_data SEQ_PXF_rtr SEQ_PXF_vector_pop 00 01 02 03 00	PXF_	_SEQ_rts																		7
SEQ_PXF_rtr SEQ_PXF_vector_pop I <td< td=""><td>PXF_SEQ_r</td><td>new_prim</td><td></td><td></td><td>Υ</td><td></td><td></td><td>/</td><td>μ</td><td></td><td></td><td>/</td><td></td><td>h</td><td></td><td></td><td></td><td></td><td></td><td></td></td<>	PXF_SEQ_r	new_prim			Υ			/	μ			/		h						
SEQ_PXF_vector_pop PMB_INT_data P0 P0 P1 P1 P1 P1 P0' P0' P0' P0' P0' P0' P1' P1	PXF_	INT_data	QO	Q0	Q1	Q2	Q3	Q0	Q1	Q2	Q3	Q0'	Q1"	Q2 "	Q3"	Q0'	Q1"	Q2"	Q3"	x
PMB_INT_data P0 P1 P1 P1 P1 P0' P0' P0' P0' P0' P1' P1' P1' P1' P1' X X X SEQ_INT_pm_load INT_param_reg x x P0 P0 P1 P1 P1 P1 P1' P1' P1' P1' X x x SEQ_INT_pm_load INT_param_reg x x P0 P0 P0 P1 P1 P1 P1 P1' P1' P1'' P1''' P1''' P1''' P1''' P1'''' P1'''' P1'''''' P1''''''''''''''''''''''''''''''''''''	SEG	_PXF_rtr		/																7
SEQ_INT_pm_load	SEQ_PXF_ve	ector_pop		Γ							/	h							1	7
INT_param_reg x P0 P0 P0 P1 P1 P1 P1 P0 P0" P0" P1' P	PMB_	INT_data	PO	PO	P1	P1	P1	P1	.0đ	. Dđ	P0,	P0.	P0"	P1'	P1'	P1'	P1"	x	x	x
SEQ_INT_px_load INT_quad_reg x x Q0 Q1 Q2 Q3 Q0 Q1 Q2 Q3 Q0' Q1" Q2" Q3" Q0' Q1" Q1" Q1" Q2" Q3" Q0' Q1" Q1" Q2" Q3" Q0' Q1" Q1" Q2" Q3" Q0' Q1"	SEQ_INT_	_pm_load		/	h			/	h			/		h		/		7		
INT_quad_reg x Q0 Q1 Q2 Q3 Q0 Q1 Q2 Q3 Q0' Q1" Q2" Q3" Q0' Q1" Q1" Q2" Q3" Q0' Q1" Q2" Q3" Q0' Q1" Q2" Q3" Q0' Q1" Q1" Q1" Q1" Q1" Q2" Q3" Q0' Q1" Q1" Q1" Q1" Q1" Q1" Q1	INT_pa	aram_reg	x	x	PO	PO	PO	PO	P1	P1	P1	P1	P0'	P0"	P0"	P0"	P1'	P1"	P1"	P1"
SEQ_SP_phase SEQ_SP_write_addr RE_SP_valid RE_SP_data RF0 write cycle mac0_phase mac1_phase	SEQ_INT	_px_load		/																٦
SEQ_SP_write_addr ID	INT_	quad_reg	x	x	QŨ	Q1	Q2	Q3	QO	Q1	Q2	Q3	Q0'	Q1"	Q2 "	Q3"	Q0'	Q1"	Q2 "	Q3"
SEQ_SP_write_addr ID																				
RE_SP_valid RE_SP_data RF0 write cycle ID TD PV PS ID TD ID ID </td <td>SEQ_S</td> <td>P_phase</td> <td></td> <td>/</td> <td>h</td> <td></td> <td></td> <td>/</td> <td>4</td> <td></td> <td></td> <td>/</td> <td>h</td> <td></td> <td></td> <td>/</td> <td>\uparrow</td> <td></td> <td></td> <td>/</td>	SEQ_S	P_phase		/	h			/	4			/	h			/	\uparrow			/
RE_SP_data Q0P0 Q1P0 Q2P0 Q3P0 Q0P1 Q1P1 Q2P1 Q3P1 Q0P0 Q1P0 Q2P0 Q3P1 Q0P1 Q1P1 Q2P1 Q3P1 Q0P0 Q1P0 Q2P0 Q3P0 Q0P1 Q1P1 Q2P1 Q3P1 Q0P0 Q1P0 Q2P0 Q3P0 Q0P1 Q0P1 Q1P1 Q2P1 Q3P1 Q0P1 Q1P1 Q1P1 Q2P0 Q3P0 Q0P1 Q0P1 Q1P1 Q1P1 Q2P0 Q3P0 Q0P1 Q0P1 Q1P1 Q1P1 Q2P0 Q3P0 Q0P1 Q0P1 Q1P1	SEQ_SP_w	/rite_addr		Γ				ID ID				DID				DI ID				- ID
RF0 write cycle mac0_phase mac1_phase mac2_phase	RE_	SP_valid						/	μ			/	h							<u></u>
mac0_phase mac1_phase mac2_phase	RE_	_SP_data						Q0P0	Q1P0	Q2P0	Q3P0	Q0P1	Q1P1	Q2P1	Q3P1	Q0P0	Q1P0"	Q2P0"	Q3P0"	Q0P1'
mac0_phase mac1_phase mac2_phase																				
mac1_phase mac2_phase					ID	TD	PV	PS	ID	TD	PV	PS	D	TD	PV	PS	ID	TD	PV	PS
mac2_phase					L	↑			ļ/	∧			ļ/	∧			↓/	↑		
						μ	┝			V	∱∕			\sim	└──			<u>ل</u>	<u>۱</u>	
							ļ/	┝			ļ/	┝			ļ/					7
	mac	c3_phase		<u>\</u>				μ	∱			ļ/	∱							

Timing Diagram 2: RE Interpolator to Shader Pipe Data Transfer

Exhibit 2013.docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page ***

AMD1044_0256794

ATI Ex. 2105 IPR2023-00922 Page 122 of 239

	ptembo	E DAT	124	4 Sept	har 20	20155		R400 S	equend	er Spe	cificatio	on		GE of 33](Format	ted: Bull	ets and Number
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
SEQ_SP_read_addr RF0_read_data	TC srcB	srcC	тс	srcA	TC srcB	srcC	TC	srcA	TC srcB	srcC	тс	srcA	TC srcB	srcC	тс	srcA	TC srcB	7-00
SP_TX_tc SEQ_TX_instr_start SEQ_TX_instruction SEQ_TX_clause SEQ_TX_write_addr SEQ_TX_last			10_0 0 r4	TC0	TC1	TC2	TC3 11_0 0 r5	TC0	TC1	TC2	ТСЗ	TCO	TC1	TC2	ТСЗ	TCO	TC1	TC2
SEQ_TX_phase	/	<u>\</u>				<u>\</u>			<u> </u>	<u>\</u>			/	<u>h</u>				
tx_phase TX_SP_write_addr TX_SP_valid			<u>`</u>				<u>`</u>			/				ļ				
TX_SP_data TX_SEQ_clause TX_SEQ_done											TO_O	T0_1	T0_2	т0_3	T1_0 0	T1_1	T1_2	T1_3
SEQ_SP_phase SEQ_SP_write_addr RF0 write cycle	PS	ID PS	- ID	PV TD	PS PV	ID PS	- ID	PV TD	PS PV	ID PS	- ID	PV TD	PS PV	ID PS	- ID	PV TD	PS PV	ID PS
	PS																10000	

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

Ehibit 2013. docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

AMD1044_0256795

ATI Ex. 2105 IPR2023-00922 Page 123 of 239

)

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 200124 September, 200114	4 September, 20155	GEN-CXXXXX-REVA	31 of 33	
14.4 Tim	ing diagrams expl	anations		4-	Formatted: Bullets and Numbering
So for examp SUO, the mos The following 1.all block 2.for regis into tl	e the most significant 512 b t significant 128 bits go to M, assumptions are made: t o block signals are register stor file reads, the RF read d ne MAC (this is the same as	its of a SP goes to SU0 and AC0 and the least significan to register ata is available in the MAC saying the read data is val	e four MACs is from left to right and 1 the least significant 512 bits go to t 128 bits go to MAC3. one clock after a RF read address id out of the RF two clocks after th	SU3; within	
	ted on the SEQ to SP interfa	,	Nee O. Chaday / Jait O. M/		
This diagram MACO is shov means that m cycles for MA	shows the basics of the Ser wn. The timing for MAC1 is ost of the signals need to be C3.	quencer to Shader Pipe inte one clock later than MACC e delayed in the SP by one (Pipe O, Shader Unit O, MA erface. For simplicity only the timir MAC2 one clock later than MAC cycle for MAC1, two cycles for MAC	ng relative to	
SEQ_SP_cor SEQ_SP_rea SEQ_SP_pha	t is asserted during the cycle	s over 4 cycles). Pipelined in Address (8 bits). Pipelined ata transfer to the RF from th	n SP for other MACs.		
RE_SP_ID[51 SU0).	1:384]: This is the most sign		P_data interface (meaning that this pelined in SP for other MACs.	MACO is in	
mac0_phase mac0_cycle_ cycle RE inter RF0_read_da mac0_vector	registered version of SEQ_ count: a counter inside the polated data is written (this r ata: data that is read out of M _result: the 32-bit output of	SP_phase used in MAC0 (th MAC that keeps track of t nay not be he actual signal i IAC0's register file (this may	action transfer. Pipelined in SP for his may not be he actual signal nam he RF write cycles; 0 here corres name). not be he actual signal name). t up over 4 cycles) (this may not	e). ponds to the	
address over RF0 write cy	te_addr: Register File Write this bus. Pipelined in SP for	other MACs. he different write sources (II	t the SEQ does not send the Textu D = Interpolated Data, TD = Textur ⊳ point on the diagram).		
14.4.2 <i>-Ti</i> ,	ming Diagram 2: RE	Interpolator to Shac	ler Pipe Data Transfer	40000000000000000000000000000000000000	
under SEQ co output of the I PXF_SEQ_rtu PXF_SEQ_no	ontrol, and how parameter de nterpolator is then shown be s: Indicates that the output of ew_prim: The current output	ata (for each vertex) is also ing sent over the RE_SP int f the pixel FIFO is valid. of the Pixel FIFO is from a (different primitive that the previous	control. The output. Tells	
PXF_INT_dat SEQ_PXF_rtt next quad of c SEQ_PXF_ve	a: Data output of the Pixel F : Indicates that the current lata will be driven the next cy	IFO – goes to the Interpolate Pixel FIFO output will be ta /cle.	⊬ prim, then new parameter data is or. iken by the Interpolator (driven by f pixels (otherwise RTRs cause th	SEQ). Then	
-	, ,	r Buffer to the Interpolator. (Note that the control of the param	eter buffer is	
SEQ_INT_pr	load: controls the loading (of parameter data into the In	terpolator.		

ATI Ex. 2105 IPR2023-00922 Page 124 of 239

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 200124 September, 200114	4 September, 20155		32 of 33	
are generated SEQ_INT_px	eg: register in the Interpol I for one or more quads (m _load: controls the loading	ator that holds the per-ve ay not be the actual signa of pixel data into the inte			
SEQ_SP_ph SEQ_SP_wri RE_SP_valid RE_SP_data RF0 write cy mac*_phase	Data from the RE interpol	TD1. Indicates that the SP sho ator to the SP. hese phase signals help	uld write the ID on the appropriate cycl to show the timing offset between the same timing).		
	ming Diagram 3: S ipe Data Transfer	Sequencer - Textu	ure Unit Interface and Text	ture Unit *	Formatted: Bullets and Numbering
This diagram transfer is the SEQ_SP_rea	, starts with the texture coo n shown, followed by the to d_addr: see above. Here	exture data transfer to the	gister file and its transfer to the TX		
SP_TX_tc:Tt SEQ_TX_ins SEQ_TX_ins SEQ_TX_cla SEQ_TX_cla SEQ_TX_wri SEQ_TX_las SEQ_TX_pha	Idr: see above. exture coordinate data sent tr_start: Asserted on the fi truction: 96 bits of texture use: the clause number as te_addr: RF write index us t: indicates that this is the li ase: synce the texture data purce the texture data to the	rst cycle of a SEQ to TX i instruction transferred ov sociated with this instruct ed by TX for returned tex ast texture instruction of a write. Note that it is ass	instruction transfer. er 4 cycles. ion. ture data. ture data. a clause. serted early enough to be registered in	ito TX and still	
TX_SP_write TX_SP_valid TX_SP_data TX_SEQ_cla TX_SEQ_dor TX_SEQ_dor		exture data.) data is being driven to the sociated with the texture (that is on the	
SEQ_SP_wri	ase: see above under TD1 te_addr: see above under cle: see above under TD1-	TD1- shown here for refe	ence.		
Ehibit 2013.docR400_Sequ	<u>iencer.doc</u> R400_Sequencer.doc 35011	_{Bytes***} © ATI Confidenti	al. Reference Copyright Notice on C 09/04/15 12:50 PM10/06/01-10:28 AM/08/1400		

ATI Ex. 2105 IPR2023-00922 Page 125 of 239

	r r	· · · · · · · · · · · · · · · · · · ·		- -	
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 200124 September, 200114	4 September, 20155	GEN-CXXXXX-REVA	33 of 33	
					Formatted: Bullets and Numbering
<u>5.20. O</u>	<u>pen issues</u>			*1	
ive the band 1) Le ba 2) W	dwith from the texture store to et the compiler handle the andwith there to operate. Thi	o feed the ALUs. Two solutio case and put those instru- is requires a significant amo ing only at a given time the v	ctions in a texture clause so we o unt of temporary storage in the regis vertices having the same constants t	can use the ster store.	
eed to do so atic).	ome testing on the size of th	e register file as well as on	the register file allocation method (dynamic VS	
aving power	?				
ze of the fif ations.	o containing the information	of a vector of pixels/vertic	es. And size of the fifos before the	reservation	
equencer Ins	struction memory, and const	ant memory.			
bitration pol	licy for the output file.				
oops and bra	anches.				
ne paramete	r cache may end up in the P	A rather than in the RS. Par	ameter cache management thus ma	ay change.	

Exhibit 2013 docR400_Sequencer.docR400_Sequencer.doc 35011 Bytes***
ATI Confidential. Reference Copyright Notice on Cover Page ***

09/04/15 12:50 PM10/05/01-10:28 AM
08/04/15 PM
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04
08/04

AMD1044_0256798

ATI Ex. 2105 IPR2023-00922 Page 126 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201517	GEN-CXXXXX-REVA	1 of 26	
Author:	Laurent Lefebvre	Coropor. / Wild Coropor 1			
Issue To:		Copy No:			
		I			
	R400 S	equencer Spe	ecification		
		SEQ			
		Version 0. <u>8</u> 7			
requ		cted uses of the block. It a	cer block (SEQ). It provides an ove lso describes the block interfaces,		
Document Lo	net Search Title: R400	erforce\r400\arch\doc\gfx\RE) Sequencer Specification APPROVALS			
	Name/Dept Signature/Date				
Remarks:					
	NTIALLY DETRIME	NTAL TO THE INTE	NFORMATION THAT CO REST OF ATI TECHNOL E OR DISCLOSURE.		
work created in unpublished wo confidential, pro	a 2001. The use of this cop ork. The copyright notice oprietary information and tra	yright notice is intended to is not an admission that ade secrets of ATI. No part	rial in this document constitutes an provide notice that ATI owns a cop publication has occurred. This w of this document may be used, re sion of ATI Technologies Inc."	oyright in this ork contains	
Exhibit 2014.docR400_S	Sequences.doc 37948 Bytes*** ⓒ AT	I Confidential. Reference C	Copyright Notice on Cover Page ©	*** <u>09/04/15 12:51</u>	

ATI 2014 LG v. ATI IPR2015-00325

AMD1044_0256799

ATI Ex. 2105 IPR2023-00922 Page 127 of 239



Table Of Contents

1. OVERVIEW
1.2 Data Flow graph
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 6.1 The control Flow Program .13 6.3 Data dependant predicate instructions .15 6.4 Register file indexing .1645 7. REGISTER FILE ALLOCATION .16 8. FETCH ARBITRATION .1847 9. ALU ARBITRATION .1847 10. HANDLING STALLS .1847 11. CONTENT OF THE RESERVATION STATION FIFOS 1847 12. THE OUTPUT FILE 13. IJ FORMAT .1947 14. THE PARAMETER CACHE .1948 15. VERTEX POSITION EXPORTING .1948 16. REAL TIME COMMANDS .2049 18.1 INTERFACES <td< td=""></td<>
2. INTERPOLATED DATA BUS 9 3. INSTRUCTION STORE 12 4. SEQUENCER INSTRUCTIONS 12 5. CONSTANT STORE 12 6. LOOPING AND BRANCHES 12 6. LOOPING AND BRANCHES 12 6.1 The control flow Program 13 6.3 Data dependant predicate instructions 15 6.4 Register file indexing 1645 7. REGISTER FILE ALLOCATION 16 8. FETCH ARBITRATION 1847 10. HANDLING STALLS 1817 11. CONTENT OF THE RESERVATION STATION FIFOS1847 12. THE OUTPUT FILE 13. IJ FORMAT 1917 14. THE PARAMETER CACHE 1918 15. VERTEX POSITION EXPORTING 1918 16. REAL TIME COMMANDS 2019 18.1 PA/SC to RE : J Dus 2019 18.1.1 PA/SC to RE : J Dus 2019 18.1.2 PA/SC to SEQ : J Control bus
3. INSTRUCTION STORE 12 4. SEQUENCER INSTRUCTIONS 12 5. CONSTANT STORE 12 6. LOOPING AND BRANCHES 12 6. LOOPING AND BRANCHES 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 1645 7. REGISTER FILE ALLOCATION 1649 8. FETCH ARBITRATION 1847 9. ALU ARBITRATION 1847 10. HANDLING STALLS 1847 11. CONTENT OF THE RESERVATION STATION FIFPOS 1847 12. THE OUTPUT FILE 1847 13. I J FORMAT 1947 14. THE PARAMETER CACHE 1948 15. VERTEX POSITION EXPORTING 1948 16. REAL TIME COMMANDS 2049 18.1 External Int
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 1615 7. REGISTER FILE ALLOCATION 16 8. FETCH ARBITRATION 1847 9. ALU ARBITRATION 1847 10. HANDLING STALLS 1847 11. CONTENT OF THE RESERVATION STATION FIFOS 1847 12. THE OUTPUT FILE 1847 13. I J FORMAT 1947 14. THE PARAMETER CACHE 1948 15. VERTEX POSITION EXPORTING 1948 16. REAL TIME COMMANDS 2049 18.1 TAKSC to RE : I J bus 2049 18.1 PA/SC to RE : I J bus 2049 18.1.1 PA/SC to RE : I J bus 2149 18.1.2
5. CONSTANT STORE
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 1645 7. REGISTER FILE ALLOCATION 16 8. FETCH ARBITRATION 1847 9. ALU ARBITRATION 1847 10. HANDLING STALLS 1847 11. CONTENT OF THE RESERVATION STATION FIFOS1847 12. THE OUTPUT FILE 1847 13. I J FORMAT 1917 14. THE PARAMETER CACHE 1918 15. VERTEX POSITION EXPORTING 1918 16. REAL TIME COMMANDS 2019 18.1 TREGISTERS 2019 18.1 External Interfaces 2019 18.1.1 PA/SC to RE : IJ bus 20149 18.1.2 PA/SC to RE : IJ bus 2149 18.1.3 VGT to RE : Vertex Bus 2120 18.1.4
6.1 The control liow Program 12 6.2 The Control Flow Program 13 6.3 Data dependant predicate instructions 15 6.4 Register file indexing 1645 7. REGISTER FILE ALLOCATION 16 8. FETCH ARBITRATION 1847 9. ALU ARBITRATION 1847 10. HANDLING STALLS 1847 11. CONTENT OF THE RESERVATION STATION FIFOS 1847 12. THE OUTPUT FILE 1847 13. IJ FORMAT 1917 14. THE PARAMETER CACHE 1918 15. VERTEX POSITION EXPORTING 1918 16. REAL TIME COMMANDS 2018 17. REGISTERS 2019 18.1 External Interfaces 2019 18.1 PA/SC to RE : IJ bus 20149 18.1.1 PA/SC to SEQ : JU Control bus 2149 18.1.3 VGT to RE : Vertex Bus 2120 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 18.1.5 CP to SEQ : Constant store load 2120
6.2 The Control Flow Program
6.3 Data dependant predicate instructions 15 6.4 Register file indexing 1645 7. REGISTER FILE ALLOCATION 1645 8. FETCH ARBITRATION 1847 9. ALU ARBITRATION 1847 10. HANDLING STALLS 1847 11. CONTENT OF THE RESERVATION STATION FIFOS 1847 12. THE OUTPUT FILE 1847 13. I J FORMAT 1947 14. THE PARAMETER CACHE 1948 15. VERTEX POSITION EXPORTING 1948 16. REAL TIME COMMANDS 2048 17. REGISTERS 2019 18.1 INTERFACES 2019 18.1 PA/SC to RE : IJ bus 2019 18.1.1 PA/SC to SEQ : J Control bus 2149 18.1.2 PA/SC to RE : IJ bus 2149 18.1.3 VGT to RE : Vertex Bus 2120 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 CP to SEQ : Constant store load 2120
6.4 Register file indexing 1645 7. REGISTER FILE ALLOCATION 16 8. FETCH ARBITRATION 1847 9. ALU ARBITRATION 1847 10. HANDLING STALLS 1847 11. CONTENT OF THE RESERVATION STATION 1847 12. THE OUTPUT FILE 1847 13. I J FORMAT 1917 14. THE PARAMETER CACHE 1918 15. VERTEX POSITION EXPORTING 1918 16. REAL TIME COMMANDS 2018 17. REGISTERS 2019 18.1 PA/SC to RE : IJ bus 2019 18.1.1 PA/SC to RE : IJ bus 2019 18.1.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to RE : IJ bus 2019 18.1.3 VGT to RE : Vertex Bus 2149 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 CP to SEQ : Constant store load 2120
7. REGISTER FILE ALLOCATION 16 8. FETCH ARBITRATION 1847 9. ALU ARBITRATION 1847 10. HANDLING STALLS 1847 11. CONTENT OF THE RESERVATION STATION FIFOS1847 12. THE OUTPUT FILE 13. IJ FORMAT 1947 14. THE PARAMETER CACHE 1948 15. VERTEX POSITION EXPORTING 1948 16. REAL TIME COMMANDS 2049 18. INTERFACES 2049 18.1 PA/SC to RE : IJ bus 2049 18.1.1 PA/SC to RE : IJ bus 2049 18.1.2 PA/SC to RE : IJ bus 2049 18.1.3 VGT to RE : Vertex Bus 2149 18.1.3 VGT to RE : Vertex Control bus 2149 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 CP to SEQ : Constant store load 2120
8. FETCH ARBITRATION 1817 9. ALU ARBITRATION 1817 10. HANDLING STALLS 1817 11. CONTENT OF THE RESERVATION STATION FIFOS1847 12. THE OUTPUT FILE 13. IJ FORMAT 1917 14. THE PARAMETER CACHE 1918 15. VERTEX POSITION EXPORTING 1918 16. REAL TIME COMMANDS 2019 18. INTERFACES 2019 18.1 PA/SC to RE : IJ bus 2019 18.1.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to RE : IJ bus 2019 18.1.3 VGT to RE : Vertex Bus 2149 18.1.3 VGT to RE : Vertex Control bus 2149 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.4 18.1.5 CP to SEQ : Constant store load 2120 18.1.5
9. ALU ARBITRATION
10. HANDLING STALLS
11. CONTENT OF THE RESERVATION STATION FIFOS1847 12. THE OUTPUT FILE 1847 13. IJ FORMAT 1917 14. 1917 14. THE PARAMETER CACHE 1918 15. VERTEX POSITION EXPORTING 1918 15. VERTEX POSITION EXPORTING 1918 16. REAL TIME COMMANDS 2018 17. REGISTERS 2019 18.1 External Interfaces 2019 18.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to SEQ : J Control bus 2149 18.1.3 VGT to RE : Vertex Bus 2120 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 CP to SEQ : Constant store load 2120 1240
FIFOS 1847 12. THE OUTPUT FILE 1847 13. IJ FORMAT 1917 14. THE PARAMETER CACHE 1918 15. VERTEX POSITION EXPORTING 1918 16. REAL TIME COMMANDS 2018 17. REGISTERS 2019 18. INTERFACES 2019 18.1 PA/SC to RE : IJ bus 20149 18.1.1 PA/SC to SEQ : IJ Control bus 2119 18.1.2 PA/SC to SEQ : IJ Control bus 2119 18.1.3 VGT to RE : Vertex Bus 2120 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 CP to SEQ : Constant store load .2120
12. THE OUTPUT FILE 1847 13. IJ FORMAT 1917 14. THE PARAMETER CACHE 1918 15. VERTEX POSITION EXPORTING 1918 16. REAL TIME COMMANDS 2019 17. REGISTERS 2019 18. INTERFACES 2019 18.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to SEQ : IJ Control bus 2149 18.1.3 VGT to RE : Vertex Bus 2120 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5
13. IJ FORMAT
14. THE PARAMETER CACHE 1918 15. VERTEX POSITION EXPORTING 1918 16. REAL TIME COMMANDS 2018 17. REGISTERS 2019 18. INTERFACES 2019 18.1 PA/SC to RE : IJ bus 2019 18.1.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to RE : IJ control bus 2149 18.1.3 VGT to RE : Vertex Bus 2120 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5
15. VERTEX POSITION EXPORTING 1948 16. REAL TIME COMMANDS 2018 17. REGISTERS 2019 18. INTERFACES 2019 18. INTERFACES 2019 18.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to SEQ : J Control bus 2149 18.1.3 VGT to SEQ : J Control bus 2149 18.1.4 VGT to SEQ : Vertex Bus 2120 18.1.5 CP to SEQ : Constant store load .2120
16. REAL TIME COMMANDS 2018 17. REGISTERS 2019 18. INTERFACES 2019 18.1 External Interfaces 2019 18.1.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to SEQ : IJ Control bus 2149 18.1.3 VGT to SEQ : IJ Control bus 2149 18.1.3 VGT to SEQ : Vertex Bus 2120 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 18.1.5 CP to SEQ : Constant store load .2120
17. REGISTERS 2019 18. INTERFACES 2019 18.1 External Interfaces 2019 18.1.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to SEQ : IJ Control bus 2119 18.1.3 VGT to RE : Vertex Bus 2120 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 18.1.5 CP to SEQ : Constant store load .2120
18. INTERFACES 2019 18.1 External Interfaces 2019 18.1.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to SEQ : IJ Control bus 2119 18.1.3 VGT to RE : Vertex Bus 2120 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 18.1.5 CP to SEQ : Constant store load .2120
18.1 External Interfaces 2019 18.1.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to SEQ : IJ Control bus 2019 18.1.3 VGT to RE : Vertex Bus 2149 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.4 18.1.5 CP to SEQ : Constant store load .2120
18.1.1 PA/SC to RE : IJ bus 2019 18.1.2 PA/SC to SEQ : IJ Control bus 2119 18.1.3 VGT to RE : Vertex Bus 2120 18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 18.1.5 CP to SEQ : Constant store load 2120
18.1.2 PA/SC to SEQ : IJ Control bus2149 18.1.3 VGT to RE : Vertex Bus
18.1.3 VGT to RE : Vertex Bus
18.1.4 VGT to SEQ : Vertex Control Bus 2120 18.1.5 CP to SEQ : Constant store load .2120
18.1.6 CP to SEQ : Fetch State store load
2220
18.1.7 CP to SEQ : Control State store load
2221
18.1.8 MH to SEQ: Instruction store Load
2224
18.1.9 SP to RB : Pixel read from RBs2221
18.1.10 SEQ to RB : Control bus
18.1.11 RB to SEQ : Output file control2224
18.1.12 SP to RB : Position return bus2321
18.1.13 Shader Engine to Fetch Unit Bus (Fast
Bus) 23 22
18.1.14 Sequencer to Fetch Unit bus (Slow
18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 2322
18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 2322 19. INTERNAL INTERFACES
18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 2322 19. INTERNAL INTERFACES
18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 2322 19. INTERNAL INTERFACES
18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 2322 19. INTERNAL INTERFACES
18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 2322 19. INTERNAL INTERFACES
18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 2322 19. INTERNAL INTERFACES
18.1.14 Sequencer to Fetch Unit bus (Slow Bus) 2322 19. INTERNAL INTERFACES

91	ODEN ISSUES 2024
<u>21.</u> 1.	OPEN ISSUES
1 1	Top Level Block Diagram
1.2	Data Flow graph
1 3	Control Graph
3	INTERPOLATED DATA BUS
	INSTRUCTION STORE
A.	SEQUENCER INSTRUCTIONS
	CONSTANT STORE 12
	LOOPING AND BRANCHES
6.1	The controlling state 12
6.2	The Control Flow Program
6.3	Data dependant predicate instructions
6.4	Register file indexing
7	REGISTER FILE ALLOCATION
	FETCH ARBITRATION
9	ALU ARBITRATION
10.	HANDLING STALLS
11	CONTENT OF THE RESERVATION STATION
FIFOS	546
12.	THE OUTPUT FILE
13	IJ FORMAT
14	THE PARAMETER CACHE
15.	VERTEX POSITION EXPORTING
16	REAL TIME COMMANDS
	REGISTERS
18.	INTERFACES
18.1	External Interfaces
181	1 PA/SC to RE · U hus 18
$\frac{18.1}{18.1}$.1 PA/SC to RE : IJ bus
$\frac{18.1}{18.1}$.1 PA/SC to RE : IJ bus
18.1 18.1 18.1 18.1	1 PA/SC to RE : IJ bus 18 .2 PA/SC to SEQ : IJ Control bus 18 .3 PA/SC to RE : Vertex Bus 19 .4 PA/SC to SEQ : Vertex Control Bus 19
$ \begin{array}{r} 18.1 \\ $	1 PA/SC to RE : IJ bus 18 .2 PA/SC to SEQ : IJ Control bus 18 .3 PA/SC to RE : Vertex Bus 19 .4 PA/SC to SEQ : Vertex Control Bus 19 .5 CP to SEQ : Constant store load 19
18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus
18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .18 .3 PA/SC to RE : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Fetch State store load .19 .7 CP to SEQ : Control State store load .19
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .18 .3 PA/SC to SEQ : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Fetch State store load .19 .7 CP to SEQ : Control State store load .29
$ \begin{array}{r} 18.1 \\ $.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .18 .3 PA/SC to RE : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Fetch State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ : Instruction store Load .20
$ \begin{array}{r} 18.1 \\ $.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .18 .3 PA/SC to RE : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Constant store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .18 .3 PA/SC to RE : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Fetch State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs
$ \begin{array}{r} 18.1 \\ 1$.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .18 .3 PA/SC to RE : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Fetch State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ : Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to PA/SC : Position return bus .20
$\begin{array}{c} 18.1\\$.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .18 .3 PA/SC to RE : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus.19 .19 .5 CP to SEQ : Constant store load19 .19 .6 CP to SEQ : Fetch State store load19 .7 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ : Instruction store Load20 .9 .9 SP to RB : Pixel read from RBs
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .19 .3 PA/SC to RE : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Constant store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to PA/SC : Position return bus .20
18.1 18.1 <t< td=""><td>.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .19 .3 PA/SC to SEQ : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Control State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to RB : Control bus .20 .11 SP to PA/SC : Position return bus .20 .12 Shader Engine to Fetch Unit Bus (Fast) .13 Sequencer to Fetch Unit bus (Slow)</td></t<>	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .19 .3 PA/SC to SEQ : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Control State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to RB : Control bus .20 .11 SP to PA/SC : Position return bus .20 .12 Shader Engine to Fetch Unit Bus (Fast) .13 Sequencer to Fetch Unit bus (Slow)
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .19 .3 PA/SC to SEQ : IJ Control bus .19 .4 PA/SC to SEQ : Vertex Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Control State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to RB : Control bus .20 .12 Shader Engine to Fetch Unit Bus (Fast .20 .20 .20
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .18 .3 PA/SC to SEQ : IJ Control bus .19 .4 PA/SC to SEQ : Vertex Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Control State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ : Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to PA/SC : Position return bus .20 .12 Shader Engine to Fetch Unit Bus (Fast .20 .20 .20 .13 Sequencer to Fetch Unit bus (Slow) .20 .13 Sequencer to Fetch Unit bus (Slow) .20 INTERNAL INTERFACES .21 .21
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .18 .3 PA/SC to RE : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus.19 .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Fetch State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : U bus .18 .2 PA/SC to SEQ : U Control bus .19 .3 PA/SC to SEQ : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Fetch State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ : Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to PA/SC : Position return bus .20 .12 Shader Engine to Fetch Unit Bus (Fast .20 .20 .13 .13 Sequencer to Fetch Unit bus (Slow) .20 .20 INTERNAL INTERFACES .21 EXAMPLES OF PROGRAM EXECUTIONS21 .1 Sequencer Control of a Vector of
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .19 .3 PA/SC to SEQ : IJ Control bus .19 .4 PA/SC to SEQ : Vertex Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Control State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to RB : Pixel read from RBs .20 .12 Shader Engine to Fetch Unit Bus (Fast) .20 .13 Sequencer to Fetch Unit bus (Slow) .20 .13 Sequencer Control of a Vector of ices .21 EXAMPLES OF PROGRAM EXECUTIONS .21 .21
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .19 .3 PA/SC to SEQ : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Control State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ : Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to PA/SC : Position return bus .20 .12 Shader Engine to Fetch Unit Bus (Fast) .20 .13 Sequencer to Fetch Unit bus (Slow) .20 INTERNAL INTERFACES .21 .21 EXAMPLES OF PROGRAM EXECUTIONS21 .1 Sequencer Control of a Vector of .2 Sequencer Control of a Vector of .21
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .19 .3 PA/SC to SEQ : IJ Control bus .19 .4 PA/SC to SEQ : Vertex Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Control State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to RB : Pixel read from RBs .20 .12 Shader Engine to Fetch Unit Bus (Fast) .20 .13 Sequencer to Fetch Unit bus (Slow) .20 .13 Sequencer Control of a Vector of ices .21 EXAMPLES OF PROGRAM EXECUTIONS .21 .21
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .19 .3 PA/SC to RE : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus.19 .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Control State store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ : Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to PA/SC : Position return bus .20 .12 Shader Engine to Fetch Unit Bus (Fast) .20 .13 Sequencer to Fetch Unit bus (Slow) .20 .13 Sequencer Control of a Vector of .21 .14 Sequencer Control of a Vector of .22 .2 Sequencer Control of a Vector of .22 .3 Notes .23
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Control bus .19 .4 PA/SC to SEQ : Vertex Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Constant store load .19 .7 CP to SEQ : Control State store load .20 .8 MH to SEQ: Instruction store Load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .11 SP to PA/SC : Position return bus .20 .12 Shader Engine to Fetch Unit Bus (Fast) .20 .21 Stageuencer to Fetch Unit bus (Slow) .20 .21 .22 .13 Sequencer Control of a Vector of ices .21 .21 Sequencer Control of a Vector of ices .21 .22 .23 Notes .23 OPEN ISSUES .23 .23
18.1 18.1 18.1 18.1 18.1 18.1 18.1 18.1	.1 PA/SC to RE : IJ bus .18 .2 PA/SC to SEQ : IJ Centrol bus .19 .3 PA/SC to SEQ : Vertex Lontrol Bus .19 .4 PA/SC to SEQ : Vertex Control Bus .19 .5 CP to SEQ : Constant store load .19 .6 CP to SEQ : Control State store load .19 .6 CP to SEQ : Control State store load .20 .7 CP to SEQ : Instruction store Load .20 .8 MH to SEQ: Instruction store Load .20 .9 SP to RB : Pixel read from RBs .20 .10 SEQ to RB : Control bus .20 .12 Shader Engine to Fetch Unit Bus (Fast) .20 .13 Sequencer to Fetch Unit bus (Slow) .20 .13 Sequencer Control of a Vector of .21 EXAMPLES OF PROGRAM EXECUTIONS21 .1 Sequencer Control of a Vector of .10 Sequencer Control of a Vector of .22 .3 .3 Notes .23 .23

Exhibit 2014 doc R400_Sequencer.doc 37948 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page ***

AMD1044_0256800

No. of Concession, Name			[
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201517	GEN-CXXXXX-REVA	3 of 26
1.2 Data FI	low graph		.2 PA/SC to SEQ : IJ Control t	us18
1.3 Control	Graph		.3 PA/SC to RE : Vertex Bus	
2. INTERI	POLATED DATA BUS		.4 PA/SC to SEQ : Vertex Con	trol Bus.18
3. INSTRI	UCTION STORE		.5 CP to SEQ : Constant store	load19
4. SEQUE	ENCER INSTRUCTIONS	16.1	.6 CP to SEQ : Texture State s	store-load
5. CONST	TANT STORE			
6. LOOPI	NG AND BRANCHES		.7 CP to SEQ : Control State s	tore load
6.1 The co	ntrolling state			
	entrol Flow Program		.8 MH to SEQ: Instruction stor	e Load 19
	ependant predicate instructi		.9 SP to RB : Pixel read from I	RBs19
	er file indexing		.10 SP to PA/SC : Position retu	rn bus19
v	TER FILE ALLOCATION		.11 Shader Engine to Texture U	Init-Bus
8. TEXTU	RE ARBITRATION		st-Bus) 19	
9. ALU A	RBITRATION		.12 Sequencer to Texture Unit I	ous (Slow
10. HANDI	ING STALLS		.)20	
11. CONTE	ENT OF THE RESERVATIO	N-STATION 17.	INTERNAL INTERFACES	
FIFOS 17		18.	EXAMPLES OF PROGRAM EXECU	TIONS20
12. THE O	UTPUT FILE		.1 Sequencer Control of a Vec	tor of
13. THE P/	ARAMETER CACHE		tices 20	
14. VERTE	X POSITION EXPORTING		.2 Sequencer Control of a Vec	tor of
15. REGIS	TERS		sls21	
16. INTERI	FACES		3 Notes	
	mal Interfaces		OPEN ISSUES	
16.1.1	PA/SC to RE : IJ bus			

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.

Exhibit 2014 doc R400_Sequencer.doc 37848 Bytes***
CATI Confidential. Reference Copyright Notice on Cover Page
Category 1 (2014) 402/16 1221 PM10/05/01 10:28 AM

AMD1044_0256801

ATI Ex. 2105 IPR2023-00922 Page 129 of 239

29	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
ш	24 September, 2001	4 September, 201517		4 of 26	

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 texturefetch 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 texturefetch 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 texturefetch reservation station to alu reservation station. A FIFO exists between each 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 texturefetch stations to choose a texturefetch 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 texturefetch 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 instruction thus there is only one sequencer for the whole chip.

Exhibit 2014.doc.8400_Sequencer.doc 37948 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page *** 09/04/15 12:51

AMD1044_0256802

ATI Ex. 2105 IPR2023-00922 Page 130 of 239

PROTECTIVE ORDER MATERIAL

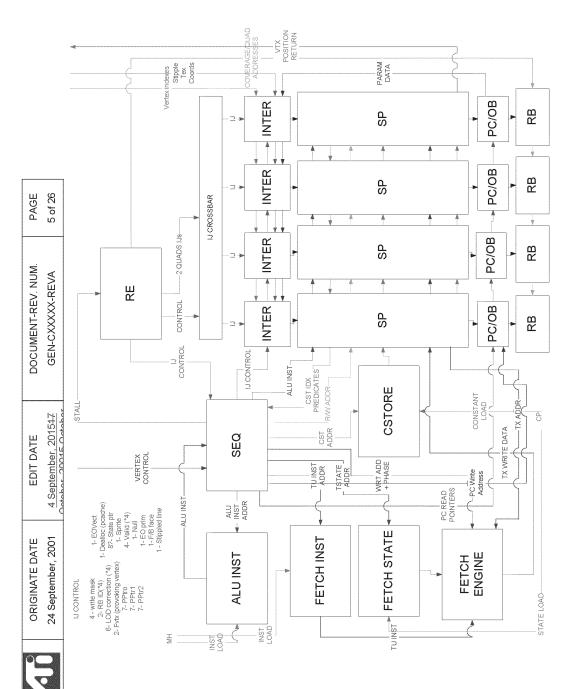


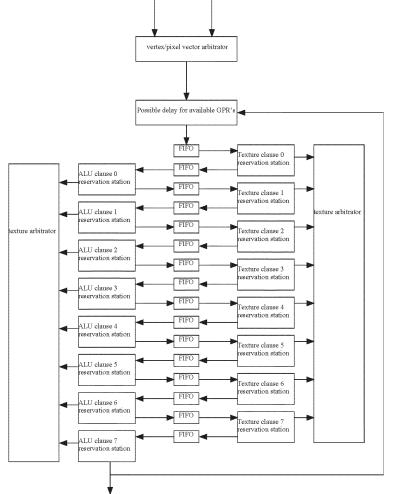
Exhibit 2014 doc R400-Sequencer doc 37948 Bytestrett © ATI Confidential. Reference Copyright Notice on Cover Page © **** 2004/15 1251 PM1005601-10:28 AM

AMD1044_0256803

ATI Ex. 2105 IPR2023-00922 Page 131 of 239



1.1 Top Level Block Diagram



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 texturefetch 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.

Exhibit 2014.doc.R400_Sequencer.doc 37948 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:51

AMD1044_0256804

ATI Ex. 2105 IPR2023-00922 Page 132 of 239

	r		·
ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201517	GEN-CXXXXX-REVA	7 of 26

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<u>fetch</u> machine issues a texure request and corresponding register address for the <u>texturefetch</u> address (ta). A small command (tcmd) is passed to the <u>texturefetch</u> system identifying the current level number (0) as well as the register write address for the <u>texturefetch</u> return data. One <u>texturefetch</u> 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 texturefetch unit writes the data to the register file using the write address that was provided by the level 0 texturefetch machine and sends the clause number (0) to the level 0 texturefetch state machine to signify that the write is done and thus the data is ready. Then, the level 0 texturefetch 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).

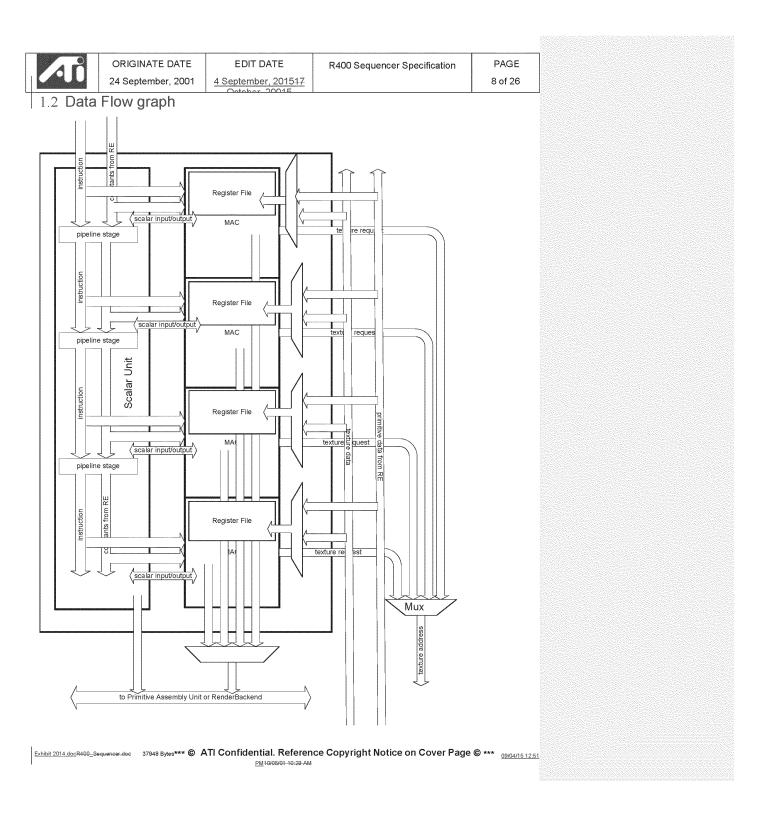
All other level process in the same way until the packet finally reaches the last ALU machine (8). On completion of the level 8 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that 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 texturefetch 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 texturefetch state machines). The arbitrers always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

Exhibit 2014.docR400_Sequencer.doc 37948 Bytes***
C ATI Confidential. Reference Copyright Notice on Cover Page
C ***
OB/04/15 12:51

AMD1044_0256805

ATI Ex. 2105 IPR2023-00922 Page 133 of 239

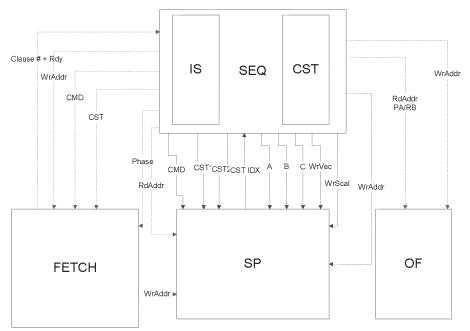


ATI Ex. 2105 IPR2023-00922 Page 134 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
<u>con</u>	24 September, 2001	4 September, 201517	GEN-CXXXXX-REVA	9 of 26

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.

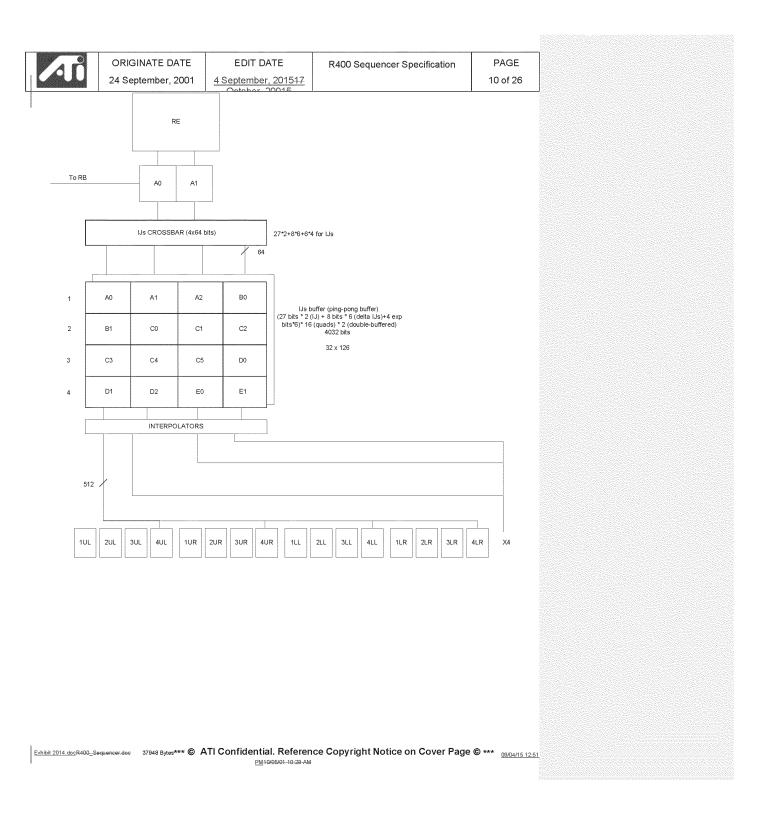
2. Interpolated data bus

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

Exhibit 2014.docR400_Sequencer.doc 37948 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:51

AMD1044_0256807

ATI Ex. 2105 IPR2023-00922 Page 135 of 239



ATI Ex. 2105 IPR2023-00922 Page 136 of 239 PROTECTIVE ORDER MATERIAL

	T19	51 - 5 51	52- 55	56- 56-	63 - 63 <	
	T18	32- 35	39 3 6 <	5 40 < <	> 4 47	
	T17	> 1 0 - 2	23 ² 0 <	24- 27	31 ²⁸⁻	
	T16	> %	> 7-4	8-11	> ¹ / ₇ ¹ / ₅ <	
	T15			Ш	Ξ	
ЗЕ 126	T14				â	
PAGE 11 of 26	T13		C	G	8	
	T12				BO	Ĕ
DOCUMENT-REV. NUM. GEN-CXXXX-REVA	T11	5	D3			LL.
OCUMENT-REV. NUN GEN-CXXXX-REVA	T10	S	5 2	C5		
CUME EN-CX	Т9	8				
	Т8	AO	A1	A2		
14	11			ЕO	ш	
ATE 2015	T6				õ	
EDIT DATE	T5		S	5	8	
EDIT DATE 4 September, 201547	Τ4				BO	0
	13	5	D2			0 d
ORIGINATE DATE 24 September, 2001	12	C3	C4	C5		
IGINAT Septem	11	B1				
0R 24 5	10 T	AO	A1	A2		
3		SPO	SP1	SP2	SP3	

AMD1044_0256809

Exhibit 2014 doo R400_Sequences doe 37948 Bytes**** C ATI Confidential. Reference Copyright Notice on Cover Page C **** 2004/15 1251 PM1005601-10-28 AM

ATI Ex. 2105 IPR2023-00922 Page 137 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September, 2001	4 September, 201517		12 of 26	

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 $1\mathbb{R}/4W$ port memory; we use 2 <u>1</u> clocks to load the ALU instruction, <u>_-and-1</u>2 clocks to load the <u>TextureFetch</u> instruction, <u>1 clock to load 2 control flow instructions and 1 clock to write instructions</u>.

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:

Exhibit 2014.doc.R400_Sequencer.doc 37948 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 00/04/15/12:61

PM10/05/01-10:28-AM

AMD1044_0256810

ATI Ex. 2105 IPR2023-00922 Page 138 of 239

	annon an Ai
	11 and 1
A. ANNESS	Albert

Boolean[25534:0] 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_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.

<u>Conditional_execute_or_Jump executes a block of instructions or jumps to an address is the condition is not met.</u> <u>Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition.</u> <u>End_of_clause marks the end of a clause.</u>

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	<u>46 42</u>	41 24	<u>23 12</u>	<u>11 0</u>	
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	<u>41 34</u>	33	<u>32 21</u>	20 12	<u>11 0</u>				
Addressing	00010	Booleans	Condition	<u>Jump</u> address	Instruction_count	Exec Address				

lif the specified boolean (68 bits can address 64256 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.

Exhibit 2014 docR400_Sequencer.doc 37548 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15 12:51

AMD1044_0256811

	ORIGINATI		EDI	Γ DATE	D 400 0	0	PAGE
	24 Septemb			ber, 201517	R400 Seq	uencer Specification	14 of 26
	24 Septemb	er, 2001	Octob	or 20015			14 01 20
			Cor	nditionnal Exe	cute		
47	46 42	41		33	32 21	20 12	110
Addressing	00011	Boolean	address	Condition	RESERVED	Instruction_count	Exec Address
	d boolean (8 bi ip to 512 instruc		ress 256 b	ooleans) meet	s the specified	I condition then exec	ute the specified
£ 100	40			inal_Execute	PROPERTY AND	00 10	
47 Addressing	<u>46 42</u> 00100		38 ate vector	37 Condition	36 21 RESERVED	20 12 Instruction count	<u>11 0</u> Exec Address
				1		the specified number	
	andition of the					specified address ir jump should be to loc	
<u>47</u>	46 42		<u>1 16</u>		<u>15 4</u>		<u> 0</u>
Addressing	00101	RE	SERVED	<u>Jı</u>	ump address		pop ID
only. Also cor	nputes the inde	x value.		Loop End		et jump to the addre	
<u>47</u>	<u>46 42</u> 00111		<u>1 16</u> SERVED	S	15 4 tart address		<u>8 0</u> pop ID
Addressing							
	is described doe				inclusion of the	<u>e loop.</u> e loop id make this e	asy to do.
<u> </u>				<u>x)</u>	<u>A</u> 1 daa		
Addressing	01000			RESE	RVED		Address
Jumps to the	specified addre	ss and pus	hes the IP of		stack.		
47	46 42			Return			
<u>++1</u>	4042						
					<u>41 0</u>		
Exhibit 2014.docR400_S	iequencer.doc 37948	Bytes*** ⓒ A	ATI Confide	ntial. Referen	ice Copyright	Notice on Cover Pa	ige © *** <u>09/04/15 12:</u>

ATI Ex. 2105 IPR2023-00922 Page 140 of 239

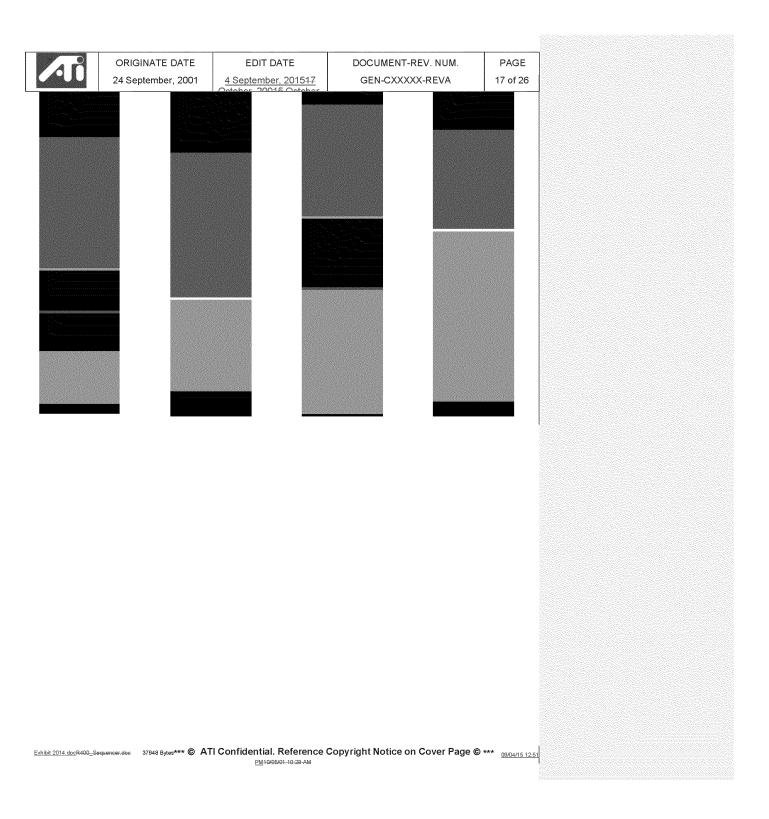
			r	F		r					
AR	ORIGINATE	E DATE	EDIT DATE	DIT DATE DOCUMENT-REV. NUM.		PAGE					
	24 Septemb	er, 2001	4 September, 201517	GEN-CXXXX	X-REVA	15 of 26					
Addressing	01001			RESERVED							
Pops the topn	nost address from	n the stack	and jumps to that address.								
4 ***?	40		Conditionnal_Jump	00 40							
47	<u>46 42</u> 01010	<u>41</u> Boolean a	00000	<u>32 12</u> <u>RESERVED</u>	<u>11</u> Addre						
Addressing											
If condition me	et, jumps to the a	address. FC	DRWARD jump only allowed	<u>-</u>							
			End of Clause								
<u>47</u>											
Addressing	01011										
Addressing			Ē	RESERVED							
Marks the end	i of a clause.										
***************************************	***************************************	will keep 9	bits loop counters instead	of 8 (we are only abl	a ta laan 256 tii	map) If the					
counter goes a loop index v	higher than 255 alue of 17 bits. 1	then the loo his will be	op end or the loop start is guided and a start is guided and a start is guided and a start of the start is a start of the	going to break the loo	p. The sequenc	er will keep					
and the register file. The way to compute this value is: Index = Loop_counter*Loop_iterator + Loop_init.											

The basic mo	del is as follows:										
Vertex_shade Vertex_shade	ate defined the c r_fetch[7:0][7:0] r_alu[7:0][7:0] _fetch[7:0][7:0] _alu[7:0][7:0]	// eight 8 // eight 8 // eight 8	daries: bit pointers to the location v bit pointers to the location v bit pointers to the location v bit pointers to the location v	vhere each clauses co vhere each clauses co	ontrol program is ontrol program is	s located s located					
			istructions in size. (There is ogram memory to allow for r								
			e is executed to completic lu execution). The control p								
The addresse same time.	s from the cont	rol-program	are added to another offs	et to allow for multiple	e-programs-resi	dent-at-the					
Under this mo	del, all subroutir	e calls mus	st be inlined into the control	program.							
6.3 Data	dependar	nt predie	cate instructions								
	ant conditionals calar predicate c		ported in the R400. The only of the form:	y way we plan to sup	port those is by	supporting					
	PRED_SETE	# - similar	to SETE except that the res	sult is 'exported' to the	sequencer.						
Exhibit 2014.docR400_S	equencer.doc 37948 B	_{vtes} *** © AT	l Confidential. Reference	Copyright Notice on	Cover Page ©	*** <u>09/04/15 12:5</u>					

ATI Ex. 2105 IPR2023-00922 Page 141 of 239

		ORIG	INATE	DATE	EDIT DAT	ΓE	R400 Sequencer	Specification	PAGE
		24 Sep	otember	, 2001	4 September, 2	201517			16 of 26
					lar to SETGT exc	cept that I	the result is 'exported at the result is 'expor		
	For the scalar	PRED	SETÉO	ve will als _ <u>#</u> – SE1 _ <u>#</u> – SE1		vo followir	ng instructions:		
	maintain the4 exposed) and	<u>sets of</u> 6 use it to 0	4 bit pre control t	edicate ve he write i	ector <u>s</u> (in fact 8 s masking (two-set	ets becau s for inter	path as the MOVA ir use we interleave two leaved operation). The icate set you want to	programs but on his predicate is no	ly 4 will be
	Then we have we execute of					s a condit	ional execute "on" bit	and the second l	bit tells us if
	P0_A	ADD_ <u>#</u> R0	,R1,R2						
I		results to	the GP	Rs whose	e predicate bit is		e predicate bit is 0. Al use of the P0 or P1 w		
	{Issue: do we	have to h	ave a N	OP betw	een PRED and tl	he first in	struction that uses a	predicate?}	
	6.4 Regi	ister fil	e ind	exing					
	retrieve the d	lata create	ed in a t	exture <u>fet</u>		nd use it	b be able to index ir into an ALU clause. these controls :		
			Bit7 0 0 1 1	Bit 6 0 1 0 1	'relative 'previou	e register register' is vector' is scalar'	,		
							s. In the case of a rel this becomes our ne		
	7. Regis	ster file	allo	ation					
	managed usi	ing two ro ertices is a	und rob allowed	ins (one to move	for pixels and o	ne for ve	static or dynamic. In rtices). In the dynam ixed to VERTEX_RE	ic case the boun	dary between
s.	xhibit 2014.doc8400_S	Sequencer.doc	37948 Byt	es*** © A		Referen	ce Copyright Notice	e on Cover Page	© *** 09/04/15 12:5

ATI Ex. 2105 IPR2023-00922 Page 142 of 239



ATI Ex. 2105 IPR2023-00922 Page 143 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201517		18 of 26

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. TextureFetch Arbitration

The texturefetch arbitration logic chooses one of the 8 potentially pending texturefetch 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<u>fetch</u> 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<u>fetch</u> return data.

9. ALU Arbitration

ALU arbitration proceeds in almost the same way than texturefetch 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<u>fetch</u> 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).

Exhibit 2014 doc R400_Sequencer.doc 37948 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:51

AMD1044_0256816

ATI Ex. 2105 IPR2023-00922 Page 144 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201517	GEN-CXXXXX-REVA	19 of 26

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 <u>19</u>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).

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

PO	P1
P2	P3

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

P0 is computed at <u>full-1924x24</u> 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-<u>19x24 (</u>Parameters): 2 Adds: 8

 $\begin{array}{rl} \mbox{FORMAT OF P0's IJ:} & \mbox{Mantissa } 23\underline{19} \mbox{ Exp 4 for } I \underline{+ Sign} \\ & \mbox{Mantissa } 23\underline{19} \mbox{ Exp 4 for } J \underline{+ Sign} \end{array}$

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

Total number of bits : <u>1923*2 + 8*6 + 4*8 + 4*2 = 126 (rounded up on the bus to 128)</u>

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

Exhibit 2014 doc R400_Sequences.doc 37948 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C ***

PM-10/05/01-10:28-AM

AMD1044_0256817

ATI Ex. 2105 IPR2023-00922 Page 145 of 239

	ORIGINATE DATE	E EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 200	01 <u>4 September, 2015</u>	17	20 of 26	
clause 5) beca exporting prog cache where t	use the registers can yram before it is don he data will be once t	nust make sure that the p n be reused at this point. T e exporting data. Along w the vertex shader exports.	regram doesn't enter ALU clause 5 (it ca The sequencer must also make sure not to vith the position is exported a pointer to The storage needed to perform the posit s for the sprite size. It is going to be tal	to dealocate an the parameter tion export is at	
				•	Formatted: Bullets and Numbering
16. Real	time comman	nds			
need to add th register bus an be able to add should be able other is raster memory to 16 view support f	aree 16x128 memories and written by type 0 paress the reatime para a able to view them a ized with. Most overla x64 or 32x64 allowing or 16 vector-4 interpo the PA/sequencer ne	s (one for each of three ve ackets, and output to the t immeter memory as well as as two banks of 16 and di ay shaders will need 2 or g only two interpolated sc plants important (true only	to way for a command stream to write inter- trices x 16 interpolants). These will be m he parameter busses (the sequencer and the regular parameter store. For higher p o double buffering allowing one to be for 4 scalar coordinates, one option might b alars per cycle, the only problem I see w if we map microsoft's high priority stream specific mode where we need to addres	apped onto the d/or PA need to performance we aded, while the e to restrict the ith this is, if we to the realtime	
A					Formatted: Bullets and Numbering
<u>16.17. R</u> €					
PIXEL_MI VERTEX_ Vshader_1 Pshader_3 Pshader_4 PSHADEF PCNTLSH VCNTLSH VWRAP PWRAP PWRAP REG_ALL REG_ALL PARAM_M	REG_SIZE Wh N_SIZE Mir MIN_SIZE Mir MIN_SIZE Mir MIN_SIZE Mir dialustrian Grading alustrian Grading alustrin Grading	hat portion of the register finimal size of the register fill himal size of the register fill ht 8-12 bit pointers to the 1 ht 8-12 bit pointers to the 2 bit pointer for the pixel shades ap point for the pixel shades mber of registers to allocat	nader t rol program oft rol program der instruction store er instruction store ie for pixel shader programs ie for vertex shader programs ie h how parameters <u>maps in</u> the pixel sha	ram is located ram is located ram is located ram is located	
GEN_TEX CYL_WRA P_EXPOR V_EXPOR	([016] for AP[064] for RT nur RT nur	wich parameters do we not wich vertices parameters mber of exports for pixel sh mber of exports for vertex	eed to generate tex coords. (and channels (xyzw)) do we do the cyl w		
V_EXPOR	T_LOC Ver	el shaders) rtex shader exporting to R			
ARBITRA	TION policy poli	icy of the arbitration betwe	een vetexes and pixels		Formatted: Bullets and Numbering
<u>17.18. Int</u>	erfaces			*1	
17.1 18.1	External Interf	faces			
<u>17.1.118.</u>	1.1_PA/SC to R	E : IJ bus			
	us that sends the IJ in goes to the sequence		the top of each shader pipe. At the same	time the control	
Name	Dir	rection Bits	Description		
Exhibit 2014.docR400_Se	quencer.doc 37948 Bytes***		erence Copyright Notice on Cover Page	e©*** 09/04/15 12:51	
		<u>PM</u> 10/06/01_10::	28-AM		

ATI Ex. 2105 IPR2023-00922 Page 146 of 239

	ORIGINATE DATE	EDIT DA	TE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	<u>4 September,</u>			21 of 26	
IJs	PA→RE	E 6 <u>3</u> 4		prmation sent over 2 clocks		
Mask	PA→RE	1	Write	Mask		
This is the con	1.2 PA/SC to SEQ throl information sent to the der program on the sent p Direction	e sequencer in ord ixels.	er to cor	trol the IJ fifos and all other information	≁- needed to	Formatted: Bullets and Numbering
Write Mask	PA→SE			Write mask left to right		
RB_ID	PA→SE		RB id	for each quad sent 2 bits per quad		
LOD_CORREC	CT PA→SE	EQ(RE) 24	LOD	correction per quad (6 bits per quad)		
FVTX	PA→SE			oking vertex for flat shading		
PPTR0	PA→SE	<u></u>		re pointer for vertex 0		
PPRT1	PA→SE			re pointer for vertex 1		
PPTR2	PA→SE			re pointer for vertex 2		
E_OFF_VECT				of the vector		
DEALLOC	PA→SE			ocation token for the P Store		
STATE	PA→SE	· /		/constant pointer (6*3+3)		
VALID	PA→SE			bits for all pixels		
NULL	PA→SE			Primitive (for PC deallocation purposes)		
E OFF PRIM	PA→SE			Of the primitive		
FBFACE	PA→SE PA→SE			face = 1, back face = 0		
STIPPLE LINE				led line and Real time command need	to load toy	
			001 : 011 : 100 : 101 :	Normal Stippled line Real Time Line AA Point AA Sprite		
RTRn	SEQ→F	PA 1		the PA in n clocks		
RTS	PA-SE			eady to send data		
QuadX	PA→SE PA→SE			X address 10-2 bits per quad		
QuadX	PA→SE PA→SE		-	Y address 10-2 bits per quad		
Quaut	rn→3t	-w(RE) 400		audiess +++ 2 pits per quad		
	1.3 PA/SVGTC to			and the se	-	Formatted: Bullets and Numbering
Name Vertex indexes	Direction VGTPA			cription ters of indexes or HOS surface information		
EOF vector	VGT⊭A VGT→F	34/10		of the vector	1	
Inputs vert	VG1→r VGT→F			ormal 128 bits per vert		
mpars ven		<u>hen</u> 1		uble 256 bits per vert		
	1.4 VGTPA/SC to		Contr	ol Bus		Formatted: Bullets and Numbering
Name	Directio			ription		
STATE	VGTPA			er State (6*3+3 for constants)		
Write MaskVerl				n vertices are valid		
Inputs_vert	<u>VGT→S</u>	<u>SEQ 1</u>		rmal 128 bits per vert uble 256 bits per vert		
This informatio	on needs to be sent over 6	4 clocks.	1.90	nner an M. MINN JEWL L. MIN]	
17.1.5 18.1	1.5_CP to SEQ : C	onstant store	load		4-	Formatted: Bullets and Numbering
Name	Directio	on Bits	Desc	ription		
				· · · · · · · · · · · · · · · · · · ·		

Address of the block of 4 constants Constant Address CP→SEQ 8 Exhibit 2014 doc R400_Sequencer.doc 37948 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

AMD1044_0256819

	RIGINATE DATE	EDIT	DATE	R400 Sequencer Specification	PAGE	
24	September, 2001	4 Septemb	per, 2015	17	22 of 26	
Constant Data	CP→S		512	Data sent over 4 clocks		┭┘
Remap Address	CP→S	EQ	10	Remaping address write address		1
Remap Data pointer	· CP→S	EQ	8	Remaping pointer		
1 7.1.6 18.1.6	CP to SEQ : 7	exture <u>Fe</u>	<u>tch</u> Sta	ate store load		Formatted: Bullets and Numbering
Name	Direct	ion	Bits	Description		7
Constant Address	CP→S	EQ	8	Address of the block of 4 state constants		
Constant Data	CP→S		512	Data sent over 4 clocks		
Remap Address	CP→S		10	Remaping address write address		_
Remap Data pointer	- CP→S	EQ	8	Remaping pointer		1
<u>17.1.7</u> 18.1.7 Name	_CP to SEQ : C		ate sto Bits	re load Description		Formatted: Bullets and Numbering
	and what is the size			Description		
. ,	_MH to SEQ: Ir	-		Load		Formatted: Bullets and Numbering
Name	Direct	ion	Bits	Description		Т
Instruction address	MH→S		12	Instruction address		1
Instruction	MH→S	EQ	96	Instruction X times]
Control Instruction a			9	Pointer to the control instruction store		
Control Instruction	MH→S	EQ	32	Control Instruction X times		
17.1.9<u>18.1.9</u>	SP to RB : Pix	el read fr	om RE	S		
	Directi		Bits	Description]
Name Pixel Data <u>Export_da</u>	<u>ata</u> SP→R	В	25664	2 pixels (or 1/2 quad)a pair of 32 bits channel]
Pixel DataExport_da		В			first or second n=0 or 1), quad) from first or 0) quad 0-3 in	
Pixel DataExport_da ExportID ExportMask	$\frac{SP \rightarrow R}{SP \rightarrow R}$	B B	256 <u>64</u> 9 2	2 pixels (or ½ quad)a pair of 32 bits channed Ocvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot If export mask is 00 data is invalid	first or second n=0 or 1), quad) from first or 0) quad 0-3 in or Z (tt=2)	
Pixel DataExport_da ExportID ExportMask	<u>ita SP→R</u>	B B	256 <u>64</u> 9	2 pixels (or ½ quad)a pair of 32 bits channed Ocvvvvhqg: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqg: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot	first or second n=0 or 1), quad) from first or 0) quad 0-3 in or Z (tt=2)	
Pixel DataExport_da ExportID ExportMask ExportLast	$\frac{SP \rightarrow R}{SP \rightarrow R}$	B B B	256 <u>64</u> 9 2	2 pixels (or ½ quad)a pair of 32 bits channed Ocvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot If export mask is 00 data is invalid	first or second n=0 or 1), quad) from first or 0) quad 0-3 in or Z (tt=2)	
Pixel DataExport_da ExportID ExportMask ExportLast 18.1.10 SEQ	$\frac{SP \rightarrow R}{SP \rightarrow R}$	B B B B bl bus	256 <u>64</u> 9 2	2 pixels (or ½ quad)a pair of 32 bits channed Ocvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot If export mask is 00 data is invalid	first or second n=0 or 1), quad) from first or 0) quad 0-3 in or Z (tt=2)	
Pixel DataExport_da ExportID ExportMask ExportLast 18.1.10 SEQ Name	to RB : Control	B B B D b b b b b b b b b b b b b b b b	256 <u>64</u> 9 2 1	2 pixels (or ½ quad)a pair of 32 bits channed Ocvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot if export mask is 00 data is invalid Last export instruction of the clause Description 0: Pixel	first or second n=0 or 1), quad) from first or 0) quad 0-3 in or Z (tt=2)	
Pixel DataExport_da ExportID ExportMask ExportLast 18.1.10 SEQ Name Type	to RB : Control SEQ \rightarrow R	B B B B D I bus R B	25664 9 2 1 <u>Bits</u> 1	2 pixels (or ½ quad)a pair of 32 bits channed Ocvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot If export mask is 00 data is invalid Last export instruction of the clause Description 0: Pixel 1: Vertex	first or second n=0 or 1), quad) from first or 0) quad 0-3 in or Z (tt=2)	
Pixel DataExport_da ExportID ExportMask ExportLast 18.1.10 SEQ Name Type	to RB : Control Direction	B B B B D I bus R B	25664 9 2 1 <u>2</u> <u>1</u>	2-pixels (or ½-quad)a pair of 32 bits channed Ocvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot if export mask is 00 data is invalid Last export instruction of the clause Description 0: Pixel 1: Vertex 0: first interleaved clause	first or second n=0 or 1), quad) from first or 0) quad 0-3 in or Z (tt=2)	
Pixel DataExport_da ExportID ExportMask ExportLast	to RB : Control SEQ \rightarrow R	B B B B D D D D D D D D D D D D D D D D	25664 9 2 1 <u>Bits</u> 1	2 pixels (or ½ quad)a pair of 32 bits channed 0cvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f) 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot If export mask is 00 data is invalid Last export instruction of the clause Description 0: Pixel 1: Vertex 0: first interleaved clause 0 thru 16 parameters exported for vertexes 0 thru 16 parameters exported for vertexes	irst or second >=0 or 1), quad > from first or 0) quad 0-3 in or Z (tt=2) h 32 bit words.	• • • • • • • • • • • • • • • • •
Pixel DataExport_da ExportID ExportMask ExportLast 18.1.10 SEQ Name Type Interleaving Export_size	to RB : Control SEQ→ SEQ→ SEQ→	B B B B D D D D D D D D D D D D D D D D	25664 9 2 1 <u>Bits</u> 1 1	2 pixels (or ½ quad)a pair of 32 bits channed 0cvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot if export mask is 00 data is invalid Last export instruction of the clause 0: Pixel 1: Vertex 0: first interleaved clause 0 thru 16 parameters exported for vertexes 0 thru 16 parameters exported for vertexes (bbzs) 1-4 color buffers (bb), two compon component colors (s=1) with z (z=1) or withe	irst or second >=0 or 1), quad > from first or 0) quad 0-3 in or Z (tt=2) h 32 bit words.	• • • • • • • • • • • • • • • • •
Pixel DataExport_da ExportID ExportMask ExportLast 18.1.10 SEQ Name Type Interleaving Export_size	to RB : Control SEQ→ SEQ→	B B B B D D D D D D D D D D D D D D D D	25664 9 2 1 <u>Bits</u> 1 1	2 pixels (or ½ quad)a pair of 32 bits channed 0cvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f) 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot If export mask is 00 data is invalid Last export instruction of the clause Description 0: Pixel 1: Vertex 0: first interleaved clause 0 thru 16 parameters exported for vertexes 0 thru 16 parameters exported for vertexes	irst or second >=0 or 1), quad > from first or 0) quad 0-3 in or Z (tt=2) h 32 bit words.	• • • • • • • • • • • • • • • • •
Pixel DataExport_da ExportID ExportID ExportLast 18.1.10 SEQ Name Type Interleaving Export_size Valid Only one exporting	to RB : Control SEQ \rightarrow SEQ \rightarrow	B B B B B B B Con RB RB RB RB RB RB RB RB RB	25664 9 2 1 1 1 1 4 y given tir	2 pixels (or ½ quad)a pair of 32 bits channed 0cvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot If export mask is 00 data is invalid Last export instruction of the clause Description 0: Pixel 1: Vertex 0: first interleaved clause 1: second interleaved clause 0thru 16 parameters exported for vertexes (bbzs) 1-4 color buffers (bb), two component component colors (s=1) with z (z=1) or withe Data valid	irst or second >=0 or 1), quad > from first or 0) quad 0-3 in or Z (tt=2) h 32 bit words.	• • • • • • • • • • • • • • • • •
Pixel DataExport_da ExportID ExportID ExportLast 18.1.10 SEQ Name Type Interleaving Export_size Valid Only one exporting	to RB : Contro SEQ→ SEQ→ SEQ→ SEQ→ SEQ→	B B B B B B B Con RB RB RB RB RB RB RB RB RB	25664 9 2 1 1 1 1 4 y given tir	2 pixels (or ½ quad)a pair of 32 bits channed 0cvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot If export mask is 00 data is invalid Last export instruction of the clause Description 0: Pixel 1: Vertex 0: first interleaved clause 1: second interleaved clause 0thru 16 parameters exported for vertexes (bbzs) 1-4 color buffers (bb), two component component colors (s=1) with z (z=1) or withe Data valid	irst or second >=0 or 1), quad > from first or 0) quad 0-3 in or Z (tt=2) h 32 bit words.	Formatted: Bullets and Numbering
Pixel DataExport_da ExportID ExportMask ExportLast 18.1.10 SEQ Name Type Interleaving Export_size Valid Only one exporting	to RB : Control SEQ \rightarrow SEQ \rightarrow	B B B B B B B R B R B R B R B R B R B R	25664 9 2 1 1 1 1 4 y given tir	2 pixels (or ½ quad)a pair of 32 bits channed 0cvvvvhqq: Vertex data vvvv 0-15 from 1 clause (c=0 or 1), XY or ZW components (f 0-3 in the shader (qq= 0-3) 1cbbkttqq: Pixel data for buffer bb (0-3 second clause (0-1) killed or not (k=1 or the shader and data is RG (tt=0), BA (tt=1) Specifies whether to write low, high or bot If export mask is 00 data is invalid Last export instruction of the clause Description 0: Pixel 1: Vertex 0: first interleaved clause 1: second interleaved clause 0thru 16 parameters exported for vertexes (bbzs) 1-4 color buffers (bb), two component component colors (s=1) with z (z=1) or withe Data valid	irst or second >=0 or 1), quad > from first or 0) quad 0-3 in or Z (tt=2) h 32 bit words.	Formatted: Bullets and Numbering

ATI Ex. 2105 IPR2023-00922 Page 148 of 239

	ORIGINATE	DATE	EDIT	DATE		DOCUMENT-REV. NUM.	PAGE	
AU I			4 Septem	ber, 201	1517	GEN-CXXXXX-REVA	23 of 26	
vail size	-	RB→SEC		6	Size ava	ilable in output buffers (in 32bits incre	ements)	T
			1					
7.1.10<u>18</u>	.1.12_SP to	PA/SC	<u>RB</u> : Posi	ition r	eturn	bus	4	Formatted: Bullets and Numbering
ame		Direction			Descrip]
osition return arameter cach	ha pointar	SP→PAF				data or sprite size (per clock) where the data will be in the paramet	tor cacho for	
					each ve			
ositions then taken from the gisters are ta	16- <u>1_point sprite</u> he output buffer. aken until the ne	sizes. The Additionna t ALU clau	e storage use ally,if needec use where th	ed is of 6 I the edg ney are g	64x128 ge flags going to	erleaved on a 16 x 16 basis. We bits for position and 64x32 bits for s are packed into the bits of the sprii be available again. Thus the seque i in the next ALU clause.	sprite size, it te sizes.The	
7.1.1118	.1.13 Shade	er Engin	e to Text i	urøFet	tch Ur	nit Bus (Fast Bus)	4	Formatted: Bullets and Numbering
within each clocks. The data is read Four Quad	n of the sub-engin Read address as d 3 clocks after the I's worth of Textu	es repeatin sociated we Instruction	ng every 4 cl ⁄ith Quad 0 m n Start. ata may be ⁻	locks. Th nust be s written t	he regis sent 1 ck	ery clock. These are sourced from a c er file index to read must precede th ick after the Instruction Start signal is egister file every clock. These are c ster file index to write must accompa	he data by 2 sent, so that directed to a	
Data and In	ndex associated w	ith the Qua			ocks afte	r the Instruction Start signal is sent.		1
Tex_Read	Decister Index			Dires				
	Register maex	SEQ->SF	>	7		o Register files for reading TextureFe	tch Address	1
Tex_RegFil	le_Read_Data	SEQ->SF SP->TEX			Index in	to Register files for reading TextureFe #e <u>Fetch</u> Addresses read from the Reg		
Tex_RegFil			(2048 7	Index in 16 ∓ext		gister file	
Tex_RegFil Tex_Write_ 7.1.1218. nce every fou registers is tos. The sequ	le_Read_Data _Register_Index . <u>1.14</u> _Seque ur clock, the textus s ready or not. Th	SP->TEX SEQ->TE PRCET to refetch un is way the les the intr	Texture	2048 7 Fetch ne seque can upd constants	Index in 16 Texts Index in Data Unit I encer of late the	reFetch Addresses read from the Reg	gister file TextureFetch if the data in ation station	Formatted: Bullets and Numbering
Tex_RegFil Tex_Write_ 7.1.1218, nce every fou e registers is ios. The sequ gister file whe	le_Read_Data _Register_Index . <u>1.14</u> _Seque ur clock, the textus s ready or not. Th uencer also provid	SP->TEX SEQ->TE Procer to rrefetch un his way the des the intr exturefetch	Texture	2048 7 Fetch ne seque can upd constants	Index in 16 Textr Index ir Data Unit I encer of late the is for the	ire <u>Fetch</u> Addresses read from the Reg to Register file for write of returned = DUS (Slow Bus) n wich clause it is now working and i texture <u>fetch</u> counters for the reserve texture fetch to execute and the ad	gister file TextureFetch if the data in ation station	Formatted: Bullets and Numbering
Tex_RegFil Tex_Write_ 7.1.1218. nce every fou re registers is ios. The sequ gister file who Name	Ie_Read_Data Register_Index .1.14_Seque ur clock, the texte s ready or not. Th iencer also provid iere to write the te	SP->TEX SEQ->TE Encer to rrefetch un lis way the les the intr exturefetch Directior	Texture	2048 7 Fetch ne seque can upd constants	Index in 16 Texts Index ir Data Unit I encer of date the s for the Descrip	ireFetch Addresses read from the Register file for write of returned a bus (Slow Bus) n wich clause it is now working and i texture <u>fetch</u> counters for the reserven- texture fetch to execute and the ad	gister file TextureFetch if the data in ation station	Formatted: Bullets and Numbering
Tex_RegFil Tex_Write_ 7.1.1218. nce every fou e registers is os. The sequ gister file who Name Tex_Ready	Ie_Read_Data _Register_Index .1.14_Seque ur clock, the textus s ready or not. Th jencer also provid iere to write the te	SP->TEX SEQ->TE Procer to rrefetch un his way the des the intr exturefetch	Texture	2048 7 Fetch ne seque can upd constants Bits	Index in 16 Textr Index ir Data Unit I encer of late the is for the	ireFetch Addresses read from the Reg to Register file for write of returned - ous (Slow Bus) in wich clause it is now working and i texturefetch counters for the reserve- texture fetch to execute and the ad tion	gister file TextureFetch if the data in ation station	Formatted: Bullets and Numbering
Tex_RegFil Tex_Write_ 7.1.1218. nce every fou re registers is ios. The sequ gister file who Name	Ie_Read_Data _Register_Index .1.14_Seque ur clock, the textus s ready or not. Th jencer also provid iere to write the te	SP->TEX SEQ->TE PRCORT for the set of the set of the set of the set of the set of the set of the set of the set of the set of the set of the s	Texture Texture Texture tit sends to the sequencer ruction and c return data.	2048 7 Fetch ne seque can upd constants Bits 1 3	Index in 16 Textu Index in Data Unit I encer of late the is for the Descrip Data rea Clause	ireFetch Addresses read from the Reg to Register file for write of returned - ous (Slow Bus) in wich clause it is now working and i texturefetch counters for the reserve- texture fetch to execute and the ad tion	gister file TextureFetch if the data in ation station Idress in the	Formatted: Bullets and Numbering
Tex_RegFil Tex_Write_ 7.1.1218. ncce every fou e registers is ios. The sequ gister file who Name Tex_Ready Tex_Clause	Ie_Read_Data _Register_Index .1.14_Seque ur clock, the textus s ready or not. Th jencer also provid iere to write the te	SP->TEX SEQ->TE SEQ->TE Procer to refetch un is way the des the intr exturefetch Direction TEX-> SI TEX-> SI	Texture Toxture Toxture sequencer ruction and c return data. EQ EQ EX	2048 7 Fetch ne seque can upd constants Bits 1 3 10 12	Index in 16 Text Index in Data Unit i encer of late the s for the Descrip Data rea Clause i Texture	ireFetch Addresses read from the Register file for write of returned - ous (Slow Bus) n wich clause it is now working and i texturefetch counters for the reserva- texture fetch to execute and the address tion idy number	gister file TextureFetch if the data in ation station tdress in the 4 clocks	Formatted: Bullets and Numbering
Tex_RegFil Tex_Write_ 7.1.1218. ncc every fou re registers is ios. The sequingister file who Name Tex_Ready Tex_Clause Tex_Clause Tex_Inst EO_CLAUS	Ie_Read_Data _Register_Index .1.14_Seque ur clock, the text⊾ s ready or not. Th jencer also provid iere to write the te / e_Num	SP->TEX SEQ->TE SEQ->TE Procer to the second second sway the des the intra- texture fetch Direction TEX-> SI SEQ->TE SEQ->TE SEQ->TE	Texture T	2048 7 Fetch ne seque can upd constants Bits 1 3 10 12 1	Index in 16 Text Index in Data Unit I encer of date the is for the Descrip Data rea Clause Texture clocks Last ins	ireFetch Addresses read from the Register file for write of returned a construction of the clause it is now working and it texturefetch counters for the reservative fetch to execute and the addition and the addition address 10 bits sent over a fetch instruction address 12 bits ruction of the clause	gister file TextureFetch if the data in ation station tdress in the 4 clocks	Image: second
Tex_RegFil Tex_Write_ 7.1.1218. nce every fou re registers is tos. The seque register file who sigister file who <u>Name</u> Tex_Ready Tex_Clause Tex_cst Tex_Inst	Ie_Read_Data _Register_Index .1.14_Seque ur clock, the text⊾ s ready or not. Th jencer also provid iere to write the te / e_Num	SP->TEX SEQ->TE SEQ->TE Procer to the second second second second second second second second second second second TEX-> SI SEQ->TE SEQ->TE	Texture T	2048 7 Fetch ne seque can upd constants Bits 1 3 10 12 1	Index in 16 Text Index in Data Unit I encer of date the is for the Descrip Data rea Clause Texture clocks Last ins	ireFetch Addresses read from the Register file for write of returned - bus (Slow Bus) in wich clause it is now working and i texturefetch counters for the reserve texture fetch to execute and the ad tion ledy number Fetch state address 10 bits sent over 4 fFetch instruction address 12 bits	gister file TextureFetch if the data in ation station tdress in the 4 clocks	
Tex_RegFil Tex_Write_ 7.1.1218. ncce every fou re registers is ios. The seque register file who Name Tex_Ready Tex_Clause Tex_Clause Tex_Inst EO_CLAUS PHASE 8.19. Inter 9.20. Ex	Ie_Read_Data _Register_Index .1.14 Seque ur clock, the textu- s ready or not. The iencer also providence to write the ter e_Num SE cernal interf	SP->TEX SEQ->TE Procer to refetch un is way the des the intro- interval of the set of the intro- control of the set of the seq of t	Texture Texture it sends to the sequencer of ruction and c return data. EQ EQ EQ EX EX EX EX EX EX	2048 7 Fetch ne seque can upd constants Bits 1 3 10 12 1 1 1 1 1 1 1	Index in 16 Text Index in Data Unit I encer of late the s for the Descrip Data rec Clause Texture clocks Last ins Write ph	ireFetch Addresses read from the Register file for write of returned - bus (Slow Bus) in wich clause it is now working and i texturefetch counters for the reserven- texture fetch to execute and the ad tion indy number -etch state address 10 bits sent over 4 ff_etch instruction address 12 bits ruction of the clause ase signal	gister file TextureFetch if the data in ation station tdress in the 4 clocks	Formatted: Bullets and Numbering
Tex_RegFil Tex_RegFil Tex_Write_ 7.1.1218. ncce every foure registers is toos. The seque register file who wighter file who Name Tex_Ready Tex_Clause Tex_	IL_Read_Data _Register_Index .1.14_Seque ur clock, the textu- s ready or not. The iencer also providence to write the ter / e_Num SE cernal interf camples of 1.1_Sequence	SP->TEX SEQ->TE Procer to refetch un is way the des the intr exture fetch Direction TEX-> SI SEQ->TE	Texture Textu	2048 7 7 Fetch ne seque seque can upd seque constants seque 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Index in 16 Text Index in Data Unit I encer of Jate the s for the Descrip Data rea Clause I Texture clocks Last ins Write ph S r of Ve 32 bits/	InterFetch Addresses read from the Register file for write of returned a cours (Slow Bus) In wich clause it is now working and is texturefetch counters for the reservent at the address for the reservent at the address 10 bits sent over a freetch instruction address 12 bits ruction of the clause as a signal services for 2048 bit total) to the RE's N	gister file TextureFetch if the data in ation station idress in the 4 clocks sent over 4	
Tex_RegFil Tex_RegFil Tex_Write_ 7.1.1218. ncc every foure registers is ios. The sequery register file who Name Tex_Ready Tex_Clause	ILE_Read_Data _Register_Index .1.14 Seque ur clock, the textu- s ready or not. The second or not. The iencer also providence to write the ter- rest or write the ter- rest of the ter- second interference of the ter- camples of 1.1 Sequence a vector of 64 ver- pointer as well as was allocated in	SP->TEX SEQ->TE Procer to refetch un is way the les the intr exture fetch Direction TEX-> SI SEQ->TE	Texture Texture Texture it sends to the sequencer of return data. TeQ EQ EQ EX X Texture	2048 7 7 Fetch ne seque can upd can upd constants Bits 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Index in 16 Text Index in Data Unit I encer of late the s for the Descrip Data rea Clause I Texture clocks Last ins Write ph S r of Ve 32 bits/ along w med pos	InterFetch Addresses read from the Register file for write of returned a cours (Slow Bus) In wich clause it is now working and is texturefetch counters for the reservent at the address for the reservent at the address 10 bits sent over a freetch instruction address 12 bits ruction of the clause as a signal services for 2048 bit total) to the RE's N	gister file TextureFetch if the data in ation station ldress in the 4 clocks sent over 4	Formatted: Bullets and Numbering

ATI Ex. 2105 IPR2023-00922 Page 149 of 239

		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE]
		24 September, 2001	<u>4 September, 201517</u>		24 of 26	
	 shade The vertice the 	er program (using the Mi ertex program is assumed e SEQ then accesses the	+?) to be loaded when we red	ing the local state pointer (provided to		J
2.	at thisthe art	point the vector is remove	ed from the Vertex FIFO t a vector to be transforme	basically the Vertex FIFO always has d if the parameter cache is full unless		
3.	 the nu state p 	mber of GPRs required by pointer that came down wi	y the program is stored in	GPRs used by the program a local state register, which is accesse e has been allocated	ed using the	
4.	 the 64 RI RI RI the income 	Pretex indices are sent to F0 of SU0, SU1, SU2, and F1 of SU0, SU1, SU2, and F2 of SU0, SU1, SU2, and F3 of SU0, SU1, SU2, and dex is written to the least s	the 64 register files over SU3 is written the first cy SU3 is written the second SU3 is written the second SU3 is written the third cy SU3 is written the fourth significant 32 bits (floating	cle d cycle ycle		
5. 	texturefeto	<u>h</u> state machine 0, or TSI	M0 FIFO)	the first reservation station (the FIFO position cache and a register file bas		
6.	store		d fetches the instructions	for t exture<u>fetch</u> clause 0 from the glob start	al instruction	
7.		ions of texturefetch clause				
			-	e FIFO in front of ALU state machine	0, or ASM0	
	FIFO)	does not wait for request for the texture <u>fetch</u> data to the TU has written all the a count greater than zero	s made to the Texture <u>Fetc</u> the TU, which will write the to the register files, it	<u>h</u> Unit to complete; it passes the regis re data to the RF as it is received increments a counter that is associate te machine can go ahead start to exe	ter file write	
9.		epts the control packet (a rom the global instruction		ASM arbiter) and gets the instructions	for ALU	
10 			ssued by ASM0, then the etch state machine 1, or TS	control packet is passed to the next re SM1 FIFO)	eservation	
11	 position shared A para going th th param pa th lo th 	on can be exported in ALU d with all four shader pipe ameter cache pointer is all to be in the parameter cac ere is a position export FII e ASM arbiter will prevent teter data is exported in cl arameter data is sent to th e SEQ allocates storage in nger a need for the param	clause 3 (or 4?); the data s) back to the PA's position so sent along with the posi- che. FO in the SP that buffers p a packet from starting an ause 7 (as well as position e Parameter Cache over a n the Parameter Cache, an eters (it is told by the PA a packet from starting on	tion data. This tells to the PA where the exporting clause if the position export a data if it was not exported earlier) a dedicated bus and the SEQ deallocates that space wh	ous (which is ne data is o the PA FIFO is full nen there is no	

Exhibit 2014. doc R400_Sequencer.doc 37948 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** <u>PM40/66/01-10:28-AM</u>

AMD1044_0256822

An generation: Der Date A generation: Der CUMENT AREV. NUM. PAGE BENCXXXXXX.REV. PAGE BENCXXXXXX.REV. 26 rd.20 21 are then abader program has completed, the SEG with the up the GPRs so that they can be used by mother and program. Numerical Science Scie	10.000000000000000000000000000000000000	T	[1		
12. And the lander program has completed, the SEC will fine us the OPRs so that they can be used by another shader program (and the second of Pixels) • • • • • • • • • • • • • • • • • • •	M AN	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
 12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program. Speel shaders are loaded into the global instruction store by the CP A this point it is assumed that the pixel program is loaded into the global instruction store and thus ready to be read. The RES Fixel FPO is totade output to be output our quade with do abardiated walker The take pointer and the LOD correction bits are also placed in the Pixel FIFO SEQ anbitrates between Pixel FIFO and encound quade with do abardering duads by the detailed walker The the take pointer and the LOD correction bits are also placed in the Pixel FIFO SEQ anbitrates between Pixel FIFO and encound quade with do abardering pixel, which is accessed using the state pointer of CPRs required by the program. The number of CPRs required by the program is loaded into the first encounds on the register file has been allocated SEQ anticates a space in the SP register file to rail the GPRs used by the program. The number of CPRs required by the program is loaded atto to the SP register file has been allocated SEQ control packet for the vector and sends it to the first reservation station (the FIFO in front of takkuweletic balaet machine 0. or SIMO FIFO. To the take machine 0. or SIMO FIFO. To the take machine 0. or SIMO FIFO. To the take the data to the state pointer, register file base pointer, register file base pointer, register file base pointer. Take data machine 0. or SIMO FIFO. To the take the data to take the take to the register file base pointer. Take another 0. or SIMO FIFO. To the take the data to take the take to require the take to the register file base pointer. Take take machine 0. or action take the take to the register file. Throne more take		24 September, 2001		GEN-CXXXXX-REVA	25 of 26	
 40.1.22012 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 A this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read. 1. The RES Fixel FICD is loaded with the barycentic coordinates for pixel quades by the detailed waterer the state pointer and the LOD correction bits are also placed in the Pixel FIFO 3. SEQ aubitrates between Pixel FIEO and Vertex FIEO - when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIEO and Vertex FIEO - when there are no vertices pending OR there is no space left in the register file for vertices data to be sent to the shader utili space in the register, which is accessed using the state pointer of ORPs required by the program is loaded atto the SP register file over the RE_SP interface (which has a bandwith of 248 bits rycel), so there or tables a bus digrams for details. 3. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwith of 248 bits rycel), so thereoptable data bus digrams for details. 3. SEQ controls the transfer of Interpolated data to the SP register file base border, and the LOD correction bits a land of the register of base border and the LOD correction bits a land of the register file base border and the LOD correction bits a land data base and the register file base border for the LOD correction bits are allocated a cortex base and fetheres the instructions for tratsfere file for the register file base base data to th				GPRs so that they can be used by ar	nother	
 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. the RE's Pixel FIFO is loaded with the baryceritic coordinates for pixel quade by the defailed walker the Pixel FIFO is loaded with the baryceritic coordinates for pixel quade by the defailed walker the Pixel FIFO is loaded with the baryceritic coordinates for pixel quade by the defailed walker the Pixel FIFO and Vertex (FIFO and Vertex (FIFO and Vertex (FIFO and Vertex (FIFO) as selected SEC additionates between Pixel (FIFO and Vertex (FIFO) as selected SEC addition the register files in the GPRs used by the program the number of GPRs required by the program is stored in a local state register. Mich is accessed using the state pointer. SEC will not allow interpolated data to be sent to the shader until space in the register file has been allocated SEC will not allow interpolated data to be adaption to over the RE_SP Therdace (which has a bandwidth of 2048 bits/cycle). See interpolated data be adaptions for totals. SEC overtice a control packet for the vector and sends in the first increased for pixels is adaptione as quad address for example Turket set is a second for pixels is all instruction for a squad address for example Turkets in a space FIFO TISMO accepts the control packet and fetches the instructions for toxium@clot burk to be first in resolved in state machines a countor that is associated to the noxi reservation station watchise the second files, it increments a countor that is associated to the toxium@clot burket for the second to the counterest to acountor packet and fetches the instructions for ALU da		-				
At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read. In RE's Fixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker In the Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker In the Pixel FIFO is wide enough to source four quad's worth of barycentrics per crycle SEC arbitrates between Pixel FIFO and Vaters FIFO – when there are no vorcices pending OR there is no space list in the register file for vortices, the Pixel FIFO is selected SEC arbitrates between Pixel FIFO is well the OPRs used by the program • the number of OFRs required by the program is stored in a local state register, which is accessed using the state pointer • SEC will not allow interpolated data to be sent to the abader until space in the register file has been allocated SEC worth there is a separate to Pixel Pixel FIFO is selected • SEC will not allow interpolated data to be degrams for details. • SEC will not allow interpolated data to be degrams for details. • SEC will not allow interpolated data to be degrams for details. • In the Hart fibre is a separate set of reservation station (the FIFO in front of pixels is a separate FIFO. • TSW0 accepts the control packet and fetches the instructions for texture@etch clause 0 from the global instruction store and was for teacetaby the TSW and by the TSW and for the next reservation station (the FIFO in fort of ALU state machine 0, or ASM0 FIFO). • TSM0 was first elected by the TSW and by the FIFO in fort of ALU state machine 0, or ASM0 FIFO). • TSM0 was first elected by the TSW and by the TSW arbitro before it could state. • TSM0 was first elected by the TSW and by the TSW arbitro before files. It increments a counter that is associated with the ASM0 clause 0 from the global instruction store files. It nervered is associated with the ASM0 clause	<u>19.1.220.</u>	<u>1.2</u> Sequencer Con	trol of a Vector of P	ixels		
 2. the RE's Pitel FIPO is baded with the barycentric coordinates for pixel quads by the dealled walker the state pointer and the LOD correction bits are also placed in the Pixel FIPO the twee FIFO is wide enough to source four quad's worth of barycentrics per cycle 3. SEC arbitrates between Pixel FIPO and Vertex FIPO – when there are no vortices pending OR there is no space left in the register files 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 attempt of GPRs required by the program is stored in a local state register (lie has been allocated 5. SEC controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of SSB eniterpolated data bus diagrams for details. 5. SEC controls packet for the vector and sends it to the first reservation station (the FIFO in front of these welface). See interpolated data bus diagrams for details. 6. SEC control packet and state the state pointer, the register file base pointer, and the LOD correction bits. a) other informations (such as quad address for example) travels in a separate FIFO. 7. TSM0 accepts the control packet and fetches the instructions for leastwefield; clause 0 from the global instruction drate. 9. The control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO. 1. TSM0 does not wait for tourset(e) requests made to the Toskweffetch Unit to complex: it passes the register file write index to file base based based in the control packet is passed to the next reservation asaocidated with the ASM OFFO a counge are than zero indicates that the ALU state machines for ALU dause 0 from the global instruction store. 1. All instructions of ALU dause (later be targe elected by the ASM athier) and gets the instr	1. As with v	ertex shader programs, pi	ixel shaders are loaded in	to the global instruction store by th	e CP	
 the testate pointer and the LOC correction bits are also placed in the Pixel FIFO the Pixel FIFO is wide enough to source focus routan's worth of barycentritis per cycle 3. SEC 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 SEC will not allow interpolated data to be sent to the shader until space in the register file has been allocated SEC will not allow interpolated data to be sent to the shader until space in the register file has been allocated SEC will not allow interpolated data to be sent to the shader until space in the register file has been allocated SEC will not allow interpolated data to be sent to the shader until space in the register file has been allocated SEC will not allow interpolated data to the SP register file over the RE_SP interface (which has a bardwidth of 2048 bits/cycle). See interpolated data to the SP register file over the RE_SP interface (which has a bardwidth of 2048 bits/cycle). See interpolated data be sentations for vertices and for pixels the control packet to activate accession and sends it to the first reservation station (the FIFO in front of FIFO in front of ALU state machine 0, or ASM0 fifO accession that the ALU state machine due to the register file with endex for the toxturefieth quaues that the ALU state machine can go ahaed and pop the FIFO accust for ALU duaue the control packet is passed to the next reservation station (the FIFO in front of ALU state machine can go ahaed and pop the FIFO and stat to execute the ALU state machine is associated with the ASM0 FIFO, accust gravest match at the register file with endex for the toxturefieth quaues that the ALU state machine and pop the FIFO and stat to execute the ALU state machine can go ahaed and pop the FIFO and stat to execute the ALU state machine can go ahaed and pop	At this	point it is assumed that the	e pixel program is loaded int	o the instruction store and thus ready	to be read.	
 In the register files for vertices, the Pixel FIPO is selected 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 units allow interpolated data to be sent to the shader until space in the register file has been allocated SEQ controls the transfer of interpolated data to due so the RE_SP interface (which has a bandwidth O246 bits/cyclo). See interpolated data but diagrams for details. SEQ controls the transfer of reservation stations/arbiters/state machines for vertices and for pixels the control packet contributes the state pointer. TSMM accepts the control packet and tetches the instructions for taxture/etch data but diagrams for details. To stature field, clause 0 are issued by TSM the control packet and tetches the instructions for taxture/etch data but diagrams. all instructions of taxture/etch clause 0 bre model. TSMM doce not wait for taxture/etch data to the Taxture-fielch Unit to complete it passes the register file write index for the toxture/etch data to the register files, it is received or one that U shates the taxture/etch data to the register files, it is received and and pop the FIFO and start to execute the ALU clause 0 the register files, and pop the FIFO and start to execute the ALU clause for the data by the resolutions of ALU clause 0 are inserved from stations until all clauses have been executed the control packet continues to travel down the path of reservation stations until all clauses have been executed the control packet continues to travel down the path of reservation stations until all clauses have been executed the control packet continues to travel down the path of reservation stations until all clauses have been executed the control packet continues to travel	 the state 	ate pointer and the LOD cor	rection bits are also placed	in the Pixel FIF0		
 the number of OPRs 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 shador until space in the register file has been allocated SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2046 bits/ycle). See interpolated data bus diagrams for details. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of texture/effch, state machine, or TSM0 FIFO) no te that there is a separate set of reservation stations/arbiter/state machines for vertices and for pixels all other informations (such as quad address for example) travels in a separate FIFO TSM0 scepts the control packet and fetches the instructions for texture/effch, clause 0 from the global instruction store of the seture/effch data to the TV which will write the data for 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 ASM of FIFO and state machine (the LU clause) for the ture/effch requests made to the Texture/effch. The register files, it increments a counter that is associated with the ASM of FICO and such associate machine cong or ahead and pop the FIFO and state machine (Lu clause) the control packet (after being selected by the SM arbiter) and gets the instructions for ALU clause 0 are issued by ASM0, then the control packet is esseed to the set of reservation station will all clauses have been executed the scort the data for a particular clause 0 are indicates is passed to the next reservation station (the FIFO and state machine), or ASM arbiter and the data for a particular clause to the restrue/effch machine cange and and pop the FIFO and state exclusite to the ASM arbiter) and gets the instructions for ALU clause 0 are issued by ASM0				re are no vertices pending OR there is	no space	
 5. SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bills/cycle). See interpolated data bus diagrams for details. 6. SEQ controls a control packet for the vector and sends it to the first reservation station (the FIFO in front of the three reservation stations/arbiters/state machines for vertices and for pixels and thor informations (such as quad address for example) travels in a separate SFIFO 7. TSM0 excepts the control packet and fetches the instructions for texture/etch clause 0 from the global instruction of texture/etch clause 0 are issued by TSM0 8. the control packet and fetches the instructions for texture/etch clause 0 from the global instruction of texture/etch clause 0 are issued by TSM0 9. TSM0 does not wait for texture/etch data to the TFO in front of ALU state machine 0, or ASM0 FFFO). • TSM0 does not wait for texture/etch data to the Texture/etch unit to complete; it passes the register file write nail the data for a paticular clause to the register file, sit increments a counter that is associated with the ASM0 FFFO, a count greater than zero indicates that the ALU state machine can go anead and pop the FIPO and stat to execute the ALU clause 10. ASM0 accepts the control packet (farer 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 FIPO in front of texture/etch mission and ASM7 if the output FIPO) 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 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. 20. Execute State and the adviters will operate ahead of time so that	 the nustate 	umber of GPRs required by t pointer	the program is stored in a lo	ocal state register, which is accessed u	-	
 6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of <u>instructedicth</u> state machine 0, or TSM0 FIFO) a. note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels b. 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 exturefetch clause 0 from the global instruction store all other like vite index is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO) b. TSM0 does not wait for texturefetch requests made to the Tox.tureFetch Unit to complete; it passes the register file with index for the texturefetch data to the register files, it increments a counter that is associated with the ASM0 FIFO, acount 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 instructions for ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of lexturefetch state machine 1, or TSM1 FIFO) 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 lexturefetch) state machine 1, or TSM1 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. H. 32.0. 1.3. Notes 14. the state machines and arbitrers will operate ahead of times on that they will be able to immediately start the real threads or stall. 	5. SEQ cont	rols the transfer of interpola	ted data to the SP register f	ile over the RE_SP interface (which ha		
 store TSM0 was first selected by the TSM arbiter before it could start all instructions of texture<u>fetch</u> clause 0 are issued by TSM0 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<u>fetch</u> requests made to the Texture<u>Fetch</u> Unit to complete; it passes the register file write index for the texture<u>fetch</u> at 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 ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instructions store 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<u>fetch</u> state machine 1, or TSM1 FIFO) the control packet continues to travel down the path of reservation stations until all clauses have been executed picel 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 after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program the state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall.	 6. SEQ constant texture fetting note t the constant texture 	structs a control packet for th <u>ch</u> state machine 0, or TSM hat there is a separate set o ntrol packet contains the sta	ne vector and sends it to the D FIFO) of reservation stations/arbite ate pointer, the register file	first reservation station (the FIFO in first reservation station (the FIFO in firs/state machines for vertices and for base pointer, and the LOD correction t	pixels	
 8. all instructions of texturefetch 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 texturefetch requests made to the TextureFetch Unit to complete; it passes the register file write index for the texturefetch 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 file, 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 texturefetch 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 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. 	store				nstruction	
 9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASMO FIFO). • TSM0 does not wait for texture/etch requests made to the Texture/Fetch Unit to complete; it passes the register file write index for the texture/etch 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 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of texture/etch 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 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. 		-				
 TSM0 does not wait for texture<u>fetch</u> requests made to the Texture<u>Fetch</u> Unit to complete; it passes the register file write index for the texture<u>fetch</u> 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<u>fetch</u> 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 19.1.320.1.3_Notes 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. 			-	IFO in front of ALU state machine 0, o	r ASM0	
 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 texturefetch 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 19.1.320.1.3 Notes 4. the state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall. 	 TSMC registe once i associ 	er file write index for the text the TU has written all the da iated with the ASM0 FIFO; a	ture <u>fetch</u> data to the TU, wh ta for a particular clause to a count greater than zero in	ich will write the data to the RF as it is the register files, it increments a count dicates that the ALU state machine ca	received ter that is	
 station (the FIFO in front of texturefetch 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 19.1.320.1.3 Notes 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. 				M arbiter) and gets the instructions for	ALU	
 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 after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program <u>19.1.320.1.3</u> Notes the state machines and arbitrers will operate ahead of time so that they will be able to immediately start the real threads or stall. Exhibit 2014 docR400_Sequence: doc 37648 Bytes*** @ ATI Confidential. Reference Copyright Notice on Cover Page @ *** gequints 12.5; 					vation	
shader program <u>19.1.3 20.1.3</u> Notes 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. <u>Exhibit 2014 doc R400_Sequences.doc</u> 37948 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** <u>gg0//15 12.51</u>	● pixel o ● it	data is exported in the last A is sent to an output FIFO w	LU clause (clause 7) here it will be picked up by	the render backend	executed	
 19.1.3 20.1.3 Notes 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. Exhibit 2014 docR400_Sequences.doc 37948 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** g904/16 12.5; 			ted, the SEQ will free up the	GPRs so that they can be used by ar	nother	
threads or stall.	19.1.3 20.	1.3_Notes			*-	Formatted: Bullets and Numbering
			operate ahead of time so tha	at they will be able to immediately start	the real	
	Exhibit 2014.docR400_S	squencer.doc 37948 Bytes*** ⓒ AT		Copyright Notice on Cover Page © 🕯	*** <u>09/04/15 12:51</u>	

ATI Ex. 2105 IPR2023-00922 Page 151 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September, 2001	4 September, 201517		26 of 26	

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

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<u>fetch</u> 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.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.

Exhibit 2014.doc.R400_Sequencer.doc 37948 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15/12/51

AMD1044_0256824

Formatted: Bullets and Numbering

ATI Ex. 2105 IPR2023-00922 Page 152 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	1 of 27
Author:	Laurent Lefebvre			
Issue To:		Copy No:		
issue io.		Сору №.		
	R400 S	equencer Spe	cification	
		SEQ		
		Version 0. <u>9</u> 8		
rec		cted uses of the block. It a	cer block (SEQ). It provides an ove lso describes the block interfaces,	
Document L		erforce\r400\arch\doc\gfx\RE) Sequencer Specification APPROVALS	\R400_Sequencer.doc Signature/Date	
	Name/Dept		Signatule/Date	
Remarks:				
	ANTIALLY DETRIME	NTAL TO THE INTE	NFORMATION THAT CO REST OF ATI TECHNOL E OR DISCLOSURE.	
work created i unpublished w confidential, pi	n 2001. The use of this cop vork. The copyright notice roprietary information and tra	yright notice is intended to is not an admission that ade secrets of ATI. No part	rial in this document constitutes an provide notice that ATI owns a cop publication has occurred. This w of this document may be used, re sion of ATI Technologies Inc."	oyright in this ork contains
Exhibit 2015.docR400_	Sequencer.doc 39737 Bytes*** ⓒ AT	1 Confidential. Reference C	Copyright Notice on Cover Page © ≝	*** <u>09/04/15 12:52</u>

ATI 2015 LG v. ATI IPR2015-00325

AMD1044_0256825

ATI Ex. 2105 IPR2023-00922 Page 153 of 239

AP	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 201519		2 of 27

Table Of Contents

1.	OVERVIEW43
1.1	Top Level Block Diagram
1.2	Data Flow graph
1.3	Control Graph
2.	INTERPOLATED DATA BUS
3.	INSTRUCTION STORE
4.	SEQUENCER INSTRUCTIONS
5.	CONSTANT STORE
6.	LOOPING AND BRANCHES
6.1	The controlling state
6.2	The Control Flow Program
6.3	Data dependant predicate instructions1614
6.4	Register file indexing
7.	REGISTER FILE ALLOCATION
8.	FETCH ARBITRATION
9.	ALU ARBITRATION
10.	HANDLING STALLS
11.	CONTENT OF THE RESERVATION STATION
FIFO	S191716
12.	THE OUTPUT FILE
13.	IJ FORMAT
14.	THE PARAMETER CACHE
15.	VERTEX POSITION EXPORTING201817
16.	REAL TIME COMMANDS201817
17.	REGISTERS
18.	INTERFACES
18.1	
18.	1.1 PA/SC to RE : IJ bus
18.	1.2 PA/SC to SEQ : IJ Control bus.221918

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 Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001

18.1.3	VGT to RE : Vertex Bus
18.1.4	VGT to SEQ : Vertex Control Bus
1015	<u>2220</u> 19
18.1.5	CP to SEQ : Constant store load 222019
18.1.6	CP to SEQ : Fetch State store load
	222019
18.1.7	CP to SEQ : Control State store load
	<u>2320</u> 20
18.1.8	MH to SEQ: Instruction store Load
	232020
18.1.9	SP to RB : Pixel read from RBs232020
18.1.10	SEQ to RB : Control bus232020
18.1.11	RB to SEQ : Output file control.232120
18.1.12	SP to RB : Position return bus. 232120
18.1.13	Shader Engine to Fetch Unit Bus (Fast
Bus)	242121
18.1.14	Sequencer to Fetch Unit bus (Slow
Bus)	242121
19. INTERN	AL INTERFACES242221
20. EXAMP	LES OF PROGRAM EXECUTIONS
242221	
20.1.1	Sequencer Control of a Vector of
Vertices	242221
20.1.2	Sequencer Control of a Vector of
Pixels	262322
20.1.3	Notes

First draft.

21.

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. Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed

Exhibit 2015.docR400_Sequencer.doc 39737 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:52

AMD1044_0256826

ATI Ex. 2105 IPR2023-00922 Page 154 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	3 of 27	
	1	the cor	nditional execute or jump. Added	debug	-
		registers	<u>>.</u>		
Exhibit 2015.docR400_	Sequencer.doc 39737 Bytes*** ⓒ A	TI Confidential. Reference (PM10/19/01-09:67-AM10/05/01-10-28-	Copyright Notice on Cover Page © ∘ ₩	09/04/15 12:5	2

ATI Ex. 2105 IPR2023-00922 Page 155 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
4 September, 2001	4 September, 201549		4 of 27
24	September, 2001		September, 2001 <u>4 September, 201549</u>

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 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 instruction thus there is only one sequencer for the whole chip.

Exhibit 2015 doc R400_Sequencer.doc 39737 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15 12:52

AMD1044_0256828

ATI Ex. 2105 IPR2023-00922 Page 156 of 239

PROTECTIVE ORDER MATERIAL

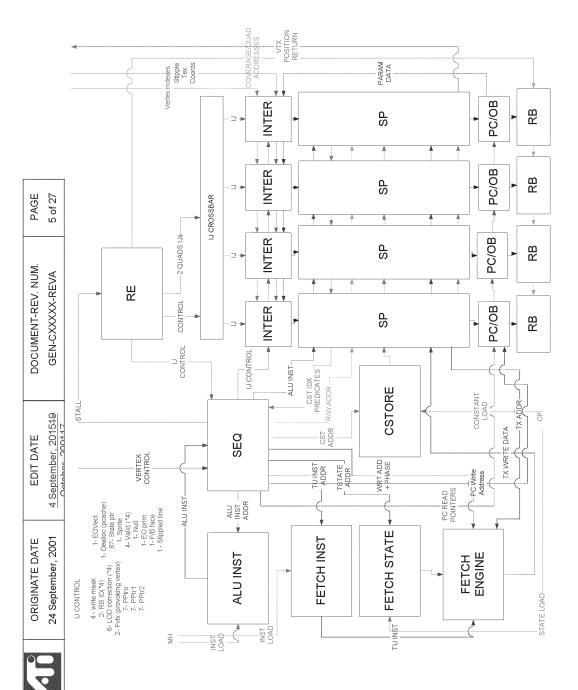


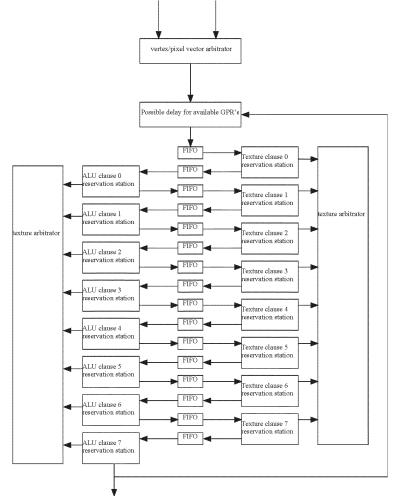
Exhibit 2015.doopR440_Sequenceer doe 39737 Bytes 444
Confidential. Reference Copyright Notice on Cover Page and 444 00004115 1252 PM14048001 40557 AM14045601 + 10;28 AM

AMD1044_0256829

ATI Ex. 2105 IPR2023-00922 Page 157 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		6 of 27

1.1 Top Level Block Diagram



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.

Exhibit 2015.docR400_Sequencer.doc 39737 Bytes**** S ATI Confidential. Reference Copyright Notice on Cover Page S *** 09/04/15.12:52

AMD1044_0256830

ATI Ex. 2105 IPR2023-00922 Page 158 of 239

					_0000000
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	7 of 27	

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 34, it can export the position if the position is ready. So the arbitrer must prevent ALU clause 34 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.

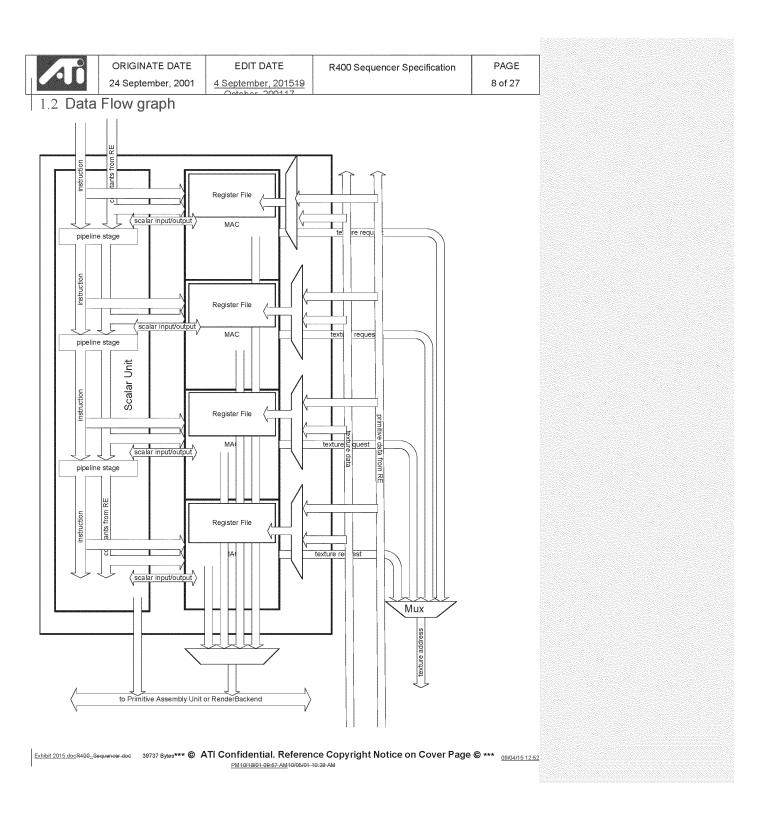
All other level process in the same way until the packet finally reaches the last ALU machine (8). On completion of the level 8 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register - a condition that 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.

Exhibit 2015 docR400_Sequences.doc 39737 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 9974/15 12:52 PM10/18/01-09:57-AM10/05/01-10:28-AM

AMD1044_0256831

ATI Ex. 2105 IPR2023-00922 Page 159 of 239

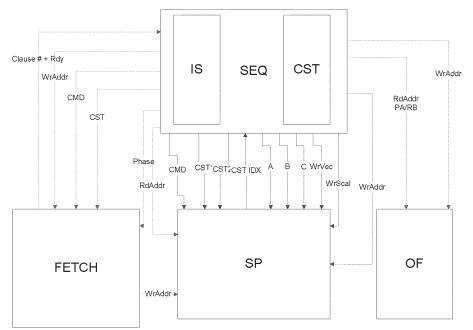


ATI Ex. 2105 IPR2023-00922 Page 160 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	9 of 27

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.

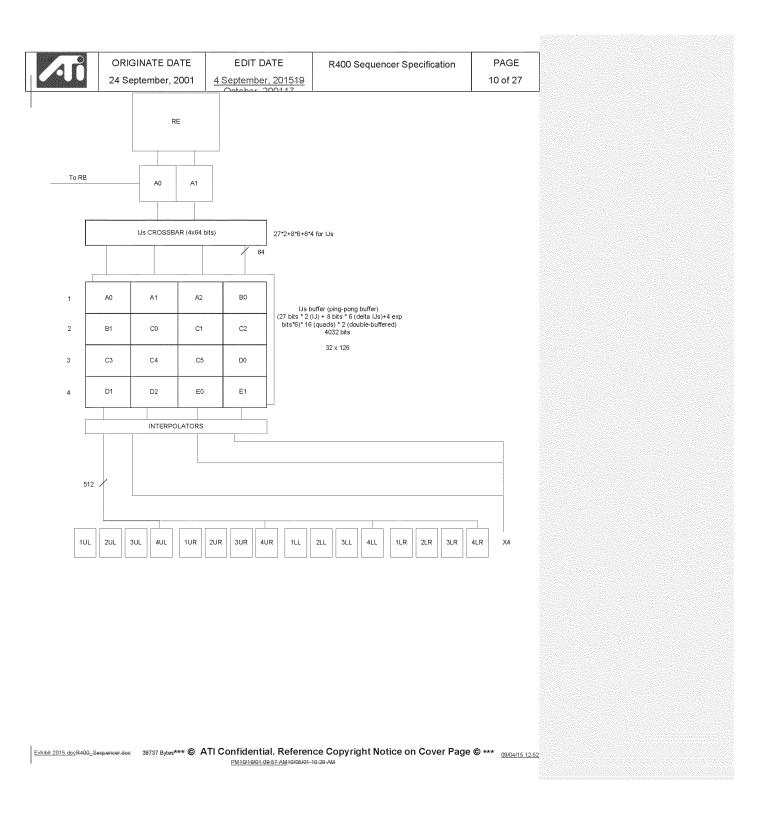
2. Interpolated data bus

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

Exhibit 2015.doc R400_Sequencer.doc 39737 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:52

AMD1044_0256833

ATI Ex. 2105 IPR2023-00922 Page 161 of 239



ATI Ex. 2105 IPR2023-00922 Page 162 of 239 PROTECTIVE ORDER MATERIAL

	T19	51 48-	52- 55	56-55 56-<	80 c <	
	T18	35 33 <	30 gg <	40 ⁴ < 4	> 44 - 47	
	T17	> 1 0 − 0 − 0	23 ² 0 <	24- 27	31 ²⁸ <	
	T16	> ⁰ .<	> 4-7	8-11	⇒ 4 5 − 5	
	T15			Ш	Ш Т	
ЗЕ 127	T14				ß	
PAGE 11 of 27	T13		CO	G	8	
	T12				BO	Σ.
DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	11	5	6			
OCUMENT-REV. NUN GEN-CXXXX-REVA	T10	ប៊	C4	C5		
EN-CX	19 1	B1				
Õ Ö	Т8	AO	A1	A2		
OPI	1			Ш	Ē	
ATE 20154	T6				â	
EDIT DATE 4 September, 201549	T5		0 C	ស	8	
4 Sep	Т4				BO	0
ш <u>Б</u>	T3	5	6			0
E DAT	12	ទ	C4	C5		
ORIGINATE DATE 24 September, 2001	F	В				
0RI 24 S	10	AO	A1	A2		
Ę		SP0	SP1	SP2	SP3	

AMD1044_0256835

ATI Ex. 2105 IPR2023-00922 Page 163 of 239

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	Contraction of the
	24 September, 2001	4 September, 201519		12 of 27	distantial.

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:

Exhibit 2015.docR400_Sequencer.doc 39737 Bytes*** S ATI Confidential. Reference Copyright Notice on Cover Page S ***

AMD1044_0256836

ATI Ex. 2105 IPR2023-00922 Page 164 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201549	GEN-CXXXXX-REVA	13 of 27	
D 1 1055 0	1				

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

Exhibit 2015 docR400_Sequencer.doc 39737 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

PM10/18/01-09:67-AM10/05/01-10:28-AM

AMD1044_0256837

ATI Ex. 2105 IPR2023-00922 Page 165 of 239

a î	ORIGINATE DATE		R400 Sequencer Specification	PAGE		
M	24 September, 200	1 <u>4 September, 201519</u> October, 200117		14 of 27		
Execute up to	4k instructions at the	pecified address in the instru	iction memory.			
			L			
		Conditionnal_Execute_or				
47	46 42		4 1 3 4			
			33			
			3221			
			20 12			
			11 0 <u>41 0</u>			
Addressing	00010		Booleans			
-			Condition			
		Jump address				
			Instruction_count			
		Exe	AddressRESERVED			

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.

Exhibit 2015. doc R400_Sequencer.doc 39737 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 00/04/15 12:52

AMD1044_0256838

ATI Ex. 2105 IPR2023-00922 Page 166 of 239

	1 1	ORIGINATE DATE 24 September, 2001		EDIT DATE <u>4 September, 201519</u> Otobor, 200117		DOCUMENT-REV. NUM. GEN-CXXXXX-REVA			PAGE 15 of 27
	Conditionnal Execute								
	47	46 42	41 34	3	3 3	32 244	203 12		11 0
Ad	dressing	00011	Boolean addr	ress Cond	dition RI	ESERVED	Instruction_count	Exe	c Address

If the specified boolean (8 bits can address 256 booleans) meets the specified condition then execute the specified instructions (up to <u>4k</u>542 instructions)

		Condition	nal_Execute_	Predicates			
47	46 42	41 38	37	36 241	203 12	11 0	
Addressing	00100	Predicate vector	Condition	RESERVED	Instruction_count	Exec Address	

Check the <u>AND/</u>OR of all current predicate bits. If <u>AND/</u>OR matches the condition execute the specified number of instructions.

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

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.

		L	.oop_End	
47	46 42	41 16	15 4	3 0
	00111	RESERVED	Start address	Loop ID
Addressing				

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	4112	11 0
	01000	RESERVED	Address
Addressing			

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

		Return	
47	46 42	41 0]
	01001	RESERVED	1
Addressing			

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.

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

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

	End_of_Clause					
47	46 42	41 0				
	01011	RESERVED				
Addressing						

Exhibit 2015.docR400_Sequencer.doc 39737 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 00/04/15 12:62

AMD1044_0256839

ATI Ex. 2105 IPR2023-00922 Page 167 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		16 of 27

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<u>and set de debug</u> registers. The sequencer will keep a-<u>two</u> loop indexes values<u>of 17 bits</u>:

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 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

 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?}

Exhibit 2015.docR4400_Sequencer.doc 38737 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:52
PM10/18/01 08:57 AM140/05/01-10:28-AM4

AMD1044_0256840

ATI Ex. 2105 IPR2023-00922 Page 168 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	17 of 27	
6.4 Regis	ster file indexing			-) 	
data created in		use it into an ALU clause. T	dex into the register file in order to he instruction will include the base		
	Bit 7 Bit 6 0 0 0 1 1 0 1 1	'absolute register' 'relative register' 'previous vector' 'previous scalar'			
			the case of a relative register read ew address that we give to the shad		
<u>7. HOS s</u>	surfaces			*-	Formatted: Bullets and Numbering
they have to d		hey can also export position	Y. If they want to export to the para in clause 3. The buffer they want t		
	ster file allocation			*-	Formatted: Bullets and Numbering
VERTEX_REG	6_SIZE for pixels.				
xhibit 2015.docR400_Se	quancar.doc 39737 Bytes*** ⓒ AT	'l Confidential. Reference С <u>Рм10/19/01-06:67-АМ</u> 10/05/01-10-28-А	opyright Notice on Cover Page @ ਅ) *** <u>09/04/15 12:52</u>	

ATI Ex. 2105 IPR2023-00922 Page 169 of 239

Aî	ORIGINATE DATE 24 September, 2001	EDIT DATE	R400 Sequencer Specification	PAGE 18 of 27

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

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.

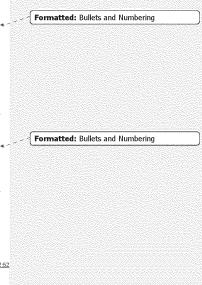
The arbitrator will not wait for the fetches to return prior to selecting another clause for execution. The fetch pipe will be able to handle up to X(?) in flight fetches and thus there can be a fair number of active clauses waiting for their fetch return data.

9.10. ALU Arbitration

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.

Exhibit 2015.docR400_Sequencer.doc 39737 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15 12:52



AMD1044_0256842

ATI Ex. 2105 IPR2023-00922 Page 170 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	19 of 27	

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.

11.12. Content of the reservation station FIFOs

3-21 bits of Render State 6-77 bits for the base address of the instruction <u>GPRs</u>store, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, <u>quad address and 1 bit to specify if the</u> <u>vector is of pixels or vertices</u>.-Every other information (such as the coverage mask, <u>quad address</u>, etc.) is put in a <u>FIFO and is retrieved when the quad exits the shader pipe to enter in the output file buffer</u>. 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.

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).

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).

Exhibit 2015 doc R400_Sequencer.doc 39737 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15 12:52

AMD1044_0256843

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

ATI Ex. 2105 IPR2023-00922 Page 171 of 239

	ORIGINATE DATE	EDIT DATE	R400	Sequencer	Specification	PAGE	
	24 September, 2001	4 September, 201519				20 of 27	
$\Delta 01I = I(1) -$	-I(0)	L Ostobor 200117					
$\Delta 01J = J(1)$	-J(0)						
$\Delta 02I = I(2)$	-I(0)		P0	P1			
$\Delta 02J = J(2)$	-J(0)				-		
$\Delta 03I = I(3)$	-I(0)						
$\Delta 03J = J(3)$	-J(0)		P2	P3			
P0 = C + I(0)))*(A-C)+J(0)*(B-C)	$\cdot C)$					
$P1 = P0 + \Delta 0$	$D1I^*(A-C) + \Delta 01J^*(A-C) + \Delta 01J^$	B-C)					
$P2 = P0 + \Delta t$	$02I^*(A-C) + \Delta 02J^*(A-C) + \Delta 02J^$	(B-C)					
$P3 = P0 + \Delta 0$	$03I^*(A-C) + \Delta 03J^*(A-C) + \Delta 03J^$	B-C)					
	ed at 19x24 mantissa prec f the image was seen usin		omputed a	t 8X24 manti	ssa precision. S	o far no visual	
	l Precision): 2 duced precision): 6 24 (Parameters): 2						
FORMAT OF	P0's IJ: Mantissa 19 Ex Mantissa 19 Ex						
FORMAT of D	eltas (x3):Mantissa 8 Exp Mantissa 8 Exp						
Total number	of bits : 19*2 + 8*6 + 4*8 +	+ 4*2 = 126					
1416 Th		- la				4-	Formatted: Bullets and Numbering
	ne parameter ca						
	r cache is where the ver jine will make it so that all					ories (1R/1W).	Formatted: Bullets and Numbering
15. 16. Ve	ertex position ex	porting				4-	· · · · · · · · · · · · · · · · · · ·
so at clause 8 data will be o	or 5) the vertex shader ca if not done at clause 4. A nce the vertex shader ex the position and 64x32 me	long with the position is ports. The storage need	exported a ed to perf	orm the position	le parameter ca tion export is a	che where the t least 64x128	
16. 17. Re	eal time commar	nds				*-	Formatted: Bullets and Numbering
need to add th register bus an be able to add should be able other is raster memory to 16 view support f	e to use the parameter me ree 16x128 memories (or nd written by type 0 packe lress the reatime paramet e able to view them as to ized with. Most overlay si x64 or 32x64 allowing on for 16 vector-4 interpolants the PA/sequencer need to stead of 16.	he for each of three vertices, and output to the the er memory as well as the vo banks of 16 and do do haders will need 2 or 4 s ly two interpolated scalars is important (true only if w	ces x 16 in parameter regular p ouble buff calar coor rs per cycl /e map mic	terpolants). ⁻ r busses (the arameter sto fering allowin rdinates, one le, the only p crosoft's high	These will be ma sequencer and, re. For higher pe g one to be loa option might be roblem I see with priority stream	pped onto the for PA need to erformance we ded, while the to restrict the th this is, if we to the realtime	
Exhibit 2015.docR400_Se	equencer.doc 39737 Bytes*** ⓒ 🗸	ATI Confidential. Refere PM10/18/01-08:57-AM10/06/0		right Notice	on Cover Page	© *** <u>09/04/15 12:52</u>	

ATI Ex. 2105 IPR2023-00922 Page 172 of 239

ORIGINATE DATE 24 September, 2001 EDIT DATE 24 September, 2001 EDIT DATE 24 September, 2001 DOCUMENT-REV. NUM. EDIT-COURCE PAGE 21 of 27 7.18. Registers Dynamic allocation (bicelvents) of the register file in reserved for vertices (static allocation only) What Rev Ref Rev Ref Rev								
Construct Construct Construct Construct 2.7.18. Registers Dynamic allocation (pixel/vertex) of the register file on or off Vertex_FES Provember 2012 VERTEX_FES Dynamic allocation (pixel/vertex) of the register file is reacred for vertices (static allocation only) Vertex_FLINK_SIZE Minimal size of the register file is vertex portion (dynamic only) Vehader_fetch[11:0](7:0) eight 12 bit pointers to the location where each clauses control program is located Pehader adit[11:0](7:0) eight 12 bit pointers to the location where each clauses control program is located Pehader adit[11:0](7:0) eight 12 bit pointers to the location where each clauses control program is located Pehader adit[11:0](7:0) eight 12 bit pointers to the location where each clauses control program is located Pehader adit[11:0](7:0) eight 12 bit pointers to the location where each clauses control program is located PWRAP wrap point for the pixel shader instruction store WWRAP wrap point for the pixel shader instruction store VEXPORTE transmeter mask to allocate for profers thader for projenting the each clauses control program is located EQN_TEX_CO_NRAP wrap point for the pixel shader instruction store VEXPORTE transmeter mask to allocate for proferab hader for projenta		ORIGINATE	DATE	EDIT DATE		DOCUMENT-REV. NUM.	PAGE	
-7.18. Registers DVNAMIC_REG VERTEX_REG_SIZE VPRTEX_REG_SIZE VPRTEX_INS_SIZE Wimal size of the register file is reserved for vertices (static allocation only) WERTEX_INS_SIZE Wimal size of the register file vertices portion (dynamic only) VPRTEX_INS_SIZE Wimal size of the register file vertice portion (dynamic only) VERTEX_INS_SIZE Wimal size of the register file vertice portion (dynamic only) VPRTEX_INS_SIZE Wimal size of the register file vertice shader portion is located Pehader_ferch110[7:0] Pehader_ferch100[7:0] PARAM_IMSCR.18] mumber of registers to allocate for price shader WWRAP wap point for the vertex shader instruction store WRAP wap point for the posel shader VVT_WRAP wap point point bort vertex shader instruction store WRAP with parameters call to allocate for price shader programs REG_ALLCC_PIX number of exports for grach call size.40 hore cyl with parameters call to the cyl with parameter make is a block of the price shader programs VEXPORTEI number of exports for grach dause. All numbers relate to the output buffer and what is it exporting. VEXPORTEI number of e		24 September	r, 2001	4 September, 20	1519	GEN-CXXXXX-REVA	21 of 27	Formatted: Bullets and Numbering
OPENANC REG Opnamic allocation (pixel/vertex) of the register file on or off. VERTEX_INE_SEE What point of the register file is reserved for vertices (static allocation only) Vertex_INE_SEE What point of the register file screenved for vertices (static allocation only) Vertex_INE_SEE What point of the register file screenved for vertices (static allocation only) Vertex_INE_SEE What point of the pointers to the location where each clauses control program is located Pehader_letch[11:0][7:0] eight 12 bit pointers to the location where each clauses control program is located Detader_letch[11:0][7:0] eight 12 bit pointers to the location where each clauses control program is located Detader_letch[11:0][7:0] eight 12 bit pointers to the location where each clauses control program is located Detader_letch[11:0][7:0] eight 12 bit pointers to the location where each clauses control program is located Detader_letch[11:0][7:0] eight 12 bit pointers to the location where each clauses control program is located Detader_letch[11:0][7:0] eight 12 bit pointers to the location where each clauses control program is located Detader_letch[11:0][7:0] eight 12 bit pointers to the location where each clauses control program is located Detader_letch[11:0][7:0] eight 12 bit pointers to the location where each clauses control program is located <td>17-18 Re</td> <td>aisters</td> <td></td> <td>Clotobor (1011</td> <td>2/ 1</td> <td></td> <td>*</td> <td></td>	17-18 Re	aisters		Clotobor (1011	2/ 1		*	
CST_SIZE_P Size of the constant store for pixels CST_SIZE_V Size of the constant store for vertexes 9. 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 8.20. Interfaces Formatted: Bullets and Numbering 8.120.1 External Interfaces 8.1.1.20.1.1 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 Iame Direction Bits Description Is PA-RE 63 I J information sent over 2 clocks	DYNAMIC VERTEX_F PIXEL_MIN VERTEX_M Vshader_fe Vshader_fe Pshader_al PSHADER VSHADER SAA	REG REG_SIZE J_SIZE J_SIZE J[11:0][7:0] Jtto[[7:0] Jtto[[7:0] Jtto[[7:0] DC_PIX DC_VERT ASK[016] JR[016] DR[064] T[8] T[8] DC T[8] DCDC DC DCD	What por Minimal s Minimal s eight 12 b eight 12 b eight 12 b base poir wrap poir number o paramete wich para for wich p for wich p 00000.(al Vertex sh policy of f Wich clau 000001 : 000001 : 000001 : 000000 (1)0000 (1)0000	allocation (pixel/ver tion of the register f ize of the register f ize of the register f ize of the register fi it pointers to the loo it pointers to the loo it pointers to the loo it pointers to the loo the for the vertex shat the for the vertex shat the tor the vertex sorts but for V EXP so the number of in ader exporting to the Not exporting (or evertes Exporting position (: Exporting BA : Exporting Z	tex) of the le is rese le's pixel le's verte> cation wh cation wh cation wh cation wh der instruct te for pixet te for vert ow param our aud sh eed to get annels (xy r pixel sh shader for <u>ORTI71 th</u> therpolate <u>B or the F</u> sen vetex: <u>ne output</u> (<u>porting o</u> <u>1</u>) <u>2</u>)	rved for vertices (static allocation of portion (dynamic only) (portion (dynamic only) ere each clauses control program ere each clauses control program eters maps in the pixel shader aded nerate tex coords. zw)) do we do the cyl wrapping. ader <i>re</i> each clause. All numbers relate f tan can relate to the PC if Exports[d parameters for pixel shaders) PCACHE es and pixels buffer and what is it exporting. nly to the PC)	is located is located is located is located	
CST_SIZE V Size of the constant store for vertexes 9. DEBUG registers Formatted: Bullets and Numbering 8.20. Interfaces Formatted: Bullets and Numbering 8.1.120.1.1 PA/SC to RE : 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 ame Direction 8 PARE 63 IJ information sent over 2 clocks					rting linea	ar to memory not to Frame Buffer.		
9. DEBUG registers PROB ADDR instruction address where the first problem occurred PROB COUNT number of problems encountered during the execution of the program 8.20. Interfaces 8.120.1 External Interfaces 8.1.120.1.1 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 Iame Direction Bits Description Is PA→RE 63 IJ information sent over 2 clocks			Size of th	e constant store for	pixels			
9. DEBUG registers PROB_ADDR instruction address where the first problem occurred PROB_COUNT number of problems encountered during the execution of the program 8.20. Interfaces 8.120.1 External Interfaces 8.1.120.1.1 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 ame Direction Bits Description s PA→RE 63 IJ information sent over 2 clocks	CST_SIZE	<u>V</u>	Size of th	e constant store for	vertexes			
PROB_ADDR instruction address where the first problem occurred PROB_COUNT number of problems encountered during the execution of the program &.20. Interfaces Formatted: Bullets and Numbering 8.120.1_External Interfaces Formatted: Bullets and Numbering 8.1.120.1.1_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 ame Direction Bits Description s PA-RE 63 IJ information sent over 2 clocks								Formatted: Bullets and Numbering
PROB_COUNT number of problems encountered during the execution of the program 8:20. Interfaces * 8:120.1 External Interfaces * 8:1.120.1.1 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 * ame Direction Bits Description is PA→RE 63 IJ information sent over 2 clocks	9. DEBU	IG register	S				*	
8.20. Interfaces 8.120.1 External Interfaces 8.1.120.1.1 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 ame Direction Is PA→RE 63 IJ information sent over 2 clocks	PROB	ADDR	instructio	n address where the	e first prol	blem occurred		
8.20. Interfaces 8.120.1 External Interfaces 8.1.120.1.1 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 ame Direction Is PA→RE 63 IJ information sent over 2 clocks	PROB	COUNT	number c	f problems encount	ered duri	ng the execution of the program		
8.120.1 External Interfaces 8.1.120.1.1 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 Iame Direction Bits Description Is PA→RE 63 IJ information sent over 2 clocks	8-20 Inte	orfaces					4	
8.1.120.1.1 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 ame Direction Bits Description is PA→RE 63 IJ information sent over 2 clocks	<u>0.20. III</u>	<u>enaces</u>						
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 ame Direction Bits Description is PA→RE 63 IJ information sent over 2 clocks	8.1 20.1_E	External Int	terface	S				
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 Image Direction Bits Description Is PA→RE 63 IJ information sent over 2 clocks	<u>8.1.1</u> 20 1] PA/SC to	RE: I	l bus				
information goes to the sequencer ame Direction Bits Description is PA→RE 63 IJ information sent over 2 clocks					the top o	f each shader pipe. At the same tin	ne the control	
s PA→RE 63 IJ information sent over 2 clocks								
	aan		FA→RE	1	white wa	lon		
				Confidential D-f		opuright Notice on Onues Dans	S	E Contraction of the second seco
and a ATI Confidential Reference Convicts Nation on Conversion and	bit 2015.docR400_Seq	uencer.doc 39737 Byte	s*** © AI				09/04/15 12:5	4
Ibit 2015.docR400_Sequencer.doc 39737 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:52 PM1001901.10.97.744.100501.10.97.444 PM1001901.10.97.444				I MILLI CONTRACT THE				
bit 2015 doc R400_Sequencer.doc 39737 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:52 PM10010901 09:57 AM10/06/01-19:28 AM								

ATI Ex. 2105 IPR2023-00922 Page 173 of 239

18.1.220.1.2 PA/ This is the control inform execute a shader program Write Mask RB_ID LOD_CORRECT FVTX PPTR0 PPRT1 PPTR2 E_OFF_VECTOR DEALLOC STATE VALID	SC to SEQ : IJ Cont ation sent to the sequence	Bits 4 8 24 2	7	22 of 27 *- Formatted: Bullets and Numbering needed to
A state of the second of the s	SC to SEQ : IJ Cont ation sent to the sequence in on the sent pixels. Direction PA \rightarrow SEQ(RE) PA \rightarrow SEQ(RE) PA \rightarrow SEQ(RE) PA \rightarrow SEQ(RE) PA \rightarrow SEQ(RE) PA \rightarrow SEQ(RE)	bits Bits Bits 4 8 24 2	to control the IJ fifos and all other information Description Quad Write mask left to right RB id for each quad sent 2 bits per quad	*
Arrise sthe control inform execute a shader program Write Mask RB_ID	ation sent to the sequence n on the sent pixels. PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE)	er in order Bits 4 8 24 2	to control the IJ fifos and all other information Description Quad Write mask left to right RB id for each quad sent 2 bits per quad	needed to
Avecute a shader progra Name Write Mask RB_ID _OD_CORRECT =VTX =VTX =PTR0 =PRT1 =PTR2 =_OFF_VECTOR DEALLOC STATE /ALID	n on the sent pixels. $\begin{array}{c} \hline \textbf{Direction} \\ PA \rightarrow SEQ(RE) \\ \end{array}$	Bits 4 8 24 2	Description Quad Write mask left to right RB id for each quad sent 2 bits per quad	
Name Write Mask RB_ID _OD_CORRECT ~VTX PPTR0 PPR1 PPR2 E_OFF_VECTOR DEALLOC STATE VALID	$\begin{array}{c} \hline \textbf{Direction} \\ PA \rightarrow SEQ(RE) \end{array}$	4 8 24 2	Quad Write mask left to right RB id for each quad sent 2 bits per quad	
Write Mask RB_ID LOD_CORRECT FVTX PPTR0 PPRT1 PPTR2 E_OFF_VECTOR DEALLOC STATE VALID	$\begin{array}{c} PA \rightarrow SEQ(RE) \\ PA \rightarrow SEQ(RE) \end{array}$	4 8 24 2	Quad Write mask left to right RB id for each quad sent 2 bits per quad	
FVTX PPTR0 PPTR1 PPTR2 E_OFF_VECTOR DEALLOC STATE VALID	PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE)	8 24 2	RB id for each quad sent 2 bits per quad	
RB_ID LOD_CORRECT FVTX PPTR0 PPRT1 PPTR2 E_OFF_VECTOR DEALLOC STATE VALID NULL	PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE)	24 2		
FVTX PPTR0 PPTR1 PPTR2 E_OFF_VECTOR DEALLOC STATE VALID	PA→SEQ(RE) PA→SEQ(RE) PA→SEQ(RE)	2	() () Correction per duad (6 bits per duad)	
PPTR0 PPRT1 PPTR2 E_OFF_VECTOR DEALLOC STATE VALID	PA→SEQ(RE) PA→SEQ(RE)			
PPRT1 PPTR2 E_OFF_VECTOR DEALLOC STATE VALID	PA→SEQ(RE)	1 1 1	Provoking vertex for flat shading P Store pointer for vertex 0	
PPTR2 E_OFF_VECTOR DEALLOC STATE VALID	`	11	P Store pointer for vertex 1	
E_OFF_VECTOR DEALLOC STATE VALID		11	P Store pointer for vertex 1	
DEALLOC STATE VALID	PA→SEQ(RE)	1	End of the vector	
STATE VALID	PA→SEQ(RE)	1	Deallocation token for the P Store	
	PA→SEQ(RE)	21	State/constant pointer (6*3+3)	
NULLI	PA→SEQ(RE)	16	Valid bits for all pixels	
	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 cords from alternate buffer	to load tex
			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 QuadY	PA→SEQ(RE) PA→SEQ(RE)	8	Quad X address 2 bits per quad Quad Y address 2 bits per quad	
Guau I	FA→SEQ(RE)	0	Guau Tauriess 2 bits per quad	
				Formatted: Bullets and Numbering
18.1.3 20.1.3_VG	to RE : Vertex Bu	IS		• • • • • • • • • • • • • • • • • • •
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	
STATE	VGT→SEQ	21	1: double 256 bits per vert Render State (6*3+3 for constants)	
VIAL			Trender State (0 STS IOI CONStants)	
101110		-		Formatted: Bullets and Numbering
18.1.4VGT to SE(: Vertex Control E	sus		*
This information needs to	be sent over 64 clocks			
				Formatted: Bullets and Numbering
	to SEQ : Constant	store l	oad	4- <u> </u>
18.1.5 <u>20.1.4</u> CP		Rite	Description	
	Direction		Address of the block of 4 constants	
Name	Direction CP→SEQ	6		
	Direction CP→SEQ CP→SEQ	8 512	Data sent over 4 clocks	
Name Constant Address Constant Data Remap Address	CP→SEQ			
Name Constant Address Constant Data Remap Address	CP→SEQ CP→SEQ	512	Data sent over 4 clocks	
Name Constant Address Constant Data Remap Address Remap Data pointer	CP→SEQ CP→SEQ CP→SEQ	512 10 8	Data sent over 4 clocks Remaping address write address Remaping pointer	Formatted: Bullets and Numbering
Constant Data Remap Address Remap Data pointer	CP→SEQ CP→SEQ CP→SEQ CP→SEQ	512 10 8	Data sent over 4 clocks Remaping address write address Remaping pointer	Formatted: Bullets and Numbering

AMD1044_0256846

	RIGINATE DATE	EDIT D	ATE	DOCUMENT-REV. NUM.	PAGE			
	September, 2001	4 Septembe	r <u>, 201519</u>	GEN-CXXXXX-REVA	23 of 27			
Constant Data	CP→SEC		200117 2 Data	sent over 4 clocks				
Remap Address	CP→SEC			aping address write address				
lemap Data pointer	CP→SEC	·		aping pointer				
· · ·	i		I					
<u>8.1.720.1.6</u>	CP to SEQ : Co	ntrol State	store lo	bad	4	(Form	natted: Bullets and Numberin	3
lame	Direction		its Desc	ription				
SSUE: How,Who a	nd what is the size of	this bus?}						
<u>8.1.820.1.7</u>	MH to SEQ: Ins	truction sto	re Load	1		Form	atted: Bullets and Numberin	1
Vame	Direction		its Desc	ription				
nstruction address	MH→SE			iction address				
struction	MH→SE			iction X times				
ontrol Instruction ad		~~~~~~		er to the control instruction store				
Control Instruction	MH→SE	ຊ 32	2 Contr	ol Instruction X times				
							attad. Dullata and Number-	<u></u>
<u>8.1.9</u> 20.1.8_9	SP to RB : Pixe	l read from	RBs		4	rorm	natted: Bullets and Numberin	3
lame	Direction	B	its Desc	ription				
xport_data	SP→RB	6,		of 32 bits channel values				
ExportID	SP→RB	9	i	/vhqq: Vertex data vvvv 0-15 from first	or second			
			claus 0-3 ir	e (c=0 or 1), XY or ZW components (h=0 the shader (qq= 0-3)	or 1), quad			
			secor	kttqq: Pixel data for buffer bb (0-3) fro nd clause (0-1) killed or not (k=1 or 0) qua	d 0-3 in the			
ExportMask	SP→RB	2	secor shade	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt=	d 0-3 in the 2)			
ExportMask		2	secor shade Spec expor	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid	d 0-3 in the 2)			
	SP→RB SP→RB	2	secor shade Spec expor	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I	d 0-3 in the 2)			
ExportLast	SP→RB SEQ to RB : C	ontrol bus	secor shade Spec expor Last e	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause	d 0-3 in the 2)	Form	natted: Bullets and Numberin	3
ExportLast 18.1.10 <u>20.1.9</u> Name	SP→RB	ontrol bus	secor shade Spec expor Last e	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause	d 0-3 in the 2)	Form	natted: Bullets and Numberin	3
ExportLast 18.1.10 <u>20.1.9</u> Name	SEQ to RB : C Direction SEQ→RI	ontrol bus	its Desc 0: Pix 1: Ve	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription rel	d 0-3 in the 2)	Form	natted: Bullets and Numberin	3
ExportLast 1 <u>8.1.1020.1.9</u> Name Type	SEQ to RB : C	ontrol bus	its Desc 0: Pix 1: Ve 0: firs	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifles whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription	d 0-3 in the 2)	Form	natted: Bullets and Numberin	3
ExportMask ExportLast 18.1.1020.1.9 Name Type nterleaving Export_size	SEQ to RB : C Direction SEQ→RI	ontrol bus B B B 1 B 1 B 1	its Desc 0: Pix 1: Ve 0: firs 1: se 0 thru (bbzs	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription tel rtex t interleaved clause cond interleaved clause or 16 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)	Form	natted: Bullets and Numberin	<u>)</u>
ExportLast 18.1.1020.1.9 Name Type Interleaving Export_size	SP→RB _SEQ to RB : C Direction SEQ→RI SEQ→RI SEQ→RI	I i B 3 1 3 1 3 4	its Desc 0: Pix 1: Ve 0: firs 1: se 0 thru (bbzs comp	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription rel tel t interleaved clause interleaved clause in 6 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without z	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)	Form	natted: Bullets and Numberin	9
ExportLast 18.1.1020.1.9 Name Type Interleaving Export_size	SP→RB _SEQ to RB : C Direction SEQ→RI SEQ→RI	I i B 3 1 3 1 3 4	its Desc 0: Pix 1: Ve 0: firs 1: se 0 thru (bbzs	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription rel tel t interleaved clause interleaved clause in 6 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without z	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)	Form	natted: Bullets and Numberin	9
ExportLast 18.1.1020.1.9 Name Type Interleaving Export_size Valid	SP→RB _SEQ to RB : C Direction SEQ→RI SEQ→RI SEQ→RI	1 ontrol bus 3 1 3 1 3 4 3 1	its Desc 0: Pix 1: Ve 0: Fix 1: Ve 0: firs 1: se 0 thru (bbzs comp Data	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription rel tel t interleaved clause interleaved clause in 6 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without z	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)	Form	natted: Bullets and Numberin	3
ExportLast 18.1.1020.1.9 Vame Type nterleaving Export_size /alid Dnly one exporting c	SP→RB _SEQ to RB : C Direction SEQ→RI SEQ→RI SEQ→RI SEQ→RI	Image: 1	its Desc 0: Pix 1: Ve 0: firs 1: set 0 thr. (bbzs comp Data	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription rel tel t interleaved clause interleaved clause in 6 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without z	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)		natted: Bullets and Numberin	
ExportLast 1.8.1.1020.1.9 Name Type Interleaving Export_size /alid Dnly one exporting c 1.8.1.1120.1.10	SP→RB _SEQ to RB : C Direction SEQ→RI SEQ→RI SEQ→RI SEQ→RI SEQ→RI	Image: control bus Image: c	its Desc Desc Desc Desc Desc Desc Desc Desc	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription rel tel t interleaved clause interleaved clause in 6 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without z	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)			
ixportLast 8.1.1020.1.9 Jame Type Interleaving ixport_size /alid Only one exporting c -8.1.1120.1.10 Jame	SP→RB SEQ to RB : C Direction SEQ→RI SEQ→RI SEQ→RI SEQ→RI SEQ→RI N SEQ→RI SEQ→RI SEQ→RI SEQ→RI SEQ→RI SEQ→RI SEQ→RI SEQ→RI	1 control bus 3 3 3 3 4 3 4 3 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 5 6 6 7 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	its Desc Desc Desc Desc Desc Desc Desc Desc	nd clause (0-1) killed or not (k=1 or 0) quarer and data is RG (tt=0), BA (tt=1) or Z (tt= files whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription real real rex t interleaved clause in 16 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without z valid	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)			
ixportLast 8.1.1020.1.9 lame ype interleaving ixport_size 'alid only one exporting c 8.1.1120.1.10 lame buff_Full	SP→RB SEQ to RB : C Direction SEQ→RI SEQ→RI SEQ→RI SEQ→RI SEQ→RI Lause (7) can be sele CRB to SEQ : Direction	I I <t< td=""><td>its Desco export Last of Last of Last of Desc 0: Pix 1: Ve 0: firs 1: set 0 thru (bbzs comp Data en time. Control its Desc Set if</td><td>nd clause (0-1) killed or not (k=1 or 0) quarer and data is RG (tt=0), BA (tt=1) or Z (tt= files whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription real real rex t interleaved clause in 16 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without z valid</td><td>d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)</td><td></td><td></td><td></td></t<>	its Desco export Last of Last of Last of Desc 0: Pix 1: Ve 0: firs 1: set 0 thru (bbzs comp Data en time. Control its Desc Set if	nd clause (0-1) killed or not (k=1 or 0) quarer and data is RG (tt=0), BA (tt=1) or Z (tt= files whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription real real rex t interleaved clause in 16 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without z valid	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)			
ExportLast 18.1.1020.1.9 Name Type Interleaving Export_size Valid Dnly one exporting c 18.1.1120.1.10 Name Buff_Full Avail_size 18.1.1220.1.11	SP→RB SEQ to RB : C Direction SEQ→RI SEQ SEQ SEQ SEQ SEQ SEQ SEQ SEQ	Image: control bus Image: c	its Desc 0: Pix 1: Ve 0: firs 1: set 0: firs 1: set 0: firs 1: set 0: firs 1: set 0: firs 1: set 0: firs 1: set 0: firs 1: set comp Data en time.	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription rel rtex t interleaved clause cond interleaved clause ord interleaved clause 16 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without z valid	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)	Form		9
ExportLast 18.1.1020.1.9 Name Type Interleaving Export_size /alid Dnly one exporting c 18.1.1120.1.10 Name Buff_Full Avail_size 18.1.1220.1.11 Vame	SP→RB SEQ to RB : C Direction SEQ→RI SEQ SEQ SEQ SEQ SEQ SEQ SEQ SEQ	Image: 1 control bus 1 3 3 3 3 3 4 3 3 4 3 4 3 4 3 1 5 1 5 1 5 1 5 1 1 1 1 1 1 1 1	its Desc export Last 0: Pix 1: Ve 0: firs 1: se 0: firs 1: se 1: s 1: s 1: se 1: s 1: s 1: s 1: se 1: s 1: s 1: se 1: se 1: se 1: se 1: se 1: se 1: se 1:	nd clause (0-1) killed or not (k=1 or 0) qualer and data is RG (tt=0), BA (tt=1) or Z (tt= files whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription rel rtex t interleaved clause 1 16 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without ; valid ription full available in output buffers (in 32bits increm ription	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)	Form	natted: Bullets and Numberin	9
ExportLast 18.1.1020.1.9 Vame Type Interleaving Export_size /alid Dnly one exporting c 18.1.1120.1.1(Vame Buff_Full Avail_size	SP→RB SEQ to RB : C Direction SEQ→RI SEQ→RI SEQ→RI SEQ→RI SEQ→RI SEQ→RI Direction RB→SEC RB→SEC C C C C C C C C C C C C C	Image: control bus Image: c	its Desc export Last of C Pix 1: Ve 0: firs 1: set 0 thru (bbzs comp Data en time. Control its Desc Set if Size m bus its Desc 28 Positi	nd clause (0-1) killed or not (k=1 or 0) qua er and data is RG (tt=0), BA (tt=1) or Z (tt= ifies whether to write low, high or both 32 I t mask is 00 data is invalid export instruction of the clause ription rel rtex t interleaved clause cond interleaved clause ord interleaved clause 16 parameters exported for vertexes (vvv) 1-4 color buffers (bb), two component ionent colors (s=1) with z (z=1) or without z valid	d 0-3 in the (2) bit words. If (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)	Form	natted: Bullets and Numberin	9

ATI Ex. 2105 IPR2023-00922 Page 175 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		24 of 27

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.

18.1.1320.1.12 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_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

18.1.1420.1.13 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	$TEX \rightarrow SEQ$	1	Data ready
Tex_Clause_Num	$TEX \rightarrow SEQ$	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

21.1.1 RE to SEQ : Vertex Control Bus

Name	Direction	Bits	Description
STATE	<u>VGT→SEQ</u>	21	Render State (6*3+3 for constants)
Vert counter	<u>VGT→SEQ</u>	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

20.1.122.1.1 Sequencer Control of a Vector of Vertices

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.

Exhibit 2015.doc.R400_Sequencer.doc 39737 Bytes***
C ATI Confidential. Reference Copyright Notice on Cover Page

09/04/15 12:52

PM10/18/01-09:57-AM10/05/01-16:28-AM

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044_0256848

ATI Ex. 2105 IPR2023-00922 Page 176 of 239

		ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE			
	<u>au</u>	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	25 of 27			
				he local state pointer (provided to a e program)	all			
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.	 the nu state p 	mber of GPRs required by pointer that came down with		al state register, which is accessed	I using the			
4.	 the 64 RI RI RI RI the inc 	vertex indices are sent to t F0 of SU0, SU1, SU2, and 3 F1 of SU0, SU1, SU2, and 3 F2 of SU0, SU1, SU2, and 3 F3 of SU0, SU1, SU2, and 3 Jex is written to the least sig	he 64 register files over 4 cyc SU3 is written the first cycle SU3 is written the second cyc SU3 is written the third cycle SU3 is written the fourth cycle gnificant 32 bits (floating poi	le	. ,			
5.	fetch state	machine 0, or TSM0 FIFO)	first reservation station (the FIFO ir ition cache and a register file base				
6.		· ·	fetches the instructions for fe SM arbiter before it could star	etch clause 0 from the global instru t	ction store			
7.	all instruct	ions of fetch clause 0 are is	sued by TSM0					
8.	 FIFO) TSM0 the fet once t 	does not wait for requests ch data to the TU, which wi he TU has written all the da a count greater than zero ir	made to the Fetch Unit to co Il write the data to the RF as ita to the register files, it incre	TO in front of ALU state machine 0, mplete; it passes the register file wr it is received ements a counter that is associated iachine can go ahead start to exect	ite index for with ASM0			
9.		epts the control packet (after om the global instruction st		l arbiter) and gets the instructions f	or ALU			
10			sued by ASM0, then the cont machine 1, or TSM1 FIFO)	rol packet is passed to the next res	ervation			
	 positionshared A paragoing the the the parameter parameter the the	In can be exported in ALU of d with all four shader pipes) ameter cache pointer is also to be in the parameter cache ere is a position export FIFC e ASM arbiter will prevent a leter data is exported in clau trameter data is sent to the e SEQ allocates storage in hger a need for the paramete e ASM arbiter will prevent a position is being exported) i	clause 3 (or 4?); the data (and back to the PA's position car sent along with the position packet from starting an expo use 7 (as well as position dat Parameter Cache over a dec the Parameter Cache, and th ters (it is told by the PA when packet from starting on ASM s full	data. This tells to the PA where the on data before it gets sent back to orting clause if the position export F a if it was not exported earlier) licated bus le SEQ deallocates that space whe n using a token). 17 if the parameter cache (or the po	s (which is e data is the PA TFO is full n there is no osition buffer			
12	after the si shader pro		ied, the SEQ will free up the	GPRs so that they can be used by	another			
<u>Exhibit</u>	2015.docR400_Se	nquencer.doc 39737 Bytes*** ⓒ AT	I Confidential. Reference C PM10/18/01-08:67-AM10/06/01-10:28-A	opyright Notice on Cover Page @ #	• *** <u>09/04/15 12:5</u>			

					DAGE	
	2 n	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
		24 September, 2001	4 September, 201519		26 of 27	Formatted: Bullets and Numbering
20) <u>.1.2</u> 22.	<u>1.2_</u> Sequencer Co	ontrol of a Vector of	f Pixels	*	
1.	As with v	ertex shader programs,	pixel shaders are loaded	I into the global instruction store by	the CP	
	At this	point it is assumed that t	he pixel program is loaded	into the instruction store and thus rea	dy to be read.	
2.	the RE's F	Pixel FIFO is loaded with t	he barycentric coordinates	for pixel quads by the detailed walker		
			orrection bits are also plac o source four quad's worth			
3		•	•	here are no vertices pending OR there	is no snace	
	left in the	register files for vertices, t	he Pixel FIFO is selected		13 110 304000	
4.	 the nu 		ster file for all the GPRs us y the program is stored in	ed by the program a local state register, which is accesse	d using the	
			data to be sent to the shac	ler until space in the register file has be	een allocated	
5.			lated data to the SP regist nterpolated data bus diagi	er file over the RE_SP interface (which ams for details.	ı has a	
6.				the first reservation station (the FIFO i	n front of	
		e machine 0, or TSM0 FIF hat there is a separate set	·	biters/state machines for vertices and f	or pixels	
		•		ile base pointer, and the LOD correction	on bits	
7				travels in a separate FIFO for fetch clause 0 from the global instru	uction store	
	 TSM0 	was first selected by the	TSM arbiter before it could			
8.		ions of fetch clause 0 are				
9.	the contro FIFO)	I packet is passed to the r	next reservation station (th	e FIFO in front of ALU state machine C), or ASM0	
				Unit to complete; it passes the register	file write	
				to the RF as it is received to the register files, it increments a co	unter that is	
			; a count greater than zero art to execute the ALU cla	o indicates that the ALU state machine use	can go	
10.		epts the control packet (a rom the global instruction		ASM arbiter) and gets the instructions	for ALU	
11.			issued by ASM0, then the ate machine 1, or TSM1 FI	control packet is passed to the next re FO)	servation	
12.			•	ation stations until all clauses have bee	en executed	
	-	lata is exported in the last is sent to an output FIFO	ALU clause (clause 7) where it will be picked up	by the render backend		
	• th	e ASM arbiter will prevent	a packet from starting on	ASM7 if the output FIFO is full		
13.	after the s shader pro		leted, the SEQ will free up	the GPRs so that they can be used by	another	
1	51.2.2 er pro					
20).1.322.	1.3_Notes			*-	Formatted: Bullets and Numbering
14.	the state r threads or		l operate ahead of time so	that they will be able to immediately s	tart the real	
15.	instruction	store base pointer does i		ne vector through the reservation static F pointer is different for all threads, bu ed via the state pointer		
Exhibit	2015.docR400_Se	equencer.doc 39737 Bytes*** ⓒ 🖡	TI Confidential. Referen	ce Copyright Notice on Cover Page	© *** 09/04/15 12:52	

ATI Ex. 2105 IPR2023-00922 Page 178 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE]
	24 September, 2001	4 September, 201519 October, 200117	GEN-CXXXXX-REVA	27 of 27	

 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.

Exhibit 2015.docR400_Sequencer.doc 39737 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:52

AMD1044_0256851

ATI Ex. 2105 IPR2023-00922 Page 179 of 239

Sector Contractor Contractor		rr		
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201526	GEN-CXXXXX-REVA	1 of 28
Author:	Laurent Lefebvre			
Issue To:		Copy No:		
	R400 S	equencer Spe	ecification	
		SEQ		
		Version 0.9 <u>1.0</u>		
req		cted uses of the block. It a	cer block (SEQ). It provides an ov so describes the block interfaces,	
Document L		erforce\r400\arch\doc\gfx\RE) Sequencer Specification APPROVALS	\R400_Sequencer.doc	
	Name/Dept		Signature/Date	
Remarks:				
	NTIALLY DETRIME	NTAL TO THE INTE	NFORMATION THAT CO REST OF ATI TECHNOL SE OR DISCLOSURE.	
work created in unpublished w confidential, pre	n 2001. The use of this cop ork. The copyright notice oprietary information and tra	yright notice is intended to is not an admission that ade secrets of ATI. No part	rial in this document constitutes ar provide notice that ATI owns a cop publication has occurred. This w of this document may be used, re sion of ATI Technologies Inc."	oyright in this ork contains
Exhibit 2016.docR400_\$	Sequencer.doc 42015 Bytes*** ⓒ AT	I Confidential. Reference C PM10/18/01-01-40 PM10/18/01-08:57 A	Copyright Notice on Cover Page € ™) *** <u>09/04/15 12:52</u>

ATI 2016 LG v. ATI IPR2015-00325

AMD1044_0256852

ATI Ex. 2105 IPR2023-00922 Page 180 of 239



Table Of Contents

	OVERVIEW4
	Top Level Block Diagram6
	Data Flow graph8
	Control Graph9
<u>2.</u>	NTERPOLATED DATA BUS9
	NSTRUCTION STORE12
4.	SEQUENCER INSTRUCTIONS12
5. (CONSTANT STORE12
	LOOPING AND BRANCHES 12
	The controlling state12
	The Control Flow Program13
	Data dependant predicate instructions16
	Register file indexing1746
<u>7.</u>	HOS SURFACES17
8.	REGISTER FILE ALLOCATION
9.	FETCH ARBITRATION18
10.	ALU ARBITRATION18
<u>11. </u>	HANDLING STALLS 19
	CONTENT OF THE RESERVATION STATION
FIFOS	19
13.	THE OUTPUT FILE
	J FORMAT19
15.	THE PARAMETER CACHE
16.	VERTEX POSITION EXPORTING
17.	REAL TIME COMMANDS
	REGISTERS
18.1	Control
18.2	Context
19.	DEBUG REGISTERS
20. 1	
20. 20.1	NTERFACES2221
	NTERFACES
20.1	NTERFACES 2221 External Interfaces 2221 I PA/SC to RE : IJ bus 2221
<u>20.1</u> 20.1.	NTERFACES 2221 External Interfaces 2221 I PA/SC to RE : IJ bus 2221 2 PA/SC to SEQ : IJ Control bus 2221
20.1 20.1.1 20.1.1 20.1.1 20.1.1	NTERFACES 2221 External Interfaces 2221 I PA/SC to RE : IJ bus 2221 2 PA/SC to SEQ : IJ Control bus 2221 3 VGT to RE : Vertex Bus 2322
20.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.4	NTERFACES 2221 External Interfaces 2221 I PA/SC to RE : IJ bus 2221 2 PA/SC to SEQ : IJ Control bus 2224 3 VGT to RE : Vertex Bus 2322 4 CP to SEQ : Constant store load 2322
20.1 20.1.1 20.1.1 20.1.1 20.1.1	NTERFACES 2221 External Interfaces 2221 I PA/SC to RE : IJ bus 2221 2 PA/SC to SEQ : IJ Control bus 2221 3 VGT to RE : Vertex Bus 2322 4 CP to SEQ : Constant store load 2322
20.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.4	NTERFACES 2221 External Interfaces 2221 I PA/SC to RE : IJ bus 2221 2 PA/SC to SEQ : IJ Control bus 2224 2 PA/SC to SEQ : IJ Control bus 2224 3 VGT to RE : Vertex Bus 2322 4 CP to SEQ : Constant store load 2322 5 CP to SEQ : Fetch State store load 2322 2322
20.1 20.1. 20.1. 20.1. 20.1. 20.1.	NTERFACES 2221 External Interfaces 2221 I PA/SC to RE : IJ bus 2221 2 PA/SC to SEQ : IJ Control bus 2224 2 PA/SC to SEQ : IJ Control bus 2224 3 VGT to RE : Vertex Bus 2322 4 CP to SEQ : Constant store load 2322 5 CP to SEQ : Fetch State store load 2322 2322
20.1 20.1.2 20.1.2 20.1.2 20.1.2 20.1.2 20.1.2 20.1.0	NTERFACES 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2221 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Fetch State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322
20.1 20.1. 20.1. 20.1. 20.1. 20.1.	NTERFACES 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2221 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Fetch State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322
20.1 20.1.2 20.1.2 20.1.2 20.1.2 20.1.2 20.1.2 20.1.0	NTERFACES 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2224 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 MH to SEQ: Instruction store Load 2322
<u>20.1</u> <u>20.1.</u> <u>20.1.</u> <u>20.1.</u> <u>20.1.</u> <u>20.1.</u> <u>20.1.</u> <u>20.1.</u>	NTERFACES 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2224 VGT to RE : Vertex Bus 2322 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 MH to SEQ: Instruction store Load 2322 SP to RB : Pixel read from RBs 23
$ \begin{array}{r} 20.1 \\ 20.1.2 \\ 20.1.2 \\ 20.1.2 \\ 20.1.4 \\ 20.1.4 \\ 20.1.4 \\ \hline 20.1.4 \\ $	NTERFACES 2221 External Interfaces 2221 I PA/SC to RE : IJ bus 2221 2 PA/SC to SEQ : IJ Control bus 2224 2 PA/SC to RE : IJ bus 2224 3 VGT to RE : Vertex Bus 2322 4 CP to SEQ : Constant store load 2322 5 CP to SEQ : Control State store load 2322 6 CP to SEQ : Instruction store Load 2322 7 MH to SEQ: Instruction store Load 2322 8 SP to RB : Pixel read from RBs
20.1 20.1.1 20.1.2 20.1.3 20.1.4 20.1.5 20.1.6 20.1.7 20.1.7 20.1.7 20.1.7 20.1.7 20.1.7	NTERFACES 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2221 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Instruction store Load 2322 MH to SEQ: Instruction store Load 2322 SP to RB : Pixel read from RBs 23 SEQ to RB : Control bus 2423 RB to SEQ : Output file control2423
20.1 20.1. 20.1.2 20.1.2 20.1.2 20.1.2 20.1.2 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.2 20.1.1 20.1.2 20.2 20	NTERFACES 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2221 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Instruction store Load 2322 MH to SEQ: Instruction store Load 2322 S SP to RB : Pixel read from RBs 23 S SEQ to RB : Control bus 2423 RB to SEQ : Output file control 2423 SP to RB : Position return bus 2523
20.1 20.1. 20.1.2 20.1.2 20.1.2 20.1.2 20.1.2 20.1.2 20.1.1 20.1	NTERFACES 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2221 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 MH to SEQ: Instruction store Load 2322 MH to SEQ: Instruction store Load 2322 Seq to RB : Pixel read from RBs 23 SeQ to RB : Control bus 2423 RB to SEQ : Output file control 2423 SP to RB : Position return bus 2523 Shader Engine to Fetch Unit Bus (Fast 2523
20.1 20.1. 20.1.2 20.1.2 20.1.2 20.1.2 20.1.2 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.1 20.1.2 20.1.1 20.1.2 20.2 20	NTERFACES 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2224 VGT to RE : Vertex Bus 2322 VGT to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 MH to SEQ: Instruction store Load 2322 SP to RB : Pixel read from RBs 233 SEQ to RB : Output file control2423 2423 RB to SEQ : Output file control2423 11 SP to RB : Position return bus 2523 Shader Engine to Fetch Unit Bus (Fast 2524
20.1 20.1. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 2	NTERFACES 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2224 VGT to RE : Vertex Bus 2322 VGT to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 MH to SEQ: Instruction store Load 2322 SP to RB : Pixel read from RBs 232 SP to RB : Octrol bus 2423 0 RB to SEQ : Control bus 2423 11 SP to RB : Position return bus 2523 12 Shader Engine to Fetch Unit Bus (Fast 2524 13 Sequencer to Fetch Unit bus (Slow 2524
20.1 20.1. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 2	NTERFACES 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2221 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Instruction store Load 2322 MH to SEQ: Instruction store Load 2322 MH to SEQ: Instruction store Load 2322 SP to RB : Pixel read from RBs23 SEQ to RB : Control bus 2423 RB to SEQ : Output file control2423 SP to RB : Position return bus2523 2524 Sequencer to Fetch Unit Bus (Fast 2524 3
20.1 20.1. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 2	NTERFACES 2221 External Interfaces 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2221 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Instruction store Load 2322 MH to SEQ: Instruction store Load 2322 SP to RB : Pixel read from RBs 23 SP to RB : Ontrol bus 2423 RB to SEQ : Output file control. 2423 SP to RB : Position return bus 2524 Sequencer to Fetch Unit Bus (Slow 2524 NTERNAL INTERFACES 2524
20.1 20.1. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 2	NTERFACES 2221 External Interfaces 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 VGT to RE : Vertex Bus 2322 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 MH to SEQ: Instruction store Load 2322 Sequencer RB : Pixel read from RBs 23 SEQ to RB : Control bus 2423 RB to SEQ : Output file control 2423 Shader Engine to Fetch Unit Bus (Fast 2524 Sequencer to Fetch Unit Bus (Slow 2524 NTERNAL INTERFACES 2524
20.1 20.1. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 2	NTERFACES 2221 External Interfaces 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2221 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Instruction store Load 2322 MH to SEQ: Instruction store Load 2322 SP to RB : Pixel read from RBs 23 SP to RB : Ontrol bus 2423 RB to SEQ : Output file control. 2423 SP to RB : Position return bus 2524 Sequencer to Fetch Unit Bus (Slow 2524 NTERNAL INTERFACES 2524
20.1 20.1. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 20.2. 2	NTERFACES 2221 External Interfaces 2221 External Interfaces 2221 PA/SC to RE : IJ bus 2221 PA/SC to SEQ : IJ Control bus 2224 VGT to RE : Vertex Bus 2322 CP to SEQ : Constant store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 CP to SEQ : Control State store load 2322 MH to SEQ: Instruction store Load 2322 SP to RB : Pixel read from RBs 233 SEQ to RB : Control bus 2423 RB to SEQ : Output file control 2423 RB to SEQ : Output file control 2423 Sequencer to Fetch Unit Bus (Fast 2524 Sequencer to Fetch Unit bus (Slow 2524 NTERNAL INTERFACES 2524 RE to SEQ : Vertex Control Bus 2524

22.1.1	Sequencer Control of a Vector of
Vertices	2624
22.1.2	Sequencer Control of a Vector of
Pixels	2725
22.1.3	Notes
	ISSUES
	VIEW
1.1 Top Le	vel Block Diagram5
	low graph7
1.2 Contro	I Graph
2 INTED	POLATED DATA BUS
	UCTION STORE 11
	ENCER INSTRUCTIONS
	TANT STORE
6. LOOP	ING AND BRANCHES
6.1 The co	ntrolling-state11
6.2 The Co	ontrol Flow Program12
6.3 Data d	ependant predicate instructions14
6.4 Regist	er file indexing14
	TER FILE ALLOCATION
	ARBITRATION
9. ALU A	RBITRATION
10. HAND	LING-STALLS17
11. CONT	ENT OF THE RESERVATION STATION
FIFOS17	
12. THE O	UTPUT FILE
	MAT17
	ARAMETER CACHE
	EX POSITION EXPORTING
	TIME COMMANDS
	TERS
10 INITED	FACES
	rnal Interfaces 19
	PA/SC to RE : IJ bus 19
10.1.4	
1012	PA/SC to SEQ : IJ Control bus
18.1.3	VGT to RE : Vertex Bus
18.1.4	VGT to RE : Vertex Bus
18.1.4 18.1.5	VGT to RE : Vertex Bus20 VGT to SEQ : Vertex Control Bus20 CP to SEQ : Constant store load20
18.1.4 18.1.5 18.1.6	VGT to RE : Vertex Bus20 VGT to SEQ : Vertex Control Bus20 CP to SEQ : Constant store load20 CP to SEQ : Fetch State store load20
18.1.4 18.1.5 18.1.6 18.1.7	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 	VGT to RE : Vertex Bus20 VGT to SEQ : Vertex Control Bus20 CP to SEQ : Constant store load20 CP to SEQ : Fetch State store load20 CP to SEQ : Control State store load 20 MH to SEQ: Instruction store Load20
18.1.4 18.1.5 18.1.6 18.1.7 18.1.8 18.1.9	VGT to RE : Vertex Bus20 VGT to SEQ : Vertex Control Bus20 CP to SEQ : Constant store load20 CP to SEQ : Fetch State store load20 CP to SEQ : Control State store load 20 MH to SEQ: Instruction store Load20 SP to RB : Pixel read from RBs20
18.1.4 18.1.5 18.1.6 18.1.7 18.1.8 18.1.9 18.1.9 18.1.10	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 18.1.8 18.1.9 18.1.10 18.1.11	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 18.1.8 18.1.9 18.1.10 18.1.11 18.1.12	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 18.1.8 18.1.9 18.1.10 18.1.11 18.1.12 18.1.13 Bue)	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 18.1.8 18.1.9 18.1.10 18.1.11 18.1.12 18.1.13 Bue)	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 18.1.8 18.1.9 18.1.10 18.1.11 18.1.12 18.1.13 Bue)	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 18.1.8 18.1.9 18.1.10 18.1.11 18.1.12 18.1.13 Bus) 18.1.14 Bus)	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 18.1.8 18.1.9 18.1.10 18.1.10 18.1.11 18.1.12 18.1.13 Bue) 18.1.14 Bue) 19. INTER	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 	VGT to RE : Vertex Bus
18.1.4 18.1.5 18.1.6 18.1.7 18.1.8 18.1.9 18.1.10 18.1.10 18.1.11 18.1.12 18.1.13 Bue) 18.1.14 Bue) 19. INTER 20. EXAM	VGT to RE : Vertex Bus

Exhibit 2016 doc R400_Sequencer.doc 42015 Bytes*** S ATI Confidential. Reference Copyright Notice on Cover Page S ***

AMD1044_0256853

ATI Ex. 2105 IPR2023-00922 Page 181 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201526	GEN-CXXXXX-REVA	3 of 28
20.1.2	Sequencer Control of a Ve	ector of 21.	OPEN ISSUES	
Pixels 20.1.3		24		
Dovision	Changes:			
1/6/19/011	Unanges.			
Rev 0.1 (Laur Date: May 7, 2	r ent Lefebvre) 2001	First dra	aft.	
Rev 0.2 (Lauro	ent Lefebvre)	Change	d the interfaces to reflect the chang	es in the
Date : July 9, 3	2001	SP. Add	led some details in the arbitration se	ction.
Rev 0.3 (Laur			ed the Sequencer spec after the me	eting on
Date : August Rev 0.4 (Laure			3, 2001. the dynamic allocation method for	rogistor
Date : August			an example (written in part by Vi	
j	,		pixels/vertices in the sequencer.	.,
Rev 0.5 (Laure		Added t	iming diagrams (Vic)	
Date : Septem Rev 0.6 (Laure		Chapter	d the spec to reflect the new	
Date : Septem			ture. Added interfaces.	V K400
Rev 0.7 (Laur			constant store management, in	struction
Date : Octobe	r 5, 2001		nanagement, control flow managem	ent and
Davi O.D. (Lavia	ant lafahana)		pendant predication.	
Rev 0.8 (Laure Date : Octobe			d the control flow method to to Also updated the external interfaces	
Rev 0.9 (Laure			rated changes made in the 10/18/0	
Date : Octobe	r 17, 2001		eeting. Added a NOP instruction,	
			nditional_execute_or_jump. Added	debug
Rev 1.0 (Laure	ent Lefebure)	register Refined	s. interfaces to RB. Added state regist	ers
Date : Octobe		<u>I (Chiled</u>	Interfaces to AD. Added state regist	010.
***************************************	*********************************			

Exhibit 2016 doc/R400_Sequences.doc 42015 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 00/04/15 12:52

AMD1044_0256854

ATI Ex. 2105 IPR2023-00922 Page 182 of 239

AR	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 2001	4 September, 201526		4 of 28	

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 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.

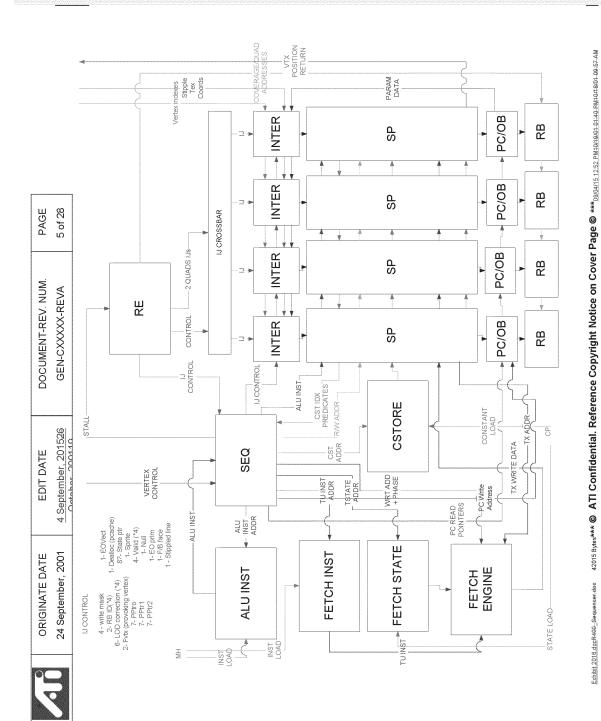
To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes. The four shader pipes also execute the same instruction thus there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

Exhibit 2016.doc/R400_Sequencer.doc 42015 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 00/04/15 12:52

AMD1044_0256855

ATI Ex. 2105 IPR2023-00922 Page 183 of 239

PROTECTIVE ORDER MATERIAL

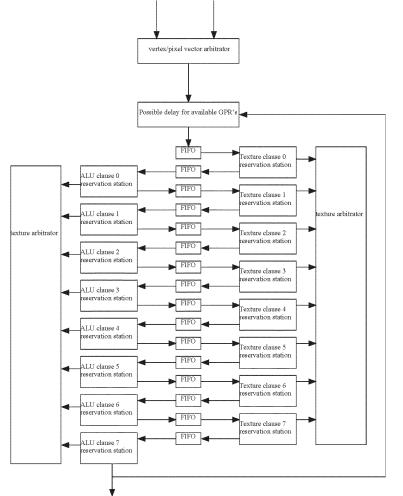


AMD1044_0256856

ATI Ex. 2105 IPR2023-00922 Page 184 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September, 2001	4 September, 201526		6 of 28	

1.1 Top Level Block Diagram



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 packetDepending 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.

Exhibit 2016 docR400_Sequencer.doc 42015 Bytes**** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15 12:52

AMD1044_0256857

ATI Ex. 2105 IPR2023-00922 Page 185 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1
24 September, 2001	4 September, 201526	GEN-CXXXXX-REVA	7 of 28	

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-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 <u>last 6</u> 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).

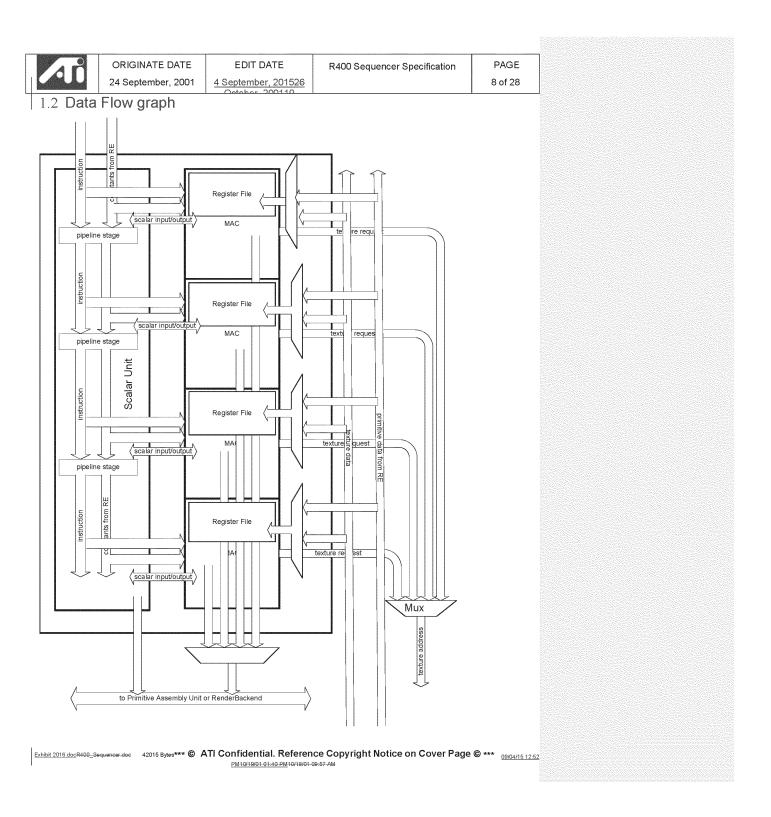
All other level process in the same way until the packet finally reaches the last ALU machine ($\underline{\otimes}$). On completion of the level $\underline{\otimes}$ -<u>A</u>LU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction 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.

Exhibit 2016.docR400_Sequencer.doc 42015 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 00/04/15 12:52

AMD1044_0256858

ATI Ex. 2105 IPR2023-00922 Page 186 of 239

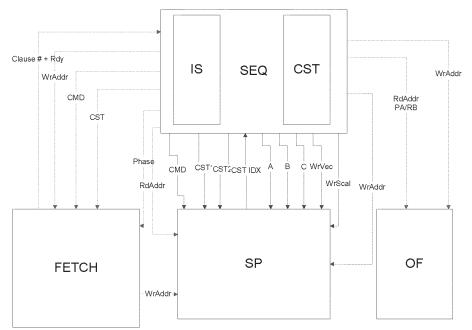


ATI Ex. 2105 IPR2023-00922 Page 187 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201526	GEN-CXXXXX-REVA	9 of 28

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.

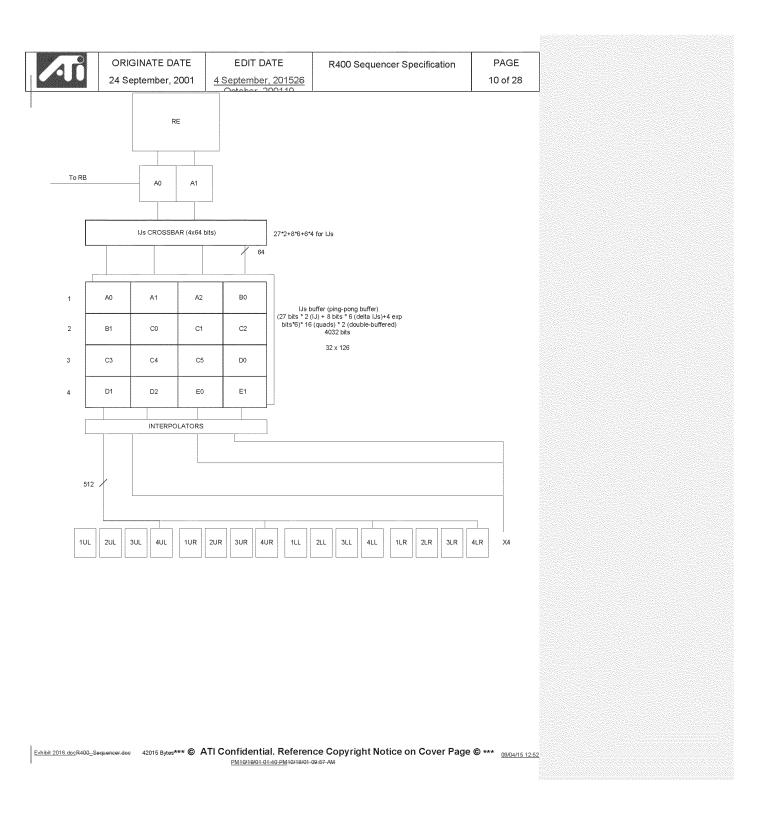
2. Interpolated data bus

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

Exhibit 2016 doc R400_Sequencer.doc 42015 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C ***

AMD1044_0256860

ATI Ex. 2105 IPR2023-00922 Page 188 of 239



ATI Ex. 2105 IPR2023-00922 Page 189 of 239 PROTECTIVE ORDER MATERIAL

	T19	51 48 <	55 55 <	59 - 56 -	63 - 63 <	
	T18	35 - 32-	30 ge <	4 ⁴ - 4	> 4 47	
	T17	> 1 0 4	23 ² 0 <	24- 27	31 28 <	
	T16	> 00	4-7	8-11 8-1	15 ¹ 7 <	
	T15			ЕО	Ш Т	
تي 38 يو	T14				ß	
PAGE 11 of 28	T13		S	G	8	
	T12				BO	2
7. NUM REVA	11	5	6			
DOCUMENT-REV. NUM. GEN-CXXXX-REVA	T10	ប៊	6	C5		
CUMEI	T9	B1				
0 0	Т8	AO	A1	A2		
କ୍ଷା	1			ЕО	ш	
ATE 20152	TG				ß	
EDIT DATE 4 September, 201526	T5		CO	ы С	3	
4 Sep	T4				BO	Ο
шБ	13	5	6			0 d
ORIGINATE DATE 24 September, 2001	12	ő	C4	C5		
ORIGINATE DATE 24 September, 2001	÷	Ε				
0R 24 S	10 T	AO	A1	A2		
Ę		SP0	SP1	SP2	SP3	

AMD1044_0256862

Exhibit 2016 doc/R400_Sequencer.doc 42015 Bytes**** C ATI Confidential. Reference Copyright Notice on Cover Page **** Georgen 51252 PM4048001 04:40 PM4048001 08:57 AM

ATI Ex. 2105 IPR2023-00922 Page 190 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September, 2001	4 September, 201526		12 of 28	1071016

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:

Exhibit 2016.docR400_Sequencer.doc 42015 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page *** 09/04/15 12:52

AMD1044_0256863

ATI Ex. 2105 IPR2023-00922 Page 191 of 239

ORIGINATE DATE EDIT DATE DOCUMENT-REV. NUM. PAGE 24 September, 2001 4 September, 201526 GEN-CXXXXX-REVA 13 of 28 Boolean[255:0] Ostebor 200110 Document[7:0][15:0] 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) Vertice of the second seco								
24 September, 2001 4 September, 201526 GEN-CXXXX-REVA 13 of 28 Boolean[255:0] Oxtober, 200110 Image: Control or Constant, register) Image: Control or Control or Constant, register) Image: Control or Contro or Control or Control or Control or Contro or Control or								
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_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_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								

ATI Ex. 2105 IPR2023-00922 Page 192 of 239

		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE			
		24 September, 2001	4 September, 201526		14 of 28			
	Execute up to 4k instructions at the specified address in the instruction memory.							
[NOP							

		NOI	1.5
47	46 42	41 0	
Addressing	00010	RESERVED	
			- 8

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.

Exhibit 2016.docR400_Sequencer.doc 42015 Bytes*** A ATI Confidential. Reference Copyright Notice on Cover Page ***

AMD1044_0256865

ATI Ex. 2105 IPR2023-00922 Page 193 of 239

		ORIGINATE DATE		EDIT DATE		DOC	DOCUMENT-REV. NUM.		
		24 September, 2001			September, 201526 GEI		N-CXXXXX-REVA		15 of 28
	Conditionnal Execute								
47		46 42	41 3	4	33	32 24	23 12		11 0
Address	sing	00011	Boolean ad	dress	Condition	RESERVED	Instruction_count	Exe	c 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				
	00101	RESERVED	Jump address	Loop ID				
Addressing								

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				
	00111	RESERVED	Start address	Loop ID				
Addressing								

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	4112	11 0				
	01000	RESERVED	Address				
Addressing							

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

	Return						
47	46 42	41 0					
	01001	RESERVED	Ĺ.				
Addressing							

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 1 <u>3</u> 2	12	11 0	
	01010	Boolean address	Condition	RESERVED	FW only	Address	
Addressing							

If condition met, jumps to the address. FORWARD jump only allowed <u>if bit 12 set</u>. <u>Bit 12 is only an optimization for the</u> <u>compiler and should NOT be exposed to the API.</u>

End_of_Clause					
47	46 42	41 0			

Exhibit 2016 docR400_Sequencer.doc 42015 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15 12:52

AMD1044_0256866

ATI Ex. 2105 IPR2023-00922 Page 194 of 239

∕ Jî	ORIGINATE DATE 24 September, 2007	 R400 Sequencer Specification	PAGE 16 of 28
	01011	RESERVED	
' Addressing			

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 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

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?}

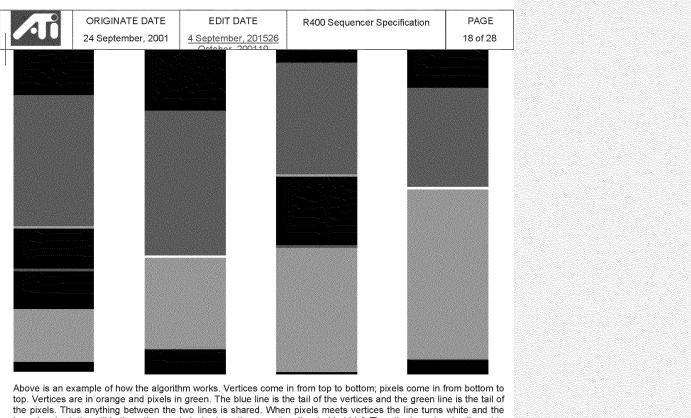
Exhibit 2016 doc R400_Sequencer.doc 42015 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15 12:52

AMD1044_0256867

ATI Ex. 2105 IPR2023-00922 Page 195 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201526	GEN-CXXXXX-REVA	17 of 28	
6.4 Regi	ster file indexing				
data created i		use it into an ALU clause.	ndex into the register file in order to The instruction will include the base		
	Bit7 Bit 6 0 0 0 1 1 0 1 1	'absolute register' 'relative register' 'previous vector' 'previous scalar'			
			the case of a relative register read v new address that we give to the shade		
7. Pixel I	Kill Mask			4-	Formatted: Bullets and Numbering
A vector of 64			to optimize the texture fetch request	s and allow	
MASK	<u>(sete</u> (setgt (setgte				
predicate vect	or and is kept across clause	e boundaries (thus allowing	then the 64 bit kill mask becomes the predicated instructions to be used in t 3s for coverage mask information.		Formatted
7.8. HOS	surfaces			4-	Formatted: Bullets and Numbering
parameter cad	che they have to do it in the		to memory ONLY. If they want to exact a second to export position in clause 3. The registers		
	ister file allocatior			*	Formatted: Bullets and Numbering
managed usir pixels and ve	ng two round robins (one fo	or pixels and one for vertice	ic or dynamic. In both cases, the reg es). In the dynamic case the bounda I to VERTEX_REG_SIZE for vertice	ry between	
Exhibit 2016.docR400_Se	aquancar.doc 42015 Bytes*** © AT	1 Confidential. Reference	Copyright Notice on Cover Page ©	*** 09/04/15 12:52	

ATI Ex. 2105 IPR2023-00922 Page 196 of 239



boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary is allowed to move again.

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.

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.

Exhibit 2016.doc/R400_Sequencer.doc 42015 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:52 PM10/19/01 01:40 PM10/18/01 09:57 AM

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044_0256869

ATI Ex. 2105 IPR2023-00922 Page 197 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 2001	4 September, 201526	GEN-CXXXXX-REVA	19 of 28	Formatted: Bullets and Numbering

11.12. 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.

12.13. 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.

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

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

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).

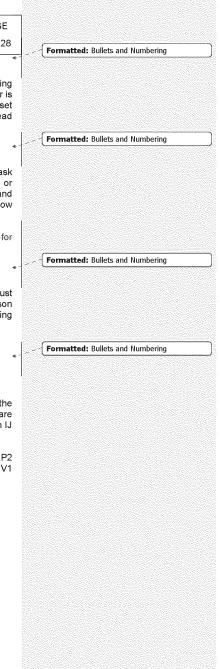
$\Delta 01I = I(1) - I(0)$		
$\Delta 01J = J(1) - J(0)$		
$\Delta 02I = I(2) - I(0)$	P0	
$\Delta 02J = J(2) - J(0)$		
$\Delta 03I = I(3) - I(0)$		
$\Delta 03J = J(3) - J(0)$	P2	
$P0 = C + I(0)^* (A - C) + J(0)^* (B - C)$		
$P1 = P0 + \Delta 01I * (A - C) + \Delta 01J * (B - C)$		
$P2 = P0 + \Delta 02I^{*}(A - C) + \Delta 02J^{*}(B - C)$		

 $P3 = P0 + \Delta 03I^{*}(A - C) + \Delta 03J^{*}(B - C)$

P0	P1
P2	P3

Exhibit 2016 doc R400_Sequencer.doc 42016 Bytes****
CATI Confidential. Reference Copyright Notice on Cover Page

00/04/16 12:3 PM10/19/01 01:40 PM10/18/01 09:57 AM



AMD1044_0256870

ATI Ex. 2105 IPR2023-00922 Page 198 of 239

ORIGINATE DATE EDIT DATE R400 Sequencer Specification	PAGE	
24 September, 2001 <u>4 September, 201526</u>	20 of 28	
P0 is computed at 19x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. S degradation of the image was seen using this scheme.	o far no visual	
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
15.16. The parameter cache	*	
The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 mem The reuse engine will make it so that all vertexes of a given primitive will hit different memories.	ories (1R/1W).	
16.17. Vertex position exporting	*-	Formatted: Bullets and Numbering
On clause 4 (or 5) the vertex shader can export to the PA both the vertex position and the point sprite. so at clause 8 if not done at clause 4. Along with the position is exported a pointer to the parameter ca data will be once the vertex shader exports. The storage needed to perform the position export is a memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel out	che where the t least 64x128	
18. 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. 2) Position exports and Z/Color 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. 4) Parameter exports and memory exports CAN be co-issued. 7) Parameter exports and memory exports CAN be co-issued. 7) Parameter exports and memory exports CAN be co-issued. 7) Parameter exports and memory exports CAN be co-issued. 7) Parameter exports and memory exports CAN be co-issued. 7) Parameter exports and memory exports CAN be co-issued. 7) Parameter exports and memory exports CAN be co-issued. 7) Parameter exports and memory exports CAN be co-issued. 7) Parameter exports and memory exports CAN be co-issued. 7) Parameter exports and memory exports CAN be co-issued. 7) Parameter exports and memory exports CAN be co-issued.	4	Formatted: Bullets and Numbering
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 need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mare register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and be able to address the reatime parameter memory as well as the regular parameter store. For higher preshould be able able to view them as two banks of 16 and do double buffering allowing one to be loa other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see wi view support for 16 vector-4 interpolants important (true only if we map microsoft's high priority stream stream), then the PA/sequencer need to support a realtime-specific mode where we need to address parameters instead of 16.	apped onto the /or PA need to erformance we ded, while the e to restrict the th this is, if we to the realtime	
Exhibit 2016.doc/R400_Sequencer.doc 42015 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page	: © *** <u>09/04/15 12:52</u>	

ATI Ex. 2105 IPR2023-00922 Page 199 of 239

		1			
	SINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 Se	ptember, 2001	4 September, 201526	GEN-CXXXXX-REVA	21 of 28	Formatted: Bullets and Numbering
8.20. Registe	<u>rs</u>			*1	
0.1 Control					
0.1 Control					
DYNAMIC_REG VERTEX_REG_SIZ	Dynamic ZE What po	(vlac			
PIXEL_MIN_SIZE	Minimal	Jiny)			
VERTEX_MIN_SIZ		size of the register file's verte			
ARBITRATION pol		the arbitration between vete ne constant store for pixels	xes and pixels		
CST_SIZE_V	Size of th	ne constant store for vertexe			
INST_STOR_ALLC		ed, separate, interleaved+sh			
PWRAP		nt for the vertex shader instr nt for the pixel shader instru			
NO INTERLEAVE			program at a time into the GPRs		
8.120.2 Conte	ext			*-	
Vshader_fetch[117]	************	8 hit pointage to the location	where each clauses control program	n is located	
Vshader_alu[117:0]			where each clauses control program		
Pshader_fetch[117]	0][7:0] eight 12-		where each clauses control program		
Pshader_alu[117:0] PSHADER			where each clauses control program	n is located	
VSHADER		nter for the pixel shader nter for the vertex shader		Ē	
Vshader_cntl_size		e vertex shader (# of instruc			
Pshader_cntl_size Pshader_size		e pixel shader (# of instructi e pixel shader (cntl+instruct			
Vshader size		e vertex shader (cntl+instruct	annanafir		
VWRAP	wrap poi	nt for the vertex shader instr	uction store		
	(I	nt for the pixel shader instru-		1	
REG_ALLOC_PIX REG_ALLOC_VER		of registers to allocate for pix of registers to allocate for ve			
PARAM_MASK[0	16] paramete	er mask to specify how para	meters maps in the pixel shader		
FLAT_GOUR[01 GEN_TEX[016]		ameters are to be gouraud s	haded eed to generate tex coords.Do we g	aparata	
texture coordinates	for 1 st parameter	or not	eed to generate tex coords.Do we g	enerate	Formatted
CYL_WRAP[064	63] f	or wich parameters (and cha	annels (xyzw)) do we do the cyl wra	pping.	
P export mode		lormal mode Iultipass mode			
			ny colors (0-4) and z is export z or r	not	
	lf multipa	ss 1-12 exports for color.			
vshader export ma vshader export ma		ne last 6 ALU clauses is exp n (1 vector), 1: position (2 ve			
vshader_export_co					
	# of expo	orted vectors to memory per	clause in multipass mode (per claus	se)	
kill_vector_on	use the r	nask kill vector to kill pixels a	and optimize texture pipe fetches OI	R use it as	
	where a state of the state of t		only predicate vector kept across cla	iuse	
P_EXPORT[8]	boundari number (<u>es.</u> of exports for pixel shader			
V_EXPORT[8]		to the output			
	buffer ex 00000.	7] is set to			
ARBITRATION pol		the arbitration between vete	xes and pixels		
anna I	Exports[3][6] Wich clause is expor	ting to the output buffer and what is	it exporting.	
		Not exporting (or exporting	only to the PC)		
		Exporting position (1) Exporting position (2)			
		Exporting RG			
		Confidential Reference	Copyright Notice on Cover Page @	9 *** 09/04/15 12:52	
bit 2016.docR400_Sequencer.doc	42015 Bytes 👻 A1	PM10/19/01-01:40 PM10/18/01-09:67.		09/04/15 12:52	

ATI Ex. 2105 IPR2023-00922 Page 200 of 239

ORIG	NATE DATE	EDIT DATE		R400 Sequencer Specification	PAGE]
	tember, 2001	4 September, 201	526	N-00 Dequencer opecification	22 of 28	
24 Sep	,	4 September, 2011 October 20011 0 : Exporting BA			22 01 20	J
		0 : Exporting BA				
	If MSB I	set pixel shader exp		near to memory not to Frame Buffer	k B	
CST_SIZE_P		the constant store for the constant store for				
00-1 <u></u> 01 <u>2-1</u> V		the constant store o	//VC(10)			
						Formatted: Bullets and Numberin
). 21. DEBUG	<u>registers</u>					
PROB_ADDR	instructi	on address where t	ne first j	problem occurred		
PROB_COUNT	number	of problems encou	ntered c	luring the execution of the program		
0.00 Interface					4	Formatted: Bullets and Numberin
0.22. Interface	<u>;></u>					
.0.1 22.1 Extern	al Intorfac	20				
<u>v.r_22.1</u> LALEIT		50				
	ISC to RES	PA · I I hus				
				op of each shader pipe. At the same t r the whole chip (SP0 thru 3)	ime the control	
lame Is	Direction PA→RI			ription ormation sent over 2 clocks		
	FA→ R €			Mask		
lask . 0.1.2<u>22.1.2</u>PA his is the control inforr	nation sent to the	: IJ Control bu	S	trol the IJ fifos and all other informa	tion needed to	(Formatted: Bullets and Numberin
Mask 20.1.2 <u>22.1.2</u> PA his is the control inforr xecute a shader progra	/SC to SEQ nation sent to the am on the sent p	: IJ Control bu	S r to cor	trol the IJ fifos and all other informa	tion needed to	(Formatted: Bullets and Numberin
lask 2 <u>0.1.222.1.2</u> PA his is the control inforr xecute a shader progra lame	/SC to SEQ nation sent to the am on the sent p Direction	: IJ Control bu e sequencer in orde ixels.	S r to cor Bits	ntrol the IJ fifos and all other informa	tion needed to	(Formatted: Bullets and Numberin
lask 10.1.222.1.2_PA his is the control inforr xecute a shader progra ame /rite Mask	/SC to SEQ nation sent to the am on the sent p Direction PA→SEC	: IJ Control bu e sequencer in orde ixels.	S r to cor	trol the IJ fifos and all other informa		(Formatted: Bullets and Numberin
ask 0.1.2 <u>22.1.2</u> PA his is the control inforr kecute a shader progra ame frite Mask B_ID	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC $\overrightarrow{PA}\rightarrow$ SEC \overrightarrow{P}	: IJ Control bu e sequencer in orde ixels. ((RE) <u>SP)</u> ((SP)PA→SEQ(RE	S r to cor Bits 4 8	trol the IJ fifos and all other informa Description Quad Write mask left to right RB id for each quad sent 2 bits pe	r quad	(Formatted: Bullets and Numberin
task :0.1.2 <u>22.1.2</u> PA his is the control inforr xecute a shader progra ame /rite Mask B_ID	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC $\overrightarrow{PA}\rightarrow$ SEC \overrightarrow{P}	: IJ Control bu e sequencer in orde ixels. ((RE)SP)	S r to cor Bits 4	trol the IJ fifos and all other informa Description Quad Write mask left to right	r quad	(Formatted: Bullets and Numberin
Mask 20.1.2 <u>22.1.2</u> PA his is the control inforr xecute a shader progra	/SC to SEQ nation sent to the am on the sent p $PA \rightarrow SEC$ $PA \rightarrow SEC$ $PA \rightarrow SEC$)	: IJ Control bu e sequencer in orde ixels. ((RE) <u>SP)</u> ((SP)PA→SEQ(RE	S r to cor Bits 4 8	trol the IJ fifos and all other informa Description Quad Write mask left to right RB id for each quad sent 2 bits pe	r quad	(Formatted: Bullets and Numberin
Mask 20.1.222.1.2 PA his is the control inforr xecute a shader progra lame Vrite Mask B_ID OD_CORRECT VTX	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC \rightarrow PA \rightarrow SEC \rightarrow PA \rightarrow SEC \rightarrow	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	5 r to cor 8 its 4 8 24 24 2	trol the IJ fifos and all other informa Description Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading	r quad	(Formatted: Bullets and Numberin
Aask 20.1.222.1.2 PA his is the control inforr xecute a shader progra lame Vrite Mask IB_ID OD_CORRECT	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC \rightarrow PA \rightarrow SEC \rightarrow PA \rightarrow SEC \rightarrow	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	S r to con Bits 4 8 24	trol the IJ fifos and all other informa Description Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe	r quad	(Formatted: Bullets and Numberin
Mask 20.1.222.1.2 PA his is the control inforr xecute a shader progra lame Vrite Mask B_ID OD_CORRECT VTX	/SC to SEQ nation sent to the am on the sent p $PA \rightarrow SEC$ $PA \rightarrow SEC$ $PA \rightarrow SEC$ $PA \rightarrow SEC$ $PA \rightarrow SEC$ $PA \rightarrow SEC$ $PA \rightarrow SEC$: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	5 r to cor 8 its 4 8 24 24 2	trol the IJ fifos and all other informa Description Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading	r quad	(Formatted: Bullets and Numberin
Aask 20.1.222.1.2 PA his is the control inforr xecute a shader progra lame Vrite Mask IB_ID OD_CORRECT VTX PTR0	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC PA \rightarrow SEC	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	S r to cor Bits 4 8 24 2 11	trol the IJ fifos and all other informa Description Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1	r quad	(Formatted: Bullets and Numberin
task 20.1.222.1.2 PA his is the control inforr xecute a shader progra ame /rite Mask B_ID OD_CORRECT VTX PTR0 PRT1 PTR2	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC PA \rightarrow SEC	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	S r to cor Bits 4 8 24 2 11 11 11	trol the IJ fifos and all other informa Description Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2	r quad	(Formatted: Bullets and Numberin
Mask 20.1.222.1.2 PA his is the control inforr xecute a shader progra lame Vrite Mask B_ID OD_CORRECT VTX PTR0 PRT1	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC PA \rightarrow SEC	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	5 r to cor 4 8 24 2 11 11	trol the IJ fifos and all other informa Description Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1	r quad	(Formatted: Bullets and Numberin
ask 0.1.222.1.2 PA his is the control inforr kecute a shader progra ame frite Mask B_ID DD_CORRECT VTX PTR0 PRT1 PTR2 _OFF_VECTOR	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC PA \rightarrow SEC	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	S r to cor Bits 4 8 24 2 11 11 11	trol the IJ fifos and all other informa Description Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2	r quad er quad)	(Formatted: Bullets and Numberin
lask 10.1.222.1.2 PA his is the control inforr xecute a shader progra ame /rite Mask B_ID OD_CORRECT VTX PTR0 PRT1 PTR2 _OFF_VECTOR EALLOC	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC PA \rightarrow SEC	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	S r to con 8its 4 8 24 2 11 11 11 11 11	trol the IJ fifos and all other informa Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store	r quad er quad)	(Formatted: Bullets and Numberin
Alask 0.1.222.1.2 PA his is the control inforr kecute a shader programe Amme DD_CORRECT VTX PTR0 PRT1 PTR2 _OFF_VECTOR EALLOC	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC PA \rightarrow SEC	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	S r to con 8 4 8 24 2 11 11 11 11	trol the IJ fifos and all other informa Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector	r quad er quad)	(Formatted: Bullets and Numberin
lask 0.1.222.1.2 PA his is the control inforr xecute a shader progra ame /rite Mask B_ID OD_CORRECT VTX PTR0 PRT1 PTR2 _OFF_VECTOR EALLOC TATE	/SC to SEQ nation sent to the am on the sent p $PA \rightarrow SEC$ $PA \rightarrow SEC$: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	S r to con 8its 4 8 24 2 11 11 11 11 1 1	trol the IJ fifos and all other informa Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store	r quad er quad)	(Formatted: Bullets and Numberin
lask 0.1.222.1.2 PA his is the control inforr xecute a shader progra ame /rite Mask B_ID OD_CORRECT VTX PTR0 PRT1 PTR2 _OFF_VECTOR EALLOC TATE ALID	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC PA \rightarrow SEC	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	S r to cor 8its 4 24 2 11 11 11 11 1 1 1 21 21 16	trol the IJ fifos and all other informa Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels	r quad er quad)	(Formatted: Bullets and Numberin
lask 0.1.222.1.2 PA his is the control inforr xecute a shader progra ame /rite Mask B_ID OD_CORRECT VTX PTR0 PRT1 PTR2 _OFF_VECTOR EALLOC TATE ALID	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC PA \rightarrow SEC	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	Bits 4 8 24 11 11 11 11 2	trol the IJ fifos and all other informa Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3)	r quad er quad)	(Formatted: Bullets and Numberin
Mask 20.1.222.1.2 PA his is the control inforr xecute a shader progra lame Vrite Mask B_ID OD_CORRECT VTX PTR0 PRT1 PTR2	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC PA \rightarrow SEC	: IJ Control bu e sequencer in orde ixels. ((RE)SP) ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE ((SP)PA→SEQ(RE	S r to cor 8its 4 24 2 11 11 11 11 1 1 1 21 21 16	trol the IJ fifos and all other informa Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels	r quad er quad)	Formatted: Bullets and Numberin
lask 0.1.222.1.2 PA his is the control inforr xecute a shader progra ame /rite Mask B_ID OD_CORRECT VTX PTR0 PRT1 PTR2 _OFF_VECTOR EALLOC TATE ALID ULL	/SC to SEQ nation sent to the am on the sent p Direction PA \rightarrow SEC PA \rightarrow SEC	: IJ Control bu e sequencer in orde ixels. $(RE)SP)$ $(SP)PA \rightarrow SEQ(RE)$	S r to cor Bits 4 24 2 11 11 11 11 1 1 1 1 1 1 1 1 1 1	trol the IJ fifos and all other informa Description Quad Write mask left to right RB id for each quad sent 2 bits pe LOD correction per quad (6 bits pe Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels Null Primitive (for PC deallocation	r quad er quad)	Formatted: Bullets and Numberin

ATI Ex. 2105 IPR2023-00922 Page 201 of 239

September, 20 PA \rightarrow SEQ(RE PA \rightarrow SEQ(RE	10 3 1 1 1 1 1 1 1 1 1 1 1 1 1	hdexes or HOS surface information ector 28 bits per vert 6 bits per vert 9 (6*3+3 for constants) Formatted: Bullets and Nu he block of 4 constants rer 4 clocks ddress write address ointer Formatted: Bullets and Nu Formatted: Bullets Formatted	umbering		
PA \rightarrow SEQ(RE PA \rightarrow SEQ(RE ertex Bus 128 128 1 1 1 21 tant store lo Bits 8 512 10 8 0 State store Bits 8 512 10 8 10 8 10 8 10 10 10 10 10 10 10 10 10 10	3 1 1 1 1 1 1 1 1 1 1 1 1 1	tex cords from alternate buffer Normal Stippled line Real Time Line AA Point AA Sprite s the PA in n clocks eady to send data ndexes or HOS surface information ector 8 bits per vert 6 bits per vert 6 bits per vert 6 bits per vert 6 constants)	umbering		
$\begin{array}{c c} PA \rightarrow SEQ(RE \\ \hline \\ ertex Bus \\ \hline \\ 128 \\ 1 \\ 1 \\ 21 \\ \hline \\ tant store lo \\ \hline \\ Bits \\ 8 \\ 512 \\ 10 \\ 8 \\ \hline \\ 8 \\ a \\ State store \\ \hline \\ Bits \\ 8 \\ \hline \end{array}$	1 1 1 Descr Pointe End of 0: Nor 1: dou Rende Data s Rema Rema Rema e Joac Descr Addree Data s	tex cords from alternate buffer Normal Stippled line Real Time Line AA Point AA Sprite s the PA in n clocks eady to send data ndexes or HOS surface information ector 8 bits per vert 6 bits per vert 6 bits per vert 6 bits per vert 6 constants)	umbering		
ertex Bus Bits 128 1 1 21 tant store lo Bits 8 512 10 8 10 8 10 8 10 8 10 8	1 Descr Pointe End of 0: Nor 1: dou Rende Data s Rema Rema Rema e Ioac Descr Addree Data s	eady to send data	umbering		
ertex Bus Bits 128 1 1 21 tant store lo Bits 8 512 10 8 10 8 10 8 10 8 10 8	Descr Pointe End of O: Nor 1: dou Rende Data s Rema Rema Rema e loac Descr Addree Data s	Formatted: Bullets and Nu	umbering		
Bits 128 1 1 21 tant store lo Bits 8 512 10 8 0 State store Bits 8	Pointe End of 0: Nor 1: dou Rende Data s Rema Poac Data s Rema Rema Data s Rema Rema	ndexes or HOS surface information ector 28 bits per vert 6 bits per vert a (6*3+3 for constants) (* Formatted: Bullets and Nu he block of 4 constants inter * Formatted: Bullets and Nu * Formatted: Bullets and Nu * Formatted: Bullets and Nu * Formatted: Bullets and Nu	umbering		
128 1 1 21 tant store lo Bits 8 512 10 8 0 State store Bits 8 8	Pointe End of 0: Nor 1: dou Rende Data s Rema Poac Data s Rema Rema Data s Rema Rema	ndexes or HOS surface information ector 28 bits per vert 6 bits per vert 6 (6*3+3 for constants) (6*3+3 for constants) (* Formatted: Bullets and Nu he block of 4 constants inter * Formatted: Bullets and Nu * Formatted: Bullets and Nu * Formatted: Bullets and Nu *			
1 1 21 tant store lo Bits 8 512 10 8 0 State store Bits 8	End of 0: Nor 1: dou Rende Data s Rema e /oac Descr Addre: Data s Rema Rema	ector 8 bits per vert 6 bits per vert 9 (6*3+3 for constants) • Formatted: Bullets and Nu he block of 4 constants er 4 clocks ddress write address ointer • Formatted: Bullets and Nu he block of 4 state constants			
1 21 tant store lo Bits 8 512 10 8 9 State store Bits 8	0: Nor 1: dou Rende Descr Addre: Data s Rema Rema <i>Rema</i> <i>Coac</i> Descr Addre: Data s	28 bits per vert 6 bits per vert 6 bits per vert 6 (6*3+3 for constants) • Formatted: Bullets and Nu he block of 4 constants rer 4 clocks ddress write address ointer • Formatted: Bullets and Nu he block of 4 state constants			
21 tant store lo Bits 8 512 10 8 9 State store Bits 8	1: dou Rende	6 bits per vert a (6*3+3 for constants) formatted: Bullets and Nu he block of 4 constants rer 4 clocks ddress write address ointer formatted: Bullets and Nu he block of 4 state constants			
tant store lo Bits 8 512 10 8 0 State store Bits 8	Rende Dad Descr Addre: Data s Rema Rema e 10a0 Descr Addre: Data s	e (6*3+3 for constants) The block of 4 constants rer 4 clocks ddress write address ointer Formatted: Bullets and Nu Formatted: Bullets Formatted: Bullets and Nu Formatted: Bullet			
tant store lo Bits 8 512 10 8 0 State store Bits 8	Dad Descr Addree Data s Rema Rema e 10a0 Descr Addree Data s	Formatted: Bullets and Nu			
Bits 8 512 10 10 8 0 State store Bits 8 8	Descr Addres Data s Rema Rema <i>Poac</i> Descr Addres Data s	he block of 4 constants rer 4 clocks ddress write address ointer Formatted: Bullets and Nu he block of 4 state constants			
8 512 10 8 9 State store Bits 8	Addres Data s Rema Rema <i>2 /020</i> Descr Addres Data s	he block of 4 constants rer 4 clocks ddress write address ointer Formatted: Bullets and Nu Formatted: Bullets and Nu he block of 4 state constants	umbering		
512 10 8 9 State store Bits 8	Data s Rema Rema <i>- loac</i> Descr Addres Data s	rer 4 clocks ddress write address ointer Formatted: Bullets and Nu Fe block of 4 state constants	umbering		
10 8 State store Bits 8	Rema Rema <i>loac</i> Descr Addres Data s	ddress write address ointer Formatted: Bullets and Nu Fe block of 4 state constants	umbering		
8 State store Bits 8	Rema 2 /0a0 Descr Addres Data s	ointer Formatted: Bullets and Nu Formatted: Bullets and Nu he block of 4 state constants	umbering		
State store	<i>∂ /0ac</i> Descr Addres Data s	Formatted: Bullets and Nu Formatted: Bullets and Nu he block of 4 state constants	umbering		
Bits 8	Descr Addre Data s	he block of 4 state constants			
8	Addre Data s	he block of 4 state constants			
	Data s				
10	Rema	ddress write address			
8	Rema				
rol State sto	ore loa	← Formatted: Bullets and Nu	umbering		
	Descr				
s bus?} ction store l	Load	← ~ ~ Formatted: Bullets and Nu	umbering		
Bits	Descr				
12					
96					
9					
32	Contro	uction X times			
ad from RB	3s	←	umbering		
Bits	Desc	l			
64*16					
ExportIDShader Dest SP→RB 94 Specifies one of the of up to 12 export destinations0cvvvvhqq: Vortex-data-wvv-0-15 from first					
-	12 96 9 32 ad from RE Bits 64*16	12 Instruction a 96 Instruction X 9 Pointer to th 32 Control Instr ad from RBs Bits Bits Description 64*16 32a pairs of 94 Specifies destination	12 Instruction address 96 Instruction X times 9 Pointer to the control instruction store 32 Control Instruction X times ad from RBs • • • • • • • • • • • • • • • • • • •		

ATI Ex. 2105 IPR2023-00922 Page 202 of 239

	1					1		
	ORIGINATE D	DATE	EDIT DATE		R400 Sequencer Specification PAGE			
	24 September,	2001	4 September		<u>6</u> 24 of 28			
Shader Coun		SP→RI	3	23	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) 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→RI	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 Pixel	Valid	$\underline{SP \rightarrow RB} \qquad \underline{4x4}$		<u>4x4</u>	Result of pixel kill in the shader pipe, wh output for all pixel exports (depth and all 4x4 because 16 pixels are computed per	color buffers).		
Shader_Word	Valid	<u>SP→R</u> I	3	2	Specifies whether to write low and/or hig of the 64-bit export data from each of the vectors	************************************		

20.1.922.1.9 SEQ to RB : Control bus

Name	Direction	Bits	Description
Export_PixelType	SEQ→RB	1	10: Pixel
			10: Vertex
Export_SENDInterleaving	SEQ→RB	1	Raised to indicate that the SQ is starting an export0:
			first interleaved clause
			1: second interleaved clause
Export_ClauseExport_size	SEQ→RB	43	Clause number, which is needed for vertex clauses0
			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_StateValid	SEQ→RB	121?	State ID, which is needed for vertex clausesData valid

{ISSUE: Where are the PC pointers}Only one exporting clause (7) can be selected at any given time.

20.1.1022.1.10 RB to SEQ : Output file control

Name	Direction	Bits	Description
Export_RTSBuff_Full	RB→SEQ	1	Raised by RB to indicate that the following two fields reflect the result of the most recent exportSet if full
Export_PositionAvail_size	RB→SEQ	<u>1</u> 6	Specifies whether there is room for another position.Size available in output buffers (in 32bits increments)
Export_Buffer	<u>RB→SEQ</u>	Z	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

Exhibit 2016.doc.R400_Sequencer.doc 42015 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:52

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044_0256875

ATI Ex. 2105 IPR2023-00922 Page 203 of 239

	ORIGINATE DAT		DATE		NT-REV. NUM.	PAGE		
	24 September, 20	01 <u>4 Septemb</u>			XXXX-REVA	25 of 28		
20.1.11 <u>22</u>	2.1.11_SP to RE	3 : Position ret	turn b	bus		4-		Formatted: Bullets and Numbering
Name	Dir	rection	Bits	Description				
Position return	SP	P→RB	128	Position data or sprite si	ze (per clock)			

Pointer where the data will be in the parameter cache for

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.

each vertex

11

20.1.1222.1.12 Shader Engine to Fetch Unit Bus (Fast Bus)

SP→RB

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	1
Tex RegFile Write Data	<u>TEX→SP</u>	2048	16 texture results	l

20.1.1322.1.13 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	$TEX \rightarrow SEQ$	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	<u>SEQ→TEX</u>	96	LOD correct 3 bits per comp 2 components per quad * 16
			quads
Mask	<u>SEQ→TEX</u>	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

Parameter cache pointer

21.1.123.1.1 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.

Exhibit 2016 doc R400_Sequences.doc 42015 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C ***

PM10/19/01-01:40-PM10/18/01-09:57-AM

- Formatted

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

---- **Formatted:** Bullets and Numbering

AMD1044_0256876

ATI Ex. 2105 IPR2023-00922 Page 204 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September, 2001	4 September, 201526		26 of 28	Formatted: Bullets and Numbering
22.24. Examples of prog			4	-
22.1.124.1.1 Sequencer Co	ontrol of a Vector o	f Vertices		
 also before the vector is sen shader program (using the N The vertex program is assume 	position cache is sent alo ition cache for transformed t to the RE, the CP has lo IH?) d to be loaded when we re b IS base for this shader us	ng with vertices I position before the vector was sent baded the global instruction store wi ceive the vertex vector. sing the local state pointer (provided to	th the vertex	
 2. SEQ arbitrates between the Pixel F at this point the vector is remover the arbitrer is not going to select nothing else to do (ie no pixels) 	red from the Vertex FIFO a vector to be transforme	 basically the Vertex FIFO always has ed if the parameter cache is full unless 		
 SEQ allocates space in the SP reg the number of GPRs required l state pointer that came down w SEQ will not send vertex data 	by the program is stored in vith the vertices	a local state register, which is accesse	ed using the	
	o the 64 register files over d SU3 is written the first cy d SU3 is written the secon d SU3 is written the third c d SU3 is written the fourth significant 32 bits (floating	4 cycles /cle d cycle :ycle		
 SEQ constructs a control packet fo fetch state machine 0, or TSM0 FIF the control packet contains the 	FO)	the first reservation station (the FIFO) e position cache and a register file bas		
 TSM0 accepts the control packet a TSM0 was first selected by the 			uction store	
7. all instructions of fetch clause 0 are	issued by TSM0			
the fetch data to the TU, whichonce the TU has written all the	ts made to the Fetch Unit t will write the data to the R data to the register files, it	to complete; it passes the register file v	vrite index for d with ASM0	
 ASMO accepts the control packet (a clause 0 from the global instruction 	u	ASM arbiter) and gets the instructions	for ALU	
10. all instructions of ALU clause 0 are station (the FIFO in front of fetch st			eservation	
 shared with all four shader pipe A parameter cache pointer is a going to be in the parameter ca 	J clause 3 (or 4?); the data es) back to the PA's position lso sent along with the position inche.	a (and the tag) is sent over a position b	us (which is ne data is	
Exhibit 2016.docR400_Sequencer.doc 42015 Bytes*** ©	ATI Confidential. Referer PM10/19/01-01:40-PM10/19/01	nce Copyright Notice on Cover Page	© *** <u>09/04/15 12:52</u>	

ATI Ex. 2105 IPR2023-00922 Page 205 of 239

	2	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	UU	24 September, 2001	4 September, 201526	GEN-CXXXXX-REVA	27 of 28	
•	parame • pa • the lor • the	eter data is exported in clau rameter data is sent to the sEQ allocates storage in ager a need for the paramet	packet from starting an exp use 7 (as well as position da Parameter Cache over a de the Parameter Cache, and t ters (it is told by the PA whe packet from starting on AS	he SEQ deallocates that space wher	there is no	
	ifter the sh hader pro		ed, the SEQ will free up the	GPRs so that they can be used by a	nother	
22.3	1 <u>.2</u> 24.1	.2_Sequencer Con	trol of a Vector of P	ixels	*-	Formatted: Bullets and Numbering
1. A	As with ve	ertex shader programs, pi	xel shaders are loaded in	to the global instruction store by t	he CP	
•	At this	point it is assumed that the	pixel program is loaded inte	o the instruction store and thus ready	to be read.	
2. tł •	the sta	te pointer and the LOD cor	barycentric coordinates for rection bits are also placed source four quad's worth of			
		ates between Pixel FIFO al egister files for vertices, the		e are no vertices pending OR there is	s no space	
4. S •	the nui state p	ates space in the SP registe mber of GPRs required by t ointer <i>r</i> ill not allow interpolated da	-			
			ed data to the SP register fi erpolated data bus diagram	le over the RE_SP interface (which h s for details.	nas a	
		ructs a control packet for th machine 0, or TSM0 FIFO)		first reservation station (the FIFO in	front of	
•	note th the cor	at there is a separate set o ntrol packet contains the sta	f reservation stations/arbite	rs/state machines for vertices and for pase pointer, and the LOD correction vels in a separate FIFO		
			fetches the instructions for SM arbiter before it could sta	fetch clause 0 from the global instruc art	tion store	
8. a	III instructi	ons of fetch clause 0 are is	sued by TSM0			
	FIFO) TSM0 index f once th associa	does not wait for fetch requ or the fetch data to the TU, ne TU has written all the da ated with the ASM0 FIFO; a	ests made to the Fetch Uni which will write the data to ta for a particular clause to	the register files, it increments a cour dicates that the ALU state machine c	ile write nter that is	

- 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

Exhibit 2016 doc R400_Sequencer.doc 42015 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 00/04/15 12:52

AMD1044_0256878

ATI Ex. 2105 IPR2023-00922 Page 206 of 239

_						1	
	V A A	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE		
		24 September, 2001	4 September, 201526		28 of 28		(<u> </u>
	22.1.3 24.	1.3_Notes			4~		Formatted: Bullets and Numbering
	14. <u>T</u> the state threads or		vill operate ahead of time s	o that they will be able to immediately	start the real		
			vector needs to travel with	the vector through the reservation stat	tions, but the		
, ,	instruction	store base pointer does		F pointer is different for all threads, but			
			cation, loops and branche	s and parameter cache de-allocation s	still needs to		
	be specke	a out.					
	00.05 O				4.		Formatted: Bullets and Numbering
	<u>23-25. O</u>	<u>pen issues</u>					
				t the same for the whole vector of vert utions exists for this problem:	ices, we don't		
	1) Le	et the compiler handle the	case and put those instru	ctions in a fetch clause so we can use			
	2) W	aterfall down the pipe allo	wing only at a given time	temporary storage in the register store the vertices having the same constants			
	pa	rrallel. This might in the v	vorst case slow us down b	y a factor of 16.			
	Need to do so static).	me testing on the size of	the register file as well as	s on the register file allocation method	(dynamic VS		
	Saving power	?					
1	Size of the fif	containing the informat	ion of a vector of nivels/v	ertices. And size of the fifos before th	e reservation		
	stations.						
	Loops and bra	nches.					
1		-					
	Exhibit 2016.docR400_S	equencer.doc 42015Bytes*** ⓒ 🖌	ATI Confidential. Referen <u>PM10/19/01-01:40 PM</u> 10/18/01-0	ce Copyright Notice on Cover Page 18:57 AM	© *** <u>09/04/15 12:52</u>	2	

ATI Ex. 2105 IPR2023-00922 Page 207 of 239

10300000000000000000000000000000000000			
ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	1 of 32
Author: Laurent Lefebvre			
Issue To:	Copy No:		
R400 S	equencer Spe	ecification	
	8 8		
	SEQ		
	Version 1. <u>1</u> 0		
	version i.je		
Overview: This is an architectural specifi			
required capabilities and expe blocks, and provides internal s		lso describes the block interfaces,	internal sub-
AUTOMATICALLY UPDATED FIELDS:			
	erforce\r400\arch\doc\gfx\RE 0 Sequencer Specification	R400_Sequencer.doc	
	APPROVALS		
Name/Dept		Signature/Date	
Remarks:			
THIS DOCUMENT CONTAIN			
SUBSTANTIALLY DETRIME		SE OR DISCLOSURE.	OGIES
"Copyright 2001, ATI Technologies Inc. A	Il rights reserved. The mate	rial in this document constitutes an	unpublished
work created in 2001. The use of this co- unpublished work. The copyright notice			
confidential, proprietary information and to transmitted in any form or by any means w			produced, or
		Ŭ	
Exhibit 2017.docR400_Sequencer.doc 46659 Bytes*** C A	TI Confidential. Reference (PM10/19/01-01:40 PM	Copyright Notice on Cover Page ©) *** <u>09/04/15 12:53</u>

ATI 2017 LG v. ATI IPR2015-00325

AMD1044_0256880

ATI Ex. 2105 IPR2023-00922 Page 208 of 239

	0000
	M A M
Ι.	

Table Of Contents

< 0\/FF	5. csm.5.c m.a
	VIEW
	evel Block Diagram
	Flow graph
	ol Graph
2. INTER	RPOLATED DATA BUS
3. INSTI	RUCTION STORE 1312
	JENCER INSTRUCTIONS
5. CONS	STANT STORE
	PING AND BRANCHES
	ontrolling state
	Control Flow Program
	dependant predicate instructions1715
	etection of PV,PS
	ter file indexing1816
	cated Instruction support for Texture
	gging the Shaders1816
<u>6.7.1</u>	
6.7.2	Method 2: Exporting the values in the
<u>GPRs (12</u>)	<u>) 1917</u>
<u>6.7.3</u>	
Dword.	1917
	_ KILL MASK2018
8. HOS	SURFACES
9. REGI	STER FILE ALLOCATION
	H ARBITRATION
11. ALU	ARBITRATION2219
	DLING STALLS
13. CON	TENT OF THE RESERVATION STATION
FIFOS2320	
14. THE (OUTPUT FILE
15. IJ FO	RMAT2320
	PARAMETER CACHE
***************************************	EX POSITION EXPORTING
	RTING ARBITRATION
	TIME COMMANDS
	STERS
	ntrol
	ntext
	JG REGISTERS
	ntrol
	ntext
	RFACES
***************************************	ernal Interfaces
22.1.1	PA/SC to SP0 : IJ bus
	PA/SC to SEQ : IJ Control bus2623
22.1.2	
22.1.3	SEQ to SP0 : Interpolator bus2623
22.1.4	SEQ to SP0 : Parameter Cache bus
00.1.5	2724
22.1.5	SEQ to SX0 : Parameter Cache Mux
control Bu	
22.1.6	SX0 to SP0 : Parameter Cache Return
bus	<u>2724</u> VGT to SP0/SEQ : Vertex Bus2724
22.1.7	VGT to SP0/SEQ . Vertex bus2/24

22.1.8	CP to SEQ : Constant store load2724
22.1.9	CP to SEQ : Fetch State store load
	<u>2724</u>
22.1.10	CP to SEQ : Control State store load 2824
22.1.11	MH to SEQ: Instruction store Load
22.1.11	2825
22.1.12	SP0 to SX0 : Pixel read from RBs2825
22.1.13	SEQ to SX0 : Control bus
22.1.14	SX0 to SEQ : Output file control2825
22.1.15	SP0 to SX0 : Position return bus2926
22.1.16	Shader Engine to Fetch Unit Bus (Fast
Bus)	2926
22.1.17	Sequencer to Fetch Unit bus (Slow
Bus)	2926
23. EAAIVII 3026	PLES OF PROGRAM EXECUTIONS
23.1.1	Sequencer Control of a Vector of
Vertices	3026
23.1.2	Sequencer Control of a Vector of
Pixels	3127
23.1.3	Notes
24. OPEN	ISSUES
1. OVER\	/IEW
	vel Block Diagram6
	ow graph8
	Graph
	POLATED DATA BUS
	UCTION STORE
	FANT STORE 12
	NG AND BRANCHES
	ntrolling_state
	ntrol Flow Program
	ependant predicate instructions
6.4 Registe	r file indexing16
	KILL-MASK
	URFACES
	TER FILE ALLOCATION
	ARBITRATION
	RBITRATION
	ING STALLS
	ENT OF THE RESERVATION STATION
FIFOS 19	UTPUT FILE
	MAT19
	ARAMETER CACHE
	X POSITION EXPORTING 20
	TING ARBITRATION
	TIME-COMMANDS
	TERS20
	rol
	ext
21. DEBUG	G-REGISTERS

Exhibit 2017.doc R400_Sequencer.doc 48659 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 00/04/15 12:53

AMD1044_0256881

	ORIGINATE DATE	EDIT DA	TE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September	20155	GEN-CXXXXX-REVA	3 of 32
22. INTER	FACES		9,	FETCH ARBITRATION	
22.1 Exter	mal-Interfaces		10.	ALU ARBITRATION	
22.1.1	PA/SC to SP0 : IJ bus		11.	HANDLING STALLS	
22.1.2	PA/SC to SEQ : U Contro	l bus22	12	CONTENT OF THE RESERVATION	STATION
22.1.3	VGT to SP : Vertex Bus		FIFOS	349	
22.1.4	CP to SEQ : Constant sto	re load 22	13	THE OUTPUT FILE	19
22.1.5	CP to SEQ · Fetch State e	tore load 22	14	J FORMAT	19
22.1.6	CP to SEQ Control State	-store-load	1.5	THE PARAMETER CACHE	
	23		16	VERTEX POSITION EXPORTING	20
2217	MH to SEQ: Instruction st	ore Load 23	17	REAL TIME COMMANDS	
22.1.8	SP to RB · Pixel read from	RBs 23	18	REGISTERS	
22.1.9	SEQ to RB : Control bus .	23	18.1	Control	
22.1.10			18.2	Context	
22.1.11	SP to RB Position return		10.2	DEBUG REGISTERS	
22112	-Shader Engine to Fetch U		20	INTERFACES	
Bus)	24		20.1	External Interfaces	
22.1.13	Sequencer to Fetch Unit t	US (Slow	20.1		
Bus)		100 (01011	20.1		
	NAL INTERFACES	24	20.1		
23.1.1	RE to SEQ : Vertex Contr		20.1		
APPENDIA FIA	PLES OF PROGRAM EXEC		20.1		
24.1.1	Sequencer Control of a V		20.1		
Vertices		50101-01	200-2		
24.1.2	Sequencer Control of a V	ector.of	20.1	Ann Ann	aload 22
Pixels		sator or	20.1		
24.1.3	Notes	27	20.1		
ALC 111.00.000	ISSUES		20.1		
	/IEW		20.1	- 1	
	vel Block Diagram		20.1		
	low graph		Bus	0	1000-(1-dot
	l Graph		20-1	,	e (Slow
	POLATED DATA BUS		Bus		- (310W
	UCTION STORE		21	INTERNAL INTERFACES	24
	ENCER INSTRUCTIONS		21_1		
	TANT STORE		22	EXAMPLES OF PROGRAM EXECU	
	NG AND BRANCHES		22.1		
	ntrolling state			tices 24	201-01
	entrol Flow Program		22.1		tar af
	ependant predicate instruction		Pixe		201-01
	ependant predicate instructions file indexing		PIX (26
	URFACES		23		
	URFACES		25.		
a	HER FILE ALLOGAHON				

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

STATION2020 .20 21 .21 21 ous.....21 load.....22 ore load...22 store load e Load ...22 RBs.....23 trol.....23 us.....23 it Bus (Fast is (Slow Bus.....24 JTIONS..24 stor of tor of

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)

Exhibit 2017.docR400_Sequencer.doc 46659 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C ***

PM10/19/01-01:40 PM

AMD1044_0256882

ATI Ex. 2105 IPR2023-00922 Page 210 of 239

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20155		4 of 32
Rev 0.6 (Laur		Cha	nged the spec to reflect the ne	ew R400
Date : Septen Rev 0.7 (Laur			itecture. Added interfaces. ed constant store management, i	notruction
Date : Octobe			e management, control flow manage	
	,		dependant predication.	
Rev 0.8 (Laur			nged the control flow method to	
Date : Octobe Rev 0.9 (Laur			ble. Also updated the external interface rporated changes made in the 10/18/0	
Date : Octobe			meeting. Added a NOP instruction,	
			conditional_execute_or_jump. Adde	d debug
Rev 1.0 (Laur	ent Lefebyre)		sters. ned interfaces to RB. Added state regis	atore
Date : Octobe		Ren	neu menaces to ND. Added state regi	51010.
Rev 1.1 (Laur			ed SEQ→SP0 interfaces. Chang	
Date : Octobe	<u>r 26, 2001</u>		sision. Changed VGT→SP0 interface hods added.	e. Debug
I		inicia	nous added.	

Exhibit 2017. doc R400_Sequencer.doc 46659 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 00/04/15 12:53

AMD1044_0256883

ATI Ex. 2105 IPR2023-00922 Page 211 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	7
24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	5 of 32	

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 station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose a neuclause 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 instruction thus there is only one sequencer for the whole chip.

Exhibit 2017.docR4400_Sequencer.doc 46659 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:53

AMD1044_0256884

ATI Ex. 2105 IPR2023-00922 Page 212 of 239

PROTECTIVE ORDER MATERIAL

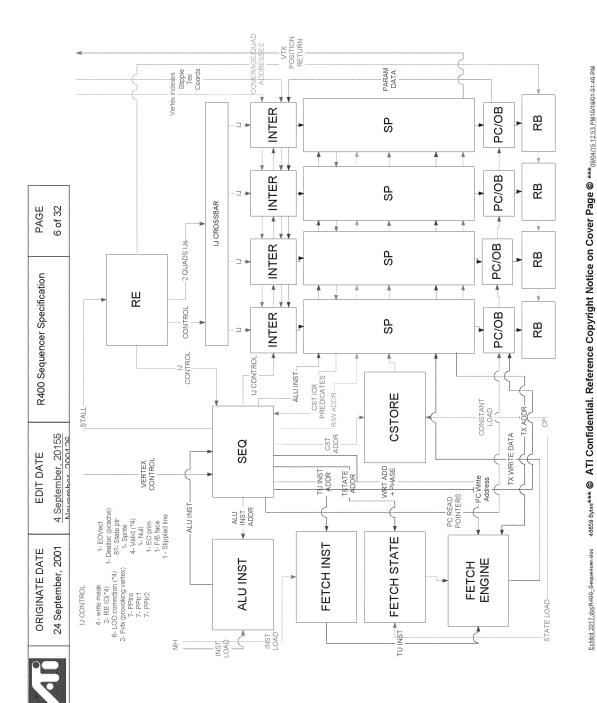
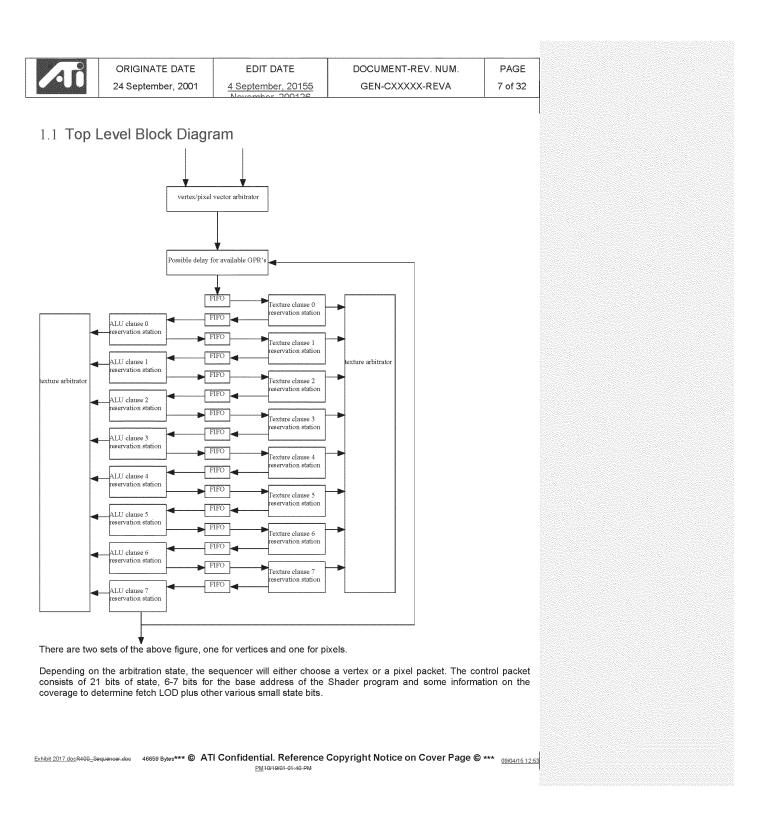


Exhibit 2017 doc/400_Sequences 400 48658 Bytes**** © ATI Confidential. Reference Copyright Notice C

AMD1044_0256885

ATI Ex. 2105 IPR2023-00922 Page 213 of 239



ATI Ex. 2105 IPR2023-00922 Page 214 of 239

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20155		8 of 32

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).

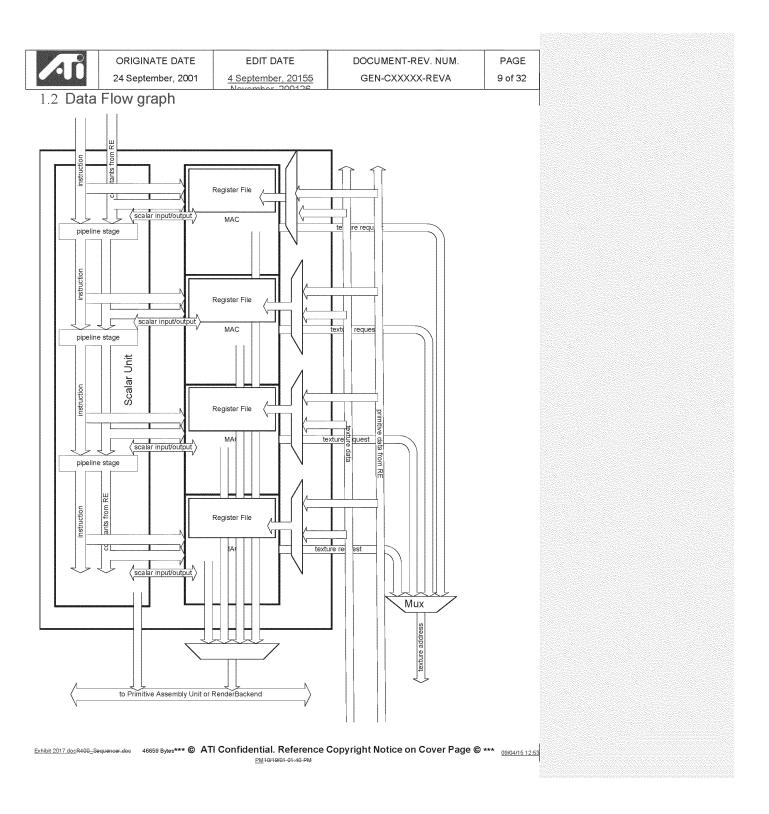
All other level process in the same way until the packet finally reaches the last ALU machine (7). On completion of the level 7 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that 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.

Exhibit 2017.docR400_Sequencer.doc 46669 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:53

AMD1044_0256887

ATI Ex. 2105 IPR2023-00922 Page 215 of 239

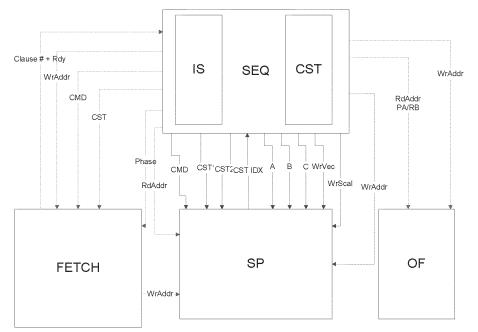


ATI Ex. 2105 IPR2023-00922 Page 216 of 239

2	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20155		10 of 32

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.

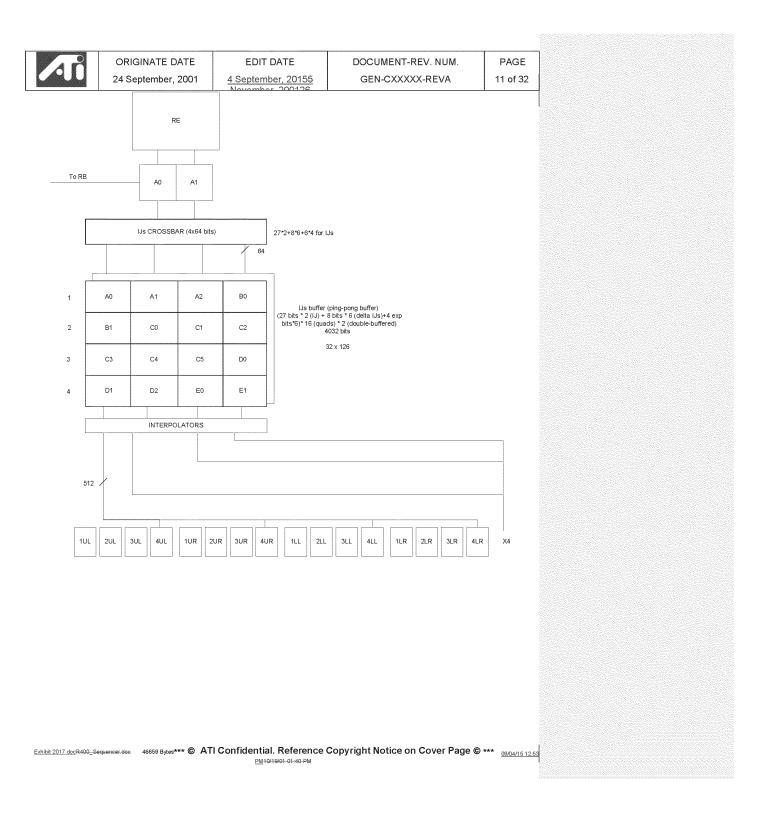
2. Interpolated data bus

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

Exhibit 2017.doc/R400_Sequencer.doc 46659 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:53

AMD1044_0256889

ATI Ex. 2105 IPR2023-00922 Page 217 of 239



ATI Ex. 2105 IPR2023-00922 Page 218 of 239 PROTECTIVE ORDER MATERIAL

	T19	5 ⁴ - 5	52- 55	56- 56-	63 - C	
	T18	32- 35	39 - 36 <	2 40 < <	> 44 74 74	
	T17	> 1 6	23 ² 0 <	24- 27	31 ²⁸ <	
	T16	°3 <	> 4 7-7	8-11 8-1	15 ⁻ 7 <	
	T15			Ш	Щ.	
ш S	T14				ß	
PAGE 12 of 32	T13		ပိ	5	3	
	T12				BO	2
fication	T11	5	D3			
r Specif	T10	S	5 C	C5		
Inence	Т9	<u>8</u>				
R400 Sequencer Specification	Т8	AO	A1	A2		
R2	4			ů	ω	
E 0155 0176	T6				â	
EDIT DATE	T5		ទ	δ	8	
4 September, 20155	Т4				BO	0
	13	5	6			0 d
E DATE er, 200	12	S	C4	C2		
ORIGINATE DATE 24 September, 2001	ŗ	<u>B</u>				
ORIC 24 Se	T0	AO	A1	A2		
P		SP0	SP1	SP2	SP3	

AMD1044_0256891

Exhibit 2017.docR400_Sequencer.doc

ATI Ex. 2105 IPR2023-00922 Page 219 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	13 of 32

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:

Exhibit 2017.docR400_Sequencer.doc 46659 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:35

PM10/18/01-01:40 PM

AMD1044_0256892

ATI Ex. 2105 IPR2023-00922 Page 220 of 239

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20155		14 of 32
Loop_Step[7:0 Loop_End[7:0]	D][15:0] •] [15:0] times 2 (one for c] [15:0] times 2 (one for c [15:0]	onstant,register)		
	Ū	ded and how many contex	ts do we have?} counters to allow for nested loops.	
			nese bits can be 0: all 0s, 1: all ones a	nd 11: mixed.
() The	Control Flow Dro	~~~~		

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 of	chosen for the R400 i	s a "control program". The control prog	gram has ten basic instructio	ons:	
	Execute_Predicates execute_or_Jump ump				
Conditional_c store to be ex Loop_start re condition and Loop_end into Call jumps to stack. Conditional_c End_of_claus	execute checks a con- cecuted. sets the correspond if met jumps over to rements (decrements an address and pus execute_or_Jump exe execute_Predicates e se marks the end of a umps jumps to an add	?) the loop counter and jumps back the hes the IP counter on the stack. On the cutes a block of instructions or jumps executes a block of instructions if all bit	pecified number of instruction on the first pass after it ch ne specified number of instru- ne return instruction, the IP to an address is the condition	necks for the end uctions. is poped from the on is not met.	
	ALL JUMPS MUST gn the jumps on even	JUMP TO EVEN CFP ADDRESSES. CFP addresses.	Thus the compiler must in	isert NOPs where	
set the debug	mp is logically bigger g registers. If an exec (clause) and set the d				
We have to f store.	it instructions into 48	bits in order to be able to put two cor	ntrol flow instruction per line	in the instruction	
		Execute			
47	46 42	41 24	23 12	11 0	
Exhibit 2017.docR400_	· · · · ·	© ATI Confidential. Reference Co		,	

PM10/18/01-01:40 PM

AMD1044_0256893

	ORIGINATE	DATE	EDIT DATE	۵	OCUMENT-REV. NUM	Л.	PAGE	10203020
CAUD	24 September	, 2001	4 September, 20155		GEN-CXXXXX-REVA		15 of 32	
Addressing	00001		RESERVED		Instruction _count	Exec	Address	100

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.

Exhibit 2017. doc R400_Sequences.doc 46659 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

AMD1044_0256894

ATI Ex. 2105 IPR2023-00922 Page 222 of 239

 A îi	ORIGINAT		4 Septe	T DATE mber, 20155	R400 Seq	uencer Specification	PAG 16 of 3	
(×			Co	nditionnal_Exe	cute			
47	46 42	41	34	33	32 24	23 12	11 0	1
Addressing	00011	Boolean a	ddress	Condition	RESERVED	Instruction_count	Exec Addre	ess

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				
	00101	RESERVED	Jump address	Loop ID				
Addressing								

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.

		L	.oop_End		
47	46 42	41 16	15 4	3 0	
	00111	RESERVED	Start address	Loop ID	
Addressing					

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	4112	11 0					
	01000	RESERVED	Address					
Addressing								

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

		Return	
47	46 42	41 0]
	01001	RESERVED	1
Addressing			

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
	01010	Boolean address	Condition	RESERVED	FW only	Address
Addressing						

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	

Exhibit 2017. doc R400_Sequencer.doc 46659 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

AMD1044_0256895

ATI Ex. 2105 IPR2023-00922 Page 223 of 239

	ORIGINATE 24 Septembe	 EDIT DATE <u>4 September, 20155</u> Nevember, 200126	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	PAGE 17 of 32
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 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 wiehwhich 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 exempleexample, 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?}

Exhibit 2017.docR400_Sequencer.doc 46659 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C ***

PM10/18/01 01:40 PM

AMD1044_0256896

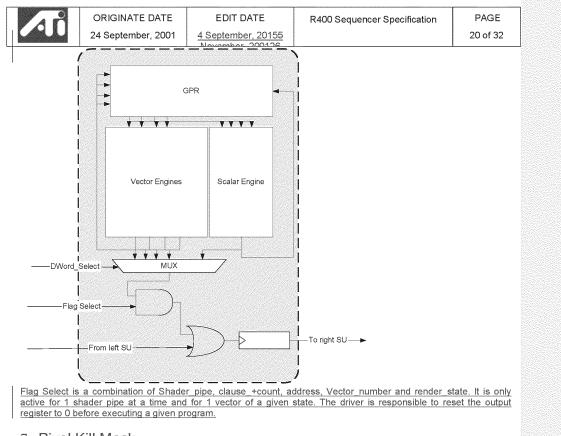
ATI Ex. 2105 IPR2023-00922 Page 224 of 239

	-				
	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 200	1 <u>4 September, 20155</u>		18 of 32	
<u>6.4 HW I</u>	Detection of P	V,PS		*	Formatted: Bullets and Numbering
			tically dependant instructions. In the		
			uses of PV,PS as needed. This with instructions. For masked writes, the		
	wherever there is a dep				
The sequence	er will also have to inse	rt NOPs between PRED_SET	and MOVA instructions and their use	<u>s.</u>	
6.4 <u>6.5</u> R	egister file inde	exing		4	Formatted: Bullets and Numbering
data created	in a fetch clause loop		to index into the register file in order e. The instruction will include the bas <u>ntrols:</u>		
	Bit7 Bit 6				
	0 0 0 1	'absolute register 'relative register'	1		
	1 0 1 1	'previous vector' 'previous scalar'			
			b. In the case of a relative register rea ur new address that we give to the shares		
66 Pred	icated Instruct	ion support for Tex	dure clauses	*	Formatted: Bullets and Numbering
***************************************			eep 1 bit (thus 4 bits for the four pred	licate vectors)	
			ns that one ore more elements in the e vector. A value of 0 means that no e		
		we can thus skip over the tex			
(Formatted: Bullets and Numbering
	ugging the Sha	aders		4	- Formatted: Bullets and Numbering
				4	Formatted: Bullets and Numbering
In order to be	able to debug the pixe	aders I/vertex shaders efficiently, we		+	Formatted: Bullets and Numbering
In order to be 6.7.1 <i>Met</i>	able to debug the pixe	aders I/vertex shaders efficiently, w ng registers	e provide 3 methods.	•-	
6.7.1 Met <u>Current plans</u>	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er	aders I/vertex shaders efficiently, w ng registers Iging, or error notification, reg	e provide 3 methods.	•	
In order to be 6.7.1 Met Current plans 1. address reg 2. count of the	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er a number of errors	aders I/vertex shaders efficiently, w ng registers Iging, or error notification, reg ror occurred	e provide 3 methods.	•	
In order to be 6.7.1 Met Current plans 1. address reg 2. count of the	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er e number of errors er will detect the followi	Aders I/vertex shaders efficiently, we ng registers Iging, or error notification, reg ror occurred	e provide 3 methods.	+	
In order to be <u>6.7.1</u> Met <u>Current plans</u> <u>1. address rec</u> <u>2. count of the</u> <u>The sequence</u> <u>- count overflo</u> <u>- jump error</u>	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er e number of errors er will detect the followi	Aders I/vertex shaders efficiently, we ing registers Iging, or error notification, reg ror occurred ng groups of errors:	e provide 3 methods.	•	
In order to be <u>6.7.1</u> Meta <u>Current plans</u> <u>1. address reg</u> <u>2. count of the</u> <u>2. count of the</u> <u>- count overfile</u> <u>- jump error</u> <u>relative jump</u> <u>relative jump</u>	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er e number of errors er will detect the followi ow address > size of the b address > length of th	Aders I/vertex shaders efficiently, we ng registers Iging, or error notification, reg ror occurred ng groups of errors: control flow program	e provide 3 methods.	•	
In order to be <u>6.7.1</u> Met <u>Current plans</u> <u>1. address reg</u> <u>2. count of the</u> <u>2. count of the</u> <u>count overflo</u> <u>-jump error</u> <u>relative jump</u> <u>relative jump</u> <u>relative jump</u> <u>relative jump</u> <u>relative jump</u>	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er e number of errors er will detect the followi ow o address > size of the following address > length of the erflow	Aders I/vertex shaders efficiently, we ng registers Iging, or error notification, reg ror occurred ng groups of errors: control flow program	e provide 3 methods.	•	
In order to be 6.7.1 Met Current plans 1. address reg 2. count of the The sequence - count overflo - jump error relative jump - constant ove	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er a number of errors er will detect the followi b address > size of the i b address > length of the erflow	Aders I/vertex shaders efficiently, we ng registers Iging, or error notification, reg ror occurred ng groups of errors: control flow program	e provide 3 methods.	•	
In order to be <u>6.7.1</u> Met <u>Current plans</u> <u>1. address reg</u> <u>2. count of the</u> <u>The sequence</u> <u>- count overfic</u> <u>- jump error</u> <u>relative jump</u> <u>- constant ove</u> <u>- register over</u> <u>- call stack</u>	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er e number of errors er will detect the followi o address > size of the o address > length of the erflow ck full	Aders I/vertex shaders efficiently, we ng registers Iging, or error notification, reg ror occurred ng groups of errors: control flow program	e provide 3 methods.	•	
In order to be 6.7.1 Met Current plans 1. address reg 2. count of the The sequence - count overfic - jump error relative jump - constant ove - register over - call stack call with stac - return with stac	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er a number of errors er will detect the followi ow o address > size of the address > length of the erflow rflow rflow ck full tack empty he errors, a jump erro	Aders I/vertex shaders efficiently, we ing registers Iging, or error notification, reg ror occurred ing groups of errors: control flow program ie shader program	e provide 3 methods. isters: cause the program to break. In this	←	
In order to be <u>6.7.1</u> Met <u>Current plans</u> <u>1. address rec</u> <u>2. count of the</u> <u>The sequence</u> <u>- count overflo</u> <u>- jump error</u> <u>relative jump</u> <u>relative jump</u> <u>relative jump</u> <u>relative jump</u> <u>- constant ove</u> <u>- register over</u> <u>- call stack</u> <u>call with stac</u> <u>return with stac</u> <u>return with stat</u> <u>with two of th</u> <u>means that a</u>	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er e number of errors er will detect the followi o address > size of the o address > length of the erflow chow charles tack empty the errors, a jump error clause will halt execution	Aders I/vertex shaders efficiently, we ing registers Iging, or error notification, reg ror occurred Ing groups of errors: <u>control flow program</u> ie shader program	e provide 3 methods. isters: cause the program to break. In this is to be executed.	case, a break	
In order to be <u>6.7.1</u> Met <u>Current plans</u> <u>1. address rec</u> <u>2. count of the</u> <u>The sequence</u> <u>- count overflo</u> <u>- jump error</u> <u>relative jump</u> <u>relative jump</u> <u>relative jump</u> <u>relative jump</u> <u>- constant ove</u> <u>- register over</u> <u>- call stack</u> <u>call with stac</u> <u>return with stac</u> <u>return with stat</u> <u>with two of th</u> <u>means that a</u>	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er e number of errors er will detect the followi o address > size of the o address > length of the erflow chow charles tack empty the errors, a jump error clause will halt execution	Aders I/vertex shaders efficiently, we ng registers Iging, or error notification, reg ror occurred ng groups of errors: control flow program le shader program e shader program in or a register overflow will on, but allowing further clause	e provide 3 methods. isters: cause the program to break. In this is to be executed.	←-	
In order to be 6.7.1 Met Current plans 1. address reg 2. count of the The sequence - count overfic - jump error relative jump - constant ove - register over - call stack call with stac - return	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er a number of errors er will detect the followi ow address > size of the following address > length of the erflow chack empty the errors, a jump error clause will halt execution r errors, program can c	Aders Nvertex shaders efficiently, we ng registers Iging, or error notification, reg ror occurred ng groups of errors: control flow program e shader program e shader program on, but allowing further clause ontinue to run, potentially to v	e provide 3 methods. isters: cause the program to break. In this es to be executed. vorst-case limits.		
In order to be <u>6.7.1</u> Met <u>Current plans</u> <u>1. address rec</u> <u>2. count of the</u> <u>The sequence</u> <u>- count overflo</u> <u>- jump error</u> <u>relative jump</u> <u>relative jump</u> <u>relative jump</u> <u>relative jump</u> <u>- constant ove</u> <u>- register over</u> <u>- call stack</u> <u>call with stac</u> <u>return with stac</u> <u>return with stat</u> <u>with two of th</u> <u>means that a</u>	able to debug the pixe thod 1: Debuggin are to expose 2 debug gister where the first er a number of errors er will detect the followi ow address > size of the following address > length of the erflow chack empty the errors, a jump error clause will halt execution r errors, program can c	Aders Nvertex shaders efficiently, we ng registers Iging, or error notification, reg ror occurred ng groups of errors: control flow program e shader program e shader program on, but allowing further clause ontinue to run, potentially to v	e provide 3 methods. isters: cause the program to break. In this is to be executed.		

ATI Ex. 2105 IPR2023-00922 Page 225 of 239

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	19 of 32	
	. This could be exploited t	causing an overflow error, I	he hardware is specified to return th by reserving and initializing the Oth		
	hod 2: Exporting the	values in the CDD	s (12)	*-	
			s [12] r this mode and 3 bits per clause sp	pecifying the	
	le for each clause. The mod				
	<u>1) Normal</u> 2) Debug Kill			4	Formatted: Bullets and Numbering
	 <u>3) Debug Addr</u> <u>4) Debug Count</u> 				
	mal mode execution follow		er the kill mode, all control flow inst ed by NOPs. Only debug export in		
clause 7 will b	e executed under the debu	g kill setting. Under the two	other modes, normal execution is d	one until we	
			n count (useful for loops) specified b kill mode for the rest of the clause.	by the count	
					Formatted: Bullets and Numbering
<u>6.7.3 Met</u>	hod 3: Selective exp	port of a 32 bit Dwoi	<u>rd.</u>	4-	·
			ode, the sequencer will keep the follo bits), clause_+count (16 bits?),addre		
Vector numbe	er (8 bits), Render_state (21	1 bits). The shader pipe rec	ister selects a shader pipe amongs use +count selects at which clause	t the 64, the	
count we exp	ort, the Render state spec	ifies which render state is	concerned, the Vector number spe		
vector is conce	erned and the mode selects	count export, or address e	<u>kport.</u>		
Exhibit 2017.docR400_Se	quencer.doc 46659 Bytes*** ⓒ AT	I Confidential. Reference	Copyright Notice on Cover Page ©) *** 09/04/15 12·53	
	-	PM10/18/01-01:40 PM	_ 0	<u></u>	

ATI Ex. 2105 IPR2023-00922 Page 226 of 239



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

Exhibit 2017.docR4400_Sequencer.doc 46659 Bytes***
ATI Confidential. Reference Copyright Notice on Cover Page

99/04/15 12:53
PM140/4901-01:40-PM

AMD1044_0256899

ATI Ex. 2105 IPR2023-00922 Page 227 of 239

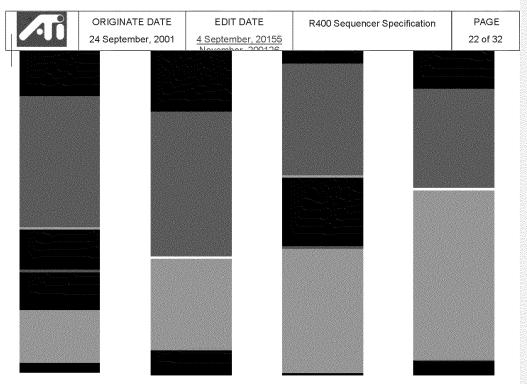
ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	21 of 32
tices is allowed to move,		to VERTEX_REG_SIZE for vertice	s and 256-

VERTEX_REG_SIZE for pixels.

Exhibit 2017. docR400_Sequencer.doc 46659 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page ***

AMD1044_0256900

ATI Ex. 2105 IPR2023-00922 Page 228 of 239



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.

Exhibit 2017.docR4400_Sequencer.doc 46659 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** <u>PM104801-01:40 PM</u>

AMD1044_0256901

ATI Ex. 2105 IPR2023-00922 Page 229 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	23 of 32	

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 49x24-20x24 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).

$\Delta 01I = I(1) - I(0)$		
$\Delta 01J = J(1) - J(0)$		
$\Delta 02I = I(2) - I(0)$	P0	P1
$\Delta 02J = J(2) - J(0)$		
$\Delta 03I = I(3) - I(0)$		
$\Delta 03J = J(3) - J(0)$	P2	P3
$P0 = C + I(0)^* (A - C) + J(0)^* (B - C)$		
$P1 = P0 + \Lambda 01I^{*}(A - C) + \Lambda 01J^{*}(B - C)$		

 $P2 = P0 + \Delta 02I * (A - C) + \Delta 02J * (B - C)$ $P3 = P0 + \Delta 03I * (A - C) + \Delta 03J * (B - C)$ P0 is computed at 49x24-20x24 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.

Exhibit 2017. docR400_Sequencer.doc 46659 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15 12:53

AMD1044_0256902

ATI Ex. 2105 IPR2023-00922 Page 230 of 239

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20155		24 of 32
Multiplion (Full	Provinian): 2			

Multiplies (Full Precision): 2 Multiplies (Reduced precision): 6 Subtracts 19x24 (Parameters): 2 Adds: 8

FORMAT OF P0's IJ : Mantissa <u>19-20</u> Exp 4 for I + Sign Mantissa <u>19-20</u> 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.
- 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 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.

Exhibit 2017 doc R400_Sequencer.doc 46659 Bytes*** C ATI Confidential. Reference Copyright Notice on Cover Page C *** 09/04/15 12:53

AMD1044_0256903

ATI Ex. 2105 IPR2023-00922 Page 231 of 239

OPCINICATE DATE 24 September, 2001 EDIT DATE 4 September, 2003 DOCUMENT-REV. NUM. GEN-CXXXXX-REVA PAGE 25 of 32 20. Registers 20.1 Control DYNAMIC, REG VENTEX, REG, SIZE What portion of the register file is reserved for vertices (static allocation only) Water CARLO, Delive VENTEX, MIN, SIZE VENTEX, MIN,						
20. Registers 20.1 Control DYNAMIC_REG_SIZE Dynamic allocation (pixel/vertex) of the register file on or off. VERTEX_REG_SIZE What portion of the register file is reserved for vertices (state allocation only) VERTEX_REG_SIZE What portion of the register file is reserved for vertices (state allocation only) VERTEX_REG_SIZE What portion of the argister file is provide (state allocation only) VERTEX_REG_SIZE What portion of the argister file is provide (state allocation only) VERTEX_REG_SIZE What portion of the argister file is provide (state allocation only) VERTEX_REG_SIZE What portion of the argister file is provide (state allocation only) VERTEX_REG_SIZE What portion of the argister file is provide (state allocation only) VERTEX_REG_SIZE Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the vertice state (state allocation where each clauses control program is located orght 8 bit pointers to the location where each clauses control program is located pith 8 bit pointers to the location where each clauses control program is located backer of the vertice state or program is located pith 8 bit pointers to the location where each clauses control program is located backer of the vertice state or program is located backer of the vertice state or program is located backer of the vertice state of the vertice stated for finstructions in control program.2) </td <td>ORIGINATE</td> <td>DATE</td> <td>EDIT DATE</td> <td>DOCUMENT-REV. NUM.</td> <td>PAGE</td> <td></td>	ORIGINATE	DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
20. Registers 20.1 Control DYNAMIC_REG Dynamic allocation (pixelverts) of the register file is reserved for vertices (static allocation only) VERTEX_REG_SUZE Winnamia size of the register file's pixel portion (dynamic only) VERTEX_NIN_SIZE Minimal size of the register file's pixel portion (dynamic only) VERTEX_NIN_SIZE Minimal size of the constant store for pixels CST_SIZE_V Size of the constant store for vertices CST_SIZE_V Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant store for vertices Size of the constant st	24 September	er, 2001		GEN-CXXXXX-REVA	25 of 32	
DYNAMIC_REG Dynamic allocation (pixel/vertex) of the register file on or off. VERTEX_REG_SIZE What portion of the register file is preaved for vertices (static allocation only) MRX_DIN_SIZE What portion of the register file is preaved for vertices (static allocation only) MRX_DIN_SIZE What portion of the register file is preaved for vertices Size of the constant store for prices Size of the constant store for prices CST_SIZE_V Size of the constant store for vertices INST_STARCE_VRAP wrapsing point for the vertexplied ishader instruction store (vertex_shader always statis at 0) PWRAPSHARE_DVRAP wrapsing point for the pixel ishadershared instruction store (vertex_shader file) for pixel ishader for file) for pixel ishader for program is located debug state register. Only allows one program at a time into the GPRs 20.2 Context eight 8 bit pointers to the location where each clauses control program is located base pointer for the vertex shader (file) for pixel ishader for program is located base pointer for the vertex shader (file) for pixel ishader program 2) Phader_static size of the vertex shader (file) instructions) Phader_static mumber of registers to allocate for vertex shader program 2) Vshader_crite_file_01.51 work and crite instruction sortid program 2) Vshader_static size of the vertex shader (file) instructions) Primeters size of the v	20. Registers					
DYNAMIC_REG_VIETEX_REG_SIZE Dynamic allocation (pixel/vertex) of the register file on or off. VERTEX_REG_SIZE What portion of the register file by portion (dynamic only) MRIDERATION_policy Size of the constant store for yearbics Size of the constant store for yearbics and with the antificity of the register file by reserved in weights Size of the constant store for yearbics Size of the constant store for yearbics Wark particular to the antificition between valexes/particles and function store (vertex_shader WARAPEMAREDWRAP wrapstart point for the vertexplied shader instruction store (vertex_shader PWRAPEMAREDWRAP wrapstart point for the yearts shader and the end of the IMI NO_INTERLEAVE debug state register. Only allows one program at a time into the GPRs 20.2 Context vspatart gift 8 bit pointers to the location where each clauses control program is located base pointer for the yearts shader (difference) Vshader_stat/701701 eight 8 bit pointers to the location where each clauses control program is located base pointer for the yearts shader (difference) Vshader_stat/701701 eight 8 bit pointers to the location where each clauses control program is located base pointer for the yearts shader (difference) Vshader_stat/701701 eight 8 bit pointers to the location where each clauses control program is located base pointer for the yearts shader (difference) Vshader_stat/701701 <td>20.1 Control</td> <td></td> <td></td> <td></td> <td></td> <td></td>	20.1 Control					
VERTEX_RES_GSZE What portion of the register file is vertex portion (dynamic only) VREL_MIN_SZE Minimal size of the register file is vertex portion (dynamic only) VRETEX_MIN_SZE Minimal size of the register file is vertex portion (dynamic only) VRETEX_MIN_SZE Minimal size of the register file is vertex portion (dynamic only) VRETEX_MIN_SZE Minimal size of the register file is vertex portion (dynamic only) VRETEX_MIN_SZE Visit portion between vertexes INST_STOR_ALLOR vargetart store for vertexes INST_STOR_ALLOR vargetart point for the vertex shader Vargetar_UNRAP start point for the vertex shader		Dynamic	allocation (pixel/vertex) of th	ne register file on or off		
20.2 Context Vshader_fetch/?017:01 Vshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fetch/?017:01 Pshader_fisze eight 8 bit pointers to the location where each clauses control program is located base pointer for the vertex shader (# of instructions in control program/2) Size of the vertex shader (# of instructions in control programs size of the vertex shader (# of instructions) number of registers to allocate for pixel shader (most or vertex shader programs FLAT_GOUR[015] REG_ALLOC_VERT Number of registers to allocate for pixel shader for or not CYL_WRAP[063] Do we generate texture coordinates for 1 th parameters or not CYL_WRAP[063] P_export_mode Doxxx : Normal mode trxxx : Nutlipass mode If normal, bbzz where bb is how many colors (0-4) and z is export z or not If muttipass 1-12 exports for color. vshader_export_mode 0: position (1 vector), 1: position (2 vectors), 3: muttipass vshader_export_mode kil_vector_on # of interpolated parameters exported in clause 7 OR # of exported vectors to memory per clause in muttipass mode (per clause) use the mask kill vector to kill pixels and optimize texture pipe fetches OR use it as the fifth proficate vector wich which is the only predicate vector kept across clause boundaries.	VERTEX_REG_SIZE PIXEL_MIN_SIZE VERTEX_MIN_SIZE ARBITRATION_policy CST_SIZE_P CST_SIZE_V INST_STOR_ALLOC WWRAP <u>PWRAP</u> always starts at 0) PWRAPSHAREDWRAP_ RTWRAP	What por Minimal s policy of Size of th Size of th interleave start poir	tion of the register file is res ize of the register file's pixe ize of the register file's verte the arbitration between vete the constant store for pixels the constant store for vertexe ad, separate, interleaved+sh wrap <u>start</u> point for the verte wrap <u>start</u> point for the t t for the RT instruction store	erved for vertices (static allocation of l portion (dynamic only) ex portion (dynamic only) xes <u>vertexes</u> and pixels s lared,separate+shared ex <u>pixel</u> shader instruction store <u>(ve</u> e pixel-shader <u>shared</u> instruction stor <u>t (RT always ends at the end of the li</u>	r <u>tex shader</u>	
Vshader_etch[7:0][7:0] eight 8 bit pointers to the location where each clauses control program is located Pshader_etch[7:0][7:0] eight 8 bit pointers to the location where each clauses control program is located Pshader_etch[7:0][7:0] eight 8 bit pointers to the location where each clauses control program is located Pshader_etch[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 (# of instructions in control program)2) Pshader_etcl_size size of the pixel shader (for linstructions) Vshader_entl_size size of the pixel shader (for linstructions) Vshader_etcl_size size of the pixel shader (for linstructions) Vshader_etcl_size size of the pixel shader (for linstructions) Vshader_etcl_size size of the pixel shader (for linstructions) REG_ALLOC_PIX number of registers to allocate for pixel shader programs REG_ALLOC_PIX number of registers to allocate for vixel shader (structons) REG_ALLOC_PIX Do we generate texture coordinates for 1 th parameter or not GEN_TEX Do we generate texture coordinates is exports or color. vshader_export_mode Oxxx : Normal mode tin rormal, bbz where bb is how many colors (0-4) and z is export z or not tif normal, bbz where b	NO_INTERLEAVE	debug st	ate register. Only allows one	program at a time into the GPRs		
Wehade_alu[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_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 vertex shader (cntl+instructions) Vshader_size size of the vertex shader programs REG_ALLOC_PIX number of registers to allocate for pixel shader VWRAP[063] for wich-which parameters are to be gouraud shaded GEN_TEX Do we generate texture coordinates for 1 st parameter or not CYL_WRAP[063] for wich-which parameters (and channels (xyzw)) do we do the cyl wrapping. Pexport_mode Oxxxx : Normal mode txxx : Multipass node tho other pointer or the memory per clause in multipass vshader_export_mode 0: position (1 vector), 1: position (2 vectors), 3:multipass vshader_export_mode 0: position (1 vector), 1: position (2 vectors), 3:multipass vshader_export_mode # of interpolated parameters exported in	20.2 Context					
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. wichwhich of the last 6 ALU clauses is exporting 0: position (1 vector), 1: position (2 vectors), 3:multipass vshader_export_count[6] Image: Color C	Vshader_fetch[7:0][7:0] Vshader_alu[7:0][7:0] Pshader_fetch[7:0][7:0] Pshader_alu[7:0][7:0] PSHADER VSHADER VSHADER Vshader_cntl_size Pshader_cntl_size Pshader_size REG_ALLOC_PIX REG_ALLOC_VERT FLAT_GOUR[015] GEN_TEX	eight 8 bi eight 8 bi eight 8 bi base poin size of th size of th size of th size of th number of wich <u>whic</u> Do we ge	t pointers to the location whit t pointers to the location whit t pointers to the location whit ter for the pixel shader inter for the vertex shader e vertex shader (# of instruction e pixel shader (cntl+instruction) of registers to allocate for pix of registers to allocate for ve h parameters are to be gout innerate texture coordinates to	ere each clauses control program is ere each clauses control program is ere each clauses control program is tions in control program/2) ions in control program/2) ions) tions) tel shader programs rtex shader programs raud shaded ior 1 st parameter or not	located located located	
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 <u>which</u> is the only predicate vector kept across clause boundaries. 21. DEBUG registers 21.1 Control Shader pipe # of the shader pipe for method 3 (064) Count +clause instruction count and clause number for method 3	vshader_export_mask vshader_export_mode	1xxxx : N If normal If multipa wich <u>whic</u> 0: positio	lultipass mode , bbbz where bbb is how ma ss 1-12 exports for color. <u>h</u> of the last 6 ALU clauses i n (1 vector), 1: position (2 ve	s exporting ectors), 3:multipass	ot	
21.1 Control Formatted: Bullets and Numbering Shader pipe # of the shader pipe for method 3 (064) Count +clause instruction count and clause number for method 3		use the r the fifth p	nask kill vector to kill pixels a redicate vector wich <u>which</u> is	and optimize texture pipe fetches OR	use it as	
21.1 Control Formatted: Bullets and Numbering Shader pipe # of the shader pipe for method 3 (064) Count +clause instruction count and clause number for method 3	21. DEBUG registe	rs				
Shader_pipe # of the shader pipe for method 3 (064) Count_+clause instruction count and clause number for method 3						
Count +clause instruction count and clause number for method 3		# of the -	hadar aina far mathad 2 (0	64)	-	
Dword_select Channel select for method 3	Count_+clause	instructio	n count and clause number			

ATI Ex. 2105 IPR2023-00922 Page 232 of 239

24 Septe	ATE DATE EI	DIT DATE	R400 Sequencer Specification	PAGE	
A Design and the second s		ember, 2015		26 of 32	
Mode	operating mode	for method 3			
Rstate	render state met				
Vector_count	vector number th	ie method 3 v	will export		Contraction of the second structure of the second stru
21.2 Context				*	Formatted: Bullets and Numbering
	instruction oddre	oo whoro the	first problem accurred		
PROB_ADDR PROB_COUNT			e first problem occurred ered during the execution of the program		
Count	instruction count				
Clause_mode[3]	clause mode for	debug metho	pd 2		
2. Interfaces					
.2. <u>Interfaces</u>					
22.1 External Int	erfaces				
22.1.1 PA/SC to S	HU: IJ DUS				
			the top of each shader pipe. At the same tir	me the control	
information goes to the	sequencer. There are 4	of these buse	es over the whole chip (SP0 thru 3)		
Vame	Direction	Bits	Description		
Js	PA→SP0	643	IJ information sent over 2 clocks		
Mask	PA→SP0	1	Write Mask		
22.1.2 PA/SC to S This is the control informa execute a shader program	ation sent to the sequer		to control the IJ fifos and all other informati	on needed to	
Fhis is the control informa execute a shader program	ation sent to the sequer n on the sent pixels.	cer in order t	Description	on needed to	
This is the control informa execute a shader program Name Write Mask	ation sent to the sequer n on the sent pixels. Direction PA→SEQ(SP)	cer in order t Bits 4	Description Quad Write mask left to right		
This is the control informa execute a shader program lame Write Mask .OD_CORRECT	ation sent to the sequer n on the sent pixels. Direction PA→SEQ(SP) PA→SEQ(SP)	cer in order t Bits 4 24	Description Quad Write mask left to right LOD correction per quad (6 bits per quad)		
This is the control informative execute a shader program lame Vrite Mask .OD_CORRECT TVTX	ation sent to the sequer n on the sent pixels. Direction PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP)	Bits 4 24 2	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading		
This is the control informative xecute a shader program lame Vrite Mask OD_CORRECT VTX PTR0	ation sent to the sequer n on the sent pixels.	Bits 4 24 2 11 1	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0		
This is the control informative execute a shader program write Mask OD_CORRECT VTX PTR0 PTR1	tion sent to the sequer n on the sent pixels.	Bits 4 24 2 11 11	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1		
This is the control informative execute a shader program lame Vrite Mask OD_CORRECT VTX PTR0 PPRT1 PPRT1 PPRT2	tion sent to the sequer n on the sent pixels. Direction PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP)	Bits 4 24 2 11 11 11 11	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2		
This is the control informative execute a shader program Vrite Mask OD_CORRECT VTX PPTR0 PPRT1 PPTR2 COFF_VECTOR	tion sent to the sequer n on the sent pixels.	Bits 4 24 2 11 11 11 11 11	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector		
This is the control informative cute a shader program lame Vrite Mask OD_CORRECT VTX PPTR0 PPR1 PPR2 _OFF_VECTOR DEALLOC	tion sent to the sequer n on the sent pixels.	Bits 4 24 2 11 11 11 1 1 1	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store		
This is the control informative cute a shader program lame Vrite Mask OD_CORRECT VTX PPTR0 PPR11 PPTR1 PPTR2 OFF_VECTOR DEALLOC STATE	tion sent to the sequer n on the sent pixels.	Bits 4 24 2 11 11 11 1 12 1	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3)		
This is the control informative execute a shader program Virte Mask COD_CORRECT VTX PPTR0 PPR1 PPTR2 COFF_VECTOR DEALLOC TATE VALID	tion sent to the sequer n on the sent pixels.	Bits 4 24 21 11 11 11 11 11 11 11 11 11 11 11 11	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store		
This is the control informative execute a shader program lame Vrite Mask COD_CORRECT VTX VTX VPTR0 PPTR1 PPTR2 	tion sent to the sequer n on the sent pixels. Direction PA→SEQ(SP) PA	Bits 4 24 2 11 11 11 11 11 11 11 11 11 11 11 11 1	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels		
This is the control informa execute a shader program Varne Vrite Mask COD_CORRECT VTX PPTR0 PPTR0 PPTR0 PPTR2 E_OFF_VECTOR DEALLOC STATE VALID VULL E_OFF_PRIM -BFACE	tion sent to the sequer n on the sent pixels. Direction PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP) PA→SEQ(SP)	Bits 4 24 2 11 11 11 11 11 11 11 11 11 11 11 11 1	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels Null Primitive (for PC deallocation purpose End Of the primitive Front face = 1, back face = 0	:S)	
This is the control informa execute a shader program Varie Mask .OD_CORRECT .VTX PPTR0 .OPTR0 .OPTR2 .OFF_VECTOR 	tion sent to the sequer n on the sent pixels. Direction PA→SEQ(SP) PA	Bits 4 24 2 11 11 11 1 1 1 21 16 1 3	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels Null Primitive (for PC deallocation purpose End Of the primitive	:S)	
This is the control informa execute a shader program Varie Mask .OD_CORRECT .VTX PPTR0 .OPTR0 .OPTR2 .OFF_VECTOR 	tion sent to the sequer n on the sent pixels. Direction PA→SEQ(SP) PA	Bits 4 24 1 11 11 11 1 11 1 11 1 11 1 11 1 11 1 11 1 1 1 1 1 16 1 1 3	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels Null Primitive (for PC deallocation purpose End of the primitive Front face = 1, back face = 0 Stippled line and Real time command need cords from alternate buffer 000 : Normal 001 : Stippled line 011 : Real Time	:S)	
This is the control informa execute a shader program Varne Vrite Mask COD_CORRECT VTX PPTR0 PPTR0 PPTR0 PPTR2 E_OFF_VECTOR DEALLOC STATE VALID VULL E_OFF_PRIM -BFACE	tion sent to the sequer n on the sent pixels. Direction PA→SEQ(SP) PA	Bits 4 24 1 11 11 11 1 11 1 11 1 11 1 11 1 11 1 11 1 1 1 1 1 16 1 1 3	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels Null Primitive (for PC deallocation purpose End of the primitive Front face = 1, back face = 0 Stippled line and Real time command need cords from alternate buffer 000 : Normal 001 : Stippled line 001 : Stippled line 011 : Real Time 100 : Line AA	:S)	
This is the control informa execute a shader program Write Mask _OD_CORRECT =VTX =PPTR0 =PPTR0 =PPTR1 =PPTR2 E_OFF_VECTOR DEALLOC STATE VALID VULL =_OFF_PRIM =BFACE	tion sent to the sequer n on the sent pixels. Direction PA→SEQ(SP) PA	Bits 4 24 1 11 11 11 1 11 1 11 1 11 1 11 1 11 1 11 1 1 1 1 1 16 1 1 3	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels Null Primitive (for PC deallocation purpose End Of the primitive Front face = 1, back face = 0 Stippled line and Real time command need cords from alternate buffer 000 : Normal 001 : Stippled line 011 : Real Time 100 : Line AA 101 : Point AA	:S)	
This is the control informa execute a shader program Write Mask .OD_CORRECT VTX PPTR0 PPTR1 PPTR2 _OFF_VECTOR DEALLOC STATE ALLOC STATE ALLO VULL _OFF_PRIM BFACE TYPE	tion sent to the sequer n on the sent pixels. Direction PA→SEQ(SP) PA	Bits 4 24 1 11 11 11 1 11 1 11 1 11 1 11 1 11 1 11 1 1 1 1 1 16 1 1 3	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels Null Primitive (for PC deallocation purpose End of the primitive Front face = 1, back face = 0 Stippled line and Real time command need cords from alternate buffer 000 : Normal 001 : Stippled line 001 : Stippled line 011 : Real Time 100 : Line AA	:S)	
This is the control informa execute a shader program Write Mask OD_CORRECT VTX PPTR0 PPTR0 PPTR2 E_OFF_VECTOR DEALLOC BTATE ALLOC STATE ALLO STATE COFF_PRIM BFACE TYPE	tion sent to the sequer n on the sent pixels. PA→SEQ(SP) PA→	Bits 4 24 11 11 11 11 1 11 1 11 1 11 1 13 1 14 1 15 1 16 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels Null Primitive (for PC deallocation purpose End Of the primitive Front face = 1, back face = 0 Stippled line and Real time command need cords from alternate buffer 000 : Normal 001 : Stippled line 011 : Real Time 102 : Line AA 101 : Point AA 110 : Sprite	:S)	
This is the control informative execute a shader program Iame Vrite Mask COD_CORRECT VTX PTR0 PTR0 PTR2 COFF_VECTOR DEALLOC STATE VALID IULL _OFF_PRIM BFACE YPE RTRn RTRn RTS	tion sent to the sequer n on the sent pixels. Direction PA→SEQ(SP)	Bits 4 24 1 11 11 11 1 11 1 11 1 11 1 11 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels Null Primitive (for PC deallocation purpose End Of the primitive Front face = 1, back face = 0 Stippled line and Real time command need cords from alternate buffer 000 : Normal 001 : Stippled line 011 : Real Time 100 : Line AA 101 : Point AA 101 : Sprite Stalls the PA in n clocks	:S)	Formatted: Bullets and Numbering
This is the control informa	tion sent to the sequer n on the sent pixels. Direction PA→SEQ(SP)	Bits 4 24 1 11 11 11 1 11 1 11 1 11 1 11 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Description Quad Write mask left to right LOD correction per quad (6 bits per quad) Provoking vertex for flat shading P Store pointer for vertex 0 P Store pointer for vertex 1 P Store pointer for vertex 2 End of the vector Deallocation token for the P Store State/constant pointer (6*3+3) Valid bits for all pixels Null Primitive (for PC deallocation purpose End Of the primitive Front face = 1, back face = 0 Stippled line and Real time command need cords from alternate buffer 000 : Normal 001 : Stippled line 011 : Real Time 100 : Line AA 101 : Point AA 101 : Sprite Stalls the PA in n clocks	:S)	Termatted: Bullets and Numbering

ATI Ex. 2105 IPR2023-00922 Page 233 of 239

	ATE DATE ED	IT DATE	DOCUMENT-REV. NUM.	PAGE	
24 Septe		ember, 201		27 of 32	
			000 : Normal 001 : Stippled line/Poly 011 : Real Time 100 : Line AA 101 : Point AA		
			110 : Sprite		- Formatted
FVTX	<u>SEQ→SPQ</u>		Provoking vertex for flat shading		Formatted
FLAT_GOURAUD	SEQ→SP0		Flat or gouraud shading Wich parameter needs to be cylindrical wra		Formatted
CYL_WRAP Inter Write Register Index	SEQ→SP0 SEQ→SP0		Index into Register file for write of Interpolated	-dandamana	Formatted
Write Mask	SEQ→SP0		Quad Write mask left to right		
IJ Line number	SEQ→SP0		Line in the IJ buffer to use to interpolate		Formatted
Swap Buffers	SEQ→SP0		Swap the IJ buffers at the end of the interp	olation	
Param_0	SEQ→SP0	1 1	We are interpolating parameter 0		
22.1.4 SEQ to SP) : Parameter Cac	he bus		4-	
Name	Direction		Description		
Ptr1	SEQ→SP0		Pointer of PC (7 LSBs of Pointer)		
Ptr2	SEQ→SP0		Pointer of PC (7 LSBs of Pointer)		
Ptr3	<u>SEQ→SP0</u>	Z	Pointer of PC (7 LSBs of Pointer)		
22.1.5 SEQ to SX) : Parameter Cac	he Mux	control Bus	*-	[Formatted: Bullets and Numbering
Name	Direction		Description		
Mux1	SEQ→SXQ		Mux control for PC (4 MSbs of Pointer)		Formatted
Mux2	SEQ→SX0	Matt	Mux control for PC (4 MSbs of Pointer)	ľ	Formatted
Mux3	<u>SEQ→SX0</u>	4	Mux control for PC (4 MSbs of Pointer)	P	Formatted
22.1.6 SX0 to SPC	: Parameter Cach	ne Retui	rn bus	*	Formatted
Name	Direction	Bits	Description		Formatted
Vtx data 1	SX0→SP0		Vertex data to interpolate		Formatted: Bullets and Numbering
Vtx_data_2	SX0→SP0	128	Vertex data to interpolate		Tomated. Dules and Numbering
Vtx_data_2	E0033338500383836038385000	128			Tomaced, buiets and Numbering
<u>Vtx_data_2</u>	SX0→SP0 SX0→SP0 to SP <u>0/SEQ</u> : Ver	128 128 tex Bus	Vertex data to interpolate Vertex data to interpolate		Formatted: Bullets and Numbering
<u>Vtx_data_2</u> Vtx_data_3 22.1.3 22.1.7_VGT Name	SX0→SP0 SX0→SP0 to SP <u>0/SEQ</u> : Ven Direction	128 128 tex Bus	Vertex data to interpolate Vertex data to interpolate Description		
<u>Vtx_data_2</u> Vtx_data_3 22.1.322.1.7_VGT Name Vertex indexes	SX0→SP0 SX0→SP0 to SP <u>0/SEQ</u> : Ver	128 128 tex Bus Bits 128	Vertex data to interpolate Vertex data to interpolate		
<u>Vtx_data_2</u> Vtx_data_3 22.1.3 22.1.7_VGT Name	SX0→SP0 SX0→SP0 to SP <u>0/SEQ</u> : Ver Direction VGT→RESP0	128 128 tex Bus Bits 128 1	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface informatic		
Vtx_data_2 Vtx_data_3 22.1.322.1.7_VGT Name Vertex indexes EOF_vector Inputs_vert	SX0→SP0 SX0→SP0 to SP0/SEQ : Ver Direction VGT→RESP0 VGT→RESP0/SEQ VGT→RESP0/SEQ	128 128 128 128 bits 1 128 1 1 1	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface informatic End of the vector 0: Normal 128 bits per vert 1: double 256 bits per vert		
Vtx_data_2 Vtx_data_3 22.1.322.1.7_VGT Name Vertex indexes EOF_vector Inputs_vert	SX0→SP0 SX0→SP0 to SP0/SEQ : Ven Direction VGT→RESP0/SEQ	128 128 128 128 bits 1 128 1 1 1	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface information End of the vector 0: Normal 128 bits per vert		
Vtx data 2 Vtx data 3 22.1.322.1.7 VGT Name Vertex indexes EOF_vector Inputs_vert STATE 22.1.422.1.8 CP to	SX0→SP0 SX0→SP0 to SP0/SEQ : Ven Direction VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→SEQ	128 128 128 128 Bits 1 128 1 1 1 21 1 store 108	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface informatic End of the vector D: Normal 128 bits per vert 1: double 256 bits per vert Render State (6*3+3 for constants)		
Vtx_data_2 Vtx_data_3 22.1.322.1.7_VGT Name Vertex_indexes EOF_vector Inputs_vert STATE 22.1.422.1.8_CP to Name	SX0→SP0 SX0→SP0 to SP0/SEQ : Ven Direction VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→SEQ VGT→SEQ DSEQ : Constant s Direction	128 128 128 128 Bits 1 128 1 1 1 21 1 store /oa Bits	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface information End of the vector 0: Normal 128 bits per vert 1: double 256 bits per vert Render State (6*3+3 for constants) ad Description		Formatted: Bullets and Numbering
Vtx_data_2 Vtx_data_3 22.1.322.1.7_VGT Name Vertex indexes EOF_vector Inputs_vert STATE 22.1.422.1.8_CP to Name Constant Address	SX0→SP0 SX0→SP0 to SP0/SEQ : Ver Direction VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→SEQ OSEQ : Constant s Direction CP→SEQ	128 1 128 1 128 1 128 1 128 1 128 1 128 1 128 1 1 1 1 1 1 1 5tore 108 8 1	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface information End of the vector 0: Normal 128 bits per vert 1: double 256 bits per vert Render State (6*3+3 for constants) ad Description Address of the block of 4 constants		Formatted: Bullets and Numbering
Vtx_data_2 Vtx_data_3 22.1.322.1.7_VGT Name Vertex_indexes EOF_vector Inputs_vert STATE 22.1.422.1.8_CP_to Name Constant Address Constant Address	SX0→SP0 SX0→SP0 to SP0/SEQ : Ver Direction VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→SEQ OSEQ : Constant s Direction CP→SEQ CP→SEQ	128 1 128 1 128 1 128 1 128 1 128 1 128 1 1 1 21 1 5tore 100 Bits 1 512 1	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface information End of the vector 0: Normal 128 bits per vert 1: double 256 bits per vert Render State (6*3+3 for constants) ad Description Address of the block of 4 constants Data sent over 4 clocks		
Vtx_data_2 Vtx_data_3 22.1.322.1.7_VGT Name Vertex_indexes EOF_vector Inputs_vert STATE 22.1.422.1.8_CP_tc Name Constant Address Constant Data Remap Address	SX0→SP0 SX0→SP0 to SP0/SEQ : Ver Direction VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→SEQ OSEQ : Constant s Direction CP→SEQ	128 128 128 128 bits 1 1 1 21 1 store loc Bits 1 1 1 21 1 store loc Bits 1 1 1 1 1 1 1 21 1	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface information End of the vector 0: Normal 128 bits per vert 1: double 256 bits per vert Render State (6*3+3 for constants) ad Description Address of the block of 4 constants		Formatted: Bullets and Numbering
Vtx_data_2 Vtx_data_3 22.1.322.1.7_VGT Name Vertex indexes EOF_vector Inputs_vert STATE 22.1.422.1.8_CP to Name Constant Address Constant Address Remap Address Remap Data pointer	SX0→SP0 SX0→SP0 to SP0/SEQ : Ver Direction VGT→RESP0 VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→SEQ CF→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ	128 128 128 128 Bits 1 128 1 128 1 128 1 128 1 128 1 1 1 1 1 21 1 5tore 10 8 1	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface information End of the vector 0: Normal 128 bits per vert 1: double 256 bits per vert Render State (6*3+3 for constants) ad Description Address of the block of 4 constants Data sent over 4 clocks Remaping address write address Remaping pointer		Formatted: Bullets and Numbering
Vtx_data_2 Vtx_data_3 22.1.322.1.7_VGT Name Vertex indexes EOF_vector Inputs_vert STATE 22.1.422.1.8_CP to Name Constant Address Constant Address Remap Address Remap Data pointer	SX0→SP0 SX0→SP0 to SP0/SEQ : Ver Direction VGT→RESP0 VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→SEQ CF→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ	128 128 128 128 bits 1 1 1 21 1 store ////////////////////////////////////	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface information End of the vector 0: Normal 128 bits per vert 1: double 256 bits per vert Render State (6*3+3 for constants) ad Description Address of the block of 4 constants Data sent over 4 clocks Remaping address write address Remaping pointer		Formatted: Bullets and Numbering
Vtx_data_2 Vtx_data_3 22.1.322.1.7_VGT Name Vertex indexes EOF_vector Inputs_vert STATE 22.1.422.1.8_CP to Name Constant Address Constant Data Remap Data pointer 22.1.522.1.9_CP to Name Constant Address	SX0→SP0 SX0→SP0 sX0→SP0 birection VGT→RESP0 VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→SEQ OSEQ : Constant s Direction CP→SEQ CP→SEQ CP→SEQ OSEQ : Fetch State Direction CP→SEQ	128 128 128 128 Bits 1 128 1 128 1 128 1 128 1 128 1 1 1 21 1 5tore 10 8 10 6 store Bits 1 8 1 9 store 8 1 8 1 8 1	Vertex data to interpolate Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface informatic End of the vector D: Normal 128 bits per vert 1: double 256 bits per vert Render State (6*3+3 for constants) ad Description Address of the block of 4 constants Data sent over 4 clocks Remaping address write address Remaping pointer IOad Description Address of the block of 4 state constants		Formatted: Bullets and Numbering
Vtx data_2 Vtx_data_3 22.1.7_VGT Name Vertex indexes EOF_vector Inputs_vert STATE 22.1.422.1.8_CP to Name Constant Address Constant Address Constant Data Remap Data pointer 22.1.522.1.9_CP to Name Constant Data	SX0→SP0 SX0→SP0 SX0→SP0 to SP0/SEQ : Ver Direction VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→RESP0/SEQ VGT→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ CP→SEQ DSEQ : Fetch State Direction	128 1 128 1 128 1 128 1 128 1 128 1 128 1 128 1 1 1 1 1 21 1 store los 8 512 1 10 1 8 1 4 store 8 1 6 store 8 1 8 1 8 1 8 1 8 1 8 1 8 1 8 1 8 1 9 512	Vertex data to interpolate Vertex data to interpolate Description Pointers of indexes or HOS surface information End of the vector 0: Normal 128 bits per vert 1: double 256 bits per vert Render State (6*3+3 for constants) ad Description Address of the block of 4 constants Data sent over 4 clocks Remaping address write address Remaping pointer /oad Description		Formatted: Bullets and Numbering

ATI Ex. 2105 IPR2023-00922 Page 234 of 239

AR	ORIGINATE	DATE	EDIT DAT	E	R400 Sequencer Specification	PAGE		
	24 September	r, 2001 <u>4 Se</u>	eptember, 2	20155		28 of 32		
Remap Data p	ointer	CP→SEQ	8	R	emaping pointer			
00 1 (00 ·	1 10 00 (~~~ ~ .						Formatted: Bullets and Numbering
22.1.6 <u>22.</u>	<u>1.10</u> CP to	SEQ : Cont	rol State	e stoi	re load			
Name		Direction	Bi	ts D	escription			~
Constant Addre Constant Data		CP→SEQ CP→SEQ	?				7 7 °	Formatted
	Who and what is]	1	Formatted
•						4		Formatted
	<u>1.11_</u> MH to	SEQ: Instru	iction st	ore L	.oad		~ ~ -	Formatted: Bullets and Numbering
Name		Direction	Bi		escription			
Instruction add	ress	MH→SEQ	12		struction address			
Instruction	Para a dalaran	MH→SEQ	96		struction X times			
Control Instruc		MH→SEQ	9		binter to the control instruction store			
Control Instruc {ISSUE: CP or		MH→SEQ	32		ontrol Instruction X times			
•	•							Formatted: Bullets and Numbering
22.1.8 22.1	<u>1.12_</u> SP <u>0</u> to	RB- <u>SX0</u> :	Pixel rea	ad fro	m RBs	*-		
Name		Direction	Bit		escription			
Export_data		SP0→RBSX0			2 pairs of 32 bits channel values	-4141		
Shader_Dest	4	SP0→SX0SP			pecifies one of the of up to 12 export de			
Shader_Coun	I.	<u>SP0→SX0</u> SP	→r(B 3	e	ach set of four pixels or vectors is a ight clocks. This field specifies where th equence.			
Shader_Last		<u>SP0→SX0</u> SP-	→RB 1	T F tl b	he current export clause is over (true fo he last export instruction creates *two* B. This needs to be set on or after the nat is produced by the last export in efore the first RB cycle of the first exp f the next clause.	cycles to the last RB cycle struction, but		
Shader_Pixel	Valid	<u>SP0→SX0</u> SP	→RB 4x	4 F 0	esult of pixel kill in the shader pipe, whi utput for all pixel exports (depth and all x4 because 16 pixels are computed per	color buffers).		
Shader_Word	Valid	SP0→SX0SP	→ RB 2	0	pecifies whether to write low and/or higl f the 64-bit export data from each of the ectors			
22.1.9 22.1	1.13_SEQ to	o RB-<u>SX0</u> :	Control	bus		*		Formatted: Bullets and Numbering
Name		Direction	В	its C	escription			
Export_Pixel		SEQ→RB <u>SX0</u>	1		: Pixel : Vertex			
Export_SEND)	<u>SEQ→SX0</u> SE	Q→RB 1		aised to indicate that the SQ is starting	an export		
Export_Clause		SEQ→SX0SE			lause number, which is needed for verte	· · · · · · · · · · · · · · · · · · ·		
Export_State		<u>SEQ→SX0</u> SE	Q→RB 2 ⁻		tate ID, which is needed for vertex claus			
{ISSUE: Wher	re sent synchron re are the PC poi 2.1.14 <i>RB</i> -S	nters}			oed in SP <u>0</u> →RB- <u>SX0 i</u> nterface	•	, i	Formatted: Bullets and Numbering
	<u></u>		·					
Name Export RTS		Direction RBSX0→SEQ			escription aised by RB-SX0 to indicate that the fol	lowing two		
Export_RTS		MBONU→SEQ			elds reflect the result of the most recent			
Export_Positio	n	SX0→SEQRB	→SEQ 1		pecifies whether there is room for anoth			
Export_Buffer		SX0→SEQRB			pecifies the space availble in the output			
					· buffers are full			

Specifies the space availble in the output buffers. 0: buffers are full

Exhibit 2017.doc R400_Sequencer.doc 46659 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

AMD1044_0256907

	ORIGINATE	DATE	EDľ	T DATE		DOCUMENT-REV. NUM.	PAGE	
	24 Septembe	r, 2001	4 Septer	nber, 20		GEN-CXXXXX-REVA	29 of 32	
					1: 2K- pixels 64: 12 64 pix	bits available (32-bits for each of the 6 in a clause) 8K-bits available (16 128-bit entries fo els) 7: RESERVED		
2.1.1122	<u>.1.15_</u> SP <u>0</u> t	o <u>RB-S</u>	<u>(0</u> : Posi	tion re	əturn	bus	4 , **	
lame Position return		Direction SP <u>0</u> →RE		Bits 128	Descri Positio	ption n data or sprite size (per clock)		
ien 1 point sp	rite sizes. The s	torage use	d is of 64x1	28 bits f	or posit	leaved on a 16 x 16 basis. We export ion and 64x32 bits for sprite size, it is ito the bits of the sprite sizes.		
2.1.1222	.1.16 Shade	ər Engin	e to Fetc	h Unit	Bus	(Fast Bus)	4-	Formatted: Bullets and Numbering
each of the Read addro 3 clocks aff Four Quad of the sub-	sub-engines repo ess associated wit ter the Instruction 's worth of Fetch engines repeating	eating every th Quad 0 n Start. Data may b g every 4 cl	4 clocks. Th nust be sent be written to ocks. The re	the regist 1 clock the Regi	er file in after the ister file e index	ock. These are sourced from a different dex to read must precede the data by 2 Instruction Start signal is sent, so that o every clock. These are directed to a dif to write must accompany the data. Data n Start signal is sent.	clocks. The data is read ferent pixel	
	le_Read_Data le Write Data	Direction SP->TEX TEX→SP				ption h Addresses read from the Register file .re results		
	.1.17_Sequ						*-	Formatted: Bullets and Numbering
egisters is rea equencer als	ady or not. This v	way the see	quencer car	update	the fet	h clause it is now working and if the ch counters for the reservation statior to execute and the address in the r	n fifos. The	
Name		Direction		Bits	Descri			
Tex_Ready		TEX→ SI		1	Data re			
Tex_Claus	e_ivum	$ $ TEX \rightarrow SI $ $ SEQ \rightarrow TE		3 10		number tate address 10 bits sent over 4 clocks		
Tex_cst Tex Inst		SEQ→TE		10		nstruction address 12 bits sent over 4 clocks	ocks	
EO CLAU	SE	SEQ→TE		12		struction of the clause	/0//3	
	w, and	SEQ→TE		1		hase signal		
PHASE	RECT	SEQ→TE		96		prrect 3 bits per comp 2 components pe	r quad * 16	
PHASE LOD CORF						ask 1 bit per pixel		
LOD CORF			X	64				
		SEQ→TE		64 3		number	1	
LOD CORF Mask Tex_Claus			ΞX		Clause	number nto Register file for write of returned Feto	h Data	
LOD CORF Mask Tex_Clause Tex_Write	e_Num	SEQ→TE	X	3	Clause Index i	nto Register file for write of returned Feto into Register files for reading Feto		
LOD CORF Mask Tex_Claus: Tex_Write Tex_Read 23Interr	e_Num Register_Index Register_Index Nal interfac	SEQ→TE SEQ->TE SEQ->SF	X X	3 7 7	Clause Index in Index	nto Register file for write of returned Feto into Register files for reading Feto		Formatted: Bullets and Numbering
LOD CORF Mask Tex_Claus Tex_Write_ Tex_Read_ 23Interr 23Interr	e_Num _Register_Index _Register_Index	SEQ→TE SEQ->TE SEQ->SF ES ertex Co	X X	3 7 7	Clause Index in Index	nto Register file for write of returned Feto into Register files for reading Feto		Formatted: Bullets and Numbering

ATI Ex. 2105 IPR2023-00922 Page 236 of 239

	RIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24	September, 2001	4 September, 20155		30 of 32	Formatted: Bullets and Numbering
24.23. Exam	ples of progr	am executions		*	
24.1.123.1.1	Sequencer Co	ontrol of a Vector of	^r Vertices		
 state pointe space was also before shader pro The vertex the SEC 	ar as well as tag into allocated in the posi the vector is sent gram (using the M program is assumed Q then accesses the	position cache is sent alor tion cache for transformed to the RE, the CP has lo H?) I to be loaded when we rea	position before the vector was sent aded the global instruction store wi ceive the vertex vector. sing the local state pointer (provided to	th the vertex	
at this pointthe arbitrer	the vector is remov	ed from the Vertex FIFO t a vector to be transforme	- basically the Vertex FIFO always has ed if the parameter cache is full unless		
 the number state pointer 	of GPRs required b r that came down w	y the program is stored in	GPRs used by the program a local state register, which is accesse le has been allocated	ed using the	
 the 64 vertee RF0 of RF1 of RF2 of RF3 of the index is 	x indices are sent t SU0, SU1, SU2, and SU0, SU1, SU2, and SU0, SU1, SU2, and SU0, SU1, SU2, and written to the least	b the 64 register files over d SU3 is written the first cy d SU3 is written the second d SU3 is written the third c d SU3 is written the fourth significant 32 bits (floating	rcle d cycle ycle		
fetch state mac	hine 0, or TSM0 FIF	O)	the first reservation station (the FIFO e position cache and a register file bas		
		nd fetches the instructions TSM arbiter before it could	for fetch clause 0 from the global instr I start	uction store	
7. all instructions of	of fetch clause 0 are	issued by TSM0			
the control pack FIFO)	ket is passed to the r	next reservation station (th	e FIFO in front of ALU state machine (), or ASM0	
the fetch da once the TL	ta to the TU, which J has written all the	will write the data to the RI data to the register files, it	o complete; it passes the register file v F as it is received increments a counter that is associate ate machine can go ahead start to exec	d with ASM0	
	the control packet (a ne global instruction		ASM arbiter) and gets the instructions	for ALU	
		issued by ASM0, then the ate machine 1, or TSM1 FI	control packet is passed to the next re FO)	servation	
 position car shared with A paramete going to be 	n be exported in ALU all four shader pipe r cache pointer is al in the parameter ca	J clause 3 (or 4?); the data s) back to the PA's positio so sent along with the pos che.	ation stations until all clauses have bee a (and the tag) is sent over a position b n cache ition data. This tells to the PA where th position data before it gets sent back to	us (which is ne data is	
Exhibit 2017.docR400_Sequencer	.doc 46659 Bytes*** © 🖌	ATI Confidential. Referen PM10/19/01-01:40 PM	ce Copyright Notice on Cover Page	© *** <u>09/04/15 12:53</u>	

ATI Ex. 2105 IPR2023-00922 Page 237 of 239

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	31 of 32	

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 exported in cladse r (as well as position data in it was not 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

24.1.223.1.2 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 FIFO
 - · 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
- 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

Exhibit 2017.docR4400_Sequencer.doc 46659 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © *** 09/04/15 12:53

--- Formatted: Bullets and Numbering

AMD1044_0256910

ATI Ex. 2105 IPR2023-00922 Page 238 of 239

ORIGINATE DATE EDIT DATE R400 Sequencer Specification PAGE 24 September, 2001 4 September, 20155 32 of 32 Formatt 24.1.323.1.3 Notes 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.	
24 September, 2001 4 September, 20155 32 of 32 24.1.323.1.3 Notes 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. Formatt 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	
 24.1.323.1.3 Notes 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 	
 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 	ed: Bullets and Numbering
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.	
Formatt	ed: Bullets and Numbering
25.24. 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: 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. 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?	

Exhibit 2017.docR400_Sequencer.doc 46659 Bytes***
ATI Confidential. Reference Copyright Notice on Cover Page

DB/004/15 12:53

PM10/1901-01:40 PM

AMD1044_0256911

ATI Ex. 2105 IPR2023-00922 Page 239 of 239