	-			
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201516	GEN-CXXXXX-REVA	1 of 35
Author:	Laurent Lefebvre			
Issue To:		Copy No:		
	R400 S	equencer Spe	ecification	
		SEQ		
		Version 1. <u>2</u> 4		
req		cted uses of the block. It al	cer block (SEQ). It provides an ove lso describes the block interfaces,	
Document Lo		erforce\r400\arch\doc\gfx\RE ) Sequencer Specification APPROVALS	\R400_Sequencer.doc	
Remarks:				
	NTIALLY DETRIME	NTAL TO THE INTE	NFORMATION THAT COUREST OF ATI TECHNOL	
work created in unpublished we confidential, pro	n 2001.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 an provide notice that ATI owns a cop publication has occurred. This we of this document may be used, rej sion of ATI Technologies Inc."	yright in this ork contains
Exhibit 2018.docR400_S	Sequencer.doc 53168 Bytes*** ⓒ AT	I Confidential. Reference C PM11/16/01-10:17 AM10/16/01-01:40 P	Copyright Notice on Cover Page © ≝	*** <u>09/04/15 12:54</u>

ATI 2018 LG v. ATI IPR2015-00325

AMD1044\_0256912

ATI Ex. 2106 IPR2023-00922 Page 1 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201516		2 of 35

#### Table Of Contents

1.	OVERVIEW4
1.1	Top Level Block Diagram6
1.2	Data Flow graph8
1.3	Control Graph
2.	INTERPOLATED DATA BUS9
3.	INSTRUCTION STORE12
4.	SEQUENCER INSTRUCTIONS <u>1412</u>
5.	CONSTANT STORE1412
6.	LOOPING AND BRANCHES1412
6.1	The controlling state <u>1412</u>
6.2	The Control Flow Program <u>15</u> 13
6.3	Data dependant predicate instructions <u>17</u> 15
6.4	HW Detection of PV,PS <u>1815</u> 16
6.5	Register file indexing <u>18</u> 16
6.6	Predicated Instruction support for Texture
	ses <u>18</u> 16
6.7	Debugging the Shaders <u>18</u> 16
	Method 1: Debugging registers <u>18</u> 16
	.2 Method 2: Exporting the values in the
	PRs (12) <u>191717</u>
6.7	
	vord. <u>191717</u>
7.	PIXEL KILL MASK
8.	HOS SURFACES
9.	REGISTER FILE ALLOCATION
10.	FETCH ARBITRATION
11. 12.	ALU ARBITRATION
12.	HANDLING STALLS <u>221920</u> CONTENT OF THE RESERVATION STATION
	S221920
14.	THE OUTPUT FILE
14.	IJ FORMAT
16.	THE PARAMETER CACHE
17.	VERTEX POSITION EXPORTING
17.	EXPORTING ARBITRATION
10.	REAL TIME COMMANDS
20.	REGISTERS 242122
20.1	Control
20.1	Context
21.	DEBUG REGISTERS
21.1	Control 252222

	ext <u>2522</u> 22
	ACES
	nal Interfaces
22.1.1	PA/SC to SP0 : IJ bus
22.1.2	PA/SC to SEQ : IJ Control bus 262223
22.1.3	SEQ to SP0 : Interpolator bus <u>2623</u> 23
22.1.4	SEQ to SP0 : Parameter Cache bus 272324
22.1.5	SEQ to SX0 : Parameter Cache Mux
control Bus	<u>2723</u> 24
22.1.6	SX0 to SP0 : Parameter Cache Return
bus	<u>27232</u> 4
22.1.7	VGT to SP0/SEQ : Vertex Bus .282324
22.1.8	CP to SEQ : Constant store load
	282424
22.1.9	CP to SEQ : Fetch State store load
	282424
22.1.10	CP to SEQ : Control State store load
	282424
22.1.11	MH to SEQ: Instruction store Load
	282425
22.1.12	SP0 to SX0 : Pixel read from RBs
	<u>282425</u>
22.1.13	SEQ to SX0 : Control bus <u>292425</u>
22.1.14	SX0 to SEQ : Output file control
00.1.15	<u>292425</u> SP0 to SX0 : Position return bus
22.1.15	
22.1.16	<u>302526</u> Shader Engine to Fetch Unit Bus (Fast
22.1.10 Bus)	302526
22.1.17	Sequencer to Fetch Unit bus (Slow
Bus)	302526
	LES OF PROGRAM EXECUTIONS
312526	
23.1.1	Sequencer Control of a Vector of
Vertices	322626
23.1.2	Sequencer Control of a Vector of
Pixels	342727
23.1.3	Notes
24. OPEN I	SSUES

# **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

#### 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

Exhibit 2018 doc R400\_Sequences.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:54

AMD1044\_0256913

ATI Ex. 2106 IPR2023-00922 Page 2 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201516	GEN-CXXXXX-REVA	3 of 35
			ixels/vertices in the sequencer.	
Rev 0.5 (Laure		Added ti	iming diagrams (Vic)	
Date : Septem Rev 0.6 (Laure		Chapge	d the spec to reflect the nev	, P400
Date : Septem			ture. Added interfaces.	/ R400
Rev 0.7 (Laure			constant store management, in	struction
Date : October			anagement, control flow managem	
			pendant predication.	
Rev 0.8 (Laure			d the control flow method to b	
Date : October Rev 0.9 (Laure			Also updated the external interfaces rated changes made in the 10/18/01	
Date : October			eting. Added a NOP instruction, r	
Date : October	117,2001		nditional execute or jump. Added	
		registers		
Rev 1.0 (Laure		Refined	interfaces to RB. Added state regist	ers.
Date : October				
Rev 1.1 (Laure Date : October			SEQ→SP0 interfaces. Changed n. Changed VGT→SP0 interface.	
Date . October	120, 2001	Methods		Debug
Rev 1.2 (Laure	ent Lefebvre)		es greatly refined. Cleaned up the sp	ec.
Date : Novemi	ber 16, 2001	***************************************		

Exhibit 2018 doc R400\_Sequences.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 00/04/15/12:54

AMD1044\_0256914

ATI Ex. 2106 IPR2023-00922 Page 3 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201516		4 of 35

### 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.

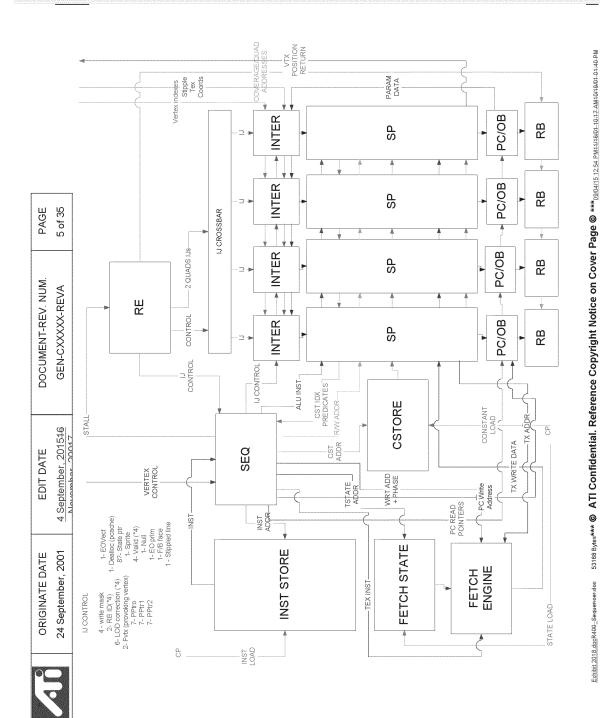
To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes. The four shader pipes also execute the same instructioninstruction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

Exhibit 2018.doc.R4400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:54

AMD1044\_0256915

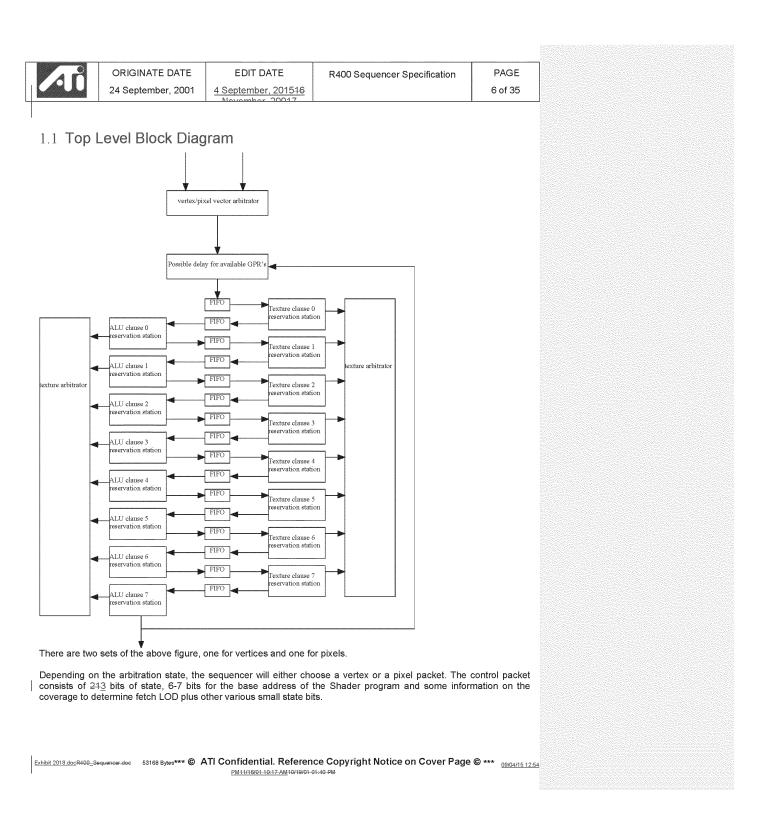
ATI Ex. 2106 IPR2023-00922 Page 4 of 223

# PROTECTIVE ORDER MATERIAL



AMD1044\_0256916

ATI Ex. 2106 IPR2023-00922 Page 5 of 223



ATI Ex. 2106 IPR2023-00922 Page 6 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201516	GEN-CXXXXX-REVA	7 of 35

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 <u>texuretexture</u> 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<u>arbiters</u>). One arbitrer<u>arbiter</u> 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 <u>arbitrersarbiters</u> 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 arbitrerarbiter 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).

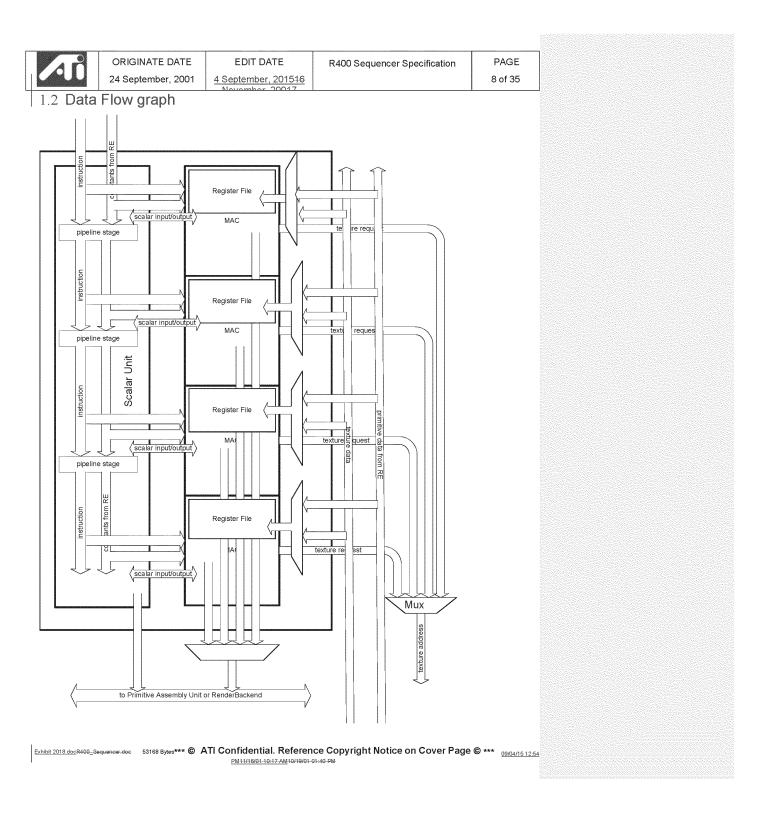
All other level process in the same way until the packet finally reaches the last ALU machine (7). On completion of the level 7 ALU clause, a valid bit is sent to the Render Backend which picks up the color data. This requires that the last instruction writes to the output register – a condition that 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<u>arbiter</u> blocks (two for the ALU state machines and one for the fetch state machines). The <u>arbitrer<u>arbiters</u> always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.</u>

Exhibit 2018 doc R400\_Sequencer.doc 53168 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:56

AMD1044\_0256918

ATI Ex. 2106 IPR2023-00922 Page 7 of 223

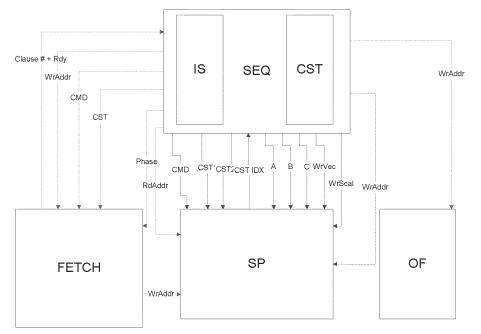


ATI Ex. 2106 IPR2023-00922 Page 8 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201516	GEN-CXXXXX-REVA	9 of 35

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.

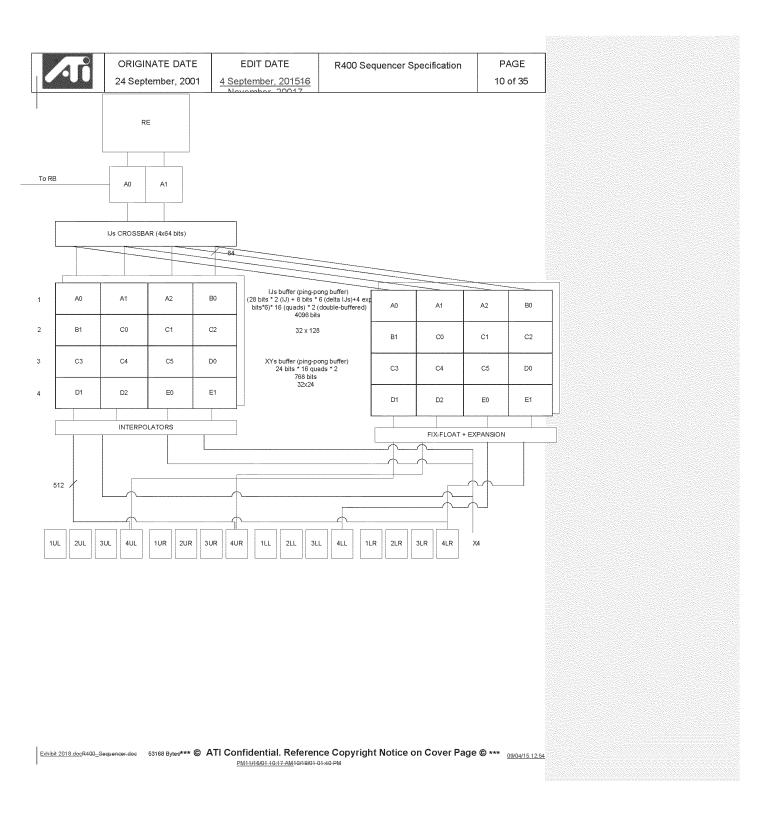
# 2. Interpolated data bus

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

Exhibit 2018 doc R400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0256920

ATI Ex. 2106 IPR2023-00922 Page 9 of 223



ATI Ex. 2106 IPR2023-00922 Page 10 of 223 PROTECTIVE ORDER MATERIAL

		53	2 % Z	245	<u>&gt;</u> 60	1 ~ ~ ~	
	-	22 T2	V V 32- 48- 35 51	V V 36-52- 39 55	40-56- 4359	V V 44-60- 47 63	×
	-	T10         T11         T13         T14         T15         T16         T17         T18         T19         T20         T21         T22         T23	> 6 €	2 % <	24- 4 27- 4	> &	L.
		20 T	>	> 4	> % £	V V 12-28- 15 31	
	-	191	51 C	XX 52-4 55 4	20 20 ×	<u>≻</u> 6α	
	-	18	XY > X2 > 32-4	33 <del>5</del> 33 <del>5</del>	X X 40-5 43 5	XY XY 44- 60- 47 63	<b>&gt;</b>
	T	17 T	5 19 19 20	% \$ \$ X	27 24	31 <sup>2</sup> 8 ×	×
PAGE 11 of 35	-	16 T	53×2	×4	,×4.5	15 2 X	
PA 11 c	•	ـــــــــــــــــــــــــــــــــــــ	~0	~ 4			
	-	4 1			ОШ	Ш Ш	
NU.	-	È				8	
ev. n X-rev	-	13		8	5	8	
T12	T12				BO	~	
DOCUMENT-REV. NUM. GEN-CXXXX-REVA	-	<b>1</b> 11	5	D2			Ω
о О	-	T10	ខ	5 2	C5		
		19	8				
1546 01546	-	Т8	AO	A	A2		
EDIT DATE	-	4			Ш	ш	
EDIT DATE 4 September, 201546	-	T6				å	
4		T5		8	5	8	
лЕ 001	-	Τ4				BO	0
ORIGINATE DATE 24 September, 2001		13	5	6			Д
IGINA'		12	ទ	2	C5		
0R 24 S	-	Ļ	Ξ				
	-	2	AO	A1	A2		
R	-		Ч O	с С С	N SP	ი წ	

Exhibit 2018 doo: Plate\_Sequences 53168 Bytes 4444 
ATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\*\* 0804/15 1254 PM11/1601110.117.AM16019001.01:40 PM

AMD1044\_0256922

ATI Ex. 2106 IPR2023-00922 Page 11 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201516		12 of 35

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 ?}.

<u>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.</u>

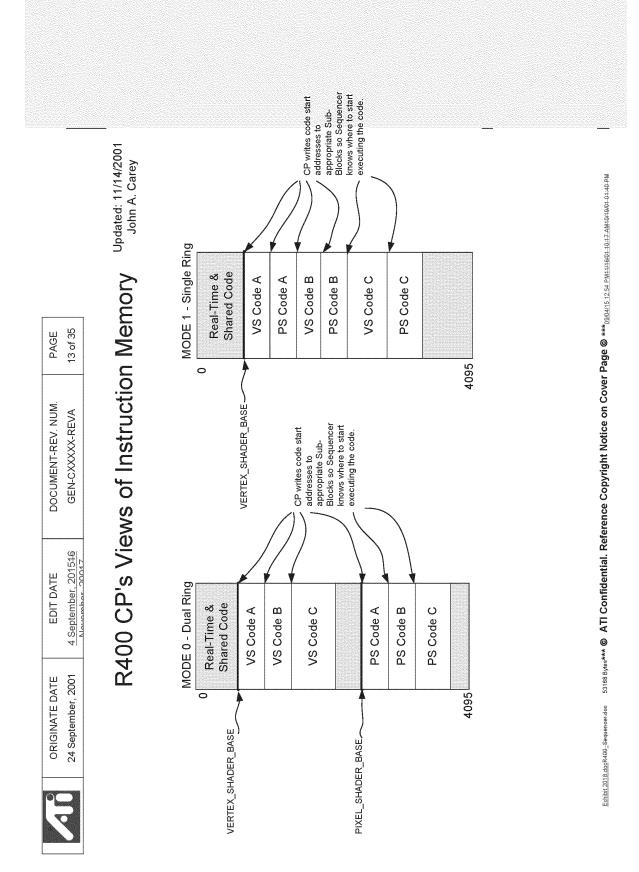
The instruction store is loaded by the CP thru the INSTRUCTION DATA, INSTRUCTION INDEX PORT control registers. The INSTRUCTION INDEX PORT is auto-incremented on both reads and writes to the INSTRUCTION DATA register.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The MSB of the INSTRUCTION\_INDEX\_PORT register contains the packet type for the sequencer to know where it must wrap around. The wrap around points are arbitrary and they are specified in the VERTEX\_SHADER\_BASE and PIXEL\_SHADER\_BASE registers.

Exhibit 2018.doc R400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:54

AMD1044\_0256923

ATI Ex. 2106 IPR2023-00922 Page 12 of 223



ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201516		14 of 35

### 4. Sequencer Instructions

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

## 5. Constant Stores

The constant store is managed by the CP. The sequencer is aware of where the constants are using a remaping table-also managed by the CP. A likely size for the constant store is 512x128-1024x128 bits. The constant store is also planned to be shared. The read BW from the constant store is 128 bits/clock and the write bandwith is 32/4 bits/clock.

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

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

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

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

The texture state is also kept in a similar memory. The size of this memory is 192x128. Which lets us load a texture state in ????

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change.

## 6. Looping and Branches

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

# 6.1 The controlling state.

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

```
Boolean[15:0]
loop_count[7:0][7:0]
In addition:
loop_start [7:0] [7:0]
loop_step [7:0] [7:0]
Exist to give more control to the controlling program.
We will extend that in the R400 to:
Boolean[255:0]
Loop_count[7:0][15:0]
Loop_Start[7:0] [15:0] times 2-3 (one for constant,registert1, register2)
Loop_Step[7:0] [15:0] times 2-3 (one for constant, registert1, register2)
Loop_End[7:0] [15:0]
```

Exhibit 2018.doc.R400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:54

AMD1044\_0256925

ATI Ex. 2106 IPR2023-00922 Page 14 of 223

00	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201546	GEN-CXXXXX-REVA	15 of 35	

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

We have a stack of 4 elements for calling subroutines and 4 loop counters to allow for nested loops.

#### 6.2 The Control Flow Program

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

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

Execute Conditional\_execute Conditional\_Execute\_Predicates Conditional\_jump Call Return Loop\_start Loop\_end End\_of\_clause NOP

Execute, causes the specified number of instructions in instruction store to be executed.

Conditional\_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.

Loop\_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.

Loop\_end increments (decrements?) the loop counter and jumps back the specified number of instructions. Call jumps to an address and pushes the IP counter on the stack. On the return instruction, the IP is poped from the stack.

Conditional\_execute\_or\_Jump executes a block of instructions or jumps to an address is the condition is not met. Conditional\_execute\_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition. End\_of\_clause marks the end of a clause.

Conditional\_jumps jumps to an address if the condition is met.

NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader\_cntl\_size (or vshader\_cntl\_size) we break the program (clause) and set the debug registers. If an execute or conditional\_execute is lower than cntl\_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

		Execute		
47	46 42	41 24	23 12	11 0
Addressing	00001	RESERVED	Instruction _count	Exec Address

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

		NOP	
47	46 42	41 0	
Addressing	00010	RESERVED	

Exhibit 2018 doc R400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:54

AMD1044\_0256926

ATI Ex. 2106 IPR2023-00922 Page 15 of 223

Γ		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	<u>C</u>	24 September, 2001	4 September, 201516		16 of 35
	If the specified	d boolean (8 bits can add	ress 256 booleans) meet	s the specified condition then execute	the specified

instructions (up to 512 instructions) or if the condition is not met jump to the jump address in the control flow program. This MUST be a forward jump. This is a regular NOP.

[	Conditionnal_Execute						
	47	46 42	41 34	33	32 24	23 12	11 0
	Addressing	00011	Boolean address	Condition	RESERVED	Instruction_count	Exec Address

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

Conditionnal_Execute_Predicates						
47	46 42	41 38	37	36 24	23 12	11 0
Addressing	00100	Predicate vector	Condition	RESERVED	Instruction_count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid.

	Loop_Start 47 4642 4116 154 30								
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	1000					
	01001	RESERVED	1000					
Addressing			Food ear					

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.

		C	onditionnal_J	Jmp		
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.

Exhibit 2018.doc R400\_Sequencer.doc 53168 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:54

AMD1044\_0256927

ATI Ex. 2106 IPR2023-00922 Page 16 of 223

	ORIGINATI 24 Septemb		EDIT DATE <u>4 September, 201516</u> Nevember, 20017	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	PAGE 17 of 35			
47	End_of_Clause							
Addressi	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 <u>instruction</u> is going to break the loop and set <u>the</u> debug registers. The sequencer will keep two loop indexes values:

IC index for constant indexing (9 bits)

IR index for register file indexing (7 bits)

This will be updated every time we loop and can only be used to index the constant store and the register file. The way to compute this value is:

Index = Loop\_counter\*Loop\_iterator + Loop\_init.

The IC for constant is going to return 0 if it is out of the constant range. The IR index is going to break the program if the index exceeds the number of requested registers.

The basic model is as follows:

The render state defined the clause boundaries:

Vertex\_shader\_fetch[7:0][7:0]// eight 8 bit pointers to the location where each clauses control program is locatedVertex\_shader\_alu[7:0][7:0]// eight 8 bit pointers to the location where each clauses control program is locatedPixel\_shader\_fetch[7:0][7:0]// eight 8 bit pointers to the location where each clauses control program is locatedPixel\_shader\_alu[7:0][7:0]// eight 8 bit pointers to the location where each clauses control program is locatedPixel\_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 which predicate set you want to use 0 thru 3.

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

P0\_ADD\_# R0,R1,R2

Exhibit 2018 docR400\_Sequencer.doc 53168 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:54

Formatted

AMD1044\_0256928

ATI Ex. 2106 IPR2023-00922 Page 17 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201516		18 of 35

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

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

# 6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of nonmasked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED\_SET and MOVA instructions and their uses.

## 6.5 Register file indexing

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

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

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

The sequencer is going to keep a loop index computed as such:

Index = Loop\_counter\*Loop\_iterator + Loop\_init.

The index is going to return 0 if it is out of the range.

#### 6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector. A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. We have to make sure the invalid pixels aren't considered with this optimization.

# 6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 3-2\_methods.

#### 6.7.1 Method 1: Debugging registers

Current plans are to expose 2 debugging, or error notification, registers: 1. address register where the first error occurred 2. count of the number of errors

The sequencer will detect the following groups of errors:

- count overflow - jump error

relative jump address > size of the control flow program relative jump address > length of the shader program

Exhibit 2018.docR400\_Sequencer.doc 53168 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09/04/15 12:54

Formatted

AMD1044\_0256929

ATI Ex. 2106 IPR2023-00922 Page 18 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201516	GEN-CXXXXX-REVA	19 of 35

- constant overflow

- register overflow

call stack

call with stack full

return with stack empty

With two of the errors, a jump error or a register overflow will cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

#### {ISSUE : Interrupt to the driver or not?}

#### 6.7.2 Method 2: Exporting the values in the GPRs (12)

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

- 1) Normal
- 2) Debug Kill
- 3) Debug Addr + Count

4)Debug Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug\_export instructions of clause 7 will be executed under the debug kill setting. Under the two-other modes, normal execution is done until we reach an address specified by the address register er-and instruction count (useful for loops) specified by the count register. After we have hit the address or the count we change to the kill mode for the rest of the clause After we have hit the instruction n times (n=count) we switch the clause to the kill mode.<sup>2</sup>

#### 6.7.3 Method 3: Selective export of a 32 bit Dword.

The third debug option will be mainly used for HW debug. For this mode, the sequencer will keep the following control debug registers: Shader\_pipe (6 bits), Mode(1 bit), Dword\_select (3 bits), clause\_+count (16 bits?),address (12 bits) Vector\_number (8 bits), Render\_state (21 bits). The shader pipe register selects a shader pipe amongst the 64, the dword\_select selects a channel (0...3 of vector or scalar), the clause\_+count selects at which clause and which count we export, the Render\_state specifies which render state is concerned, the Vector\_number specifies which vector is concerned and the mode selects count export, or address export.

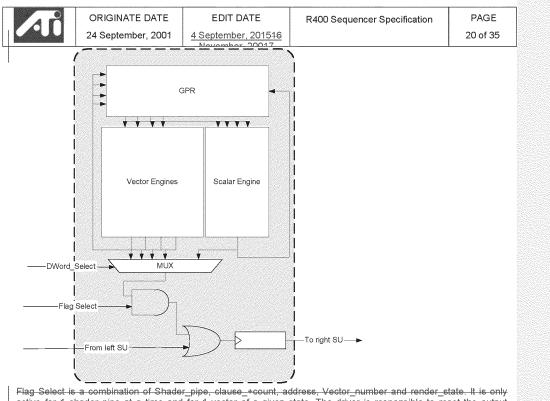
Exhibit 2018 doc R400\_Sequencer.doc 53168 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 00/04/15 12:54

AMD1044\_0256930

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

ATI Ex. 2106 IPR2023-00922 Page 19 of 223



Flag Select is a combination of Shader\_pipe, clause\_+count, address, Vector\_number and render\_state. It is only active for 1 shader pipe at a time and for 1 vector of a given state. The driver is responsible to reset the output register to 0 before executing a given program.

## 7. Pixel Kill Mask

A vector of 64 bits is kept per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK\_SETE MASK\_SETGT MASK\_SETGTE

# 8. HOS surfaces

HOS surfaces are able to export from the 6 last clauses but to memory ONLY. If they want to export to the parameter cache they have to do it in the last clause (7). They can also export position in clause 3.

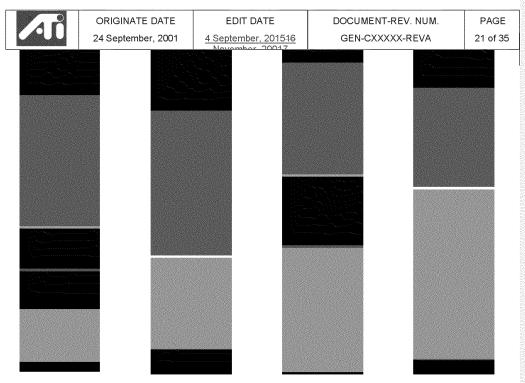
#### 9. Register file allocation

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

Exhibit 2018.doc.R400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:54

AMD1044\_0256931

ATI Ex. 2106 IPR2023-00922 Page 20 of 223



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 2018 docR400\_Sequencer.doc 53168 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09/04/15 12:54

AMD1044\_0256932

ATI Ex. 2106 IPR2023-00922 Page 21 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201516		22 of 35

## 12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS\_FULL) then the sequencer also prevents a thread to enter the exporting clause (43?). The sequencer will set the OUT\_FILE\_FULL signal n clocks before the output file is actually full and thus the ALU arbitrerarbiter 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. <del>For this reason only ONE concurrent program can be of clause 8 (exporting clause) the other program MUST not.</del> The staging registers are 4x128 (and there are 16 of those on the whole chip).

# 15. IJ Format

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

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

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

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

PO	P1
P2	P3

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

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

#### Multiplies (Full Precision): 2

Exhibit 2018.docR400\_Sequencer.doc 53168 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 00/04/15.12:54

AMD1044\_0256933

ATI Ex. 2106 IPR2023-00922 Page 22 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201516	GEN-CXXXXX-REVA	23 of 35

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

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign Mantissa 20 Exp 4 for J + Sign

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

Total number of bits : 19\*2 + 8\*6 + 4\*8 + 4\*2 = 128

The Deltas have a leading 1, the Full precision IJs don't. This means that in the case of the deltas we MUST be able to shift 8 right (exponent value of 0 means number = 0, exponent value of 1 means shift right 8). This means that the maximum range for the IJs (Full precision) is +/- 64 and the range for the Deltas is +/- 128.

#### 16. The parameter cache

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

## 17. Vertex position exporting

On clause 4 (or 5)3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 43.<u>Along with the position is exported a pointer to the parameter cache where</u> 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<u>from</u> the SX blocks.

## 18. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

- 1) Position exports and position exports cannot be co-issued.
- 2) Position exports and memory exports cannot be co-issued.
- 3) Position exports and Z/Color exports cannot be co-issued.
- 4) Memory exports and Z/Color exports cannot be co-issued.
- 5) Memory exports and memory exports cannot be co-issued.
- 6) Z/color exports and Z/color exports cannot be co-issued.
- 7) Parameter exports and Z/Color exports CAN be co-issued
- 8) Parameter exports and parameter exports CAN be co-issued.
- 9) Parameter exports and memory exports CAN be co-issued.

# 19. Real time commands

We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the 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 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 2018 docR400\_Sequencer.doc 53168 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:54

AMD1044\_0256934

ATI Ex. 2106 IPR2023-00922 Page 23 of 223

ORIGINAT	E DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 Septem	ber, 2001	4 September, 201516		24 of 35	Formatted: Bullets and Numbering
20. State manage	ement			*	
		to the CP the oldest st	tates still in the pipe. These are the	states of the	
programs as they enter the	ast ALU cla	use.			
					Formatted: Bullets and Numbering
21. XY Address in	nports			4. ~	
buffer) with XY writes (to interpolate the IJ data or pa	the XY buf ss the XY d	fer). Then when writing ata thru a Fix→float conve	oes so by interleaving the writes of the the data to the GPRs, the sequence enter and expander and write the conve ES. The values in the XY buffers will w	er is going to erted values to	
20.22. Registers				4-	Formatted: Bullets and Numbering
20.122.1_Control					
DYNAMIC_REG VERTEX_REG_SIZE PIXEL_MIN_SIZE VERTEX_MIN_SIZE ARBITRATION_policy CST_SIZE_P CST_SIZE_V INST_STOR_ALLOC VERTEX_WRAP	What p Minima policy o Size of Size of interlea start po	ortion of the register file is I size of the register file's I size of the register file's of the arbitration between the constant store for pixe the constant store for vert ved, separate, interleaved oint for the vertex instruction	els		
PIXEL WRAP	Begins		hader instruction store (vertex shader	alwavs-starts	
at-0)					
SHAREDWRAP RTWRAP NO_INTERLEAVE NO_INTERLEAVE_ALL	start po debug debug	state register. Only allows	on store store (RT always ends at the end of the one program at a time into the GPRs one ALU program at a time to be exe		
NO_PRED_OPTIMIZE	of 2) turns o executo		nization (conditional_execute_predica	tes is always	
INSTRUCTION_INDEX					
PORT		ented on reads/writes)	ase address of the instruction writes a	nd type (auto-	
INSTRUCTION DATA CONSTANT DATA		where the CP puts the act where the CP puts consta	tual data going to the instruction memo	ory	
	111315	where the Or puts collsta	un uara		
20.222.2 Context 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 Vshader_size REG_ALLOC_PIX REG_ALLOC_PIX REG_ALLOC_VERT FLAT_GOUR[015] CYL_WRAP[063]	eight 8 eight 8 base p base p size of size of size of numbe numbe which p for whic	bit pointers to the location bit pointers to the location bit pointers to the location pointer for the pixel shader pointer for the vertex shade the vertex shader (# of inst the pixel shader (# of inst the pixel shader (cntl+inst the vertex shader (cntl+inst the vertex shader (cntl+inst r of registers to allocate fo parameters are to be gour ch parameters (and channel	structions in control program/2) 'uctions in control program/2) ructions) structions) r pixel shader programs r vertex shader programs	is located is located is located	
P_export_mode		Normal mode	• • • • • • •		
Exhibit 2018.docR400_Sequencer.doc 5316	3 Bytes*** © 🖌	TI Confidential. Referen	ice Copyright Notice on Cover Page	© *** 09/04/15 12:54	

ATI Ex. 2106 IPR2023-00922 Page 24 of 223

ORIGINA	TE DATE	EDIT DATE		DOCUMENT-REV. NUM.	PAGE	
24 Septem	ber, 2001	4 September, 201		GEN-CXXXXX-REVA	25 of 35	
vshader_export_mask vshader_export_mode vshader_export_count[6 Control_Flow	If normal, k If multipase which of th 0: position # of interpo # of export 24 Dwords vector to ki	Itipass mode bbz where bbb is h s 1-12 exports for co the last 6 ALU clauses (1 vector), 1: positic blated parameters event vectors to memore that contain the co ill pixels and optimis	how many co color. es is exportin ion (2 vectors exported in cl ory per clause ontrol flow co ze texture pig	s), 3:multipass	use) ne mask kill ifth predicate	Formatted: Bullets and Numbering
21.23. DEBUG re	gisters					
21.1 Control						
Shader_pipe Count_+clause Dword_select Mode Rstate Vector_count	instruction channel se operating r render stat	ader pipe for metho count and clause n blect for method 3 mode for method 3 te method 3 is opera bler the method 3 v	aumber for me	ethod 3		
<del>21.2</del> 23.1 Context	Vootor Hum		moxport			Formatted: Bullets and Numbering
PROB ADDR		address where the	first problem	occurred		
PROB_COUNT Count <u>Addr</u> Clause mode[3]	instruction break addr	problems encounte counter for debug r ress for method nun de for debug metho	ered during th method 2 mber 2	e execution of the program		
Count	instruction break addr	counter for debug r	ered during th method 2 mber 2			Formatted: Bullets and Numbering
Count Addr Clause_mode[3] 22.24. Interfaces 22.124.1_External	instruction break addr clause mod	counter for debug r ress for method nun de for debug metho	ered during th method 2 <u>mber 2</u> od 2	e execution of the program		Formatted: Bullets and Numbering
Count Addr Clause_mode[3] 22.24. Interfaces 22.124.1_External	instruction break addr clause mod	counter for debug r ress for method nun de for debug metho bus is broadcast to	ered during th method 2 <u>mber 2</u> od 2 to all units of	e execution of the program	ple, if a bus is	Formatted: Bullets and Numbering
Count <u>Addr</u> Clause_mode[3] 22.24. Interfaces 22.124.1 External Whenever an x is used, it mamed SQ→SPx it means th	instruction break addr clause mod Interfaces heans that the lat SQ is going	counter for debug r ress for method nun de for debug metho bus is broadcast to to broadcast the se	ered during th method 2 <u>mber 2</u> od 2 to all units of	e execution of the program	ple, if a bus is	Formatted: Bullets and Numbering
Count <u>Addr</u> Clause_mode[3] 22.24. Interfaces 22.124.1 External Whenever an x is used, it mamed SQ→SPx it means the set of the set	Interfaces to SP0 : Jo the IJ information	counter for debug r ress for method nun de for debug metho bus is broadcast to to broadcast the sa J bus on to the IJ fifos on f	ered during the method 2 mber 2 od 2 to all units of ame informat the top of eace	the execution of the program the same name. For exam ion to all SP instances. ch shader pipe. At the same t	4	
Count Addr Clause_mode[3] 22.24. Interfaces 22.124.1 External Whenever an x is used, it m hamed SQ-SPx it means th 22.1.124.1.1 PA/SC This is a bus that sends information goes to the se	Interfaces to SP0 : Jo the IJ information	counter for debug r ress for method num de for debug metho bus is broadcast to to broadcast the sa J bus on to the IJ fifos on to are 4 of these buses	ered during the method 2 mber 2 od 2 to all units of ame informat the top of eace	the execution of the program the same name. For exam ion to all SP instances. ch shader pipe. At the same t	4	
Count <u>Addr</u> Clause_mode[3] 22.24. Interfaces 22.124.1 External Whenever an x is used, it m hamed SQ→SPx it means th 22.1.124.1.1 PA/SC This is a bus that sends information goes to the set Name	Interfaces beans that the lat SQ is going to SP0 : Jo the IJ informatic quencer. There	counter for debug r ress for method num de for debug metho bus is broadcast to to broadcast the sa J bus on to the IJ fifos on to are 4 of these buses Bits 1 P0 64 1	ered during the method 2 mber 2 and 2 to all units of ame informat the top of each is over the wh Description IJ information	the execution of the program the same name. For exam ion to all SP instances. ch shader pipe. At the same t	time the control	
Count Addr Clause_mode[3] 22.24. Interfaces 22.124.1 External Whenever an x is used, it mamed SQ→SPx it means th 22.1.124.1.1 PA/SC This is a bus that sends information goes to the set Name USSC_SP0_data MaskSC_SP0_g_wr_mask	Interfaces Interfaces heans that the hat SQ is going to SP0 : Ju he IJ informatic quencer. There Direction PASC-SF	counter for debug r ress for method num de for debug metho bus is broadcast to to broadcast the sa J bus on to the IJ fifos on f are 4 of these buses Bits P0 64 1 P0 1 1	ered during the method 2 mber 2 od 2 to all units of ame informat the top of ead is over the whe Description IJ information clock in the lo Write Mask	the same name. For example in the same name. For example in the same name. For example in the same toole chip (SP0 thru 3)	time the control	
Count <u>Addr</u> Clause_mode[3] <u>22.24. Interfaces</u> <u>22.124.1</u> External Whenever an x is used, it r named SQ→SPx it means th <u>22.1.124.1.1</u> PA/SC This is a bus that sends	Interfaces instruction break addr clause mod interfaces heans that the lat SQ is going to SP0 : Jo the IJ information quencer. There Direction PASC → SF	counter for debug r ress for method num de for debug metho bus is broadcast to to broadcast the sa J bus on to the IJ fifos on t are 4 of these buser Bits P0 64 1 P0 1 1 64 1	ered during the method 2 mber 2 and 2 and 2 and 2 and 1 and 1 and 1 be constant and	the same name. For example the same name. For example ion to all SP instances. The shader pipe. At the same to lole chip (SP0 thru 3) The sent over 2 clocks (or XY in the interface) write destination (XY buffer, IJ to sent over 2 clocks (or XY in the sent over 2 c	time the control	
Count Addr Clause_mode[3] 22.24. Interfaces 22.124.1 External Mhenever an x is used, it r hamed SQ→SPx it means th 22.1.124.1.1 PA/SC This is a bus that sends to information goes to the set Name UsSC_SP0_data MaskSC_SP0_g_wr_mask SC_SP0_dest SC_SP1_data	Interfaces Interfaces heans that the lat SQ is going to SP0 : Ic he IJ informatio quencer. There Direction PASC-SP PASC-SP SC-SP1	counter for debug r ress for method num de for debug metho bus is broadcast the to broadcast the sa J bus on to the IJ fifos on f are 4 of these buses Bits P0 64 1 P0 1 1 1 1 64 1	ered during the method 2 mber 2 and 2 and 2 and 2 and 1 ame information clock in the lo Write Mask Controls the v Quinformation clock in the lo	the same name. For example in the same name. For example ion to all SP instances. The shader pipe. At the same the sole chip (SP0 thru 3) The sent over 2 clocks (or XY in the sent over 2 clocks (or	time the control	
Count Addr Clause_mode[3] 2224. Interfaces 22124.1 External Whenever an x is used, it mamed SQ→SPx it means th 221.124.1.1 PA/SC This is a bus that sends information goes to the set Name JSSC_SP0_data MaskSC_SP0_g_wr_mask SC_SP1_data SC_SP1_dwr_mask	Interfaces Interf	counter for debug r ress for method num de for debug metho bus is broadcast to to broadcast the sa J bus on to the IJ fifos on to are 4 of these buses Bits P0 64 1 	ered during the method 2 mber 2 od 2 to all units of ame information construction by the top of ead is over the whether Description IJ information colock in the lo Write Mask Controls the v IJ information colock in the lo Write Mask	the same name. For example the same name. For example ion to all SP instances. The shader pipe. At the same to lole chip (SP0 thru 3) The sent over 2 clocks (or XY in the interface) write destination (XY buffer, IJ to sent over 2 clocks (or XY in the sent over 2 c	time the control	
Count Addr Clause_mode[3] 22.24. Interfaces 22.124.1 External Whenever an x is used, it mamed SQ→SPx it means th 22.1.124.1.1 PA/SC This is a bus that sends information goes to the se Name UsSC_SP0_data MaskSC_SP0_g_wr_mask SC_SP1_data SC_SP1_data SC_SP1_dest	Interfaces Interfaces heans that the lat SQ is going to SP0 : Ic he IJ informatio quencer. There Direction PASC-SP PASC-SP SC-SP1	counter for debug r ress for method num de for debug metho bus is broadcast to to broadcast the sa J bus on to the IJ fifos on to are 4 of these buses Bits PO 64 1 PO 1 0 64 1 1 1 64 1	ered during the method 2 mber 2 and 2 and 2 and 2 and 2 and 1 ame information clock in the lo Write Mask Controls the v IJ information clock in the lo	the same name. For example in the same name. For example is the same name. For example is the same toole chip (SP0 thru 3) The sent over 2 clocks (or XY in the same toole chip (SP0 thru 3) The sent over 2 clocks (or XY in the same toole solution (XY buffer, IJ) is sent over 2 clocks (or XY in the same toole clocks (o	time the control	
Count Addr Clause_mode[3] 2224. Interfaces 22124.1 External Whenever an x is used, it mamed SQ→SPx it means the 221.124.1.1 PA/SC This is a bus that sends information goes to the set information goes to the set Name USSC SP0 data MaskSC SP0 dest SC SP1 data SC SP1 data SC SP1 dest SC SP2 data	Interfaces Interf	counter for debug r ress for method num de for debug metho bus is broadcast to to broadcast the sa J bus on to the IJ fifos on t are 4 of these buses Bits P0 64 1 P0 1 1 20 1 1 20 1 1 1 1 64 1	ered during the method 2 mber 2 and 2 and 2 and 2 and 2 and 1 ame information clock in the lo Write Mask Controls the v IJ information clock in the lo	the same name. For example in the same name. For example ion to all SP instances. The shader pipe. At the same to ole chip (SP0 thru 3) In sent over 2 clocks (or XY in over 24 LSBs of the interface) write destination (XY buffer, IJ in sent over 2 clocks (or XY in over 24 LSBs of the interface) write destination (XY buffer, IJ in sent over 2 clocks (or XY in over 24 LSBs of the interface)	time the control	
Count Addr Clause_mode[3] 22.24. Interfaces 22.124.1 External Menever an x is used, it maned SQ→SPx it means th 22.1.124.1.1 PA/SC This is a bus that sends information goes to the set Name USSC SP0 data MaskSC SP0 q wr mask SC SP1 data SC SP1 data SC SP1 dest SC SP2 data SC SP2 data	instruction break addr clause mod lnterfaces heans that the hat SQ is going to SP0 : I. he IJ informatic quencer. There Direction PASC $\rightarrow$ SP SC $\rightarrow$ SP1 SC $\rightarrow$ SP1 SC $\rightarrow$ SP1 SC $\rightarrow$ SP2	counter for debug r ress for method num de for debug metho bus is broadcast to to broadcast the sa J bus on to the IJ fifos on t are 4 of these buses Bits PO 64 1 20 1 1 20 1 20	ered during the method 2 mber 2 and	the same name. For example in the same name. For example ion to all SP instances. The shader pipe. At the same to ole chip (SP0 thru 3) In sent over 2 clocks (or XY in over 24 LSBs of the interface) write destination (XY buffer, IJ in sent over 2 clocks (or XY in over 24 LSBs of the interface) write destination (XY buffer, IJ in sent over 2 clocks (or XY in over 24 LSBs of the interface)	time the control nfo sent over 1 buffer) nfo sent over 1 buffer) nfo sent over 1	
Count Addr Clause_mode[3] 22.24. Interfaces 22.124.1 External Whenever an x is used, it r named SQ→SPx it means th 22.1.124.1.1 PA/SC This is a bus that sends information goes to the se Name NaskSC SP0_data MaskSC SP0_q wr_mask SC SP0_dest	Interfaces Interf	counter for debug r ress for method num de for debug metho bus is broadcast to to broadcast the sa J bus on to the IJ fifos on to are 4 of these buses Bits PO 64 1 PO 1 1 1 64 1 1 1 64 1 1 1 64 1	ered during the method 2 mber 2 and	the same name. For example in the same name. For example is the same name. For example is the same table of the same tab	time the control	

ATI Ex. 2106 IPR2023-00922 Page 25 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201516		26 of 35

#### Formatted: Bullets and Numbering

22.1.224.1.2 PA/SC to SEQ : IJ Control bus

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. <u>This information is sent over 2 clocks</u>, if <u>SENDXY</u> is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quads that where just sent).

Name	Direction	Bits	Description
Write MaskSC SQ q wr mask	PASC→SEQ(SP)	4	Quad Write mask left to right
LOD_CORRECTSC_SQ_lod_correct	<u>SC→SQ</u> PA→SEQ(SP }	24	LOD correction per quad (6 bits per quad)
FVTXSC SQ flat vertex	$SC \rightarrow SQ PA \rightarrow SEQ(SP)$	2	Provoking vertex for flat shading
PPTR0SC SQ param ptr0	$\frac{SC \rightarrow SQ}{PA} \rightarrow SEQ(SP)$	11	P Store pointer for vertex 0
SC SQ param ptr1PPRT1	$SC \rightarrow SQ PA \rightarrow SEQ(SP)$	11	P Store pointer for vertex 1
SC SQ param ptr2PPTR2	$SC \rightarrow SQPA \rightarrow SEQ(SP)$	11	P Store pointer for vertex 2
SCE_OFF_VECTOR_SQ_end_of_vec t	$SC \rightarrow SQPA \rightarrow SEQ(SP)$	1	End of the vector
DEALLOCSC SQ store dealloc	$\frac{SC \rightarrow SQ}{PA} \rightarrow SEQ(SP)$	1	Deallocation token for the P Store
STATESC SQ state	<u>SC→SQ</u> PA→SEQ(SP }	<u>213</u>	State/constant pointer (6*3+3)
VALIDSC SQ valid pixel	$\frac{SC \rightarrow SQ}{PA} \rightarrow SEQ(SP)$	16	Valid bits for all pixels
NULLSC_SQ_null_prim	$\frac{SC \rightarrow SQ}{PA} \rightarrow SEQ(SP)$	1	Null Primitive (for PC deallocation purposes)
SC SQ end of primE_OFF_PRIM	$SC \rightarrow SQ PA \rightarrow SEQ(SP)$	1	End Of the primitive
FBFACESC_SQ_fbface	<u>SC→SQ</u> PA→SEQ(SP }	1	Front face = 1, back face = 0
SC SQ send xy	<u>SC→SQ</u>	1	Sending XY information [XY information is going to be sent on the next clock]
TYPE <u>SC SQ prim type</u>	<u>SC→SQ</u> PA→SEQ(SP )	3	Stippled line and Real time command need to load tex cords from alternate buffer 000 : Normal 001 : Stippled line 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
<u>SC_SQ_</u> RTRn	SEQ→PA <u>SQ→SC</u>	1	Stalls the PA in n clocks
<u>SC_SQ_RTS</u>	PA <u>SC</u> →SEQ(SP)SQ	1	PASC ready to send data
22.1.324.1.3 SEQ to SPO : 1	nterpolator bus		4
Name	Direction	Bit	
TYPESQ SPx_interp_prim_type	SEQ→SP <u>x</u> 0	3	Type of the primitive 000 : Normal 001 : Stippled line/Poly 011 : Real Time

Formatted: Bullets and Numbering

Exhibit 2018.docR400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:54

2

SEQ→SP0x

FVTXSQ SPx interp flat vtx

100 : Line AA 101 : Point AA 110 : Sprite

Provoking vertex for flat shading

AMD1044\_0256937

ATI Ex. 2106 IPR2023-00922 Page 26 of 223

OI	RIGINATE DATE	E		Ξ		DOCUMENT-REV. NUM.	PAGE		
24	September, 2001		tember, 20			GEN-CXXXXX-REVA	27 of 35		
LAT_GOURAUDSC	Q SPx interp flat go		SEQ→SP0		1	Flat or gouraud shading		-	
SQ_SPx_interp_cyl	wrapCYL_WRAP	s	SEQ→SP0	<u>Þx</u>	4	Wich parameter needs to b wrapped	e cylindrical		
SQ_SPx_interp_ijline	JJ_Line number	S	SEQ→SP <u>x</u>	0	2	Line in the IJ/XY buffer interpolate	to use to		
Q_SPx_interp_buff	swapSwap_Buffers	S	SEQ→SP0	) <u>x</u>	1	Swap the IJ/XY buffers at th interpolation	e end of the		
SQ SPx interp ij xv	L	S	<u>SQ→SPx</u>		1	Read from the IJ buffer or buffer	from the XY		
SQ SPx interp para	am0Param_0	S	SEQ→SP0	λX	1	We are interpolating parame	ter 0		
2 <u>2.1.424.1.4</u>	SEQ to SPO : P	aramei	terCad	che <u>R</u>	ead o	<u>control</u> bus	4	For	matted: Bullets and Numbering
The four following int	erfaces (SQ→SP, S	Q→SX,SF	⊃—SX and	d SX—lr	nterpol	ators) are all SYNCHRONIZE	) together.		
			Dit	<b>D</b>					
Vame	Direction		Bits	Descri		(7.1.CDa of Dainter)			
SQ_SPx_Pptr04	SEQ→SI		79			(7 LSBs of Pointer)			
SQ SPx Pptr12	SEQ→SI		79			(7 LSBs of Pointer)			
SQ_SPx_Pptr32	SEQ→SI		97			(7 LSBs of Pointer)			
SQ SP0 read ena	SQ→SP(		4			for the 4 memories in the SPO			
SQ_SP1_read_ena	SQ→SP'		4			for the 4 memories in the SP1			
SQ SP2 read ena	SQ→SP2		4			for the 4 memories in the SP2			
SQ SP3 read ena	SQ→SP3							Contraction and Contraction of the	
2 <u>2.1.5</u> 24.1.5_8	SEQ to SX0-: P	arame		he Mu	их со	of or the 4 memories in the SP3	*-	For	matted: Bullets and Numbering
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21	SEQ to SXO-: P Direction SEQ-S3 SEQ-S3	Paramei n X <u>x</u> 0 X <u>x</u> 0	ter Cac Bits 4 4	Descri Mux co	UX CC iption ontrol fo	ontrol Bus or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer)	*-	For	matted: Bullets and Numbering
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux32	SEQ to SXO-: F Direction SEQ-S SEQ-S SEQ-S	Paramei n K <u>x</u> 0 K <u>x</u> 0 K <u>x</u> 0	ter Cac Bits	Descri Mux co	UX CC iption ontrol fo	ontrol Bus			matted: Bullets and Numbering matted: Bullets and Numbering
22.1.524.1.5 Name SQ SXx Mmux01 SQ SXx Mmux21 SQ SXx Mmux32 24.1.6 SP to S	SEQ to SX0-: F Direction SEQ-S2 SEQ-S2 SEQ-S2 SX: Parameter (	Paramen n X <u>x</u> 0 X <u>x0</u> X <u>x0</u> data	ter Cac Bits 4 4 4	Descri Mux co Mux co Mux co	UX CO iption ontrol fo ontrol fo	ontrol Bus or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer)			
22.1.524.1.5 Name SQ SXx Mmux01 SQ SXx Mmux21 SQ SXx Mmux32 24.1.6 SP to S Name	SEQ to SXO-: P Direction SEQ-S3 SEQ-S3 SEQ-S3 SX: Parameter of Direction	Paramer n X <u>x0</u> X <u>x0</u> X <u>x0</u> data 1	ter Cac Bits 4 4 4 Bits	Descri Mux co Mux co Mux co Descri	UX CO iption ontrol fo ontrol fo ontrol fo	ontrol Bus or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer)			
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux32 24.1.6 SP to S Name SP0_SX0_data0	SEQ to SX0-: F Direction SEQ-S SEQ-S SEQ-S SEQ-S SX: Parameter of Direction SP0-SX	Paramei n x <u>x0</u> x <u>x0</u> data 1	ter Cac Bits 4 4 4 Bits 128	Descri Mux co Mux co Mux co Descri Param	UX CC iption ontrol fo ontrol fo ontrol fo iption iption	ontrol Bus or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer)			
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux32 24.1.6 SP to S Name SP0_SX0_data0 SP0_SX0_data1	SEQ to SX0-: F Direction SEQ-S2 SEQ-S2 SEQ-S2 SX: Parameter ( Direction SP0-S2 SP0-S2 SP0-S2	Paramei NXX0 XX0 XX0 XX0 data 1 1 X0 X0	Bits           4           4           4           128	be Mu Descri Mux cc Mux cc Mux cc Descri Param Param	UX CC iption ontrol fo ontrol fo ontrol fo iption eter da eter da	ontrol Bus or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) ta 0	· ·		
22.1.5 <u>2</u> 4.1.5 SQ SXx Mmux01 SQ SXx Mmux21 SQ SXx Mmux32 24.1.6 SP to S Vame SP0 SX0 data0 SP0 SX0 data1 SP0 SX0 data2	SEQ to SX0-: F Direction SEQ-S3 SEQ-S3 SEQ-S3 SX: Parameter ( Direction SP0-S2 SP0-S2 SP0-S2 SP0-S2	2aramei n x <sub>x</sub> 0 xx0 xx0 xx0 data 1 (0 (0 (0 (0 (0)	Eits 128 128 128 128 128 128 128 128	be Mu Descri Mux cc Mux cc Mux cc Mux cc Descri Param Param	UX CC iption ontrol fo ontrol fo iption eter da eter da eter da	ontrol Bus or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) ta 0 ta 0 ta 1 ta 2			
22.1.524.1.5         SQ           SQ         SXx         Mmux01           SQ         SXx         Mmux21           SQ         SXx         Mmux21           SQ         SXx         Mmux32           24.1.6         SP to S           Vame         SP0 SX0 data0           SP0         SX0 data1           SP0         SX0 data2           SP0         SX0 data3	SEQ to SX0-: F Direction SEQ-S3 SEQ-S3 SEQ-S3 SX: Parameter ( Direction SP0-S2 SP0-S2 SP0-S2 SP0-S2 SP0-S2	Paramei NXX0 XX0 XX0 XX0 XX0 XX0 X0 X0 X0 X0 X0 X	Bits           4           4           4           128           128           128           128	be Mu Descri Mux cc Mux cc Mux cc Mux cc Descri Param Param Param	UX CC iption ontrol fo ontrol fo iption eter da eter da eter da eter da	ontrol Bus or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) ta 0 ta 1 ta 2 ta 3			
22.1.524.1.5         Sume           SQ_SXx_Mmux01         SQ_SXx_Mmux21           SQ_SXx_Mmux32         SXx_Mmux32           24.1.6         SP to S           SP0_SX0_data1         SP0_SX0_data2           SP0_SX0_data3         SP0_SX0_data3           SP1_SX1_data0         SX1_data0	SEQ to SXO-: F Direction SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SP0-S3 SP0-S3 SP0-S3 SP1-S3	2 Paramer Xx0 Xx0 data 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Eits Eits 4 4 4 4 228 128 128 128 128 128 128	be Mu Descri Mux cc Mux cc Mux cc Mux cc Descri Param Param Param	UX CC iption ontrol fo ontrol fo iption eter da eter da eter da	ontrol Bus or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) ta 0 ta 1 ta 2 ta 3			
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux32 24.1.6 SP to S Name SP0_SX0_data0 SP0_SX0_data1 SP0_SX0_data2 SP0_SX0_data3 SP1_SX1_data0 SP1_SX1_data1	SEQ to SX0-: F Direction SEQ-S3 SEQ-S3 SEQ-S3 SX: Parameter ( Direction SP0-S2 SP0-S2 SP0-S2 SP0-S2 SP0-S2	2 Paramer Xx0 Xx0 data 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Bits           4           4           4           128           128           128           128	be Mu Descri Mux cc Mux cc Mux cc Mux cc Param Param Param Param Param	UX CC iption ontrol fo ontrol fo iption eter da eter da eter da eter da	ontrol Bus or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) or PC (4 MSbs of Pointer) ta 0 ta 1 ta 2 ta 3 ta 0			
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux32 24.1.6 SP to S Name SP0_SX0_data0 SP0_SX0_data1 SP0_SX0_data2 SP0_SX0_data3 SP1_SX1_data0 SP1_SX1_data1	SEQ to SXO-: F Direction SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SP0-S3 SP0-S3 SP0-S3 SP1-S3	2 Paramet Xx0 Xx0 data 0 0 0 0 0 0 0 0 0 0 0 0 0	Eits Eits 4 4 4 4 228 128 128 128 128 128 128	be Mu Descri Mux cc Mux cc Mux cc Descri Param Param Param Param Param Param	UX CC iption ontrol fo ontrol fo ontrol fo iption ieter da eter da eter da eter da eter da	ta 0 ta 2 ta 2 ta 2 ta 2 ta 1 ta 1 ta 2 ta 1 ta 1			
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux32 24.1.6 SP to S Name SP0_SX0_data0 SP0_SX0_data1 SP0_SX0_data2 SP0_SX0_data3 SP1_SX1_data1 SP1_SX1_data2 SP1_SX1_data2	SEQ to SXO-: F Direction SEQ-S2 SEQ-S2 SX: Parameter ( Direction SP0-S2 SP0-S2 SP0-S2 SP0-S2 SP0-S2 SP0-S2 SP1-S2 SP1-S2 SP1-S2	Paramet xx0 xx0 <u>data</u> <u>1</u> 00 00 11 11 11 11 11 11 11 11	Eits 4 4 4 4 128 128 128 128 128 128 128 128 128 128	be Mu Descri Mux cc Mux cc Mux cc Descri Param Param Param Param Param Param Param Param	UX CC iption ontrol fo ontrol fo ontrol fo iption eter da eter da eter da eter da eter da eter da	ta 0 ta 2 ta 2 ta 2 ta 2 ta 2 ta 2 ta 2 ta 2			
22.1.524.1.5 SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux22 24.1.6 SP to S Name SP0_SX0_data0 SP0_SX0_data1 SP0_SX0_data3 SP1_SX1_data1 SP1_SX1_data2 SP1_SX1_data3 SP1_SX1_data3	SEQ to SX0-: F Direction SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SP0-S: SP0-S: SP0-S: SP1-S: SP1-S: SP1-S:	Paramet NXx0 Xx0 data 1 00 00 00 00 00 11 11 11 11 11	Eits 4 4 4 4 4 4 28 128 128 128 128 128 128 128	be Mu Descri Mux cc Mux cc Mux cc Mux cc Param Param Param Param Param Param Param Param Param	UX CC iption ontrol fo ontrol fo ontrol fo iption eter da eter da eter da eter da eter da eter da	ta 0 ta 1 ta 2 ta 2 ta 2 ta 3 ta 3			
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux32 24.1.6 SP to S Name SP0_SX0_data0 SP0_SX0_data1 SP0_SX0_data2 SP0_SX0_data2 SP1_SX1_data0 SP1_SX1_data2 SP1_SX1_data3 SP1_SX1_data3 SP1_SX1_data3 SP1_SX1_data3 SP2_SX0_data0	Direction           SEQ to SX0-: F           Direction           SEQ-S3           SP0-S3           SP0-S3           SP0-S3           SP0-S3           SP0-S3           SP1-S3           SP1-S3           SP1-S3           SP1-S3           SP1-S3	Paramet NXx0 Xx0 data 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Bits           4           4           4           4           4           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128	be Mu Descri Mux cc Mux cc Mux cc Param Param Param Param Param Param Param Param Param Param Param	LIX CC iption ontrol fo ontrol fo ontrol fo iption eter da eter da eter da eter da eter da eter da eter da eter da eter da	ta 0 ta 2 ta 2 ta 2 ta 2 ta 3 ta 0 ta 2 ta 3 ta 0 ta 2 ta 3 ta 0 ta 2 ta 3 ta 0 ta 2 ta 3 ta 0			
22.1.524.1.5 SQ SXx Mmux01 SQ SXx Mmux21 SQ SXx Mmux22 24.1.6 SP to S Name SP0 SX0 data0 SP0 SX0 data1 SP0 SX0 data3 SP1 SX1 data1 SP1 SX1 data2 SP1 SX1 data3 SP1 SX1 data3 SP2 SX0 data1 SP2 SX0 data1	SEQ to SX0-: F Direction SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SP0-S2 SP0-S2 SP0-S2 SP0-S2 SP0-S2 SP1-S2 SP1-S2 SP1-S2 SP1-S2 SP1-S2 SP1-S2 SP2-S2 SP2-S2	Paramet NXX0 XX0 XX0 data data 0 0 0 0 0 0 0 0 0 0 0 0 0	Bits           4           4           4           4           4           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128	be Mu Descri Mux cc Mux cc Mux cc Param Param Param Param Param Param Param Param Param Param Param Param Param Param	LIX CC iption ontrol fo ontrol fo ontrol fo ontrol fo iption eter da eter da eter da eter da eter da eter da eter da eter da eter da	ta 0 ta 1 ta 2 ta 2 ta 3 ta 0 ta 1 ta 2 ta 3 ta 0 ta 1			
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux32 24.1.6 <i>SP to S</i> Name SP0 SX0 data0 SP0 SX0 data1 SP0 SX0 data3 SP1 SX1 data2 SP1 SX1 data2 SP1 SX1 data3 SP1 SX1 data3 SP2 SX0 data1 SP2 SX0 data1 SP2 SX0 data2	SEQ to SX0-: F Direction SEQ-S) SEQ-S) SEQ-S) SEQ-S) SX: Parameter ( Direction SP0-SX SP0-SX SP0-SX SP0-SX SP0-SX SP1-SX SP1-SX SP1-SX SP1-SX SP1-SX SP1-SX SP1-SX SP2-SX SP2-SX	Paramet NXX0 XX0 data data 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Eits 4 4 4 4 4 4 4 4 4 128 128 128 128 128 128 128 128	be Mu Descri Mux cc Mux cc Mux cc Param Param Param Param Param Param Param Param Param Param Param Param Param Param Param	LIX CC iption ontrol fo ontrol fo ontrol fo ontrol fo iption eter da eter da	ta 0 ta 1 ta 2 ta 2 ta 3 ta 0 ta 1 ta 2 ta 1 ta 2 ta 2 ta 1 ta 2 ta 2 ta 1 ta 2 ta 2 ta 2 ta 1 ta 2 ta 2 ta 3 ta 0 ta 1 ta 2 ta 3 ta 0 ta 1 ta 2 ta 3 ta 0 ta 1 ta 1 ta 2 ta 1 ta 2 ta 1 ta 1 ta 2 ta 1 ta 1 ta 2 ta 1 ta 1 ta 1 ta 2 ta 1 ta 1 ta 1 ta 2 ta 1 ta 1 ta 1 ta 2 ta 1 ta 1 ta 1 ta 1 ta 1 ta 1 ta 1 ta 1			
22.1.524.1.5 Name SQ SXx Mmux01 SQ SXx Mmux21 SQ SXx Mmux32 24.1.6 <i>SP to S</i> Name SP0 SX0 data0 SP0 SX0 data1 SP0 SX0 data3 SP1 SX1 data3 SP1 SX1 data3 SP1 SX1 data3 SP1 SX1 data3 SP1 SX1 data3 SP1 SX1 data3 SP2 SX0 data1 SP2 SX0 data1 SP2 SX0 data3	SEQ to SXO-: F Direction SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SEQ-S3 SP0-S2 SP0-S2 SP0-S2 SP0-S2 SP1-S3 SP1-S3 SP1-S3 SP1-S3 SP1-S3 SP2-S3 SP2-S3 SP2-S3	Paramet NXX0 XX0 XX0 data data 0 0 0 0 0 0 0 0 0 0 0 0 0	Bits           4           4           4           4           128	be Mu Descri Mux cc Mux	LIX CCC iption pontrol fo pontrol fo introl for introl	ta 0 ta 2 ta 2 ta 2 ta 3 ta 0 ta 1 ta 2 ta 3 ta 3 ta 1 ta 2 ta 3 ta 4 ta 5 ta 4 ta 5 ta 5			
22.1.524.1.5           SQ_SXx_Mmux01           SQ_SXx_Mmux21           SQ_SXx_Mmux22           24.1.6           SP0_SX0_data0           SP0_SX0_data1           SP0_SX0_data1           SP0_SX0_data2           SP1_SX1_data1           SP1_SX1_data3           SP1_SX1_data3           SP2_SX0_data3           SP2_SX0_data1           SP2_SX0_data3           SP2_SX0_data3           SP2_SX0_data3           SP2_SX0_data3           SP2_SX0_data3           SP3_SX1_data0	Direction           SEQ to SX0-: F           Direction           SEQ-S3           SEQ-S3           SEQ-S3           SEQ-S3           SEQ-S3           SEQ-S3           SEQ-S3           SEQ-S3           SEQ-S3           SP0-S2           SP0-S2           SP0-S2           SP1-S2           SP1-S2           SP1-S2           SP1-S2           SP2-S2           SP2-S2           SP2-S2           SP3-S2	Paramet Paramet Xx0 Xx0 <u>Xx0</u> <u>data</u> <u>data</u> <u>0</u> <u>0</u> <u>0</u> <u>0</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u>11</u> <u></u>	Bits           4           4           4           4           128	be Mu Descri Mux cc Mux cc Mux cc Param Param Param Param Param Param Param Param Param Param Param Param Param Param Param Param Param Param	UX CC iption ontrol fo pontrol fo iption eter da eter da	ta 0 ta 1 ta 2 ta 3 ta 0 ta 2 ta 3 ta 0 ta 3 ta 0 ta 3 ta 0 ta 0 ta 3 ta 0 ta 0 ta 3 ta 0 ta 0			
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux32 24.1.6 SP to S Name SP0 SX0 data0 SP0 SX0 data1 SP0 SX0 data1 SP0 SX0 data3 SP1 SX1 data1 SP1 SX1 data2 SP1 SX1 data2 SP1 SX1 data2 SP1 SX1 data2 SP1 SX1 data3 SP2 SX0 data1 SP2 SX0 data1 SP2 SX0 data2 SP2 SX0 data2 SP3 SX1 data0 SP3 SX1 data1	SEQ to SX0-: F Direction SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SP0-S: SP0-S: SP0-S: SP0-S: SP1-S: SP1-S: SP1-S: SP1-S: SP1-S: SP1-S: SP1-S: SP1-S: SP1-S: SP2-S: SP2-S: SP2-S: SP2-S: SP2-S: SP3-S: SP3-S:	Paramet xx0 xx0 data 0 0 0 0 0 0 0 0 0 0 0 0 0	Eits 4 4 4 4 4 4 28 128 128 128 128 128 128 128	be Mu Descri Mux cc Mux cc Mux cc Mux cc Param P	LIX CCC iption pontrol fc iption iption fc iption fc iption eter da eter da	ta 0 ta 1 ta 2 ta 2 ta 3 ta 0 ta 1 ta 2 ta 3 ta 3 ta 0 ta 1 ta 3 ta 3 ta 0 ta 3 ta 3 ta 3 ta 0 ta 3 ta 3 ta 3 ta 0 ta 3 ta 3 ta 0 ta 3 ta 0 ta 3 ta 0 ta 3 ta 0 ta 1 ta 3 ta 0 ta 1 ta 3 ta 0 ta 1 ta 3 ta 0 ta 1 ta 1 ta 3 ta 0 ta 1 ta 3 ta 0 ta 1 ta 1 ta 3 ta 0 ta 1 ta 3 ta 1 ta 3 ta 0 ta 1 ta 3 ta 1 ta 3 ta 0 ta 1 ta 3 ta 1 ta 3 ta 1 ta 1 ta 3 ta 1 ta 1 ta 3 ta 4 ta 3 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4			
SQ         S24.1.5         SQ           SQ         SXx         Mmux01           SQ         SXx         Mmux21           SQ         SXx         Mmux32           24.1.6         SP to S           SP0         SX0           SP0         SX0           SP0         SX0           SP0         SX0           SP0         SX0           SP1         SX1           SP1         SX1           SP1         SX1           SP2         SX0           SP2         SX0           SP2         SX0           SP2         SX0           SP3         SX1           SP3         SX1           SP3         SX1           SP3         SX1	SEQ to SX0-: F Direction SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SP0-S? SP0-S? SP0-S? SP0-S? SP1-S? SP1-S? SP1-S? SP1-S? SP1-S? SP1-S? SP1-S? SP2-S? SP2-S? SP2-S? SP2-S? SP3-S? SP3-S?	Paramet NX20 X20 X20 data 1 00 00 00 11 11 11 11 11 11	Bits           4           4           4           4           4           128	be Mu Descri Mux cc Mux cc Mux cc Mux cc Mux cc Mux cc Param	LIX CCC iption pontrol fc iption iption teter da eter da	ta 0 ta 1 ta 2 ta 3 ta 0 ta 2 ta 3 ta 0 ta 2 ta 2 ta 3 ta 0 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 3 ta 2 ta 3 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4			
22.1.524.1.5         \$           SQ_SXx_Mmux01           SQ_SXx_Mmux21           SQ_SXx_Mmux22           24.1.6         SP to S           24.1.6         SP to S           SP0_SX0_data0           SP0_SX0_data1           SP0_SX0_data2           SP0_SX0_data3           SP1_SX1_data1           SP1_SX1_data2           SP1_SX1_data3           SP2_SX0_data1           SP2_SX0_data2           SP2_SX0_data2           SP2_SX0_data3           SP2_SX0_data1           SP2_SX0_data2           SP3_SX1_data1           SP3_SX1_data1           SP3_SX1_data2	SEQ to SX0-: F Direction SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SP0-S: SP0-S: SP0-S: SP0-S: SP1-S: SP1-S: SP1-S: SP1-S: SP1-S: SP1-S: SP1-S: SP1-S: SP1-S: SP2-S: SP2-S: SP2-S: SP2-S: SP2-S: SP3-S: SP3-S:	Paramet NX20 X20 X20 data 1 00 00 00 11 11 11 11 11 11	Eits 4 4 4 4 4 4 28 128 128 128 128 128 128 128	be Mu Descri Mux cc Mux cc Mux cc Mux cc Mux cc Mux cc Param	LIX CCC iption pontrol fc iption iption fc iption fc iption eter da eter da	ta 0 ta 1 ta 2 ta 3 ta 0 ta 2 ta 3 ta 0 ta 2 ta 2 ta 3 ta 0 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 3 ta 2 ta 3 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4			
22.1.524.1.5         \$           SQ_SXx_Mmux01           SQ_SXx_Mmux21           SQ_SXx_Mmux22           24.1.6         SP to S           24.1.6         SP to S           SP0_SX0_data0           SP0_SX0_data1           SP0_SX0_data2           SP0_SX0_data3           SP1_SX1_data1           SP1_SX1_data2           SP1_SX1_data3           SP2_SX0_data1           SP2_SX0_data2           SP2_SX0_data2           SP2_SX0_data3           SP2_SX0_data1           SP2_SX0_data2           SP3_SX1_data1           SP3_SX1_data1           SP3_SX1_data2	SEQ to SX0-: F Direction SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SEQ-S? SP0-S? SP0-S? SP0-S? SP0-S? SP1-S? SP1-S? SP1-S? SP1-S? SP1-S? SP1-S? SP1-S? SP2-S? SP2-S? SP2-S? SP2-S? SP3-S? SP3-S?	Paramet NX20 X20 X20 data 1 00 00 00 11 11 11 11 11 11	Bits           4           4           4           4           4           128	be Mu Descri Mux cc Mux cc Mux cc Mux cc Mux cc Mux cc Param	LIX CCC iption pontrol fc iption iption teter da eter da	ta 0 ta 1 ta 2 ta 3 ta 0 ta 2 ta 3 ta 0 ta 2 ta 2 ta 3 ta 0 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 3 ta 2 ta 3 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4		For	matted: Bullets and Numbering
22.1.5         24.1.5         5           SQ_SXx_Mmux01         SQ_SXx_Mmux21           SQ_SXx_Mmux32         SXx_Mmux32           24.1.6         SP to S           24.1.6         SP to S           SP0_SX0_data0         SP0_SX0_data1           SP0_SX0_data1         SP0_SX0_data3           SP1_SX1_data1         SP1_SX1_data2           SP1_SX1_data2         SP1_SX1_data3           SP2_SX0_data1         SP2_SX0_data1           SP2_SX0_data3         SP3_SX1_data1           SP2_SX0_data3         SP3_SX1_data1           SP3_SX1_data1         SP3_SX1_data1           SP3_SX1_data3         SP3_SX1_data3           SP3_SX1_data3         SP3_SX1_data3	SEQ to SX0-: F Direction SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SP0-SX SP0-SX SP0-SX SP0-SX SP0-SX SP1-SX SP1-SX SP1-SX SP1-SX SP1-SX SP1-SX SP2-SX SP2-SX SP2-SX SP2-SX SP3-SX SP3-SX SP3-SX	Paramet xx0 xx0 data 0 0 0 0 0 0 0 0 0 0 0 0 0	Bits           4           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128	be Mil Descri Mux cc Mux cc Mux cc Mux cc Mux cc Param	LIX CCC iption pontrol fc iption eter da eter da	ta 0 ta 1 ta 2 ta 3 ta 0 ta 2 ta 3 ta 0 ta 2 ta 2 ta 3 ta 0 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 2 ta 3 ta 2 ta 2 ta 3 ta 2 ta 2 ta 3 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4		For	
22.1.524.1.5 Name SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux32 24.1.6 SP to S Name SP0 SX0 data0 SP0 SX0 data1 SP0 SX0 data2 SP0 SX0 data2 SP0 SX0 data3 SP1 SX1 data0 SP1 SX1 data1 SP1 SX1 data3 SP2 SX0 data1 SP2 SX0 data1 SP2 SX0 data2 SP2 SX0 data1 SP2 SX0 data2 SP3 SX1 data0 SP3 SX1 data3 SP3 SX1 data3	SEQ to SX0-: F Direction SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SEQ-S: SP0-SX SP0-SX SP0-SX SP0-SX SP0-SX SP1-SX SP1-SX SP1-SX SP1-SX SP1-SX SP1-SX SP2-SX SP2-SX SP2-SX SP2-SX SP3-SX SP3-SX SP3-SX	Paramet Axo Xx0 data 0 0 0 0 0 0 0 0 0 0 0 0 0	Bits           4           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128           128	be Mil Descri Mux cc Mux cc Mux cc Mux cc Mux cc Param	LIX CCC iption pontrol fc iption teter da eter da	ta 0 ta 1 ta 2 ta 2 ta 3 ta 0 ta 1 ta 2 ta 3 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4		For	matted: Bullets and Numbering
22.1.524.1.5         S           SQ_SXx         Mmux01           SQ_SXx         Mmux21           SQ_SXx         Mmux32           24.1.6         SP to S           SQ_SXx         Mmux32           24.1.6         SP to S           SP0_SX0         data0           SP0_SX0         data1           SP0_SX0         data1           SP0_SX0         data1           SP0_SX0         data1           SP1_SX1         data2           SP1_SX1         data2           SP1_SX1         data3           SP2_SX0         data1           SP2_SX0         data1           SP2_SX0         data3           SP2_SX0         data3           SP3_SX1         data1           SP3_SX1         data3           SP3_SX1         data3	SEQ to SX0-: F Direction SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SEQ-S2 SP0-S2 SP2-S2 SP2-S2 SP3-S	Paramet Paramet Xx0 Xx0 data data 0 0 0 0 0 0 0 0 0 0 0 0 0	ter Cac Bits 4 4 4 4 128 128 128 128 128 128 128 128 128 128	bescri Mux cc Mux cc Param	LIX CC iption pontrol for pontrol for pont	ta 0 ta 1 ta 2 ta 2 ta 3 ta 0 ta 1 ta 2 ta 3 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4 ta 4		For	matted: Bullets and Numbering
22.1.524.1.5 SQ_SXx_Mmux01 SQ_SXx_Mmux21 SQ_SXx_Mmux22 24.1.6 SP to S Name SP0 SX0 data0 SP0 SX0 data1 SP0 SX0 data2 SP0 SX0 data2 SP1 SX1 data1 SP1 SX1 data2 SP1 SX1 data3 SP2 SX0 data1 SP2 SX0 data1 SP2 SX0 data1 SP2 SX0 data2 SP2 SX0 data2 SP2 SX0 data3 SP3 SX1 data3 SY1 SY1 SY1 SY1 SY1 SY1 SY1 SY1 SY1 SY1	SEQ to SX0-: F           Direction           SEQ-S:           SP0-SX           SP0-SX           SP1-SX           SP1-SX           SP1-SX           SP2-SX           SP2-SX           SP2-SX           SP3-SX           SY3-SX           SY3-SX           SY3-SX           SY3-SX           SY3-SX           SY3-SX	Paramet Axx0 Axx0 Axx0 data 1 0 0 0 0 0 0 0 0 0 0 0 0 0	ter Cac Bits 4 4 4 4 128 128 128 128 128 128 128 128 128 128	be Mil Descri Mux cc Mux cc Mux cc Mux cc Param	UX CC iption pontrol fc iption iption fc iption fc iption eter da eter da	ta 0 ta 1 ta 2 ta 2 ta 3 ta 0 ta 1 ta 2 ta 3 ta 0 ta 2 ta 3 ta 0 ta 3 ta 0 ta 3 ta 4 ta 4 t		For	matted: Bullets and Numbering

Exhibit 2018 doc R400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*
OBIO4/15 12:56
PM11/16/01-10:17-AM10/19/01-01:40-PM

AMD1044\_0256938

ATI Ex. 2106 IPR2023-00922 Page 27 of 223

	RIGINATE DATE	Ξ	EDIT DATE		R400 Sequencer Specification	PAGE		
24	September, 200	01 <u>4 S</u>	eptember, 20	1516		28 of 35		
22.1.724.1.8	VGT to SP	00/SEG	): Vertex	Bus		*	Formatted: Bullets and Nu	umbering
lame		Direction		Bits	Description			
/GT_SP0_vrtx_inde ndexes	exesVertex	VGT→SF		128	Pointers of indexes or HOS surface infor	mation		
/GT SP0 end of	vectEOF_vector	VGT→SF	PO/SEQ	1	End of the vector			
/GT SP0 vrtx form	nat-Inputs_vert	VGT→SF	PO/SEQ	1	0: Normal 128 bits per vert 1: double 256 bits per vert			
VGT_SQ_end_of_v		VGT→SC		1	End of the vector			
/GT SQ vrtx form	at	<u>VGT→SC</u>	<u>z</u>   :	1	0: Normal 128 bits per vert 1: double 256 bits per vert			
/GT_SQ_stateSTA	TE	VGT→SE	EQ 1	21 <u>3</u>	Render State (6*3+3 for constants)			
22.1.8 CP to	SEQ : Cons	stant-ste	o <del>ro load</del>			*	Formatted: Bullets and Nu	umbering
22.1.9CP to S	SEO · Eatab	Stata a	toro lood					
<u>22.1.00F 10 C</u>	n_Q_FotUH-	<del>0:018-</del> 8	n <del>ure 10au</del>		oon			
22 1 1000 40	SEO · Cont	rol Ctot	a atam la	24		4	Formatted: Bullets and Nu	umbering
22.1.10CP to	•			<del>du</del>	***			
(ISSUE: How,Who							Formatted: Bullets and Nu	umbering
22.1.11 MH (	to SEQ: Insti	ruction	store Loa	d		4		
24.1.9 SEQ (	to CP <sup>.</sup> State	report						
		rection	Bits	e na	escription			
Namo								
SQ_CP_vrtx_state	SE	Q→CP	3	0	dest vertex state still in the pipe			
SQ_CP_vrtx_state	SE			0				
<u>Name</u> SQ_CP_vrtx_state SQ_CP_pix_state {ISSUE: CP_or_MH	SE SE	Q→CP	3	0	dest vertex state still in the pipe			
SQ_CP_vrtx_state SQ_CP_pix_state {ISSUE: CP_or_MH	2} SE 2}	Q→CP Q→CP	3	0	dest vertex state still in the pipe	4	~ - <b>Formatted:</b> Bullets and Nu	umbering
SQ_CP_vrtx_state SQ_CP_pix_state [ISSUE: CP_or_MH	2} SE 2}	Q→CP Q→CP	3	0	Idest vertex state still in the pipe Idest pixel state still in the pipe	4	<b>Formatted:</b> Bullets and Nu	umbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP or MH' 22.1.1224.1.1 Name	2} 10_SP0 to S	<u>iQ→CP</u> i <u>Q→CP</u> :X <i>Q : Pi</i> . rection	3 3 xel <u>/Verte</u> > Bits	<u>o</u> <u>o</u>	Idest vertex state still in the pipe Idest pixel state still in the pipe In the pipe Identification of	4	<b>Formatted:</b> Builets and Nu	.mbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP or MH 22.1.1224.1.1 Name SP0_SX0_Export_d	SE           ?}           10_SP0 to S	Q→CP Q→CP X <i>Q : Pi</i> ection 0→SX0	3 3 xel/ <u>/verte</u> > Bits 64*1	<u>o</u> i oi	Idest vertex state still in the pipe Idest pixel state still in the pipe Ideat from RBs <u>write to SX</u> Description 432-pairss of 32 bits channel values	4	<b>Formatted:</b> Builets and Nu	umbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP_or_MH 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Export_d	SE           ?}           10         SP0 to S           lata         SP           Dest         SP	$Q \rightarrow CP$ $Q \rightarrow CP$ $C \rightarrow CP$	3 3 xel/Verte> Bits 64*4 4	<u>o</u> <u>o</u>	Idest vertex state still in the pipe Idest pixel state still in the pipe In th	destinations	(Formatted: Bullets and Nu	umbering
SQ_CP_vrtx_state SQ_CP_pix_state ISSUE: CP_or_MH 22.1.1224.1.1 Name SPO_SX0_Export_d SPO_SX0_Shader SP1_SX1_Export_d	2} 10 SPO to S Lata SP Lata SP Lata SP Lata SP	$\begin{array}{c} Q \rightarrow CP \\ \hline Q \rightarrow CP \\ \hline \hline Q \rightarrow CP \\ \hline \hline C \\ \hline C \\ \hline \hline C \\ \hline C \\ \hline C \\ \hline \hline C \\ \hline C \\ \hline \hline \hline C \\ \hline \hline \hline C \hline \hline \hline C \\ \hline \hline \hline C \hline \hline \hline \hline$	3 3 xel/Vertex Bits 64*1 4 256	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBs <u>write to SX</u> Description 432_pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values		( Formatted: Bullets and Nu	.mbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP or MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Export_d SP1_SX1_Export_d SP1_SX1_Export_d SP1_SX1_Shader	SE       ?}       10_SP0 to S       lata       SP       Jata       Dest       Jata       SP	$\begin{array}{c} Q \rightarrow CP \\ \hline Q \rightarrow CP \\ \hline \hline Q \rightarrow CP \\ \hline \hline \\ \hline \\ ection \\ \hline 0 \rightarrow SX0 \\ \hline 0 \rightarrow SX0 \\ \hline 1 \rightarrow SX1 \\ \hline 1 \rightarrow SX1 \\ \hline \end{array}$	3 3 xel/Verte) Bits 64*1 4 256 4	<u>o</u> <u>o</u>	dest vertex state still in the pipe idest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export		(Formatted: Bullets and Nu	umbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP_or_MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Shader SP1_SX1_Shader SP1_SX1_Shader SP1_SX1_Shader SP1_SX1_Shader	SE       ?}       10_SP0 to S       lata       SP       lata       Dest       SP       lata       SP       lata       SP       lata       SP       lata       SP       lata       SP       lata	$\begin{array}{c} Q \rightarrow CP \\ \hline Q \rightarrow CP \\ \hline Q \rightarrow CP \\ \hline \hline Q \rightarrow CP \\ \hline \hline Q \rightarrow CP \\ \hline \hline Q \rightarrow SX0 \\ \hline 0 \rightarrow SX0 \\ \hline 1 \rightarrow SX1 \\ \hline 1 \rightarrow SX1 \\ \hline 2 \rightarrow SX0 \\ \hline \end{array}$	3 3 xel/Vertex 64*1 4 256 4 256	<u>o</u> <u>o</u>	dest vertex state still in the pipe idest pixel state still in the pipe ad from RBswrite to SX Description 432_pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values	destinations	<b>Formatted:</b> Bullets and Nu	umbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP or MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Export_d SP1_SX1_Export_d SP1_SX1_Shader SP2_SX0_Export_d SP2_SX0_Shader	SE       ?}       10_SP0 to S       Dest       SP       lata       SP       Dest       SP       Lata       SP       Dest       SP       Dest       SP       Dest       SP       Dest       SP       Dest       SP	$\begin{array}{c} Q \rightarrow CP \\ \hline Q \rightarrow CP \\ \hline Q \rightarrow CP \\ \hline \hline \\ ection \\ \hline 0 \rightarrow SX0 \\ \hline 0 \rightarrow SX0 \\ \hline 1 \rightarrow SX1 \\ \hline 1 \rightarrow SX1 \\ \hline 2 \rightarrow SX0 \\ \hline 2 \rightarrow SX0 \\ \hline \end{array}$	3 3 xel/Verte) Bits 64*1 4 256 4 256 4	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export	destinations	<b>Formatted:</b> Bullets and Nu	.mbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP_or_MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Export_d SP1_SX1_Shader SP2_SX0_Shader1 SP2_SX0_Shader1 SP2_SX0_Shader1 SP3_SX1_Export_d	SE       ?}       10     SP0 to S       lata     SP       Dest     SP       lata     SP       lata     SP       lata     SP	$\begin{array}{c} Q \rightarrow CP \\ \hline Q \rightarrow CP \\ \hline Q \rightarrow CP \\ \hline \hline Q \rightarrow CP \\ \hline \hline Q \rightarrow CP \\ \hline \hline Q \rightarrow SX0 \\ \hline 0 \rightarrow SX0 \\ \hline 1 \rightarrow SX1 \\ \hline 1 \rightarrow SX1 \\ \hline 2 \rightarrow SX0 \\ \hline \end{array}$	3 3 xel/Vertex 64*1 4 256 4 256	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values	destinations destinations		umbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP_or_MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Export_d SP1_SX1_Shader SP2_SX0_Shader1 SP2_SX0_Shader1 SP2_SX0_Shader1 SP3_SX1_Export_d	SE       ?}       10     SP0 to S       lata     SP       Dest     SP       lata     SP       Dest     SP       lata     SP       Dest     SP       lata     SP       lata     SP       lata     SP       lata     SP       lata     SP       lata     SP	$\begin{array}{c} Q \rightarrow CP \\ \hline Q \rightarrow CP \\ \hline Q \rightarrow CP \\ \hline \hline \\ ection \\ \hline 0 \rightarrow SX0 \\ \hline 0 \rightarrow SX0 \\ \hline 1 \rightarrow SX1 \\ \hline 1 \rightarrow SX1 \\ \hline 2 \rightarrow SX0 \\ \hline 2 \rightarrow SX0 \\ \hline \end{array}$	3 3 xel/Verte) Bits 64*1 4 256 4 256 4	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export	destinations destinations	( Formatted: Bullets and Nu	umbering
SQ_CP_vrtx_state SQ_CP_pix_state SSUE: CP_or_MH 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Export_d SP1_SX1_Export_d SP1_SX1_Shader SP2_SX0_Export_d SP2_SX0_Shader SP3_SX1_Export_d SP3_SX1_Export_d SP3_SX1_Export_d	SE       ?}       10     SP0 to S       lata     SP       Dest     SP	$C \rightarrow CP$ $C \rightarrow $	3 3 xel/Verte) Bits 64*1 4 256 4 256 4 256	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values	destinations destinations destinations	<b>Formatted:</b> Bullets and Nu	Imbering
SQ_CP_vrtx_state SQ_CP_pix_state SQ_CP_pix_state (ISSUE: CP_or_MH' 22.1.1224.1.1 SP0_SX0_Export_d SP0_SX0_Export_d SP1_SX1_Export_d SP1_SX1_Shader SP2_SX0_Export_d SP2_SX0_Shader SP3_SX1_Export_d SP3_SX1_Export_d SP3_SX1_Export_d	SE       ?}       10     SP0 to S       lata     SP       Dest     SP	iQ→CP iQ→CP iQ→CP iQ→CP iQ→CP iQ→SX0 iQ→SX0 iQ→SX0 iQ→SX1 iQ→SX1 iQ→SX1 iQ→SX1 iQ→CP iQ→SX0 iQ→SX0 iQ→SX1 iQ→SX1 iQ→SX0 iQ→SX1 iQ→	3 3 xel/Verte> Bits 64*1 4 256 4 256 4 256 4 256 4	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432_pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export each set of four pixels or vectors is eight clocks. This field specifies when	destinations destinations destinations exported over	<b>Formatted:</b> Bullets and Nu	umbering
SQ_CP_vrtx_state SQ_CP_pix_state {ISSUE: CP_or_MH	SE       ?}       [0] SP0 to S       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Dest       SP       Lata       SP       Dest       SP       Count	iQ→CP iQ→CP iQ→CP iQ→CP iQ→CP iQ→SX0 iQ→SX0 iQ→SX0 iQ→SX1 iQ→SX1 iQ→SX1 iQ→SX1 iQ→CP iQ→SX0 iQ→SX0 iQ→SX1 iQ→SX1 iQ→SX0 iQ→SX1 iQ→	3 3 xel/Verte> Bits 64*1 4 256 4 256 4 256 4 256 4	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export that sequence. The current export clause is over	destinations destinations destinations exported over e the SP is in	<b>Formatted:</b> Bullets and Nu	.mbering
SQ_CP_vrtx_state <u>SQ_CP_pix_state</u> {ISSUE: CP_or_MH' <u>22.1.1224.1.1</u> Name <u>SP0_SX0_Export_d</u> <u>SP0_SX0_Shader</u> <u>SP1_SX1_Shader</u> <u>SP1_SX1_Shader</u> <u>SP2_SX0_Export_d</u> <u>SP2_SX0_Export_d</u> <u>SP2_SX0_Export_d</u> <u>SP3_SX1_Export_d</u> <u>SP3_SX1_Export_d</u> <u>SP3_SX1_Shader</u> <u>SP3_SX1_Shader</u>	SE       ?}       [0] SP0 to S       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Dest       SP       Lata       SP       Dest       SP       Count	iQ→CP iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Vertex         Bits           64*1         4           256         4           256         4           256         4           3         3	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export Each set of four pixels or vectors is eight clocks. This field specifies when that sequence. The current export clause is over clock)	destinations destinations destinations exported over e the SP is in (true for one	<b>Formatted:</b> Bullets and Nu	mbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP or MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Export_d SP1_SX1_Export_d SP2_SX0_Export_d SP2_SX0_Export_d SP3_SX1_Export_d SP3_SX1_Export_d SP3_SX1_Shader_ SP3_SX1_Shader_	SE       ?}       [0] SP0 to S       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Dest       SP       Lata       SP       Dest       SP       Count	iQ→CP iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Vertex         Bits           64*1         4           256         4           256         4           256         4           3         3	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432_pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 p	destinations destinations destinations exported over e the SP is in (true for one wo* cycles to er the last RB	<b>Formatted:</b> Bullets and Nu	mbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP or MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Export_d SP1_SX1_Export_d SP2_SX0_Export_d SP2_SX0_Export_d SP3_SX1_Export_d SP3_SX1_Export_d SP3_SX1_Shader_ SP3_SX1_Shader_	SE       ?}       [0] SP0 to S       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Dest       SP       Lata       SP       Dest       SP       Count	iQ→CP iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Vertex         Bits           64*1         4           256         4           256         4           256         4           3         3	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 6 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 6 p	destinations destinations exported over e the SP is in (true for one wo* cycles to er the last RB ort instruction,	<b>Formatted:</b> Bullets and Nu	mbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP or MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Export_d SP1_SX1_Export_d SP2_SX0_Export_d SP2_SX0_Export_d SP3_SX1_Export_d SP3_SX1_Export_d SP3_SX1_Shader_ SP3_SX1_Shader_	SE       ?}       [0] SP0 to S       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Dest       SP       Lata       SP       Dest       SP       Count	iQ→CP iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Vertex         Bits           64*1         4           256         4           256         4           256         4           3         3	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432_pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 p	destinations destinations exported over e the SP is in (true for one wo* cycles to er the last RB ort instruction,		<u>.mbering</u>
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP or MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Export_d SP1_SX1_Export_d SP2_SX0_Export_d SP2_SX0_Export_d SP3_SX1_Export_d SP3_SX1_Export_d SP3_SX1_Shader_ SP3_SX1_Shader_	SE       ?}       [0] SP0 to S       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Dest       SP       Lata       SP       Dest       SP       Count	iQ→CP iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Vertex         Bits           64*1         4           256         4           256         4           256         4           3         3	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 6 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 6 p	destinations destinations exported over e the SP is in (true for one wo* cycles to er the last RB ort instruction, e first export	<b>Formatted:</b> Bullets and Nu	Imbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP or MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Export_d SP1_SX1_Export_d SP2_SX0_Export_d SP2_SX0_Export_d SP3_SX1_Export_d SP3_SX1_Export_d SP3_SX1_Shader_ SP3_SX1_Shader_	SE       ?}       [0] SP0 to S       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Dest       SP       Lata       SP       Dest       SP       Count	iQ→CP iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Vertex         Bits           64*1         4           256         4           256         4           256         4           3         3	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of	destinations destinations exported over e the SP is in (true for one wo* cycles to er the last RB ort instruction, e first export ed on the first	<b>Formatted:</b> Bullets and Nu	Imbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP_or_MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Shader SP1_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader_ SP3_SXX_Shader_	SE       ?}       10_SP0 to S       lata       SP       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Lata       SP       Last	iQ→CP iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Vertex         Bits           64*1         4           256         4           256         4           256         4           3         3	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export Each set of four pixels or vectors is eight clocks. This field specifies when that sequence. The current export clause is over clock) The last export instruction creates *t the RB. This needs to be set on or aff cycle that is produced by the last expo but before the first RB cycle of the instruction of the next clause.Asserte	destinations destinations exported over e the SP is in (true for one wo* cycles to er the last RB ort instruction, e first export ed on the first clause	<b>Formatted:</b> Bullets and Nu	mbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP or MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Export_d SP1_SX1_Export_d SP2_SX0_Export_d SP2_SX0_Export_d SP3_SX1_Export_d SP3_SX1_Export_d SP3_SX1_Shader_ SP3_SX1_Shader_	SE       ?}       10_SP0 to S       lata       SP       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Lata       SP       Last	iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Verte;         8           64*1         4           256         4           256         4           256         4           3         3           1         1	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432_pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of the of the tast export of the of the of the of the of the tast export of the of the of the tast export of the of the tast export of the of the tas	destinations destinations exported over e the SP is in (true for one wo* cycles to ter the last RB ort instruction, e first export ad on the first clause which must be	<b>Formatted:</b> Bullets and Nu	mbering
SQ_CP_vrtx_state SQ_CP_pix_state SQ_CP_pix_state SSUE: CP_or_MH 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader_ SP3_SXX_Shader_	SE       ?}       10_SP0 to S       lata       SP       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Lata       SP       Last	iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Verte;         8           64*1         4           256         4           256         4           256         4           3         3           1         1	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432_pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 pecifies one of the of up to 12 export 5 p	destinations destinations exported over e the SP is in (true for one wo* cycles to ter the last RB ort instruction, e first export clause which must be all color	<b>Formatted:</b> Bullets and Nu	imbering
SQ_CP_vrtx_state SQ_CP_pix_state (ISSUE: CP_or_MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Shader SP1_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader_ SP3_SXX_Shader_	SE       ?}       10_SP0 to S       lata       SP       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Lata       SP       Last	iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Verte;         8           64*1         4           256         4           256         4           256         4           3         3           1         1	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export ta pairs of 32 bits channel values Specifies one of the of up to 12 export ta pairs of 32 bits channel values Specifies one of the of up to 12 export Each set of four pixels or vectors is eight clocks. This field specifies when that sequence. The current export clause is over clock) The last export instruction creates *t the RB. This needs to be set on or aff cycle that is produced by the last export but before the first RB cycle of th instruction of the next clause_Asserted shader count of the last export of the c Result of pixel kill in the shader pipe, v output for all pixel exports (depth and	destinations destinations exported over e the SP is in (true for one wo* cycles to ter the last RB ort instruction, e first export clause which must be all color	<b>Formatted:</b> Bullets and Nu	Imbering
SQ_CP_vrtx_state SQ_CP_pix_state SQ_CP_pix_state SSUE: CP_or_MH 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader_ SP3_SXX_Shader_	SE       ?}       10_SP0 to S       lata       SP       lata       SP       lata       Dest       SP       lata       Dest       SP       lata       SP       Lata       SP       Last	iQ→CP iQ→CP iQ→CP iQ→CP i i i i i i i i i i i i i	3         3           3         3           xel/Verte;         8           64*1         4           256         4           256         4           256         4           3         3           1         1	<u>o</u> <u>o</u>	dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export Each set of four pixels or vectors is eight clocks. This field specifies when that sequence. The current export clause is over clock) The last export instruction creates *t the RB. This needs to be set on or aft the RB. This needs to be set on or aft but before the first RB cycle of th instruction of the next clause.Assertes shader count of the last export of the c Result of pixel kill in the shader pipe, v output for all pixel exports (depth and a buffers). 4x4 because 16 pixels are co	destinations destinations exported over e the SP is in (true for one wo* cycles to ter the last RB ort instruction, e first export clause which must be all color	<b>Formatted:</b> Bullets and Nu	mbering
SQ_CP_vrtx_state SQ_CP_pix_state ISSUE: CP_or_MH' 22.1.1224.1.1 Name SP0_SX0_Export_d SP0_SX0_Shader SP1_SX1_Shader SP1_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader SP3_SX1_Shader_ SP3_SXX_Shader_	SE       ?}       10_SP0 to S       lata       SP       Last       PixelValid	<u>iQ</u> → <u>CP</u> <u>iQ</u> → <u>CP</u> <u>iQ</u> → <u>CP</u> <u>iQ</u> → <u>SX0</u> <u>0</u> → <u>SX0</u> <u>1</u> → <u>SX1</u> <u>1</u> → <u>SX1</u> <u>2</u> → <u>SX0</u> <u>3</u> → <u>SX1</u> <u>3</u> → <u>SX1</u> <u>0</u> →SX0 0→SX0 0→SX0	3         3           3         3           xel/Vertex         8its           64*1         4           256         4           256         4           256         4           3         3           1         1		dest vertex state still in the pipe dest pixel state still in the pipe ad from RBswrite to SX Description 432-pairss of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export 4 pairs of 32 bits channel values Specifies one of the of up to 12 export Each set of four pixels or vectors is eight clocks. This field specifies when that sequence. The current export clause is over clock) The last export instruction creates *t the RB. This needs to be set on or aft the RB. This needs to be set on or aft but before the first RB cycle of th instruction of the next clause.Assertes shader count of the last export of the c Result of pixel kill in the shader pipe, v output for all pixel exports (depth and a buffers). 4x4 because 16 pixels are co	destinations destinations exported over e the SP is in (true for one wo* cycles to er the last RB ort instruction, e first export d on the first clause which must be all color imputed per		mbering

ATI Ex. 2106 IPR2023-00922 Page 28 of 223

	r						r
AP	ORIGINATE	DATE	ED	IT DATE		DOCUMENT-REV. NUM.	PAGE
	24 September			4 September, 201516		GEN-CXXXXX-REVA	29 of 35
SP0_SX0_Sha	ader_WordValid	SP0→SX	D	2	wo	ecifies whether to write low and/or higl rd of the 64-bit export data from each els or vectors	
SP1_SX1_Sha	ader_PixelValid	<u>SP1→SX</u>	1	4	out	sult of pixel kill in the shader pipe, whi put for all pixel exports (depth and all fers). 4x4 because 16 pixels are comp ck	color
SP1_SX1_Sha	ader_WordValid	<u>SP1→SX</u>	1	2	wo	ecifies whether to write low and/or hig rd of the 64-bit export data from each els or vectors	
SP2 SX0 Sha	ader_PíxelValid	<u>SP2→SX</u>	0	4	out	sult of pixel kill in the shader pipe, whi put for all pixel exports (depth and all fers). 4x4 because 16 pixels are comp ck	color
SP2_SX0_Sha	ader WordValid	<u>SP2→SX</u>	0	2	wo	ecifies whether to write low and/or hig rd of the 64-bit export data from each els or vectors	
SP3 SX1 Sha	ader_PixelValid	<u>SP3→SX</u>	1	4	out	sult of pixel kill in the shader pipe, whi put for all pixel exports (depth and all fers). 4x4 because 16 pixels are comp ck	color
SP3_SX1_Sha	ader_WordValid	<u>SP3→SX</u>	1	2	woi	ecifies whether to write low and/or hig rd of the 64-bit export data from each els or vectors	

# 22.1.1324.1.11 SEQ to SXX0-: Control bus

Name	Direction	Bits	Description
SQ_SXx_Export_Pixel	SEQ→SXx0	1	1: Pixel
			0: Vertex
SQ_SXx_Export_SEND	SEQ→SX <u>x</u> 0	1	Raised to indicate that the SQ is starting an export
SQ_SXx_Export_Clause	SEQ→SX⊻0	3	Clause number, which is needed for vertex clauses
SQ_SXx_Export_State	SEQ→SX <u>×</u> 0	21? <u>3</u>	State ID, which is needed for vertex clauses

These fields are sent synchronously with SP export data, described in SP0 $\rightarrow$ SX0 interface {ISSUE: Where are the PC pointers}

22.1.1424.1.12 SX0 to SEQ : Output file control

Name	Direction	Bits	Description
SXx_SQ_Export_RTScount_rd Y	<u>SX0SXx</u> →SEQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_Export_Position	SX <u>x</u> 0→SEQ	1	Specifies whether there is room for another position.
<u>SXx SQ</u> Export_Buffer	SX0 <u>x</u> →SEQ	7	Specifies the space available <u>available</u> 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 2018 doc R400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*
OBIO4/15 12:56
PM11/16/01-10:17-AM10/19/01-01:40-PM

Formatted: Bullets and Numbering

....

Formatted: Bullets and Numbering

AMD1044\_0256940

ATI Ex. 2106 IPR2023-00922 Page 29 of 223

		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
		24 September, 2001	4 September, 201516		30 of 35	
-	00116		Neversbor 2001		1	Formatted: Bullets and Numbering

22.1.15 SP0 to SX0 : Position return bus

#### 22.1.1624.1.13 Shader Engine to Fetch Unit Bus (Fast Bus)

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

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

Name	Direction	Bits	Description
Tex_RegFile_Read_DataSP0_TP0_fetch_addr	SP0->TEXP0	2048512	16-4 Fetch Addresses read from the
			Register file
Tex_RegFile_Write_DataTP0_SP0_data	T <u>P0</u> EX→SP0	2048512	16 4 texture results
SP1 TP1 fetch addr	<u>SP1-&gt;TP1</u>	512	4 Fetch Addresses read from the
			Register file
TP1_SP1_data	<u>TP1→SP1</u>	512	4 texture results
SP2_TP2_fetch_addr	SP2->TP2	512	4 Fetch Addresses read from the
			Register file
TP2 SP2 data	TP2→SP2	512	4 texture results
SP3_TP3_fetch_addr	SP3->TP3	512	4 Fetch Addresses read from the
			Register file
TP3_SP3_data	TP3→SP3	512	4 texture results
TPx_SPx_gpr_dst	TPx→SPx	7	Write address into the gprs
TPx_SPx_gpr_cmask	TPx→SPx	4	Channel mask

# 22.1.1724.1.14 Sequencer to Fetch Unit bus (Slow Bus)

Once every four-clock, the fetch unit sends to the sequencer on wich<u>which</u> 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<u>instruction</u> 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_ReadyTPx_SQ_data_rdy	TEXPx→ SEQ	1	Data ready
TPx_SQ_clause_numTex_Clause_Num	TEXPx→ SEQ	3	Clause number
<u>SQ TPx const</u> Tex_est	SEQ→TEX <u>Px</u>	10 <u>64</u>	Fetch state address 10 bits ser over 4 clocks
Tex_InstSQ_TPx_instuct	SEQ→TEX <u>Px</u>	1 <u>224</u>	Fetch instruction address 12 bit sent over 4 clocks
SQ_TPx_end_of_clauseEO_CLAUSE	SEQ→TEXPx	1	Last instruction of the clause
PHASESQ TPx phase	SEQ→TEXPx	12	Write phase signal
SQ_TP0_lod_correctLOD_CORRECT	SEQ→TEXP0	696	LOD correct 3 bits per comp components per quad * 16 quads
MaskSQ TP0 pmask	SEQ→TEXP0	644	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	<u>SQ→TP1</u>	6	LOD correct 3 bits per comp components per guad guads
SQ TP1 pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	<u>SQ→TP2</u>	<u>6</u>	LOD correct 3 bits per comp components per quad quads
SQ_TP2_pmask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ TP3 lod correct	<u>SQ→TP3</u>	6	LOD correct 3 bits per comp components per quad quads
SQ TP3 pmask	SQ→TP3	4	Pixel mask 1 bit per pixel
Tex_Clause_NumSQ_TPx_clause_num	SEQ→TEXPx	3	Clause number
Tex Write Register IndexSQ TPx write gpr index	SEQ->TEXPx	7	Index into Register file for write

Exhibit 2018.doc R400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:54

Formatted: Bullets and Numbering

ATI Ex. 2106 IPR2023-00922 Page 30 of 223

AMD1044\_0256941

5 [ ]	ORIGINATE D	2001 <u>4 Septer</u>	IT DATE nber, 201		35
1.15 Seq	uencer to S	SP: GPR contr	<i>'ol</i>		Formatted: Bullets and Numbering
Vame		Direction	Bits	Description	
SQ SPx gpr v	/r_addr	<u>SQ→SPx</u>	7	Write address	-1
SQ_SPx_gpr_r		<u>SQ→SPx</u>	7	Read address	
<u>SQ_SPx_gpr_r</u> SQ_SPx_gpr_v		<u>SQ→SPx</u> SQ→SPx	1	Read Enable Write Enable	
SQ SPX gpr p		<u>SQ→SFX</u> SQ→SPX	2	The phase mux	-
SQ SPx gpr c	hannel mask	<u>SQ→SPx</u>	4	The channel mask	
SQ SP0 gpr p		SQ→SP0	4	The pixel mask	
<u>SQ SP1 gpr p</u> SQ SP2 gpr p		$SQ \rightarrow SP1$ SQ $\rightarrow SP2$	4	The pixel mask The pixel mask	
SQ SP3 gpr p		SQ→SP3	4	The pixel mask	
1.16 Seq	uencer to S	SPx: Paramete		e write control	Formatted: Bullets and Numbering
Vame		Direction	Bits	Description	
SQ SPx pc wi SQ SPx pc wi		<u>SQ→SPx</u> SQ→SPx	1	Write Enable	
SQ SPx pc pl	CONTRACTOR CONTRACTOR	SQ→SPx	1	The output selector mux (gpr vs parameter cache)	
1.17 Seq	uencer to S	SPx: Instructio	ns		Formatted: Bullets and Numbering
Vame		Direction	Bits	Description	
SQ SPx instru	11 11 11 11 11 11 11 11 11 11 11 11 11	<u>SQ→SPx</u>	1	Instruction start	
<u>SQ_SP_instruc</u> SQ_SPx_stall	<u>t</u>	SQ→SPx SQ→SPx	20	Instruction sent over 4 clocks Stall signal	
SQ_SPx_Shad	er_Count	<u>SQ→SPx</u> SQ→SPx	3	Each set of four pixels or vectors is exported ov eight clocks. This field specifies where the SP is that sequence.	
SQ_SPx_Shad	er_Last	<u>SQ→SPx</u>	1	Asserted on the first shader count of the last expo	<u>nt</u>
SQ_SP0_Shad	er_PixelValid	<u>SQ→SP0</u>	4	Result of pixel kill in the shader pipe, which must is output for all pixel exports (depth and all col buffers). 4x4 because 16 pixels are computed p clock	or
SQ_SP0_Shad	er_WordValid	<u>SQ→SP0</u>	2	Specifies whether to write low and/or high 32- word of the 64-bit export data from each of the pixels or vectors	
SQ SP1 Shad	er_PixelValid	<u>SQ→SP1</u>	4	Result of pixel kill in the shader pipe, which must l output for all pixel exports (depth and all col buffers). 4x4 because 16 pixels are computed p clock	or
SQ SP1 Shad	er WordValid	<u>SQ→SP1</u>	2	Specifies whether to write low and/or high 32- word of the 64-bit export data from each of the pixels or vectors	
SQ_SP2_Shad	er_PixelValid	<u>SQ→SP2</u>	4	Result of pixel kill in the shader pipe, which must a output for all pixel exports (depth and all col buffers). 4x4 because 16 pixels are computed p clock	or
	ar Mard Calid	<u>SQ→SP2</u>	2	Specifies whether to write low and/or high 32- word of the 64-bit export data from each of the	
				pixels or vectors	
SQ_SP3_Shad	er PixelValid	<u>SQ→SP3</u> SQ→SP3	4	pixels or vectors Result of pixel kill in the shader pipe, which must if output for all pixel exports (depth and all col buffers). 4x4 because 16 pixels are computed p clock Specifies whether to write low and/or high 32-	or er

ATI Ex. 2106 IPR2023-00922 Page 31 of 223

	ORIGINATE DATE	EDIT I		R400 Sequencer Specification	PAGE	
	24 September, 2001	4 Septembe		z	32 of 35	
				word of the 64-bit export data from ea pixels or vectors	ich of the 16	
.1.18 S	SP to Sequencer:	Constant a	ddres	s load	+	Formatted: Bullets and Numbering
Name	Dire	ction	Bits	Description		
AC435743753994530		→SQ	36	Constant address load to the sequencer		Formatted
SPO SQ V		→SQ	1	Data valid		Formatted
SP1 SQ ( SP1 SQ )		<u>→SQ</u>	36	Constant address load to the sequencer		Formatted
And a state of the second		<u>→SQ</u> →SQ	1 36	Data valid Constant address load to the sequencer		(1)
SP2 SQ 1		→SQ	1	Data valid		Formatted
		→SQ	36	Constant address load to the sequencer		
SP3 SQ V	valid SP3	<u>→SQ</u>	1	Data valid	]	
.1.19 S	Sequencer to SP>	c constant l	oroad	cast	4	- Formatted: Bullets and Numbering
Name	Dire	ction	Bits	Description	]	
SQ SPx o		→SPx	128	Constant broadcast		- Formatted
1 20 0	DO to Securação	r Kill unata	land		*	Formatted
.1.20 8	SP0 to Sequence		iuau		) (	Formatted
Name		ction	Bits	Description		Formatted
SPO SQ I		→SQ	4	Kill vector load		
SP1_SQ_I SP2_SQ_I		<u>→SQ</u> →SQ	4 4	Kill vector load		Formatted: Bullets and Numbering
SP3 SQ 1		<u>→SQ</u>	4	Kill vector load		
			12	Tim Veeter load	[	- Formatted: Bullets and Numbering
	SQ to CP: RBBM					( · · · · · · · · · · · · · · · · · · ·
Name	Accession and a second second	ction	Bits	Description		
SQ RBB			<u>1</u> 32	Read Strobe Read Data		
SQ RBBA	Moorana		1	Optional		
SQ RBBN			1	Real-Time (Optional)		
1 22 0	CP to SQ: RBBM	bus	,	and an	 	Formatted: Bullets and Numbering
		and the second se	10:40	Deparintian		
Name rbbm we	CP-	<u>ction</u> →SQ	Bits 1	Description Write Enable		
rbbm a	CP		18	Address Upper Extent is TBD		
	CP-		32	Data		
CONTRACTOR OF THE OWNER OWNE	00	0.0	4	how 5 here 3 2		
rbbm_wd rbbm_be	CP-		4	Byte Enables		
rbbm wd rbbm be rbbm re	CP-	→SQ	1	Read Enable		
rbbm_wd rbbm_be rbbm_re rbb_rs0	CP- CP-	→SQ →SQ	1	Read Enable Read Return Strobe 0		
rbbm wd rbbm be rbbm re rbb rs0 rbb rs1	CP CP CP	<u>→SQ</u> →SQ →SQ	1	Read Enable Read Return Strobe 0 Read Return Strobe 1		
rbbm_wd rbbm_be rbbm_re rbb_rs0 rbb_rs1 rbb_rd0	CP- CP- CP- CP- CP-	>SQ >SQ >SQ >SQ >SQ	1 1 32	Read Enable Read Return Strobe 0 Read Return Strobe 1 Read Data 0		
rbbm_wd rbbm_be rbbm_re rbb_rs0 rbb_rs1 rbb_rd0 rbb_rd1	CP- CP- CP- CP- CP-	-SQ -SQ -SQ -SQ -SQ -SQ	1	Read Enable Read Return Strobe 0 Read Return Strobe 1 Read Data 0 Read Data 0		
rbbm wd rbbm be rbbm re rbb rs0 rbb rs1 rbb rd0 rbb rd1	CP- CP- CP- CP- CP-	-SQ -SQ -SQ -SQ -SQ -SQ	1 1 1 32 32 32	Read Enable Read Return Strobe 0 Read Return Strobe 1 Read Data 0		→ Formatted: Bullets and Numbering
rbbm wd rbbm be rbbm re rbb rs0 rbb rs1 rbb rd0 rbb rd1 RBBM S0	CP- CP- CP- CP- CP- 2 soft reset CP-	+SQ +SQ +SQ +SQ +SQ +SQ +SQ	1 1 32 32 1	Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Read Data 0         Soft Reset		<b>Formatted:</b> Bullets and Numbering
rbbm wd rbbm be rbbm re rbb rs0 rbb rs1 rbb rd0 rbb rd1 RBBM_S0	CP- CP- CP- CP- CP-	+SQ +SQ +SQ +SQ +SQ +SQ +SQ	1 1 32 32 1	Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Read Data 0         Soft Reset		- <b>Formatted:</b> Bullets and Numbering
rbbm wd rbbm be rbbm re rbb rs0 rbb rs1 rbb rd0 rbb rd1 RBBM S0	CP- CP- CP- 2 soft reset CP- 2 soft reset CP- CP- CP- CP- CP- CP- CP- CP- CP- CP-	<u>-50</u> <u>-50</u> <u>-50</u> <u>-50</u> <u>-50</u> gram exec	1 1 32 32 1 	Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Read Data 0         Soft Reset		<b>Formatted:</b> Bullets and Numbering
rbbm wd rbbm be rbbm re rbb rs0 rbb rs1 rbb rd0 rbb rd1 RBBM SC	CP- CP- CP- CP- CP- 2 soft reset CP-	<u>-50</u> <u>-50</u> <u>-50</u> <u>-50</u> <u>-50</u> gram exec	1 1 32 32 1 	Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Read Data 0         Soft Reset		<b>Formatted:</b> Bullets and Numbering
rbbm wd rbbm re rbbm rs0 rbb rs1 rbb rd1 RBBM SC -25. E) -1.125.	CP CP CP 2 soft reset 2 soft reset CP 2 soft reset CP CP CP CP CP CP CP CP CP CP CP CP CP	<u>-sq</u> <u>-sq</u> <u>-sq</u> <u>-sq</u> <u>-sq</u> gram exec Control of a	1 1 32 32 1 Vecto	Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Read Data 0         Soft Reset	Vertex FIFO	<b>Formatted:</b> Bullets and Numbering
rbbm wd rbbm re rbbm re rbb rs0 rbb rd0 rbb rd0 -25. E> -1.125. PA sends • state p	CP         CP	Image: square       Image: square	1 1 32 32 1	Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Soft Reset         S         r of Vertices         - 32 bits/index for 2048 bit total) to the RE's talong with vertices	Vertex FIFO	Formatted: Bullets and Numbering
rbbm wd rbbm re rbbm re rbb rs0 rbb rd0 rbb rd0 -25. E> -1.125. PA sends • state p	CP         CP	Image: square       Image: square	1 1 32 32 1	Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Soft Reset	Vertex FIFO	Formatted: Bullets and Numbering
rbbm wd rbbm re rbbm re rbb rs0 rbb rd0 rbb rd0 rbb rd1 RBBM SC -25. E> -1.125. PA sends • state p	CP         CP         CP         CP         CP         Q soft reset         CP         xamples of prog         1.1         Sequencer C         a vector of 64 vertices         pointer as well as tag in         was allocated in the pointer	Image: SQ	1 1 1 32 32 1 wution	Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Soft Reset         S         r of Vertices         - 32 bits/index for 2048 bit total) to the RE's talong with vertices		Formatted: Bullets and Numbering

ATI Ex. 2106 IPR2023-00922 Page 32 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201516	GEN-CXXXXX-REVA	33 of 35

- 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 arbitrerarbiter 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 2048 bits/cycle)
   the 64 vertex indices are sent to the 64 register files over 4 cycles
  - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
  - RF1 of SU0, SU1, SU2, and SU3 is written the list cycle
     RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
  - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
  - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
  - the index is written to the least significant 32 bits (floating point format?) (what about compound indices) of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
- SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
  - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
- TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   TSM0 was first selected by the TSM arbiter before it could start
- 7. all instructions of fetch clause 0 are issued by TSM0
- the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
  - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
  - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
- ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
- 10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
- the control packet continues to travel down the path of reservation stations until all clauses have been executed
   position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
  - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
  - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
  - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
  - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
  - parameter data is sent to the Parameter Cache over a dedicated bus
  - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
  - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full

Exhibit 2018.docR400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:54

AMD1044\_0256944

601000	Sector Contraction	I	[		1	
	An	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
		24 September, 2001	4 September, 201516		34 of 35	
12	after the s shader pr		leted, the SEQ will free up	the GPRs so that they can be used b	y another	
22		-	ntral of a Vactor a	FDivolo	*	Formatted: Bullets and Numbering
Tent -	<del>, 1, <u>4</u> <u>4</u> <u>3</u>.</del>	1.2_Sequencer Co	ontrol of a Vector of	FIXEIS		
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.	<ul> <li>the state</li> </ul>	ate pointer and the LOD c	orrection bits are also plac	o for pixel quads by the detailed walke ed in the Pixel FIFO n of barycentrics per cycle	r	
3.		rates between Pixel FIFO register files for vertices, t		here are no vertices pending OR ther	e is no space	
4.	<ul> <li>the nu state</li> </ul>	imber of GPRs required b pointer		eed by the program a local state register, which is access ler until space in the register file has b	-	
5.	SEQ cont	rols the transfer of interpo		er file over the RE_SP interface (which		
6.	SEQ cons fetch state • note t • the co	tructs a control packet for e machine 0, or TSM0 FIF hat there is a separate set ntrol packet contains the	the vector and sends it to O) t of reservation stations/ari state pointer, the register f	the first reservation station (the FIFO biters/state machines for vertices and ile base pointer, and the LOD correction or example) travels in a separate FIFC	for pixels on bits	
7.	TSM0 acc	epts the control packet ar		for fetch clause 0 from the global instr		
8.	all instruct	tions of fetch clause 0 are	issued by TSM0			
9.		I packet is passed to the r	next reservation station (th	e FIFO in front of ALU state machine	0, or ASM0	
	<ul> <li>once factor</li> <li>association</li> </ul>	for the fetch data to the T the TU has written all the iated with the ASM0 FIFC	Ú, which will write the data data for a particular clause	Unit to complete; it passes the registe to the RF as it is received to the register files, it increments a co indicates that the ALU state machine use	ounter that is	
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 Fl	control packet is passed to the next re FO)	eservation	
12	<ul> <li>pixel o</li> <li>it</li> </ul>	lata is exported in the last is sent to an output FIFO	ALU clause (clause 7) where it will be picked up	ation stations until all clauses have be by the render backend ASM7 if the output FIFO is full	en executed	
13	after the s shader pr		leted, the SEQ will free up	the GPRs so that they can be used b	y another	
23	<del>3.1.3</del> 25.	1.3_Notes			4	<b>Formatted:</b> Bullets and Numbering
14		machines and a <del>rbitrers<u>arl</u> reads or stall</del> .	<u>oiters</u> will operate ahead o	f time so that they will be able to imme	ediately start	
Exhibit	2018.docR400_S	equencer.doc 53168 Bytes*** ⓒ 🖡	ATI Confidential. Referen PM11/16/01-10:17-AM10/19/01-1	ce Copyright Notice on Cover Page সংগ্ৰহম্ম	© *** <u>09/04/15 12:54</u>	

ATI Ex. 2106 IPR2023-00922 Page 34 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	]
24 September, 2001	4 September, 201516	GEN-CXXXXX-REVA	35 of 35	

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

16. Waterfalling still needs to be specked out.

# 24.26. Open issues

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<u>bandwidth</u> 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<u>bandwidth</u> 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 parrallelparallel. 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 2018 doc R400\_Sequencer.doc 53168 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:54

AMD1044\_0256946

Formatted: Bullets and Numbering

ATI Ex. 2106 IPR2023-00922 Page 35 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	1 of 43
Author:	Laurent Lefebvre	LDocombor200126		1
Issue To:		Copy No:		
		I		
	R400 S	equencer Spe	ecification	
		SEQ		
		Version 1. <u>4</u> 3		
rei		cted uses of the block. It a	cer block (SEQ). It provides an ov Iso describes the block interfaces,	
	C:\pe	erforce\r400\doc_lib\design\t Parforce\r400\orphites\design\t APPROVALS	blocks\sq\R400_Sequencer.docC:\p E\R400_Sequencer.des Signature/Date	erforce\r400\
Remarks:				
	ANTIALLY DETRIME	NTAL TO THE INTE	NFORMATION THAT CO REST OF ATI TECHNOL SE OR DISCLOSURE.	
work created i unpublished v 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 ar provide notice that ATI owns a co publication has occurred. This w of this document may be used, re ssion of ATI Technologies Inc."	oyright in this ork contains
Exhibit 2020.docR400_	Sequencer.doc 62852 Bytes*** © AT	I Confidential. Reference ( PM12/05/01-04:58 PM11/16/01-10.17/	Copyright Notice on Cover Page @	) *** <u>09/04/15 12:50</u>

ATI 2020 LG v. ATI IPR2015-00325

AMD1044\_0256947

ATI Ex. 2106 IPR2023-00922 Page 36 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20155	·····	2 of 43

# Table Of Contents

1. OVERVIEW	C A
I.         OVERVIEW           1.1         Top Level Block Diagram	20
1.2 Data Flow graph	
1.2         Data Flow graph	
	440
	.113
3. INSTRUCTION STORE	1414
4. SEQUENCER INSTRUCTIONS	
5. CONSTANT STORES	
5.1 Memory organizations	
5.2 Management of the remaping tables	
5.3 Constant Store Indexing	
5.4	
5.5 Real Time Commands	
6. LOOPING AND BRANCHES	
6.1 The controlling state	
6.2 The Control Flow Program	
6.3 Data dependant predicate instructions2	
6.4 HW Detection of PV,PS	
6.5 Register file indexing	2118
6.6 Predicated Instruction support for Texture	
clauses	
6.7 Debugging the Shaders	2249
6.7.1 Method 1: Debugging registers2	
6.7.2 Method 2: Exporting the values in the	e
GPRs (12) 2220	
7. PIXEL KILL MASK	2320
8. MULTIPASS VERTEX SHADERS (HOS)	2320
9. REGISTER FILE ALLOCATION	
10. FETCH ARBITRATION	2421
11. ALU ARBITRATION	2421
12. HANDLING STALLS	2522
13. CONTENT OF THE RESERVATION STAT	
FIFOS2522	*******
14. THE OUTPUT FILE	2522
15. IJ FORMAT	
15.1 Interpolation of constant attributes	2623
16. THE PARAMETER CACHE	
17. VERTEX POSITION EXPORTING	
18. EXPORTING ARBITRATION	
19. EXPORT TYPES	
19.1 Vertex Shading	
19.2 Pixel Shading	
20. SPECIAL INTERPOLATION MODES	
20.1 Real time commands	
20.2 Sprites/ XY screen coordinates/ FB	es ( Aur - Y
information	2825
20.3 Auto generated counters	
20.3.1 Vertex shaders	
20.3.2 Pixel shaders	
	2020
21 STATE MANAGEMENT	2020
21. STATE MANAGEMENT	
22. XY ADDRESS IMPORTS	2926
22.         XY ADDRESS IMPORTS           22.1         Vertex indexes imports	<b>2926</b> 2926
22. XY ADDRESS IMPORTS	2926 2926 2926

23.2 Conte	ext					
24. DEBUG	REGISTERS					
24.1 Conte	ext					
25. INTERF	ACES					
25.1 Extern	nal Interfaces					
25.1.1	SC to SP : IJ bus					
25.1.2	SC to SQ : IJ Control bus					
25.1.3	SQ to SC: Vertex/Pixel shader					
synchronizat	tion bus Error! Bookmark not					
defined.29						
25.1.4	SQ to SP: Interpolator bus					
25.1.5	SQ to SP: GPR Input Mux select .3329					
25.1.6	SQ to SPx: Parameter cache write					
control	3330					
25.1.7	SQ to SP: Parameter Cache Read					
control bus	3330					
25.1.8	SQ to SX: Parameter Cache Mux					
control Bus	3330					
25.1.9	SP to SX: Parameter data					
25.1.10	SX to Interpolators: Parameter Cache					
Return bus	3430					
25.1.11	SQ to SP0: Staging Register Data					
	3431					
25.1.12	PA to SQ : Vertex interface					
25.1.13	SQ to CP: State report					
25.1.14	SP to SX : Pixel/Vertex write to SX					
	3734					
25.1.15	SQ to SX: Control bus					
25.1.16	SX to SQ : Output file control3835					
25.1.17	Shader Engine to Fetch Unit Bus.3835					
25.1.18	Sequencer to Fetch Unit bus					
25.1.19	Sequencer to SP: GPR control3936					
25.1.20	Sequencer to SPx: Instructions3936					
25.1.21	SP to Sequencer: Constant address					
load	4037					
25.1.22	Sequencer to SPx: constant broadcast					
	4037					
25.1.23	SP0 to Sequencer: Kill vector load					
	4037					
25.1.24	SQ to CP: RBBM bus4037					
25.1.25	CP to SQ: RBBM bus4037					
	LES OF PROGRAM EXECUTIONS					
4137						
26.1.1	Sequencer Control of a Vector of					
Vertices	4137					
26.1.2	Sequencer Control of a Vector of					
Pixels	4238					
26.1.3	Notes					
	SSUES4340					
	IEW					
1.1 Top Lev	el Block Diagram					
1.2 Data Flow graph8						
Lis Control	Graph					
2. INTERPOLATED DATA BUS						

Exhibit 2020.doc/R400\_Sequencer.doc 62852 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*
<u>09/04/15.12:56</u>
<u>PM12/05/01-04:59-PM11/16/01-10.17-AM</u>

AMD1044\_0256948

ATI Ex. 2106 IPR2023-00922 Page 37 of 223

	ORIGINATE DATE	EDIT DATE		
	24 September, 2001	4 September, 20	155	
		December 200-	126	
	UCTION STORE			.14
	ENCER INSTRUCTIONS			.15
	FANT STORES			.16
	NG AND BRANCHES			.17
6.1 Ine-co	ntrolling state		24.1	.18
	ntrol Flow Program		load	10
	ependant predicate instruction		24.1	.19
6.4 HW De	tection of PV,PS		2.1.1	20
	er file indexing			.20
	ted Instruction support for T		24.1	.21
			25.	
	ging the Shaders			
0.7.1	Method 1: Debugging registe Method 2: Exporting the value			.1 ices
6.7.2 GPRs (12)		<del>162 III (116</del>		.2
	+9 KILL MASK	20		
	URFACES			.3
	TER FILE ALLOCATION		26.	
	ARBITRATION		1.	
	RBITRATION		1.1	
	ING STALLS		1.2	
	INT OF THE RESERVATIO		1.2	
FIFOS22		TE STATISTIC	2	
	UTPUT FILE	22	3	
	MAT		4	
	ARAMETER CACHE		5	
	X POSITION EXPORTING		6	
	TING ARBITRATION		6.1	
	AL INTERPOLATION MOD		6.2	The (
	time commands		6.3	Data
19.2 Sprit	es/ XY screen coordinates/	FB	6.4	HW C
			6.5	Regis
20. STATE	MANAGEMENT		6.6	Predi
21. XY AD	DRESS IMPORTS		clause	s
	TERS		6.7	Debu
	rol			
	ext			<u>}</u>
	G-REGISTERS			<del>Rs (12</del>
	ext			)
	FACES			ord.
	nal Interfaces		7.	
	SC to SP : IJ bus		8	
	SC to SEQ : IJ Control bu		9,	
	SQ to SP: Interpolator but		10	
	Sequencer to SPx: Param	leter-cache	11. 12.	
write contro		ha Dand	12.	
	SQ to SP: Parameter Cac	ne rteac	FIFOS	
control bus		ho Mur	14.	
control-Bus		He-Mux-	14. 15.	
		20		
24.1.9		ator Cacho	16. 17.	
Return bus	20	010000000	18.	
	VGT to SP0/SQ : Vertex E	3us 29	10.	
	SEQ to CP: State report		20.	
	SP to SX : Pixel/Vertex wi		20.1	
	-SQ to SX: Control bus		20.2	
	-SX to SQ : Output file con		21.	
an esderden.			<i></i>	min tout 4

SP0 to Sequencer: Kill vector load....36 CP to SQ: RBBM bus ......36 MPLES OF PROGRAM EXECUTIONS...36 Sequencer Control of a Vector of -36 Sequencer Control of a Vector of 37 38 Notes RVIEW......4 Level Block Diagram......6 Flow graph..... .....8 Irol-Graph ......9 UENCER INSTRUCTIONS ......12 ISTANT STORE 12 PING AND BRANCHES ......12 dependant predicate instructions ......15 Detection of PV,PS ......15 ister file indexing.....16 licated Instruction support for Texture ugging the Shaders .....16 Method 1: Debugging registers......16 Method 2: Exporting the values in the 2) 17 Method 3: Selective export of a 32 bit 17 ARBITRATION......18 IDLING STALLS ......19 ITENT OF THE RESERVATION STATION OUTPUT FILE ......19 ORMAT......19 əntrol..... 21 

DOCUMENT-REV. NUM.

GEN-CXXXXX-REVA

36

36

Shader Engine to Fetch Unit Bus ..... 34 Sequencer to Fetch Unit bus......34 Sequencer to SP: GPR control......35 Sequencer to SPx: Instructions......35 SP to Sequencer: Constant address

Sequencer to SPx: constant broadcast

PAGE

3 of 43

Exhibit 2020 doc R400\_Sequencer.doc 62852 Bytes\*\*\* 
CATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\* 
090/4/15 12:68 PM12/05/01-04:59 PM11/16/01-10:17 AV

AMD1044\_0256949

ATI Ex. 2106 IPR2023-00922 Page 38 of 223

		ORIGINATE DATE	EDIT DATE	R400 S	equencer Specification	PAGE
	2	24 September, 2001	4 September, 20155			4 of 43
_	21.1 Control			22111	MH to SEQ: Instruction s	tore Load 24
	21.2 Context		22	22112	-SPO to SX0 : Pixel read f	
	22. INTERFAC	***************************************	~~	22.1.12	SEQ to SX0 Control bus	
			····· ha ha	AND AND J. F X. C.		
	22.1 External	Interfaces		22.1.14	SX0 to SEQ : Output file	control24
	22.1.1 P/	A/SC to SP0 : IJ bus		22.1.15	SP0 to SX0 : Position ret	urn bus25
	22.1.2 P/	A/SC to SEQ : IJ Contr	ol-bus22	22.1.16	Shader Engine to Fetch U	Jnit Bus (Fast
	22.1.3 SI	EQ to SP0 : Interpolato	r bus23	Bus)		
	22.1.4 SI	EQ to SP0 : Parameter	Cache bus 23	22.1.17	-Sequencer to Fetch Unit-	bus (Slow
	22.1.5 SI	EQ to SX0 : Parameter	Cache Mux	Bus)		
	control Bus 23	3	2	3. EXAMP	LES OF PROGRAM EXE	CUTIONS25
	22.1.6 S	X0 to SP0 : Parameter	Cache Return	23.1.1	Sequencer Control of a V	ector of
	bus 23	3		Vertices		
	22.1.7 V(	GT to SP0/SEQ : Verte	x Bus23	23.1.2	-Sequencer Control of a V	ector of
	22.1.8 Cl	P to SEQ : Constant st	ore-load24	Pixels	-27	
	22.1.9 Cl	P to SEQ : Fetch State	store load .24	23.1.3	Notes	
	22.1.10 Cl	P to SEQ : Control Stat	te-store-load 2	4. OPEN I	SSUES	
	24	4				

Exhibit 2020. doc R400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*
<u>09/04/15.12:56</u>
<u>PM12/05/01-04:59-PM114/6/01-10:17-AM</u>

AMD1044\_0256950

ATI Ex. 2106 IPR2023-00922 Page 39 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	5 of 43
<b>—</b> · ·				
Revision	Changes:			
Rev 0.1 (Lau Date: May 7,	<b>rent Lefebvre)</b> 2001	First dra	ft.	
Rev 0.2 (Laur Date : July 9, Rev 0.3 (Laur Date : August Rev 0.4 (Laur Date : August Rev 0.5 (Laur Date : Septen Rev 0.6 (Laur Date : Septen Rev 0.7 (Laur Date : Octobe Rev 0.8 (Laur Date : Octobe Rev 0.8 (Laur Date : Octobe Rev 0.9 (Laur Date : Octobe Rev 1.0 (Laur Date : Octobe Rev 1.1 (Laur Date : Octobe Rev 1.1 (Laur Date : Octobe Rev 1.2 (Laur Date : Novem Rev 1.3 (Laur	rent Lefebvre) 2001 rent Lefebvre) t 6, 2001 rent Lefebvre) t 24, 2001 rent Lefebvre) nber 7, 2001 rent Lefebvre) mber 24, 2001 rent Lefebvre) er 5, 2001 rent Lefebvre) er 8, 2001 rent Lefebvre) er 17, 2001 rent Lefebvre) er 19, 2001 rent Lefebvre) er 26, 2001 rent Lefebvre) ber 16, 2001 rent Lefebvre) ber 16, 2001 rent Lefebvre) ber 26, 2001 rent Lefebvre) ber 26, 2001	SP. Add Reviewe August Added t file and flow of p Added ti Change architec Added store m data dep Change flexible. Incorpor flow me the con registers Refined Added precision Methods Interface Added ti	the dynamic allocation method for r an example (written in part by Vic) pixels/vertices in the sequencer. iming diagrams (Vic) d the spec to reflect the new ture. Added interfaces. constant store management, insi anagement, control flow management pendant predication. d the control flow method to be Also updated the external interfaces. rated changes made in the 10/18/01 reting. Added a NOP instruction, re nditional_execute_or_jump. Added	tion. ting on register of the R400 truction ent and control moved debug rs. delta Debug cc.

Exhibit 2020 doc R400\_Sequences.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:55

AMD1044\_0256951

ATI Ex. 2106 IPR2023-00922 Page 40 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20155		6 of 43

### 1. Overview

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

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

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

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

Exhibit 2020 doc R400\_Sequences.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:56

AMD1044\_0256952

ATI Ex. 2106 IPR2023-00922 Page 41 of 223

# PROTECTIVE ORDER MATERIAL

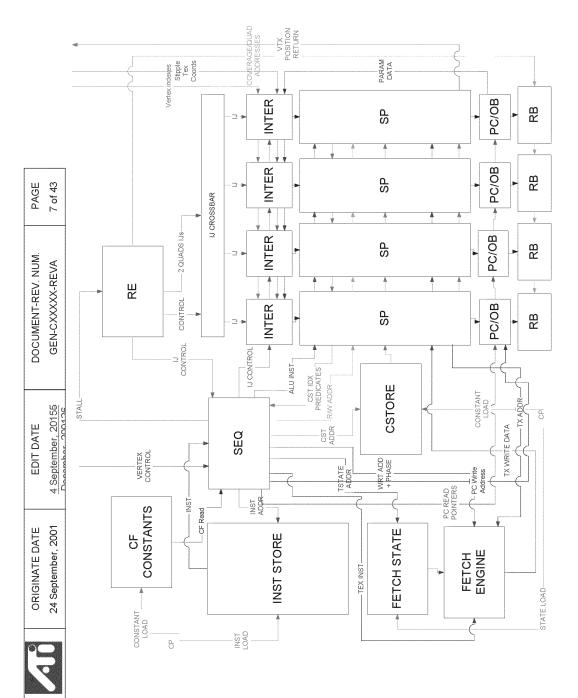
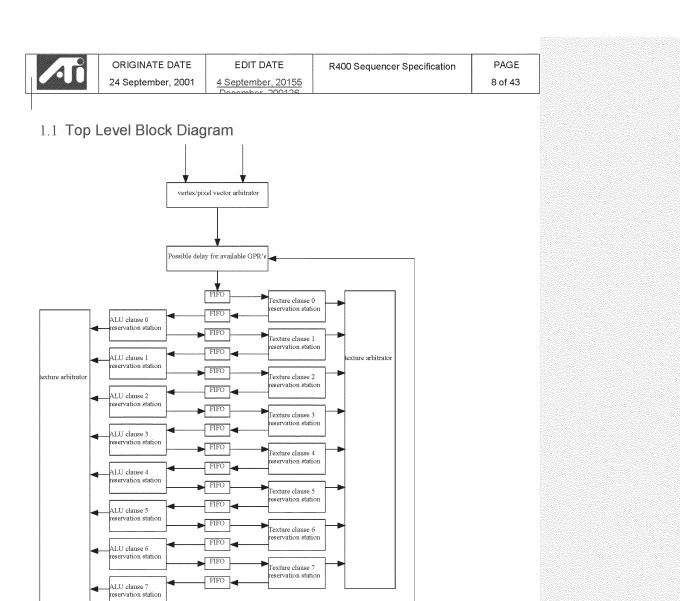


Exhibit 2020 doc R400\_Sequences doc 62852 Bytes 444 
Confidential. Reference Copyright Notice on Cover Page a 444 00004115 1256 PM120501 0458 PM1116011-10177 AM

AMD1044\_0256953

ATI Ex. 2106 IPR2023-00922 Page 42 of 223



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

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

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

Exhibit 2020 docR400\_Sequencer.doc 62852 Bytes\*\*\* 
CATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\* 09/04/15 12:58
PM12/05/01-04:58 PM11/16/01-10-17-AM

AMD1044\_0256954

ATI Ex. 2106 IPR2023-00922 Page 43 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	9 of 43	

screen position if needed) are sent to the interpolator buffers which are going to use these barycentric coordinates to interpolate the parameters and place the interpolated values into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

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

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

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

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

{ISSUE: How do we handle parameter cache pointers (computed, semi-computed or not computed)?}

A special case is for multipass vertex shaders which shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

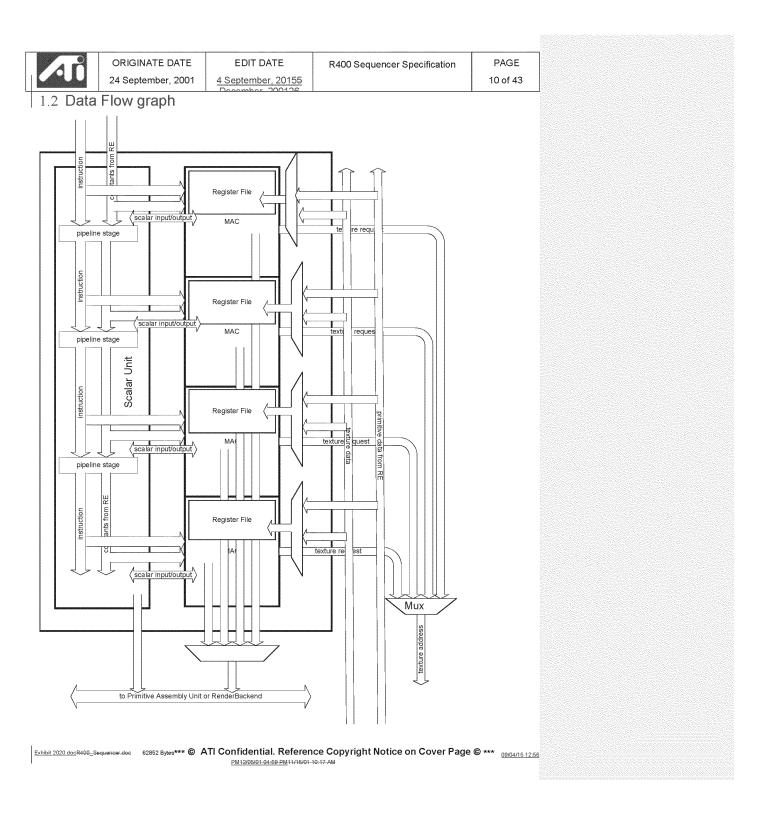
All other level process in the same way until the packet finally reaches the last ALU machine (7). Upon completion of a vertex shader, a bit is sent to the SC to let it know that it can begin sending pixels of this group to the sequencer.

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

Exhibit 2020 docR400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:56

AMD1044\_0256955

ATI Ex. 2106 IPR2023-00922 Page 44 of 223

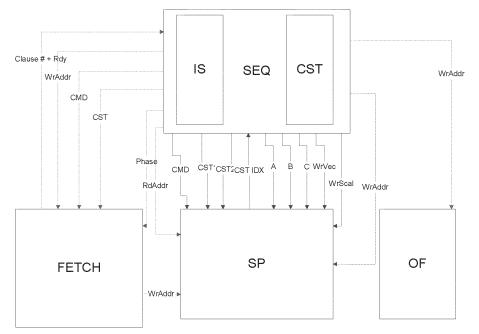


ATI Ex. 2106 IPR2023-00922 Page 45 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	11 of 43

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.

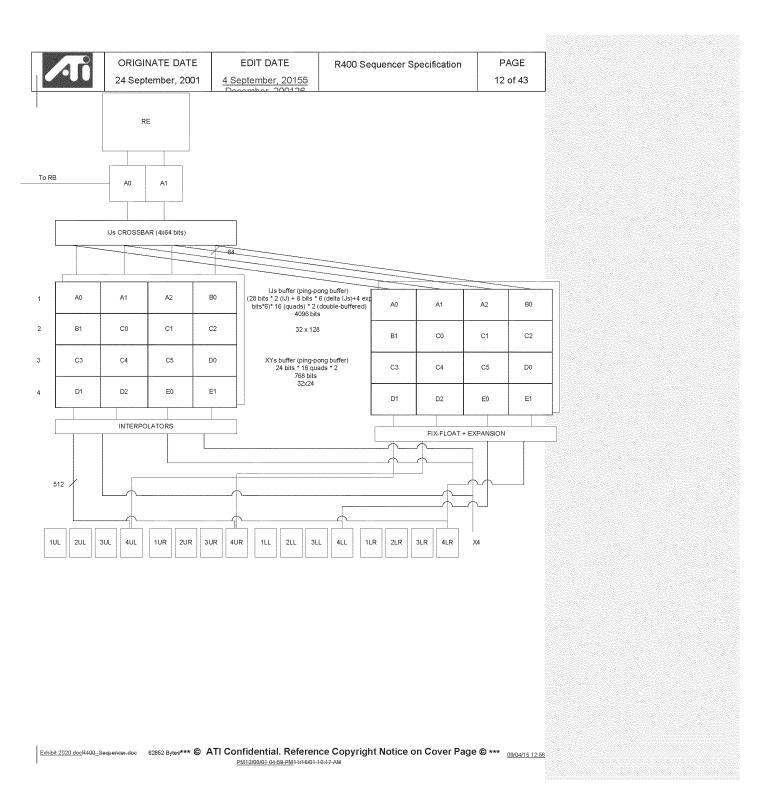
## 2. Interpolated data bus

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

Exhibit 2020 doc R400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0256957

ATI Ex. 2106 IPR2023-00922 Page 46 of 223



ATI Ex. 2106 IPR2023-00922 Page 47 of 223 PROTECTIVE ORDER MATERIAL

		[23	51 -48 -	52- 52-	59 - 5 59 <	S ရို S	
		T22	V V 32-48- 35 51	V V 36-52- 39-55	40-56- 4359	<pre>&lt; &lt; </pre> <pre>&lt; &lt; </pre> <pre>44-60-</pre> <pre>47</pre> <pre>63</pre>	X
		T21	> <sup>1</sup> <sub>0</sub> <sup>1</sup> €	33 <sup>2</sup> 0 <	24- 27	28- 31	5
		T20	>°°<	> <sup>4</sup> -7	≓ ∞ <	> 4 4 5	
		T19			ШO	Ш	
		T18				ã	
	-	T17		ပိ	ъ С	S	
PAGE 13 of 43	-	T16				BO	D D
	-	T15	5	D3			<u>с</u>
Š	-	T10 T11 T12 T13 T14 T15 T16 T17 T18 T19 T20 T21 T22 T23	S	<u>5</u>	C5		
V. NUI	-	T13	ñ				
DOCUMENT-REV. NUM. GEN-CXXXX-REVA		T12	AO	A1	A2		
UMEN N-CX	•	- 			СШ	ш	
DOC GE	-	T10				8	
	-	6 L		8	5	ß	
0155	361	T8				BO	
EDIT DATE 4 September, 20155	JUC -	1	5	5			à
EDIT	-donaca	T6	S	5	CS		
4 0	ď	T5	<u>m</u>				
ш 2	-	T4	AO	A1	A2		
E DAT er, 20(		T3	51 Å8 X	55 <sup>5</sup> 2 ≺	59 °5 56 × √	3 % ≾	
ilNATE otembu	-	12	35,32,≾	≳ ဗို ဗိ	X 4 €	XY XY 44- 60- 47 63	$\succ$
ORIGINATE DATE 24 September, 2001		Ę	5 <del>0</del> 5	33 20 X	24- 27	31 28 X	
	-	To	53 ×		≍ ∾ ≿		
13			d O	P ←	N SP	იფ	
$\mathbf{N}$			I	I	I	I	1

AMD1044\_0256959

Exhibit 2020 doo:R406\_Sequencer-doe 62852 Bytes\*\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\*\* 0600415 1256 PM1205001 0458 PM1416001 10:17.AM

ATI Ex. 2106 IPR2023-00922 Page 48 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20155		14 of 43

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

### 3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the <u>INST\_DATAINSTRUCTION\_DATA</u>, <u>INST\_INDEX\_PORT</u> <u>INSTRUCTION\_INDEX\_PORT</u> control registers. The <u>INST\_INDEX\_PORTINSTRUCTION\_INDEX\_PORT</u> is autoincremented on both reads and writes to the <u>INST\_DATAINSTRUCTION\_DATA</u> register.

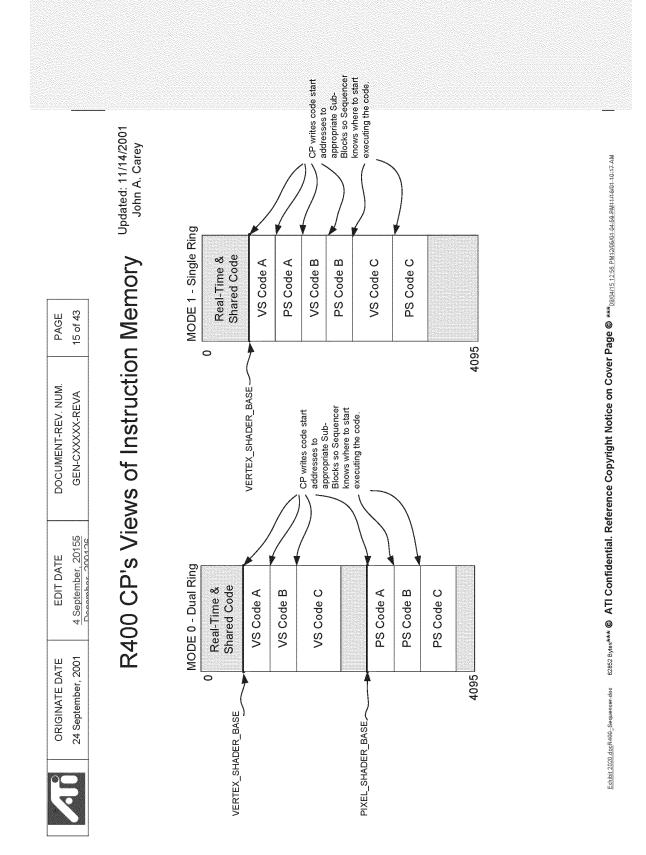
The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The MSB of the <u>INST INDEX PORT</u> INSTRUCTION\_INDEX\_PORT\_register contains the packet type for the sequencer to know where it must wrap around. The wrap around points are arbitrary and they are specified in the <u>VERTEX\_SHADERVS\_BASE</u> and <u>PIXEL\_SHADERX\_BASE</u> registers.

For the Real time commands the story is quite the same but for some small differences. The CP will use the INST\_INDEX\_PORT\_RT and INST\_DATA\_RT register pair instead of the regular ones and there are no wrap around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

Exhibit 2020.doc/R400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:56

AMD1044\_0256960

ATI Ex. 2106 IPR2023-00922 Page 49 of 223



PROTECTIVE ORDER MATERIAL

AMD1044\_0256961

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	]
24 September, 2001	4 September, 20155		16 of 43	

# 4. Sequencer Instructions

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

# 5. Constant Stores

### 5.1 Memory organizations

The sequencer is aware of where the constants are using a remaping table. A likely size for the <u>ALU</u> constant store is 1024x128 bits. The read BW from the <u>ALU</u> constant store is 128 bits/clock and the write bandwidth is <u>32 bits/clock</u> (directed by the CP bus size not by memory ports)32/4 bits/clock.

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the remaping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants.

The texture state is also kept in a similar memory. The size of this memory is 192x128. The memory thus holds 128 texture states (192 bits per state). The logical size exposed 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the remaping table to for the texture state memory is 16 lines (each line addresses 2 texture state lines in the real memory). The write granularity is 2 texture state lines (or 384 bits). The driver sends 512 bits but the CP ignores the top 128 bits. It thus takes 12 clocks to write the two texture states.

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change. Its size is 256\*32 because it must hold 8 copies of the 32 dwords of control flow constants.

The CP is loading the constant store using the CONST\_DATA and CONST\_ADDR registers. It does so by writing to the CONST\_ADDR register the logical address for the constant block it wants to update and then writes 16 times to the CONST\_DATA register. The CONST\_ADDR is auto-incremented on both reads and writes to the CONST\_DATA register.

## 5.2 Management of the remaping tables

The sequencer is responsible to manage two remaping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its remaping tables to a new one. We have 8 different remaping tables we can use concurrently. More details and a diagram to come....

## 5.15.3 Constant Store Indexing

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

 MOVA
 R1.X,R2.X
 // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X

 NOP
 // latency of the float to fixed conversion

 ADD
 R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3

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

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

Exhibit 2020.docR440\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:56

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0256962

ATI Ex. 2106 IPR2023-00922 Page 51 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
$\langle A \rangle \rangle$	24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	17 of 43	
5.4				4	
55 Real	Time Commands				
			e CONST_DATA_RT and CONST	ADDR RT	
<u>registers. It w</u>	<u>orks is the same way than</u>	when dealing with regular	constant loads BUT in this case th are not passing thru the remaping t	ne CP is not	
			nes is defined by the CONST EO		
register.		co	NST_EO_RT		
		ECTON	)		
	(Reads/Wri	tes are direct)			
	REGULA	R SECTION			
	(Reads/Writ	es are passing aping table)			
		memory. The size of this m 96 texture states (2*128 bits	emory is 192x128. Which lets us lo per state)	ad a texture	
The control-fle	ow constant memory doesn'	t sit behind a renaming table	. It is register mapped and thus the	driver must	
reload its con of control flow		te change. Its size is 192*32	because it must hold 8 copies of the	e 24 dwords	
6 1 0001	ng and Branches				
		unported and will have to be	e dealt with at the sequencer level.	We plan on	
	nstant loops and branches u				
6.1 The	controlling state.				
	ntroling state consists of: e following state is available				
Boolean[15:0] loop_count[7:					
In add					
Exhibit 2020.docR400_S	equencer.doc 62852 Bytes*** © AT	I Confidential. Reference C PM12/05/01-04:59 PM11/16/01-10:17-A	opyright Notice on Cover Page ©	) *** 09/04/15 12:56	

ATI Ex. 2106 IPR2023-00922 Page 52 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20155		18 of 43

loop\_start [7:0] [7:0] loop\_step [7:0] [7:0]

Exist to give more control to the controlling program.

We will extend that in the R400 to: Boolean[255256:0] Loop\_count[7:0][31:031:0] Loop\_Start[7:0]-[[31:031:0] Loop\_Step[7:0]-[31:31:0] Loop\_End[7:0]-[31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested callings of subroutines and 4 loop counters to allow for nested loops.

### 6.2 The Control Flow Program

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

The basic model is as follows:

The render state defined the clause boundaries:

 Vertex\_shader\_fetch[7:0][7:0]
 // eight 8 bit pointers to the location where each clauses control program is located

 Vertex\_shader\_alu[7:0][7:0]
 // eight 8 bit pointers to the location where each clauses control program is located

 Pixel\_shader\_fetch[7:0][7:0]
 // eight 8 bit pointers to the location where each clauses control program is located

 Pixel\_shader\_alu[7:0][7:0]
 // eight 8 bit pointers to the location where each clauses control program is located

 Pixel\_shader\_alu[7:0][7:0]
 // eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

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

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

Execute Conditional\_execute\_Predicates Conditional\_jump Call Return Loop\_start Loop\_end End\_of\_clause Conditional\_End\_of\_clause NOP

Execute, causes the specified number of instructions in instruction store to be executed.

Conditional\_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.

Loop\_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.

Loop\_end increments (decrements?) the loop counter and jumps back the specified number of instructions.

Call jumps to an address and pushes the IP counter on the stack. On the return instruction, the IP is popped from the stack.

Conditional\_execute\_or\_Jump executes a block of instructions or jumps to an address is the condition is not met. Conditional\_execute\_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition. End\_of\_clause marks the end of a clause.

Exhibit 2020.docR400\_Sequencer.doc 62852 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 08/04/15.12:56

AMD1044\_0256964

ATI Ex. 2106 IPR2023-00922 Page 53 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	19 of 43

Conditional End of clause marks the end of a clause if the condition is met. Conditional\_jumps jumps to an address if the condition is met. NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader\_cntl\_size (or vshader\_cntl\_size) we break the program (clause) and set the debug registers. If an execute or conditional\_execute is lower than cntl\_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0),

Formatted

		Execute		
47	46 42	41 24	23 12	11 0
Addressing	00001	RESERVED	Instruction	Exec Address
			count	

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

		NOP	
47	46 42	41 0	
Addressing	00010	RESERVED	

This is a regular NOP.

	Conditionnal_Execute						
47	46 42	<u>41</u> 41 34	<u>40 33</u>	33 <u>32</u>	32 <u>31</u> 24	23 12	11 0
Addressing	00011	RESERVED Boolean address	Boolean address	Condition	RESERVED	Instruction count	Exec Address

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

	Conditionnal_Execute_Predicates						
47	46 42	<u>41 35</u>	41 <u>34</u> 3833	37 <u>32</u>	36- <u>31</u> 24	23 12	11 0
Addressing	00100	RESERVED	Predicate vector	Condition	RESERVED	Instruction_ count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid.

		Loop_Start		
47	46 42	41 17	16 5 <u>16 12</u>	4 0 <u>11 0</u>
	00101	RESERVED	Jump	Loop-IDJump
Addressing			addressloop ID	address

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

		Loop_End		
47	46 42	41 17	16 5 <u>12</u>	4 0 <u>11 0</u>
				8

Exhibit 2020 doc R400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:56

AMD1044\_0256965

ATI Ex. 2106 IPR2023-00922 Page 54 of 223

		ORIGINATE DATE 24 September, 2001		EDIT DATE R400 Seque		encer Specification	PAGE
	CAUD			4 September, 20155			20 of 43
П		00111		RESERVED		Start	Loop IDstart
	Addressing					addressloop ID	address

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

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

		Call		100
47	46 42	4112	11 0	
	01000	RESERVED	Jump	10.000
Addressing			addressAddres	1 a status
			s	2010

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

		Return	20200
47	46 42	41 0	
	01001	RESERVED	10000
Addressing			10000

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	47 46 42 <u>4141 34</u> 40 <u>33</u> <u>3332</u> <u>32 1331</u> <u>1230 12</u> 11 0						
Addressing	01010	RESERVED Boolean address	Boolean address	Condition	<u>FW</u> onlyRESE RVED	RESERVEDFW only	Jump addressAddres s

If condition met, jumps to the address. FORWARD jump only allowed if bit 42-31 set. Bit 42-31 is only an optimization for the compiler and should NOT be exposed to the API.

	Conditional_End_of_Clause						
47	4/ 46 42   414 <del>1 3</del> 4   40 33   3332   32-31 U						
	01011	RESERVED	Boolean	Condition	RESERVED		
Addressing		Boolean	address				

This is an optimization in the case of very short shaders (where the control flow instruction can't be hidden anymore and thus are not free. In this case, if the condition is met, the clause is ended, else we continue the execution of the clause.

		End_of_Clause	1000
47	46 42	41 0	672.005
Addressing	01011	RESERVED	107455

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop\_end or the loop\_start instruction is going to break the loop and set the debug registers.

The basic model is as follows:

The render state defined the clause boundaries:

Vertex\_shader\_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located Vertex\_shader\_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located Pixel\_shader\_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located Pixel\_shader\_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located 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 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 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 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 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 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 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 Pixel\_

Exhibit 2020 doc R400\_Sequencer.doc 62852 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

PM12/05/01-04:59 PM11/16/01-10:17 AM

AMD1044\_0256966

ATI Ex. 2106 IPR2023-00922 Page 55 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
~~UU	24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	21 of 43	

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. <u>PRED\_SETNE</u> # - similar to <u>SETNE</u> except that the result is 'exported' to the sequencer. PRED\_SETGT\_# - similar to <u>SETGT</u> except that the result is 'exported' to the sequencer PRED\_SETGTE\_# - similar to <u>SETGT</u> except that the result is 'exported' to the sequencer.

For the scalar operations only we will also support the two following instructions: PRED SETE0 #-SETE0

PRED\_SETE1\_# - SETE1

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

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

### P0\_ADD\_# R0,R1,R2

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

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

### 6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of nonmasked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED\_SET and MOVA instructions and their uses.

### 6.5 Register file indexing

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

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

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

Exhibit 2020 docR400\_Sequencer.doc 62852 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:56

AMD1044\_0256967

ATI Ex. 2106 IPR2023-00922 Page 56 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20155		22 of 43

The sequencer is going to keep a loop index computed as such:

Index = Loop\_counter\*Loop\_iterator + Loop\_init.

The index is going to return 0 if it is out of the range.

### 6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector. A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. We have to make sure the invalid pixels aren't considered with this optimization.

# 6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

### 6.7.1 Method 1: Debugging registers

Current plans are to expose 2 debugging, or error notification, registers: 1. address register where the first error occurred 2. count of the number of errors

The sequencer will detect the following groups of errors:

- count overflow

- jump error

relative jump address > size of the control flow program

relative jump address > length of the shader program

- constant overflow

- register overflow

- call stack

call with stack full return with stack empty

With two of the errors, a jump error or a register overflow will cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

### 6.7.2 Method 2: Exporting the values in the GPRs (12)

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

- 1) Normal
- 2) Debug Kill
- 3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug\_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Exhibit 2020.docR400\_Sequencer.doc 62852 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:56

AMD1044\_0256968

ATI Ex. 2106 IPR2023-00922 Page 57 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	23 of 43

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, *z*, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

# 7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK\_SETE MASK\_SETNE MASK\_SETGT MASK\_SETGTE

### 8. HOS surfaces Multipass vertex shaders (HOS)

HOS<u>Multipass vertex shaders</u>-surfaces are able to export from the 6 last clauses but to memory ONLY. If they want to export to the parameter cache they have to do it in the last clause (7). They can also export position in clause 3.

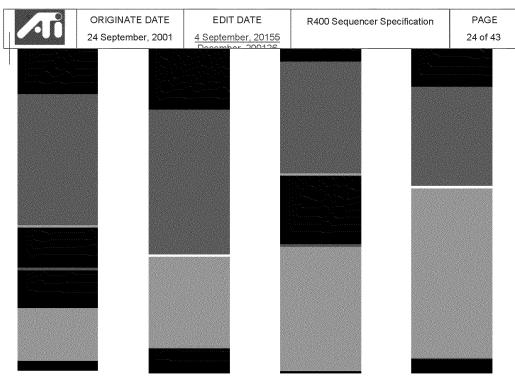
### 9. Register file allocation

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

Exhibit 2020 docR400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:56

AMD1044\_0256969

ATI Ex. 2106 IPR2023-00922 Page 58 of 223



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 2020.doc/R400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 00/04/15.12:56

AMD1044\_0256970

ATI Ex. 2106 IPR2023-00922 Page 59 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1
<b>C4U</b>	24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	25 of 43	

# 12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic to select the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS\_FULL) then the sequencer also prevents a thread to enter the exporting clause (3?). The sequencer will set the OUT\_FILE\_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

# 13. Content of the reservation station FIFOs

24.3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, quad address and 1 bit to specify if the vector is of pixels or vertices. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

# 14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

# 15. IJ Format

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

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

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

$\Delta 01I = I(1) - I(0)$		
$\Delta 01J = J(1) - J(0)$		
$\Delta 02I = I(2) - I(0)$	P0	P1
$\Delta 02J = J(2) - J(0)$		
$\Delta 03I = I(3) - I(0)$		
$\Delta 03J = J(3) - J(0)$	P2	P3
$P0 = C + I(0)^* (A - C) + J(0)^* (B - C)$		
D1 = D0 + A01I * (A = C) + A01I * (D = C)		

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

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Multiplies (Full Precision): 2 Multiplies (Reduced precision): 6

Exhibit 2020 docR400\_Sequencer.doc 62852 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09/04/15 12:56

AMD1044\_0256971

ATI Ex. 2106 IPR2023-00922 Page 60 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
Subtracts 19x24 (Parameters): 2	4 September, 20155		26 of 43	
Adds: 8				
FORMAT OF P0's IJ: Mantissa 20 E Mantissa 20 E	αp 4 for I + Sign αp 4 for J + Sign			
FORMAT of Deltas (x3):Mantissa 8 Exp Mantissa 8 Exp				
Total number of bits : 1920*2 + 8*6 + 4	<u>8 + 4*28*6 + 4*8 + 4*2</u> =	128		
The Deltas have a leading 1, the Full p to shift 8 right (exponent value of 0 mea maximum range for the IJs (Full precisi	ins number = 0, exponent	value of 1 means shift right 88). This n		
15.1 Interpolation of con	stant attributes		*-	
Because of the floating point imprecision	n, we need to take specia	provisions if all the interpolated terms	are the same	
or if two of the barycentric coordinates				
We start with the premise that if A = B a may be zero, so we extend this to:	Ind B = C and C = A, then	PU,1,2,3 = A. Since one or more of the	e ij terms	
if (A=B and B=C and C=A)				
$\frac{P0,1,2,3 = A;}{else if ((l = 0) or (J = 0)) and}$				
((J = 0) or (1-I-J = 0)) and ((1-J-I = 0) or (I = 0))) {				
$\frac{if(1 = 0) \{}{P0 = A;}$				
} else if(J != 0) { P0 = B;				
<u>} else {</u> P0 = C;				
//rest of the guad interpolated n	ormaliv			
} else	<u></u>			
Inormal interpolation				
16. The parameter cache	2004.			
The parameter cache is where the ver The reuse engine will make it so that al			nies (1R/1W).	
17. Vertex position expo	rting			
On clause 3 the vertex shader can exp clause 7 if not done at clause 3. The s the position and 64x32 memories for th The clause where the position export of the clause the position export of	torage needed to perform e sprite size. It is going to occurs is specified by the	the position export is at least 64x128 be taken in the pixel output fifo from t EXPORT_LATE register. If turned on,	memories for he SX blocks.	
the export is going to occur at ALU clau	se / ir unset position expo	n occurs at clause 3.		
18. Exporting Arbitration				
Here are the rules for co-issuing export	ing ALU clauses.			
Exhibit 2020.docR400_Sequencer.doc 62852 Bytes*** O	ATI Confidential. Referen PM12/05/01-04:59 PM11/16/01		© *** 09/04/15 12:56	

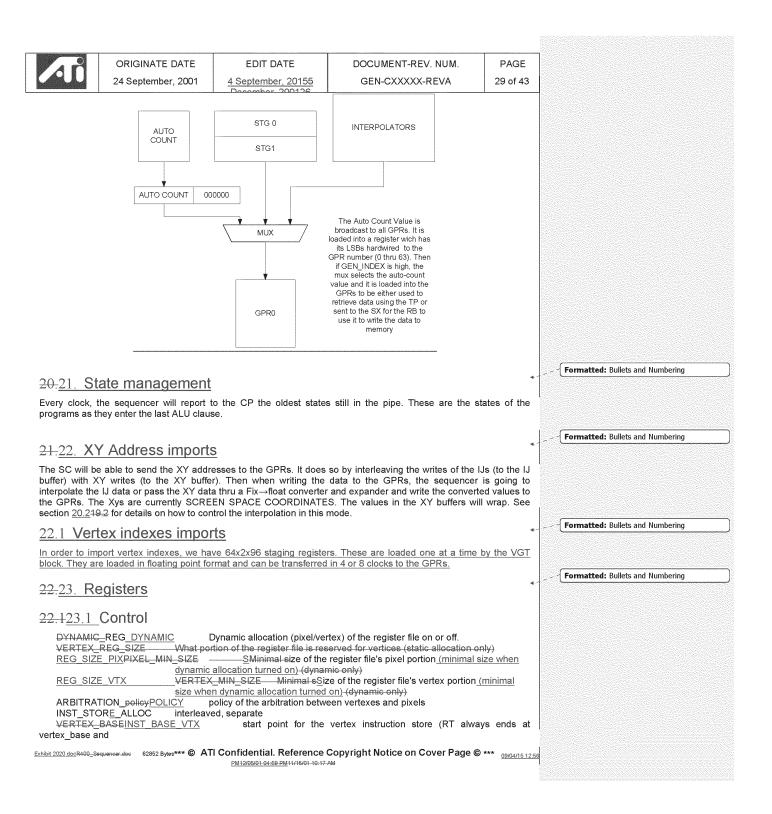
ATI Ex. 2106 IPR2023-00922 Page 61 of 223

ORIGINATE DATE EDIT DATE DOCUMENT-REV. NUM. PAGE	
24 September, 2001         4 September, 20155         GEN-CXXXXX-REVA         27 of 43	
<ol> <li>Position exports and position exports cannot be co-issued.</li> <li>Position exports and memory exports cannot be co-issued.</li> <li>Position exports and Z/Color exports cannot be co-issued.</li> <li>Memory exports and Z/Color exports cannot be co-issued.</li> <li>Memory exports and Z/Color exports cannot be co-issued.</li> <li>Memory exports and Z/color exports cannot be co-issued.</li> <li>Z/color exports and Z/color exports cannot be co-issued.</li> <li>Parameter exports and Z/color exports cannot be co-issued.</li> <li>Parameter exports and Z/color exports CAN be co-issued.</li> <li>Parameter exports and parameter exports CAN be co-issued.</li> <li>Parameter exports and memory exports CAN be co-issued.</li> </ol>	
10 Expect Types	Formatted: Bullets and Numbering
<u>19. Export Types</u> The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:	
10.1 Vertex Shading	Formatted: Bullets and Numbering
19.1 Vertex Shading         0:15       - 16 parameter cache         16:31       - Empty (Reserved?)         32:43       - 12 vertex exports to the frame buffer and index         44:47       - Empty         48:59       - 12 debug export (interpret as normal vertex export)         60       - export addressing mode         61       - Empty         62       - sprite size export that goes with position export         (point h,point w,edgeflag,misc)         63       - position	
19.2 Pixel Shading	Formatted: Bullets and Numbering
0       - Color for buffer 0 (primary)         1       - Color for buffer 1         2       - Color for buffer 2         3       - Color for buffer 3         4:7       - Empty         8       - Buffer 1 Color/Fog         10       - Buffer 2 Color/Fog         11       - Buffer 2 Color/Fog         11       - Buffer 2 Color/Fog         11       - Buffer 3 Color/Fog         11       - Buffer 3 Color/Fog         12:15       - Empty         16:31       - Empty (Reserved?)         32:43       - 12 exports for multipass pixel shaders.         44:47       - Empty         48:59       - 12 debug exports (interpret as normal pixel export)         60       - export addressing mode         61:62       - Empty         63       - Z for primary buffer (Z exported to 'alpha' component)	- Formatted: Bullets and Numbering
19.20. Special Interpolation modes	
<u>19.120.1</u> 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 <u>Exhibit 2020 doc R400_Sequencer.doc</u> <b>20</b> 52 Bytes*** <b>© ATI Confidential. Reference Copyright Notice on Cover Page ©</b> *** <u>OMAILS 1246</u>	

ATI Ex. 2106 IPR2023-00922 Page 62 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September, 2001	4 September, 20155		28 of 43	
other is rasterized with. Most overlay s memory to 16x64 or 32x64 allowing or view support for 16 vector-4 interpolani stream), then the PA/sequencer need parameters instead of 16. This mode is	nly two interpolated scalars is important (true only if we to support a realtime-spec	s per cycle, the only problem I see wit map Microsoft's high priority stream t cific mode where we need to address	h this is, if we o the realtime	
19.220.2 Sprites/ XY scre	een coordinates/	FB information	*	- Formatted: Bullets and Numbering
When working with sprites, one may of coordinates may be needed in the sha conjunction with the SND_XY register ( special operations) to the shader usin interact together:	der program. This function in SC). Also it is possible t	nality is controlled by the gen_I0 regis o send the faceness information (for C	ster (in SQ) in GL front/back	
Gen_st is a bit taken from the interface set, it means we are dealing with Point from the GEN_S and GEN_T state re- and 1.	t AA, Line AA or sprite and	I in this case the vertex values are go	ing to be-read	
Param_Gen_I0 disable, snd_xy disable Param_Gen_I0 disable, snd_xy disable Param_Gen_I0 disable, snd_xy enable Param_Gen_I0 disable, snd_xy enable Param_Gen_I0 enable, snd_xy disable Param_Gen_I0 enable, snd_xy disable Param_Gen_I0 enable, snd_xy enable, Param_Gen_I0 enable, snd_xy enable,	, gen_st – I0 = No modifica , no gen_st – I0 = No modifica , gen_st – I0 = No modifica , no gen_st – I0 = garbage , gen_st – I0 = garbage, ga no gen_st – I0 = screen x	ation fication ttion , garbage, garbage, faceness rrbage, s, t , screen y, garbage, faceness		
20.3 Auto generated cou	inters		*	Formatted: Bullets and Numbering
In the cases we are dealing with multip both use this count to write the 1 <sup>st</sup> pas				
The count is always generated in the s the shader type (pixel or vertex). This count broadcast to the registers, the Ls in the vector.	ame way but it is passed to is toggled on and off using	o the shader in a slightly different way g the GEN_INDEX register. While the	depending on re is only one	Formatted
			4	- Formatted: Bullets and Numbering
20.3.1 Vertex shaders In the case of vertex shaders, if GEN	INDEX is set the data will	he put into the x field of the third real	stor (it moons	
that the compiler must allocate 3 GPRs			ster (it means	(
20.3.2 Pixel shaders			*	- Formatted: Bullets and Numbering
In the case of pixel shaders, if GEN_IN	DEX is set, the data will be	put in the x field of the 2 <sup>nd</sup> register (I1	<u>x).</u>	Formatted
Exhibit 2020.doc.R400_Sequencer.doc 62852 Bytes*** © /	ATI Confidential. Referen PM12/05/01-04:58 PM11/16/01-	ce Copyright Notice on Cover Page	© *** <u>09/04/15 12:56</u>	

ATI Ex. 2106 IPR2023-00922 Page 63 of 223



ATI Ex. 2106 IPR2023-00922 Page 64 of 223

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 2001	4 September, 20155	· ·	30 of 43	
	Begins	at 0)	II		
	ASEINST_BASE_PIX RLEAVEONE_THREAD	start point for the	e pixel shader instruction store ister. Only allows one program at a	time into the	
	RLEAVE_ALUONE_ALU (instea		ly allows one ALU program at a time to	o be executed	
INSTRUC _PORT			ase address of the instruction writes a	nd type (auto-	
INSTRUC CONSTAN	TION_DATA This is	ented on reads/writes) where the CP puts the ac where the CP puts consta	tual data going to the instruction memo	ory	
CONSTAN			jical constant address (9 bits)		
PORT RT		where the CP puts the me (auto-incremented on	base address of the instruction writes	and type for	_ ~ Formatted
INSTRUC			tual data going to the instruction memo	ory for	
CONSTAN	NT_DATA_RT This is		int data for Real Time (32 bits)		
	NT_ADDR_RT This is	where the CP puts the log	ical constant address for Real Time (9		
CONSTAN			served for real time in the constant sto ping table operates on the rest of the n		
EXPORT	LATE Contro	ls whether or not we are	exporting position from clause 3. If		
	export	occur at clause 7.			
22.223.2	Context			<b>4</b>	
	fetch[7:0][7:0]VS FETCH	{07} eight 8	bit pointers to the location where	each clauses	
	control program	n is located			
control progra	alu[7:0][7:0]VS_ALU_{0 m is located	/} eight 8	bit pointers to the location where	each clauses	
PS_FETC	H_{07} Pshader_fetch	<del>7:0][7:0]</del> eight 8 bit poir	nters to the location where each cla	auses control	
program is loc PS_ALU		0117:01 eight 8 bit poir	nters to the location where each cla	auses control	
program is loc	ated				
	<u>₹PS_BASE</u> ₹VS_BASE		the pixel shader in the instruction store the vertex shader in the instruction stor		
Vshader_c	entl_sizeVS_CF_SIZE		shader (# of instructions in control pro		
	cntl_sizePS_CF_SIZE sizePS_SIZE		shader (# of instructions in control prog shader (cntl+instructions)	ıram/2)	
	sizeVS_SIZE		shader (cntl+instructions)		
REG_ALL	OC_PIXPS_NUM_REG		ers to allocate for pixel shader program		
	OC_VERTVS_NUM_REC	···· ·	ers to allocate for vertex shader progra bit register specifying which parameter		
CYL WR		d shaded (0 = flat, 1 = gou // WRAP64 bits: 1	uraud) or which parameters (and channels (x)	vzw)) do we	
	do the	cyl wrapping (0=linear, 1=	cylindrical).	, .,,	
<del>\_export_</del>	modePS_EXPORT_MOI 1xxxx	Multipass mode	lormal mode		
		al, bbbz where bbb is how bass 1-12 exports for colo	r many colors (0-4) and z is export z or r.	not	
	export_maskVS_EXPOR	MASK which of the last	6 ALU clauses is exporting (multipass ctor), 1: position (2 vectors), 3:multipas		
vshader_e	export_count[6]VS_EXPC		ctor), 1. position (2 vectors), 3.multipas	5	
COUNT	{06} Six 4 b		e## of interpolated parameters of COUNT_6) OR	exported in	
	# of ex	ported vectors to memory	per clause in multipass mode (per clau	use)	
Control_F		ords that contain the contr alue interpolated in the T o			
GEN_S		alue interpolated in the S of			
Exhibit 2020.docR400_Se	equencer.doc 62852 Bytes*** ©	ATI Confidential. Referer	ce Copyright Notice on Cover Page	© *** 09/04/15 12:56	
1		PM12/05/01-04:58-PM11/16/01			

ATI Ex. 2106 IPR2023-00922 Page 65 of 223

	ORIGINATE DATE	EDIT DATE	DOCU	MENT-REV. NUM.	PAGE	
200	24 September, 2001	4 September, 20		I-CXXXXX-REVA	31 of 43	
GEN_IO GEN_INDEX CONST_BAS PIX_CST_BA PIX_CST_SIZ VTX_CST_SIZ VTX_CST_SIZ VTX_CST_SIZ VTX_CST_SIZ INST_PRED CF_BOOLEA CF_LOOP_C CF_LOOP_S CF_LO	Do we ov Auto gen and R3 fd E VTXVTX_CST_BASE SECONST_BASE_PIX (8 ZECONST_SIZE_PIX (8 ZECONST_SIZE_VTX ( OPTIMIZE_Turns on always ex NS_256 boold OUNT_32x8 bit of COUNT_32x8 bit of CUG registers CUG registers CONTin B_ADDR in INST_COUNT INST_COUNT BREAK_ADDR	Decombar 2004     renvrite or not the pa erates an address fre or vertex shaders     (9 bits) Logical Base     (9 bits) Size of the log the predicate bit opt kecuted),     san bits     counters (number of log     counters (init value un counters (step value	nee I and the set of t	lata and generated T an e results into R1 for pixe r the constants of the Ve istants of the Pixel shad for pixel shaders for vertex shaders litional_execute_predica he loop) tation) utation) m occurred the execution of the pro	d S values d S values entex shader er ates is ates is	Formatted: Bullets and Numbering
	17771111111111111111111111111111111111				əs is always	
24. <u>25. Inter</u>	faces				4	Formatted: Bullets and Numbering
Vhenever an x is	ternal Interface used, it means that th it means that SQ is goin	e bus is broadcast t			e, if a bus is	
4 1 1 2 5 1 1	SC to SP : IJ bu	s			4	
This is a bus t	that sends the IJ information in the sequencer. The	tion to the IJ fifos on			ne the control	
lame	Direction	n Bits	Description			
C_SP0_data	SC→SP0	) 64	J information sent	over 2 clocks (or XY info LSBs of the interface)	sent over 1	
C_SP0_q_wr_ma	ask SC→SP0		Write Mask	cobs of the interface)		
C_SP0_dest	SC→SP0	) 1	Controls the write de	estination (XY buffer, IJ bu		
C_SP1_data	SC→SP			over 2 clocks (or XY info LSBs of the interface)	sent over 1	
C_SP1_q_wr_ma		1 1	Write Mask			
C_SP1_dest	SC→SP	1 1	Controls the write de	stination (XY buffer, IJ bu		
C_SP2_data	SC→SP2			over 2 clocks (or XY info LSBs of the interface)	sent over 1	
C_SP2_q_wr_ma		2 1	Write Mask	,		
C_SP2_dest	SC→SP2			estination (XY buffer, IJ bu		
C_SP3_data	SC→SP3			over 2 clocks (or XY info LSBs of the interface)	sent over 1	
C_SP3_q_wr_ma	ask SC→SP3		Write Mask			
SC_SP3_dest	SC→SP3			stination (XY buffer, IJ bi	uffer)	
ibit 2020.docR400_Sequen	cer.doc 62852 Bytes*** ⓒ AT	Confidential. Refe		otice on Cover Page @	•*** <u>09/04/15 12:56</u>	

ATI Ex. 2106 IPR2023-00922 Page 66 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20155		32 of 43

### 24.1.225.1.2 SC to SEQ : IJ Control bus

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. This information is sent over 2 clocks, if SENDXY is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quads that where just sent).

Name	Direction	Bits	Description
SC SQ q wr mask	SC→SQ	4	Quad Write mask left to right
SC SQ lod correct	SC→SQ	24	LOD correction per quad (6 bits per quad)
SC SQ flat vertex	SC→SQ	2	Provoking vertex for flat shading
SC_SQ_param_ptr0	SC→SQ	11	P Store pointer for vertex 0
SC_SQ_param_ptr1	SC→SQ	11	P Store pointer for vertex 1
SC_SQ_param_ptr2	SC→SQ	11	P Store pointer for vertex 2
SC_SQ_end_of_vect	SC→SQ	1	End of the vector
SC_SQ_store_dealloc	SC→SQ	1	Deallocation token for the P Store
SC_SQ_state	SC→SQ	3	State/constant pointer (6*3+3)
SC_SQ_valid_pixel	SC→SQ	16	Valid bits for all pixels
SC_SQ_null_prim	SC-→SQ	1	Null Primitive (for PC deallocation purposes)
SC_SQ_end_of_prim	SC→SQ	1	End Of the primitive
SC_SQ_fbface	SC→SQ	1	Front face = 1, back face = 0
SC_SQ_send_xy	SC→SQ	1	Sending XY information [XY information is going to be sent on the next clock]
SC_SQ_prim_type	SC→SQ	3	Real time command need to load tex cords from alternate buffer. Line AA, Point AA and Sprite reads their parameters from GEN_T and GEN_S registers. 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite
SC SQ new vector	<u>SC→SQ</u>	1	This primitive comes from a new vector of vertices. Make sure that the corresponding vertex shader has finished before starting the group of pixels.
SC_SQ_RTRn	SQ→SC	1	Stalls the PA in n clocks
SC_SQ_RTS	SC→SQ	1	SC ready to send data

# 24.1.325.1.3 SQ to SP: Interpolator bus

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive
			000 : Normal
			011 : Real Time
			100 : Line AA
			101 : Point AA
			110 : Sprite
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich parameter needs to be cylindrical wrapped
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_gen_I0	SQ→SPx	1	Generate IO or not. This tells the interpolators not to
			use the parameter cache but rather overwrite the data
			with interpolated 1 and 0. Overwrite if gen_10 is high.

Exhibit 2020 doc R400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:56

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0256978

ATI Ex. 2106 IPR2023-00922 Page 67 of 223

This interface is synchronized with the I         data are: generated index, Interpolated c         Name       Directio         SQ_SPx_data_type       SQ_SP         SQ_SPx_data_type       SQ_SP         SQ_SPx_index_count       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_proc_wr_addr       SQ_SP         SQ_SPx_pc_wr_addr       SQ_SP         SQ_SPx_pc_wr_addr       SQ_SP         SQ_SPx_pc_wr_addr       SQ_SP         SQ_SPx_pc_wr_addr       SQ_SP         SQ_SPx_pc_wr_addr       SQ_SP         SQ_SPx_pc_wr_addr       SQ_SP         SQ_SPx_ptro       SQ_SP         SQ_SPx_ptro       SQ_SP         SQ_SPx_ptr0       SQ_SP         SQ_SPx_ptr1       SQ_SP         SQ_SP1_read_ena       SQ_SP         SQ_SP2_read_ena       SQ_SP         SQ_SP3_read_ena       SQ_SP         SQ_SXx_mux0       SQ_SX         SQ_SXx_mux1       SQ_SX         SQ_SX_mux2       SQ_SX         SQ_SX_data0       SP0_S	4 Septemb       Mux selection       Interpolator bus       iata, vertex index       n       x       y       x       y       <	Bits         1           12?         1           1         1 <th>controls the input mux to the GPRs         a (coming from the staging registers         Description         00: Interpolated data         01: Staging register data         1x: Count         Index count, common for all shader p         Staging register address         0: First staging register         1: second staging register         1: second staging register         0: Mrite address         Write Enable         The output selector mux (gpr vs par         Read control bus         SX→Interpolators) are all SYNCHF         Description         Pointer of PC         Pointer of PC         Pointer of PC         Pointer of PC         Read enables for the 4 memories in 1         Read enables for the 4 memories in 1         Read enables for the 4 memories in 1</th> <th>A 33 of 43 <u>33 of 43</u> <u>33 of 43</u> <u>34 of 43</u> <u>35 of 43</u> <u>36 of 43</u> <u>37 of 43</u> <u>38 of 43</u> <u>38 of 43</u> <u>39 of 43 of 43 <u>39 of 43</u> <u>39 of 43</u> <u>39 of 43</u> <u>39 of 43 of 43 <u>39 of 43 of 43 </u> <u>39 of </u></u></u></th> <th>3</th> <th>Formatted: Bullets and Numbering Formatted Formatted Formatted Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering</th>	controls the input mux to the GPRs         a (coming from the staging registers         Description         00: Interpolated data         01: Staging register data         1x: Count         Index count, common for all shader p         Staging register address         0: First staging register         1: second staging register         1: second staging register         0: Mrite address         Write Enable         The output selector mux (gpr vs par         Read control bus         SX→Interpolators) are all SYNCHF         Description         Pointer of PC         Pointer of PC         Pointer of PC         Pointer of PC         Read enables for the 4 memories in 1         Read enables for the 4 memories in 1         Read enables for the 4 memories in 1	A 33 of 43 <u>33 of 43</u> <u>33 of 43</u> <u>34 of 43</u> <u>35 of 43</u> <u>36 of 43</u> <u>37 of 43</u> <u>38 of 43</u> <u>38 of 43</u> <u>39 of 43 of 43 <u>39 of 43</u> <u>39 of 43</u> <u>39 of 43</u> <u>39 of 43 of 43 <u>39 of 43 of 43 </u> <u>39 of </u></u></u>	3	Formatted: Bullets and Numbering Formatted Formatted Formatted Formatted: Bullets and Numbering Formatted: Bullets and Numbering Formatted: Bullets and Numbering
25.1.4       SQ to SP: GPR Input         This interface is synchronized with the I         data are: generated index, Interpolated of         SQ SPx data type,       SQ $\rightarrow$ SF         SQ SPx data type,       SQ $\rightarrow$ SF         SQ SPx index_count       SQ $\rightarrow$ SF         SQ SPx index_count       SQ $\rightarrow$ SF         SQ SPx index_count       SQ $\rightarrow$ SF         SQ SPx stage_addr       SQ $\rightarrow$ SF         SQ SPx pc wr addr       SQ $\rightarrow$ SF         SQ SPx pc wr addr       SQ $\rightarrow$ SF         SQ SPx pc we addr       SQ $\rightarrow$ SF         SQ SPx pc phase mux       SQ $\rightarrow$ SF         SQ SPx pc not see mux       SQ $\rightarrow$ SF         SQ SPx ptr0       SQ $\rightarrow$ SF         SQ SPx ptr1       SQ $\rightarrow$ SF         SQ SPx ptr1       SQ $\rightarrow$ SF         SQ SP1_read_ena       SQ $\rightarrow$ SF         SQ SP1_read_ena       SQ $\rightarrow$ SF         SQ SP2_read_ena       SQ $\rightarrow$ SF         SQ SP3_read_ena       SQ $\rightarrow$ SF         SQ SX_mux0       SQ $\rightarrow$ SF	Description       Mux selection       Interpolator bus       Interpolator	Bits         1           12?         1           1         1 <td>controls the input mux to the GPRs         a (coming from the staging registers         Description         00: Interpolated data         01: Staging register data         1x: Count         Index count, common for all shader p         Staging register address         0: First staging register         1: second staging register         1: second staging register         0: Mrite address         Write Enable         The output selector mux (gpr vs par         Read control bus         SX→Interpolators) are all SYNCHF         Description         Pointer of PC         Pointer of PC         Pointer of PC         Pointer of PC         Read enables for the 4 memories in 1         Read enables for the 4 memories in 1         Read enables for the 4 memories in 1</td> <td>ameter cache) ONIZED together. he SP0 he SP1 he SP2</td> <td>•</td> <td>Formatted Formatted Formatted Formatted Formatted: Bullets and Numbering</td>	controls the input mux to the GPRs         a (coming from the staging registers         Description         00: Interpolated data         01: Staging register data         1x: Count         Index count, common for all shader p         Staging register address         0: First staging register         1: second staging register         1: second staging register         0: Mrite address         Write Enable         The output selector mux (gpr vs par         Read control bus         SX→Interpolators) are all SYNCHF         Description         Pointer of PC         Pointer of PC         Pointer of PC         Pointer of PC         Read enables for the 4 memories in 1         Read enables for the 4 memories in 1         Read enables for the 4 memories in 1	ameter cache) ONIZED together. he SP0 he SP1 he SP2	•	Formatted Formatted Formatted Formatted Formatted: Bullets and Numbering
SQ_SPx_data_type_       SQ_SPx_spressed         SQ_SPx_index_count       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_pc_wr_addr       SQ_SQ_SP         SQ_SPx_pc_wr_addr       SQ_SQ_SP         SQ_SPx_pc_wr_addr       SQ_SQ_SP         SQ_SPx_pc_phase_mux       SQ_SP         24.1.425.1.6       SQ to SP. Para         The four following interfaces (SQ_SP, S         Name       Directio         SQ_SPx_ptr0       SQ_SP         SQ_SPx_ptr1       SQ_SP         SQ_SP_ptr2       SQ_SP         SQ_SP1_read_ena       SQ_SP         SQ_SP2_read_ena       SQ_SP         SQ_SP3_read_ena       SQ_SP         SQ_SX_mux1       SQ_SX         SQ_SX_mux2       SQ_SX         SQ_SX_mux2       SQ_SX         24.1.625.1.8       SP to SX: Para         Name       Directio         SPO_SX0_data0       SP0_S	$\begin{array}{c c c c c c c c c c c c c c c c c c c $	Bits         12?           1         1           1         1           0         0           8         1           0         0           1         1           1         1           0         0           9         9           9         9           4         4           4         4	a (coming from the staging registers  Description 00: Interpolated data 01: Staging register data 1x: Count, Index count, common for all shader p Staging register address 0: First staging register 1: second staging register  Control  Description Write address Write Enable The output selector mux (gpr vs par Read control bus SX→Interpolators) are all SYNCHF  Description Pointer of PC Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1	). ipes ameter cache) ONIZED together. he SP0 he SP1 he SP2		Formatted Formatted Formatted Formatted Formatted: Bullets and Numbering
This interface is synchronized with the I         Iata are: generated index, Interpolated c         Name       Directio         SQ_SPx_data_type,       SQ_SP         SQ_SPx_index_count       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_pr_cwr_addr       SQ_SP         SQ_SPx_pc_wr_addr       SQ_SP         SQ_SPx_ptr1       SQ_SP         SQ_SP1_read_ena       SQ_SP <td><math display="block">\begin{array}{c c c c c c c c c c c c c c c c c c c </math></td> <td>Bits         12?           1         1           1         1           0         0           8         1           0         0           1         1           1         1           0         0           9         9           9         9           4         4           4         4</td> <td>a (coming from the staging registers  Description 00: Interpolated data 01: Staging register data 1x: Count, Index count, common for all shader p Staging register address 0: First staging register 1: second staging register  Control  Description Write address Write Enable The output selector mux (gpr vs par Read control bus SX→Interpolators) are all SYNCHF  Description Pointer of PC Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories in 1</td> <td>). ipes ameter cache) ONIZED together. he SP0 he SP1 he SP2</td> <td></td> <td>Formatted Formatted Formatted Formatted: Bullets and Numbering</td>	$\begin{array}{c c c c c c c c c c c c c c c c c c c $	Bits         12?           1         1           1         1           0         0           8         1           0         0           1         1           1         1           0         0           9         9           9         9           4         4           4         4	a (coming from the staging registers  Description 00: Interpolated data 01: Staging register data 1x: Count, Index count, common for all shader p Staging register address 0: First staging register 1: second staging register  Control  Description Write address Write Enable The output selector mux (gpr vs par Read control bus SX→Interpolators) are all SYNCHF  Description Pointer of PC Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1	). ipes ameter cache) ONIZED together. he SP0 he SP1 he SP2		Formatted Formatted Formatted Formatted: Bullets and Numbering
data are: generated index, Interpolated c         Vame       Directio         SQ_SPx_data_type,       SQ_SP         SQ_SPx_index_count       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_pc_wr_addr       SQ_SQ_SP         SQ_SPx_pc_wr_addr       SQ_SQ_SP         SQ_SPx_pc_wr_addr       SQ_SQ_SP         SQ_SPx_pc_phase_mux       SQ_SQ_SP         24.1.425.1.6       SQ to SP: Para         SQ_SPx_ptr0       SQ_SP         SQ_SPx_ptr1       SQ_SP         SQ_SP_pread_ena       SQ_SP         SQ_SP1_read_ena       SQ_SP         SQ_SP1_read_ena       SQ_SP         SQ_SP3_read_ena       SQ_SP         SQ_SX_mux0       SQ_SX         SQ_SXx_mux1       SQ_SX         SQ_SX_mux2       SQ_SX         SQ_SX0_data0       SP0_SX	n     n       x     n       x     n       x     n       x     n       x     n       x     n       x     n       x     n       Px     n       Px     n       x     n       x     n       x     n       x     n       x     n       x     n       x     n       x     n       x     n       x     n       1     n       3     n	Bits         2           12?         1           1         1           Bits         7           1         1           SX and         8           9         9           9         9           4         4           4         4	a (coming from the staging registers  Description 00: Interpolated data 01: Staging register data 1x: Count, Index count, common for all shader p Staging register address 0: First staging register 1: second staging register  Control  Description Write address Write Enable The output selector mux (gpr vs par Read control bus SX→Interpolators) are all SYNCHF  Description Pointer of PC Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1	). ipes ameter cache) ONIZED together. he SP0 he SP1 he SP2		Formatted Formatted Formatted Formatted: Bullets and Numbering
NameDirectioSQ_SPx_data_type_SQ_SPSQ_SPx_data_type_SQ_SPSQ_SPx_index_countSQ_SPSQ_SPx_stage_addrSQ_SPSQ_SPx_stage_addrSQ_SPSQ_SPx_pc_wr_addrSQ_SSSQ_SPx_pc_wr_addrSQ_SSSQ_SPx_pc_wr_addrSQ_SSSQ_SPx_pc_wr_addrSQ_SSSQ_SPx_pc_phase_muxSQ_SS24.1.425.1.6SQ to SP: ParaChe four following interfaces (SQ_SP, SSQ_SPx_ptr0SQ_SP, SSQ_SPx_ptr1SQ_SP, SSQ_SP_read_enaSQ_SFSQ_SP2_read_enaSQ_SFSQ_SP3_read_enaSQ_SF24.1.525.1.7SQ to SX: ParaSQ_SXx_mux1SQ_SXSQ_SXx_mux2SQ_SX24.1.625.1.8SP to SX: ParaSQ_SX_mux2SQ_SXSQ_SX_mux1SQ_SXSQ_SX_mux2SQ_SXSQ_SX_data0SP0_SSP0_SX0_data1SP0_SX	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	Bits           2           12?           1 <td>Description         00: Interpolated data         01: Staging register data         1x: Count         Index count, common for all shader p         Staging register address         0: First staging register         1: second staging register         1: second staging register         0: Description         Write address         Write Enable         The output selector mux (gpr vs pair)         Read control bus         SX→Interpolators) are all SYNCHF         Pointer of PC         Pointer of PC         Pointer of PC         Pointer of PC         Read enables for the 4 memories in 1         Read enables for the 4 memories in 1</td> <td>ameter cache) ONIZED together.</td> <td></td> <td>Formatted Formatted Formatted Formatted: Bullets and Numbering</td>	Description         00: Interpolated data         01: Staging register data         1x: Count         Index count, common for all shader p         Staging register address         0: First staging register         1: second staging register         1: second staging register         0: Description         Write address         Write Enable         The output selector mux (gpr vs pair)         Read control bus         SX→Interpolators) are all SYNCHF         Pointer of PC         Pointer of PC         Pointer of PC         Pointer of PC         Read enables for the 4 memories in 1	ameter cache) ONIZED together.		Formatted Formatted Formatted Formatted: Bullets and Numbering
SQ_SPx_data_type_       SQ_SPx         SQ_SPx_index_count       SQ_SP         SQ_SPx_index_count       SQ_SP         SQ_SPx_index_count       SQ_SP         SQ_SPx_stage_addr       SQ_SP         SQ_SPx_index_count       SQ_SP         SQ_SPx_index_count       SQ_SP         SQ_SPx_index_count       SQ_SP         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_wr_addr       SQ_SQ_SP         SQ_SPx_pc_pr_addr       SQ_SQ_SP         24.1.425.1.6       SQ to SP. Para         SQ_SPx_ptr0       SQ_SP, S         Name       Directio         SQ_SPx_ptr1       SQ_SPS         SQ_SP_ptr2       SQ_SP         SQ_SP1_read_ena       SQ_SP         SQ_SP3_read_ena       SQ_SP         SQ_SX_mux1       SQ_SX         SQ_SX_mux2       SQ_SX         SQ_SX_mux2       SQ_SX         SQ_SX_data0       SP0_SX         SP0_SX0_data1       SP0_SX	x x x x x x x x x x	Image: 2         Image: 2           12?         1           1         1           Bits         7           1         1           0         Che F           SX and         9           9         9           9         9           4         4           4         4	00: Interpolated data 01: Staging register data 1x: Count Index count, common for all shader p Staging register address 0: First staging register 1: second staging register 0. The output selector mux (gpr vs part Read control bus SX→Interpolators) are all SYNCHF Description Pointer of PC Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memo	ameter cache) ONIZED together. he SP0 he SP1 he SP2		Formatted Formatted Formatted Formatted: Bullets and Numbering
SQ_SPx_index_count       SQ_SF         SQ_SPx_stage_addr       SQ_SF         SQ_SPx_stage_addr       SQ_SF         SQ_SPx_stage_addr       SQ_SF         SQ_SPx_stage_addr       SQ_SF         SQ_SPx_pc_wr_addr       SQ_SS         SQ_SPx_pc_wr_addr       SQ_SS         SQ_SPx_pc_wr_addr       SQ_SS         SQ_SPx_pc_wr_addr       SQ_SS         SQ_SPx_pc_wr_addr       SQ_SS         SQ_SPx_pc_phase_mux       SQ_SS         24.1.425.1.6       SQ to SP: Para         Che four following interfaces (SQ_SP, S         Vame       Directio         SQ_SPx_ptr0       SQ_SP         SQ_SPx_ptr1       SQ_SF         SQ_SP_pread_ena       SQ_SF         SQ_SP1_read_ena       SQ_SF         SQ_SP3_read_ena       SQ_SF         24.1.525.1.7       SQ to SX: Para         Xame       Directio         SQ_SX_mux1       SQ_SX         SQ_SX_mux2       SQ_SX         24.1.625.1.8       SP to SX: Para         Xame       Directio         SPO_SX0_data1       SPO_SX	$\begin{array}{c c} x \\ \hline x \\ x \\$	12?         1           1         1           2         1           1         1           1         1           0         0	01: Staging register data 1x: Count Index count, common for all shader p Staging register address 0: First staging register 1: second staging register 0 mtrol Description Write address Write Enable The output selector mux (gpr vs par Read control bus SX→Interpolators) are all SYNCHF Description Pointer of PC Pointer of PC Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 mem	ameter cache) ONIZED together. he SP0 he SP1 he SP2		Formatted Formatted Formatted Formatted: Bullets and Numbering
SQ_SPx_stage_addr       SQ_SP         SQ_SPx_stage_addr       SQ_SP         Name       Directi         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_phase_mux       SQ_SS         24.1.425.1.6_SQ_to_SP: Para         The four following interfaces (SQ_SP, S         Name       Directio         SQ_SPx_ptr1       SQ_SP_SP         SQ_SPx_ptr2       SQ_SP         SQ_SP1_read_ena       SQ_SP         SQ_SP3_read_ena       SQ_SP         SQ_SXx_mux0       SQ_SX         SQ_SXx_mux1       SQ_SX         SQ_SXx_mux2       SQ_SX         SQ_SX_20_data0       SP0_SX         SP0_SX0_data1       SP0_SX	$\begin{array}{c c} x \\ \hline \\$	1           Bits           7           1           1           1           1           1           1           0	1x: Count Index count, common for all shader p Staging register address 0: First staging register 1: second staging register 0. Description Write address Write Enable The output selector mux (gpr vs par Read control bus SX→Interpolators) are all SYNCHR Description Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	ameter cache) ONIZED together. he SP0 he SP1 he SP2		Formatted Formatted Formatted: Bullets and Numbering
SQ_SPx_stage_addr       SQ_SP         SQ_SPx_stage_addr       SQ_SP         Name       Directi         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_wr_addr       SQ_SQ_SS         SQ_SPx_pc_phase_mux       SQ_SS         24.1.425.1.6_SQ to SP: Para         The four following interfaces (SQ_SP, S         Name       Directio         SQ_SPx_ptr1       SQ_SP         SQ_SPx_ptr2       SQ_SF         SQ_SP1_read_ena       SQ_SP         SQ_SP3_read_ena       SQ_SP         SQ_SXx_mux0       SQ_SX         SQ_SXx_mux1       SQ_SX         SQ_SXx_mux2       SQ_SX         SQ_SX_20_data0       SP0_SX         SP0_SX0_data1       SP0_SX	$\begin{array}{c c} x \\ \hline \\$	1           Bits           7           1           1           1           1           1           1           0	Staging register address         0: First staging register         1: second staging register         1: second staging register         Ontrol         Description         Write address         Write Enable         The output selector mux (gpr vs pail         Read control bus         SX→Interpolators) are all SYNCHF         Description         Pointer of PC         Pointer of PC         Pointer of PC         Read enables for the 4 memories in 1	ameter cache) ONIZED together. he SP0 he SP1 he SP2		Formatted
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	$\begin{array}{c c} \hline pr cache with one of the second s$	Bits           7           1           1           1           0	0: First staging register 1: second staging register Description Write address Write Enable The output selector mux (gpr vs par Read control bus SX→Interpolators) are all SYNCHF Description Pointer of PC Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	ONIZED together. he SP0 he SP1 he SP2		- Formatted: Bullets and Numbering
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	on Px Px Px Px ameter Cac sq→SX,SP→S n 1 x 1 x 1 x 1 x 1 x 1 x 1 x 1 x	Bits           7           1	<u>1: second staging register</u> <u>ontrol</u> <u>Description</u> <u>Write address</u> <u>Write Enable</u> <u>The output selector mux (gpr vs par</u> <u>Read control bus</u> SX→Interpolators) are all SYNCHR <u>Description</u> Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for 1	ONIZED together. he SP0 he SP1 he SP2		
NameDirectiSQ_SPx_pc_wr_addrSQ_SQ_SSQ_SPx_pc_we_addrSQ_SQ_SSQ_SPx_pc_phase_muxSQ_S $24.1.425.1.6$ SQ to SP: ParaThe four following interfaces (SQ_SP, SNameDirectioSQ_SPx_ptr0SQ_SFSQ_SPx_ptr1SQ_SFSQ_SPx_ptr2SQ_SFSQ_SP1_read_enaSQ_SFSQ_SP3_read_enaSQ_SF24.1.525.1.7SQ to SX: ParaSQ_SXx_mux0SQ_SXSQ_SXx_mux1SQ_SXSQ_SXx_mux2SQ_SXSQ_SXx_mux2SQ_SXSQ_SXx_mux2SP to SX: ParaSQ_SX_mux2SP to SX: ParaNameDirectioSQ_SXx_mux1SQ_SXSQ_SXx_mux2SP to SX: ParaNameDirectioSP0_SX0_data0SPO_SSP0_SX0_data1SPO_S	on Px Px Px Px ameter Cac sq→SX,SP→S n 1 x 1 x 1 x 1 x 1 x 1 x 1 x 1 x	Bits           7           1	Description Write address Write Enable The output selector mux (gpr vs par Read control bus SX→Interpolators) are all SYNCHR Description Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories	ONIZED together. he SP0 he SP1 he SP2	+	
NameDirectionSQ_SPx_pc_wr_addrSQ_SQ_SSSQ_SPx_pc_we_addrSQ_SQ_SSSQ_SPx_pc_phase_muxSQ_SQ_SS24.1.425.1.6SQ to SP: ParaThe four following interfaces (SQ_SP, SNameDirectionSQ_SPx_ptr0SQ_SPSQ_SPx_ptr0SQ_SPSQ_SPx_ptr1SQ_SPSQ_SPx_ptr2SQ_SPSQ_SP_ptr2SQ_SPSQ_SP1_read_enaSQ_SPSQ_SP2_read_enaSQ_SPSQ_SP3_read_enaSQ_SPSQ_SX_mux1SQ_SXSQ_SXx_mux1SQ_SXSQ_SXx_mux2SQ_SX24.1.625.1.8SP to SX: ParaNameDirectionSQ_SX_data0SPO_SSSPO_SX0_data1SPO_SS	on Px Px Px Px ameter Cac sq→SX,SP→S n 1 x 1 x 1 x 1 x 1 x 1 x 1 x 1 x	Bits           7           1	Description Write address Write Enable The output selector mux (gpr vs par Read control bus SX→Interpolators) are all SYNCHR Description Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories	ONIZED together. he SP0 he SP1 he SP2		
SQ_SPx_pc_wr_addr       SQ_SS_SQ_SPx         SQ_SPx_pc_we_addr       SQ_SS_SQ_SS_SQ_SS_SQ_SS_SQ_SS_SQ_SS_SQ_SS_SQ_SS_SQ_SS_SS	Px         Px           Px         Px           ameter Cac         GQ→SX,SP→S           n         1           x         9           x         9           x         9           x         9           x         9           3         3	7 1 1 SX and Bits 9 9 9 9 9 4 4 4 4	Write address         Write Enable         The output selector         mux (gpr vs pail         Read control bus         SX→Interpolators) are all SYNCHF         Description         Pointer of PC         Pointer of PC         Pointer of PC         Read enables for the 4 memories in 1	ONIZED together. he SP0 he SP1 he SP2		- Formatted: Bullets and Numbering
SQ_SPx_pc_wr_addr       SQ_SS_SQ_SPx         SQ_SPx_pc_we_addr       SQ_SS_SQ_SS_SQ_SS_SQ_SS_SQ_SS_SQ_SS_SQ_SS_SQ_SS_SQ_SS_SS	Px         Px           Px         Px           ameter Cac         GQ→SX,SP→S           n         1           x         9           x         9           x         9           x         9           x         9           3         3	7 1 1 SX and Bits 9 9 9 9 9 4 4 4 4	Write address         Write Enable         The output selector         mux (gpr vs pail         Read control bus         SX→Interpolators) are all SYNCHF         Description         Pointer of PC         Pointer of PC         Pointer of PC         Read enables for the 4 memories in 1	ONIZED together. he SP0 he SP1 he SP2		- Formatted: Bullets and Numbering
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	Px ameter Cac cq→sX,SP→s n 1 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9	1           ache F           SX and           Bits           9           9           9           9           4           4           4	The output selector mux (gpr vs par Read control bus SX→Interpolators) are all SYNCHF Description Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	ONIZED together. he SP0 he SP1 he SP2	+ + - - - - - - - - - - - - - - - - - -	Formatted: Bullets and Numbering
24.1.425.1.6SQ to SP: ParaThe four following interfaces (SQ $\rightarrow$ SP, SThe four following interfaces (SQ $\rightarrow$ SP, SSQ SPx_ptr0SQ $\rightarrow$ SPSQ SPx_ptr1SQ $\rightarrow$ SPSQ SPx_ptr1SQ $\rightarrow$ SPSQ SPx_ptr2SQ $\rightarrow$ SPSQ SP1_read_enaSQ $\rightarrow$ SPSQ SP1_read_enaSQ $\rightarrow$ SPSQ SP3_read_enaSQ $\rightarrow$ SPSQ SS SP3_read_enaSQ $\rightarrow$ SPSQ SS SP3_read_enaSQ $\rightarrow$ SSSQ SXx_mux0SQ $\rightarrow$ SXSQ SXx_mux1SQ $\rightarrow$ SXSQ SXx_mux2SQ $\rightarrow$ SX24.1.625.1.8SP to SX: ParaNameDirectionSP $=$ SX0_data0SP0 $\rightarrow$ SP0 $=$ SX0_data1	ameter Cac sq→sX,sP→s n     x   ! x   ! x   ! x   ! 2     3	Che F           SX and           Bits           9           9           9           4           4           4           4	Read control bus SX→Interpolators) are all SYNCHR Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	ONIZED together. he SP0 he SP1 he SP2		- Formatted: Bullets and Numbering
The four following interfaces (SQ $\rightarrow$ SP, SNameDirectionSQ_SPx_ptr0SQ $\rightarrow$ SFSQ_SPx_ptr1SQ $\rightarrow$ SFSQ_SPx_ptr2SQ $\rightarrow$ SFSQ_SP_read_enaSQ $\rightarrow$ SFSQ_SP3_read_enaSQ $\rightarrow$ SF24.1.525.1.7SQ $\rightarrow$ SRSQ_SXx_mux0SQ $\rightarrow$ SXSQ_SXx_mux1SQ $\rightarrow$ SRSQ_SXx_mux2SQ $\rightarrow$ SR24.1.625.1.8SP to SX: ParaNameDirectionSQ_SXx_mux2SQ $\rightarrow$ SXSQ_SXx_mux2SQ $\rightarrow$ SXSP_SX0_data0SP0 $\rightarrow$ SP0_SX0_data1	$\begin{array}{c c} \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{y} \\ \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{y} \\ \mathbf{x} & \mathbf{x} \\ \mathbf{x} & $	Bits         9         9         9         9         9         9         9         4	SX→Interpolators) are all SYNCHR Description Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	he SP0 he SP1 he SP2		<b>- ormatted:</b> Bullets and Numbering
Name       Direction         SQ_SPx_ptr0       SQ_SPx         SQ_SPx_ptr1       SQ_SPx         SQ_SPx_ptr1       SQ_SPx         SQ_SPx_ptr2       SQ_SPx         SQ_SP1_read_ena       SQ_SP         SQ_SP2_read_ena       SQ_SP         SQ_SP3_read_ena       SQ_SP         SQ_SSX_mux0       SQ_SX         SQ_SXx_mux1       SQ_SX         SQ_SXx_mux2       SQ_SX         SQ_SX_2       SP to SX: Para         Name       Direction         SP0_SX0_data0       SP0_S         SP0_SX0_data1       SP0_SX	$\begin{array}{c c} \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{y} \\ \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{y} \\ \mathbf{x} & \mathbf{x} \\ \mathbf{x} & $	Bits         9         9         9         9         9         9         9         4	SX→Interpolators) are all SYNCHR Description Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	he SP0 he SP1 he SP2		
SQ_SPx_ptr0         SQ→SF           SQ_SPx_ptr1         SQ→SF           SQ_SPx_ptr2         SQ→SF           SQ_SP1_read_ena         SQ→SF           SQ_SP1_read_ena         SQ→SF           SQ_SP1_read_ena         SQ→SF           SQ_SP2_read_ena         SQ→SF           SQ_SP3_read_ena         SQ→SF           SQ_SP3_read_ena         SQ→SF           SQ_SP3_read_ena         SQ→SF           SQ_SSX_mux0         SQ→SX           SQ_SXx_mux1         SQ→SX           SQ_SXx_mux2         SQ→SX           24.1.625.1.8         SP to SX: Para           Name         Direction           SP0_SX0_data0         SP0→S           SP0_SX0_data1         SP0→S	x 9 x 9 0 4 1 4 3 4	9 9 9 4 4 4 4 4	Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	he SP1 he SP2		
SQ_SPx_ptr0         SQ→SF           SQ_SPx_ptr1         SQ→SF           SQ_SPx_ptr2         SQ→SF           SQ_SP1_read_ena         SQ→SF           SQ_SP1_read_ena         SQ→SF           SQ_SP1_read_ena         SQ→SF           SQ_SP1_read_ena         SQ→SF           SQ_SP3_read_ena         SQ→SF           SQ_SP3_read_ena         SQ→SF           SQ_SP3_read_ena         SQ→SF           SQ_SSX_mux0         SQ→SX           SQ_SXx_mux1         SQ→SX           SQ_SXx_mux2         SQ→SX           24.1.625.1.8         SP to SX: Para           Name         Directio           SP0_SX0_data0         SP0→S           SP0_SX0_data1         SP0→S	x 9 x 9 0 4 1 4 3 4	9 9 9 4 4 4 4 4	Pointer of PC Pointer of PC Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	he SP1 he SP2		
$\begin{array}{c ccccc} & \text{SQ}\_\text{SPx\_ptr2} & \text{SQ}\_\text{SPx}\_ptr2 & \text{SQ}\_\text{SP}\\ & \text{SQ}\_\text{SP0}\_read\_ena & \text{SQ}\_\text{SF}\\ & \text{SQ}\_\text{SP1}\_read\_ena & \text{SQ}\_\text{SF}\\ & \text{SQ}\_\text{SP2}\_read\_ena & \text{SQ}\_\text{SF}\\ & \text{SQ}\_\text{SP3}\_read\_ena & \text{SQ}\_\text{SF}\\ & 24.1.5\underline{2}5.1.7 & SQ \ to \ SX: \ Para\\ \hline \\ & \text{Name } & \text{Directio}\\ & \text{SQ}\_\text{SXx\_mux0} & \text{SQ}\_\text{SX}\\ & \text{SQ}\_\text{SXx\_mux1} & \text{SQ}\_\text{SX}\\ & \text{SQ}\_\text{SXx\_mux2} & \text{SQ}\_\text{SX}\\ & 24.1.6\underline{2}5.1.8 & SP \ to \ SX: \ Para\\ \hline \\ & \text{Name } & \text{Directio}\\ & \text{SP0}\_\text{SX0}\_data0 & \text{SP0}\_\text{S}\\ & \text{SP0}\_\text{SX0}\_data1 & \text{SP0}\_\text{SX}\\ \hline \end{array}$	x 9 0 4 1 4 2 4 3 4	9 4 4 4 4 4	Pointer of PC Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	he SP1 he SP2		
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	0 / · · · · · · · · · · · · · · · · · ·	4 4 4 4	Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	he SP1 he SP2		
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	1 · 2 · 3 ·	4 4 4	Read enables for the 4 memories in 1 Read enables for the 4 memories in 1 Read enables for the 4 memories in 1	he SP1 he SP2		
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	2 3	4 4	Read enables for the 4 memories in t Read enables for the 4 memories in t	he SP2		
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	3	4	Read enables for the 4 memories in t			
Name         Directio           SQ_SXx_mux0         SQ→SX           SQ_SXx_mux1         SQ→SX           SQ_SXx_mux2         SQ→SX           24.1.625.1.8         SP to SX: Para           Name         Directio           SP0_SX0_data0         SP0→S           SP0_SX0_data1         SP0→S		- 1	Aux control Rus		1888	
SQ_SXx_mux0         SQ→SX           SQ_SXx_mux1         SQ→SX           SQ_SXx_mux2         SQ→SX           24.1.625.1.8         SP to SX: Para           Name         Directio           SP0_SX0_data0         SP0→S           SP0_SX0_data1         SP0→S	imeter Cat	cne n	nux control Duo		*	Formatted: Bullets and Numbering
SQ_SXx_mux1         SQ→SX           SQ_SXx_mux2         SQ→SX           24.1.625.1.8         SP to SX: Para           Name         Direction           SP0_SX0_data0         SP0→S           SP0_SX0_data1         SP0→S	n	Bits	Description			
SQ_SXx_mux2         SQ→SX           24.1.625.1.8         SP to SX: Para           Name         Direction           SP0_SX0_data0         SP0→S           SP0_SX0_data1         SP0→S		4	Mux control for PC (4 MSbs of Pointe	r)		
$24.1.625.1.8$ SP to SX: ParaNameDirectionSP0_SX0_data0SP0 $\rightarrow$ SSP0_SX0_data1SP0 $\rightarrow$ S		4	Mux control for PC (4 MSbs of Pointe			
NameDirectioSP0_SX0_data0SP0 $\rightarrow$ SSP0_SX0_data1SP0 $\rightarrow$ S	x I	4	Mux control for PC (4 MSbs of Pointe	<u>r)</u>		[
SP0_SX0_data0         SP0→S           SP0_SX0_data1         SP0→S	meter data	ta			*	Formatted: Bullets and Numbering
SP0_SX0_data0         SP0→S           SP0_SX0_data1         SP0→S	n	Bits	Description			
SP0_SX0_data1 SP0→S		128	Parameter data 0			
SP0_SX0_data2 SP0→S	X0	128	Parameter data 1			
		128	Parameter data 2			
SP0_SX0_data3 SP0→S		128	Parameter data 3			
SP1_SX1_data0 SP1→S		128	Parameter data 0			
SP1_SX1_data1 SP1→S		128	Parameter data 1			
SP1_SX1_data2 SP1→S		128	Parameter data 2			
SP1_SX1_data3 SP1→S		128	Parameter data 3			
SP2_SX0_data0 SP2→S		128	Parameter data 0			
SP2_SX0_data1 SP2→S		128	Parameter data 1			
SP2_SX0_data2 SP2→S SP2 SX0 data3 SP2→S		128	Parameter data 2			
SP2_SX0_data3 SP2→S SP3_SX1_data0 SP3→S	l	128 128	Parameter data 3 Parameter data 0		-	
	V 1	128	Parameter data 0			
SP3_SX1_data1         SP3→S           SP3_SX1_data2         SP3→S		120	Parameter data 1		- 88	
0F5_0A1_uala2 0F3→5	X1	140	Farameter udta 2			

ATI Ex. 2106 IPR2023-00922 Page 68 of 223

	ORIGINATE	DATE	EDIT	DATE	R	400 Sequencer Specification	PAGE	
	24 September	r, 2001	4 Septemb		5		34 of 43	
SP3_SX1_data	13	SP3→		128	Paramete	er data 3		4
								- Formatted: Bullets and Numbering
<u>24.1.725.</u>	<u>1.9_</u> SX to In	nterpol	ators: Para	amete	r Cach	e Return bus		
Name	data 0	Direct		Bits	Descript			
SXx_SPx_vtx_ SXx_SPx_vtx_	data_1	SXx→ SXx→				ita to interpolate ita to interpolate		
SXx_SPx_vtx_	data_2	SXx→	SPx	128	Vertex da	ta to interpolate		
<u>25.1.10</u> S	SQ to SPO: S	Staging	g Register	<u>Data</u>			4-	
Name	· · · ·	Direct		Bits	Descript	constant		
SQ_SP0_vgt_v SQ_SP0_vgt_v		SQ→S		<u>96</u> 1		of indexes or HOS surface inform I 96 bits per vert 1: double 192 b		
SQ SP0 data		SQ→S	CONTRACTOR OF CONT	1	Data is va			
25111 F	PA to SQ : V	'ertev i	interface				4-	<b>Formatted:</b> Bullets and Numbering
25.1.11.1	Interface Sign	<u>ial Lab</u>	ble					
The area diffe	erence between	the two	methods is n	ot suffic	ient to w	arrant complicating the interfa	ce or the state	
						s that the VGT will transmit		
						ormat. The VGT can transmit e two transmission clocks. The		
bits wide.	141000 10 0401, 2						<u> </u>	
Name		Bits	Description					
PA_SQ_vgt_vs		96	Pointers of ind					
PA SQ vgt vs PA SQ vgt ei		1				ble 192 bits per vert t for the current process vector (i	for double vector	
			data, "end_of	vector" i	s set on th	e second vector)		
PA SQ vgt st	ate	<u>3</u>	Render State			ts). This signal is guaranteed to igh	be correct when	
PA SQ vgt se	end	1		GT_SQ i	s valid rec	eive (see write-up for standard F	R400 SEND/RTR	
<u>SQ PA vqt rt</u>	<u>r</u>	1	Ready to re handshaking)	ceive (s	ee write-	up for standard R400 SEN	D/RTR interface	
25.1.11.2	Interface Dia	<u>grams</u>					4	
1		_					-	
Exhibit 2020.docR400_Se	equencer.doc 62852 By	tes*** © /	ATI Confident PM12/05/01	1al. Refe	erence Co	opyright Notice on Cover Pag	e © *** 09/04/15 12:50	3

ATI Ex. 2106 IPR2023-00922 Page 69 of 223

# PROTECTIVE ORDER MATERIAL

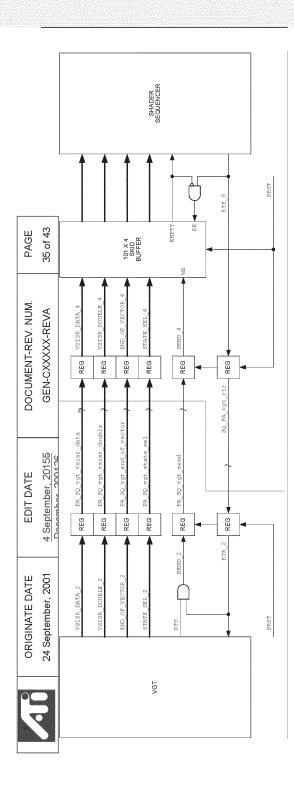
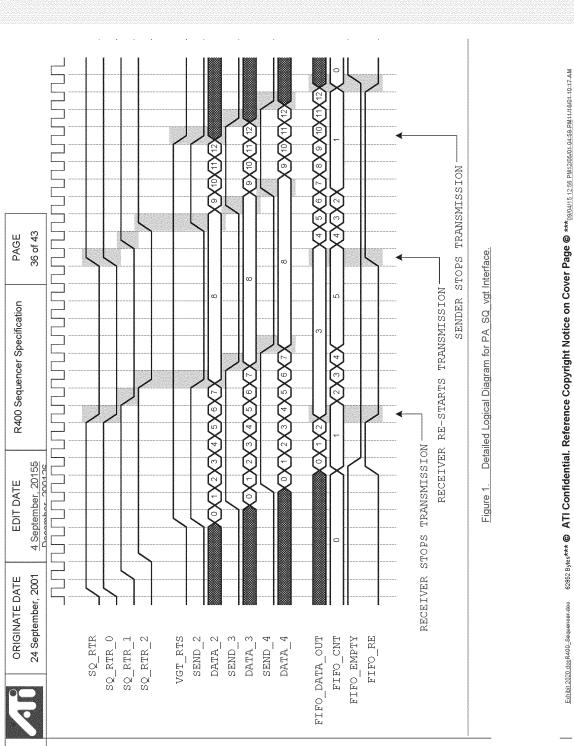


Exhibit 2020 doo?H406\_Sequences: 62852 Bytes\*\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*\* OB004/15 1256 PM1205601 0459 PM1414601-10477 AM

AMD1044\_0256981

ATI Ex. 2106 IPR2023-00922 Page 70 of 223





ATI Ex. 2106 IPR2023-00922 Page 71 of 223

	PAGE	DOCUMENT-REV. NUM.	ATE	EDIT DA	GINATE DATE	ORIGINATE
	37 of 43	GEN-CXXXXX-REVA	r, 20155	4 September	eptember, 2001	24 September
- Formatted: Bullets and Numbering	*			tex Bus	P0/SQ : Ven	24.1.8 VGT to SP0/S
						24.1.9-
				State report	SEQ to CP: 3	<del>24.1.10</del> 25.1.12 SEQ
		ption		-	Directio	Name
		vertex state still in the pipe			SEQ→C	SQ_CP_vrtx_state
		pixel state still in the pipe	Oldest	2P 3	SEQ→C	SQ_CP_pix_state
Formatted: Bullets and Numbering	*	SX	write to S	lixel/Vertex w	SP to SX : P	24.1.10 <u>25.1.13</u> SP to
		iption	ts Descr	n Bite	Directio	Name
		of 32 bits channel values	6 4 pairs	X0 256	SP0→S)	SP0_SX0_Export_data
	tinations	fies one of the of up to 12 export desi				SP0_SX0_Shader_Dest
	tinations	of 32 bits channel values fies one of the of up to 12 export desi			SP1→SX SP1→SX	SP1_SX1_Export_data SP1_SX1_Shader_Dest
		of 32 bits channel values			SP2→S	SP2 SX0 Export data
	tinations	fies one of the of up to 12 export des				SP2_SX0_Shader_Dest
		of 32 bits channel values			SP3→S>	SP3_SX1_Export_data
		ies one of the of up to 12 export desi				SP3_SX1_Shader_Dest
		set of four pixels or vectors is ex clocks. This field specifies where the nce.		X0 3	nt SP0→S)	SPx_SXx_Shader_Count
	ast export of	ted on the first shader count of the la		X0 1	SP0→S>	SPx_SXx_Shader_Last
	olor buffers).	t of pixel kill in the shader pipe, which for all pixel exports (depth and all co ecause 16 pixels are computed per c	output	X0 4	elValid SP0→S)	SP0_SX0_Shader_PixelValid
	32-bit word	ies whether to write low and/or high 64-bit export data from each of the 1	Specif	X0 2	rdValid SP0→S>	SP0_SX0_Shader_WordValid
	olor buffers).	t of pixel kill in the shader pipe, which for all pixel exports (depth and all co ecause 16 pixels are computed per c	Result output	X1 4	elValid SP1→S)	SP1_SX1_Shader_PixelValid
	32-bit word	fies whether to write low and/or high 64-bit export data from each of the 1	Specif	X1 2	rdValid SP1→S>	SP1_SX1_Shader_WordValid
	olor buffers).	s t of pixel kill in the shader pipe, which for all pixel exports (depth and all co ecause 16 pixels are computed per c	Result output	X0 4	elValid SP2→S)	SP2_SX0_Shader_PixelValid
	32-bit word	ies whether to write low and/or high 64-bit export data from each of the 1	Specif	X0 2	dValid SP2→S>	SP2_SX0_Shader_WordValid
	olor buffers).	o t of pixel kill in the shader pipe, which for all pixel exports (depth and all co ecause 16 pixels are computed per c	Resulf output	X1 4	elValid SP3→S>	SP3_SX1_Shader_PixelValid
	32-bit word	ies whether to write low and/or high 64-bit export data from each of the 1	Specif	X1 2	rdValid SP3→S>	SP3_SX1_Shader_WordValid
- Formatted: Bullets and Numbering	*-			ontrol bus	SQ to SX: C	24.1.11 <u>25.1.14</u> SQ to
	100		ts Descri	n Bit	Directio	Name

ATI Ex. 2106 IPR2023-00922 Page 72 of 223

	AÎ	ORIGINATE D		EDIT D		55	R400 Sequencer Specification	PAGE 38 of 43
		El copionibol,	2001	December				00 01 10
Τ	SQ_SXx_exp	Pixel	SQ→S	Xx	1	1: F	Pixel	
1		-				0: \	/ertex	
	SQ_SXx_exp_	start	SQ→S	Хх	1	Rai	sed to indicate that the SQ is starting a	an export
	SQ_SXx_exp_	Clause	SQ→S	Xx	3	Cla	use number, which is needed for verte	x clauses
	SQ_SXx_exp_	State	SQ→S	Xx	3	Sta	te ID, which is needed for vertex claus	es

These fields are sent synchronously with SP export data, described in SP0 $\rightarrow$ SX0 interface {ISSUE: Where are the PC pointers}

24.1.1225.1.15 SX to SQ : Output file control

Name	Direction	Bits	Description
SXx_SQ_Export_count_rdy	SXx→SQ	1	Raised by SX0 to indicate that the following two fields reflect the result of the most recent export
SXx_SQ_Export_Position	SXx→SQ	1	Specifies whether there is room for another position.
SXx_SQ_Export_Buffer	SXx→SQ	7	Specifies the space available in the output buffers. 0: buffers are full 1: 2K-bits available (32-bits for each of the 64 pixels in a clause)
			64: 128K-bits available (16 128-bit entries for each of 64 pixels) 65-127: RESERVED

#### 24.1.1325.1.16 Shader Engine to Fetch Unit Bus

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

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

Name	Direction	Bits	Description
SP0_TP0_fetch_addr	SP0->TP0	512	4 Fetch Addresses read from the Register file
TP0_SP0_data	TP0→SP0	512	4 texture results
SP1_TP1_fetch_addr	SP1->TP1	512	4 Fetch Addresses read from the Register file
TP1_SP1_data	TP1→SP1	512	4 texture results
SP2_TP2_fetch_addr	SP2->TP2	512	4 Fetch Addresses read from the Register file
TP2_SP2_data	TP2→SP2	512	4 texture results
SP3_TP3_fetch_addr	SP3->TP3	512	4 Fetch Addresses read from the Register file
TP3_SP3_data	TP3→SP3	512	4 texture results
TPx_SPx_gpr_dst	TPx→SPx	7	Write address into the gprs
TPx SPx opr cmask	TPx→SPx	4	Channel mask

#### 24.1.1425.1.17 Sequencer to Fetch Unit bus

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

Name	Direction	Bits	Description
TPx_SQ_data_rdy	$TPx \rightarrow SQ$	1	Data ready
TPx_SQ_clause_num	TPx→ SQ	3	Clause number
SQ_TPx_const	SQ→TPx	64	Fetch state sent over 4 clocks
SQ_TPx_instuct	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause

Exhibit 2020 doc R400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:56

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0256984

ATI Ex. 2106 IPR2023-00922 Page 73 of 223

	ORIGINATE	DATE	EDI	T DATE	Ξ	DOCUMENT-REV. NUM.	PAGE
	24 Septembe	r, 2001	4 Septer			GEN-CXXXXX-REVA	39 of 43
SQ TPx p	hase	SQ→TPx	(	2	Write p	hase signal	
SQ_TP0_k	od_correct	SQ→TP0	)	6	LOD co	rrect 3 bits per comp 2 components per	r quad
SQ_TP0_p	mask	SQ→TP0	)	4	Pixel m	ask 1 bit per pixel	
SQ TP1 k	od_correct	SQ→TP1		6	LOD co	rrect 3 bits per comp 2 components per	r quad
SQ_TP1_p	mask	SQ→TP1		4	Pixel m	ask 1 bit per pixel	
SQ_TP2_k	od_correct	SQ→TP2	2	6	LOD co	rrect 3 bits per comp 2 components per	r quad
SQ_TP2_p	mask	SQ→TP2	2	4	Pixel m	ask 1 bit per pixel	
SQ_TP3_k	od_correct	SQ→TP3	3	6	LOD co	rrect 3 bits per comp 2 components per	r quad
SQ_TP3_p	mask	SQ→TP3	}	4	Pixel m	ask 1 bit per pixel	
SQ_TPx_c	lause_num	SQ→TP×	(	3	Clause	number	
SQ TPX W	rite gpr index	SQ->TPx		7	Index in	nto Register file for write of returned Fet	ch Data

# 24.1.1525.1.18 Sequencer to SP: GPR control

				1
Name	Direction	Bits	Description	1
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address	
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address	
SQ_SPx_gpr_re_addr	SQ→SPx	1	Read Enable	
SQ_SPx_gpr_we_addr	SQ→SPx	1	Write Enable	
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux	]
SQ_SPx_gpr_channel_mask	SQ→SPx	4	The channel mask	
SQ_SP0_gpr_pixel_mask	SQ→SP0	4	The pixel mask	
SQ_SP1_gpr_pixel_mask	SQ→SP1	4	The pixel mask	
SQ_SP2_gpr_pixel_mask	SQ→SP2	4	The pixel mask	
SQ_SP3_gpr_pixel_mask	SQ→SP3	4	The pixel mask	1000

24.1.16 Sequencer to SPx: Parameter cache write control

24.1.1725.1.19 Sequencer to SPx: Instructions

Name	Direction	Bits	Description
SQ_SPx_instruct_start	SQ→SPx	1	Instruction start
SQ_SP_instruct	SQ→SPx	20	Instruction sent over 4 clocks
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SPx_Shader_Count	SQ→SPx	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SQ_SPx_Shader_Last	SQ→SPx	1	Asserted on the first shader count of the last export of the clause
SQ_SP0_Shader_PixelValid	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP0_Shader_WordValid	SQ→SP0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP1_Shader_PixelValid	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_Shader_WordValid	SQ→SP1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP2_Shader_PixelValid	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_Shader_WordValid	SQ→SP2	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16

Exhibit 2020 doc R400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:55

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

4

.

AMD1044\_0256985

ATI Ex. 2106 IPR2023-00922 Page 74 of 223

Name SP0_SQ_const_addr SP0_SQ_valid SP1_SQ_const_addr	lid SQ $\rightarrow$ SP3 lid SQ $\rightarrow$ SP3	2 2 Constan		nd all color mputed per high 32-bit	
SQ_SP3_Shader_WordVa 1.1.1825.1.20_SP to Name SP0_SQ_const_addr SP0_SQ_valid SP1_SQ_const_addr	lid SQ→SP3 lid SQ→SP3 D Sequencer: Direction	2	pixels or vectors         Result of pixel kill in the shader pipe, whoutput for all pixel exports (depth ar buffers). 4x4 because 16 pixels are colock         Specifies whether to write low and/or word of the 64-bit export data from each	nd all color mputed per high 32-bit	
4.1.1825.1.20 SP to Name SP0_SQ_const_addr SP0_SQ_valid SP1_SQ_const_addr	o Sequencer:		buffers). 4x4 because 16 pixels are co clock Specifies whether to write low and/or word of the 64-bit export data from eac	mputed per high 32-bit	
4.1.1825.1.20 SP to Name SP0_SQ_const_addr SP0_SQ_valid SP1_SQ_const_addr	o Sequencer:		word of the 64-bit export data from eac		
Name SP0_SQ_const_addr SP0_SQ_valid SP1_SQ_const_addr	Direction	Constan		I (2)	,
SP0_SQ_const_addr SP0_SQ_valid SP1_SQ_const_addr			t address load	4-5	Formatted: Bullets and Numbering
SP0_SQ_valid SP1_SQ_const_addr		Bits	Description		
SP1_SQ_const_addr	SP0→SQ	36	Constant address load to the sequencer		
	SP0→SQ	1	Data valid		
LAPT SELVAR	SP1→SQ SP1→SQ	36	Constant address load to the sequencer Data valid		
SP1_SQ_valid SP2 SQ const addr	SP2→SQ	36	Constant address load to the sequencer		
SP2 SQ valid	SP2→SQ	1	Data valid		
SP3_SQ_const_addr	SP3→SQ	36	Constant address load to the sequencer		
SP3_SQ_valid	SP3→SQ	1	Data valid		
4.1.1925.1.21 Sequ	iancar to SPv	· constai	at broadcast	4	Formatted: Bullets and Numbering
Name SQ_SPx_constant	Direction SQ→SPx	Bits 128	Description Constant broadcast		
					- Formatted: Bullets and Numbering
24.1.2025.1.22_SP0				*	<u> </u>
Name	Direction	Bits	Description		
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load		
SP1_SQ_kill_vect SP2_SQ_kill_vect	SP1→SQ SP2→SQ	4	Kill vector load Kill vector load		
SP2_SQ_kill_vect	SP3→SQ	4	Kill vector load		
24.1.2125.1.23 SQ to		bus			
Name	Direction				
		Bits	Description		
	SQ→CP	Bits 1	Description Read Strobe		
SQ_RBB_rs SQ_RBB_rd			Description Read Strobe Read Data		
SQ_RBB_rs	SQ→CP	1	Read Strobe		
SQ_RBB_rs SQ_RBB_rd	SQ→CP SQ→CP	1 32	Read Strobe Read Data		
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_nrtrtr	$\begin{array}{c} SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \end{array}$	1 32 1 1	Read Strobe Read Data Optional	- 	<b>Formatted:</b> Bullets and Numbering
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_ntttr SQ_RBBM_rtr 24.1.2225.1.24 CP to Name	$\begin{array}{c} SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \end{array}$	1 32 1 1	Read Strobe Read Data Optional		
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_nttr SQ_RBBM_rtr 24.1.2225.1.24_CP to	$SQ \rightarrow CP$ $SQ \rightarrow CP$ $SQ \rightarrow CP$ $SQ \rightarrow CP$ o SQ: RBBM I Direction CP \rightarrow SQ	1 32 1 1 5005	Read Strobe Read Data Optional Real-Time (Optional) Description Write Enable		
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_nttr SQ_RBBM_rtr 24.1.2225.1.24 CP to Name rbbm_we rbbm_a	$\begin{array}{c} SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ o \ SQ : RBBM \\ \hline \\ \hline \\ Direction \\ CP \rightarrow SQ \\ CP \rightarrow SQ \\ \end{array}$	1 32 1 1 0 <i>US</i> Bits 1 18	Read Strobe Read Data Optional Real-Time (Optional) Description Write Enable Address Upper Extent is TBD		Formatted: Bullets and Numbering
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_nttr SQ_RBBM_rtr 24.1.2225.1.24 CP to Name rbbm_we rbbm_a rbbm_wd	$\begin{array}{c} SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ O SQ: RBBM \\ \hline \\ Direction \\ CP \rightarrow SQ \\ CP \rightarrow SQ \\ CP \rightarrow SQ \\ \end{array}$	1 32 1 1 2005 Bits 1 18 32	Read Strobe Read Data Optional Real-Time (Optional) Description Write Enable Address Upper Extent is TBD Data		<b>Formatted:</b> Bullets and Numbering
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_nttr SQ_RBBM_rtr 24.1.2225.1.24 CP to Name rbbm_we rbbm_a rbbm_a rbbm_be	$\begin{array}{c} SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ O \ \ SQ: \ RBBM \ I \\ \hline \\ \textbf{Direction} \\ CP \rightarrow SQ \\ CP \rightarrow SQ \\ CP \rightarrow SQ \\ CP \rightarrow SQ \\ \hline \\ \end{array}$	1 32 1 1 0US Bits 1 18 32 4	Read Strobe Read Data Optional Real-Time (Optional) Use Enable Address Upper Extent is TBD Data Byte Enables		Formatted: Bullets and Numbering
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_nttr SQ_RBBM_rtr 24.1.2225.1.24 CP to Name rbbm_we rbbm_a rbbm_wd rbbm_be rbbm_re	$\begin{array}{c} SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ O \ \ SQ: \ RBBM \ I \\ \hline \\ \textbf{Direction} \\ CP \rightarrow SQ \\ \hline \\ CP \rightarrow SQ \\ \hline \\ CP \rightarrow SQ \\ \hline \end{array}$	1 32 1 1 0US Bits 1 18 32 4 1	Read Strobe Read Data Optional Real-Time (Optional) Description Write Enable Address Upper Extent is TBD Data Byte Enables Read Enable		Formatted: Bullets and Numbering
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_nttr SQ_RBBM_rtr 24.1.2225.1.24 CP to Name rbbm_we rbbm_a rbbm_wd rbbm_be rbbm_be rbbm_re rbb_rs0	$\begin{array}{c} SQ \rightarrow CP \\ O  SQ: RBBM \ I \\ \hline Direction \\ CP \rightarrow SQ \\ \hline CP \rightarrow SQ \\ \hline CP \rightarrow SQ \\ \hline CP \rightarrow SQ \end{array}$	1 32 1 1 0US Bits 1 18 32 4 1 1	Read Strobe Read Data Optional Real-Time (Optional) Urite Enable Address Upper Extent is TBD Data Byte Enables Read Enable Read Return Strobe 0		
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_nttr SQ_RBBM_nttr 24.1.2225.1.24 CP to Name rbbm_we rbbm_a rbbm_wd rbbm_be rbbm_re rbbm_rs0 rbb_rs1	$\begin{array}{c} SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ \hline \\ o SQ: RBBM \\ \hline \\ o SQ: RBBM \\ \hline \\ cP \rightarrow SQ \\ \hline \end{array}$	1 32 1 1 50US Bits 1 18 32 4 1 1 1 1	Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD         Data         Byte Enables         Read Enable         Read Return Strobe 0         Read Return Strobe 1		<b>Formatted:</b> Bullets and Numbering
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_nttr SQ_RBBM_rtr 24.1.2225.1.24 CP to Name rbbm_we rbbm_a rbbm_wd rbbm_be rbbm_re rbbm_re rbbm_rs1 rbb_rd0	$\begin{array}{c} SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ \hline \\ SQ \rightarrow CP \\ \hline \\ O SQ: RBBM \\ B \\ \hline \\ O SQ: RBBM \\ CP \\ O SQ \\ CP \rightarrow SQ \\ \hline \\ \end{array}$	1 32 1 1 2005 Bits 1 1 18 32 4 1 1 1 1 1 1 32	Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD         Data         Byte Enables         Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0		Formatted: Bullets and Numbering
SQ_RBB_rs SQ_RBB_rd SQ_RBBM_nttr SQ_RBBM_nttr 24.1.2225.1.24 CP to Name rbbm_we rbbm_a rbbm_wd rbbm_be rbbm_re rbbm_rs0 rbb_rs1	$\begin{array}{c} SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ SQ \rightarrow CP \\ \hline \\ o SQ: RBBM \\ \hline \\ o SQ: RBBM \\ \hline \\ cP \rightarrow SQ \\ \hline \end{array}$	1 32 1 1 50US Bits 1 18 32 4 1 1 1 1	Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD         Data         Byte Enables         Read Enable         Read Return Strobe 0         Read Return Strobe 1		<b>Formatted:</b> Bullets and Numbering

ATI Ex. 2106 IPR2023-00922 Page 75 of 223

Open Information         Did That Te         Did Call Te Are the Vision         Packed Te Are the Vision </th <th></th> <th></th> <th></th> <th></th> <th></th> <th></th>						
<ul> <li>25-26_Examples of program executions</li> <li>25-14-26.1.1 Sequencer Control of a Vector of Vertices</li> <li>1. PA sends a vector of 64 vertices (actually vertex indices - 32 bitsindex for 2048 bit total) to the RE's Vertex FIPO</li> <li>state pointer as well as tag into position cache is sent along with vertices</li> <li>appear on well as tag into position cache is sent along with vertices</li> <li>appear on well as tag into position cache is sent along with vertices</li> <li>appear on well as tag into position cache is sent along with vertices</li> <li>appear on well as tag into position cache is sent along with vertices</li> <li>the sector well as tag into position cache is sent along with vertices</li> <li>the sector well as tag into position cache is the loaded the global instruction store with the vertex is a sent of the RE, the CP has loaded the global instruction store with the vertex is the sector was sent as the sector was sent as the sector well and the position cache is full unless the pipe as nothing bet to do (in posities the vertex vector)</li> <li>the arbitre in close are in the position (and the parameter cache is full unless the pipe as nothing the sector to be 3P register file tor index data plus GPRs used by the program</li> <li>the arbitre in close are ent to be 4 angitter file to raide and aloue for the sector ways has priordly</li> <li>the 64 of (in the sector be 2P register file to raide the register file has been allocated</li> <li>SEQ aloues the vector to the 3P register file to raide and aloue GPRs used by the program</li> <li>the or the train sector to the 3P register file vertice (sector the raide sector) to the 3P register file vertice (sector well) the sector and sends (sector 4P register)</li> <li>the of 4000, SUI, SU2, and SU3 is written the first cycle</li> <li>RPa of 5000, SUI, SU2, and SU3 is written the first cycle</li> <li>RPa of 5000, SUI, SU2, and SU3 is written the first cycle</li> <li></li></ul>		ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
<ul> <li>25-1-12. Sequencer Control of a Vector of Vertices</li> <li>1. PA sends a vector of divertices (actually vertex indices - 32 bitsindex for 2048 bit total) to the RE's Vertex FIFO estate protein case will as tag into position cache is sent along with vertices</li> <li>also before the vector is sent to the RE. the CP has loaded the global instruction solve with the vertex shaded on the prostion cache divert have been events.</li> <li>also before the vector is sent to the RE. the CP has loaded the global instruction solve with the vertex cache are program (asing the MP).</li> <li>The SEO that accesses the IS base for this shader using the load istate pointer (provided to all sources by the RBM when the CPI is done loading that program.</li> <li>25. EEO arbitrates between the Fivel FIFO and the Vertex FIFO - basically the Vertex FIFO elvays has priority.</li> <li>at this point the vector is event to be transformed of the parameter cache is full unless the pipe a nothing fiels to do (in point bar Vertex FIFO - basically the Vertex FIFO elvays has priority.</li> <li>35. EEO arbitrates between the Fivel FIFO and the Vertex FIFO - basically the transformed parameter active is a load vector to be transformed of the parameter cache is full unless the pipe a nothing of top sector a vector to be transformed of the parameter cache is full unless the pipe an onthing else to do (in or picks are in the point of the tas been allocated.</li> <li>35. EEO arbitrates between the Transformed fitthe parameter cache is full unless the pipe as nothing of top sector active by the program.</li> <li>36. EEO arbitrates between the the register file to raide all table register, which is accessed using the state pointer that came down with the vertice.</li> <li>37. FO of SUU, SUI, SU2, and SU3 is written the tract cycle.</li> <li>38. EEO arbitrates between the SP register file to raide add the parameter cache is full unless the prior.</li> <li>39. EEO constructs a control packet for the vector and sends it to the first reservation s</li></ul>		24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	41 of 43	Formatted: Bullets and Numbering
<ol> <li>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 easily with vertices         space well as tag into position cache for the vector vas sent         also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex         shader program (arguing the MH7)         The vertex program (arguing the NH7)         Ste2 divertifies between the Vertex FIFO - alsocally the Vertex FIFO always has promy         at this point the vector is removed from the Vertex FIFO always has promy         at this point is not going to solical a vector to be transformed if the parameter cache is full unless the pipe as         nothing due to do (on o picel are in the pixel file).         Ste2 divertifies the vector is Progleter file or vertices         set of Vertex program (arguing the Program)         Ste2 divertifies the vector is the Progleter file solical vector is abandwith of 2048 bits/cycle)         the divertex indices are sent to the 64 register file sole of vector         set program (arguing the MH7)         set of SUB, SUB, SUB, SUB, and SUB awritten the second cycle         REF of SUB, SUB, SUB, SUB, SUB, and SUB awritten the second cycle         REF of SUB, SUB, SUB, SUB, SUB, and SUB awritten the file cycle         REF of SUB, SUB, SUB, SUB, SUB, SUB, and SUB awritten the file cycle         REF of SUB, SUB, SUB, SUB</li></ol>	25.26.	Examples of progra	m executions		*	-
<ul> <li>state pointer as well as tag into position cache is sent along with writces</li> <li>space we allocated in the position cache for transformed position before the vector was sent</li> <li>also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH7)</li> <li>The vertex program (using the MH7)</li> <li>The vertex program (using the MH7)</li> <li>SEO arbitrates between the Poke IFPO and the Vertex FIPO basicably the Vertex FIPO always has priority</li> <li>at this point the vector is removed from the Vertex FIPO basicably the Vertex FIPO always has priority</li> <li>the number of CPRs required by the program is setted in a local state register, which is accessed using the state point that were always the program (always has priority)</li> <li>SEO always base space in the SP register file how the Vertex FIPO always has priority</li> <li>the number of CPRs required by the program is stored in a local state register, which is accessed using the state point that came the SP register file how the Alex Piese Sente (always has a priority)</li> <li>SEO and vertex data until space in the register file has been allocated</li> <li>SEO and vertex data until space in the register file has been allocated</li> <li>SEO and vertex indices are sent to the Grigitar files over 4 cycles</li> <li>RE7 of SUD, SUT, SU2, and SU3 as written the first reservation state. (the FIPO in front of table vector in the state significant 20 bits (bound on the state significant 20 bits (bound on the setter significant 20 bits (bound on the setter significant a comb pacel bar the vector and sends in the first reservation station (the FIPO in front of table as earlied to the vector and sends in the first reservation state (the FIPO in front of table as pointer).</li> <li>SEO constructs a comb pacel and there here the reservation state (the FIPO in front of ALU state machine 0, or ASM0 FIPO, a count practer and p</li></ul>		-			/ertex FIFO	
<ul> <li>at this point the vector is removed from the Vertex FIFO</li> <li>the arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (in no pixels are in the pixel fflo).</li> <li>SEC allocates space in the SP register file for index data plus GPRs used by the program</li> <li>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</li> <li>SEQ will not send vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)</li> <li>the 64 vertex indices are sent to the 64 register files over 4 cycles</li> <li>RFD of SU0, SU1, SU2, and SU3 is written the second cycle</li> <li>RF2 of SU0, SU1, SU2, and SU3 is written the second cycle</li> <li>RF2 of SU0, SU1, SU2, and SU3 is written the formating data bits are set to zero (x, y, z)</li> <li>SEC constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fich state machine 0, or TSM0 FIFO)</li> <li>the control packet for the vector and sends it to the first reservation station (the FIFO in front of fich state machine 0, or TSM0 FIFO)</li> <li>the control packet and forches the instructions for fetch clause 0 from the global instruction store</li> <li>TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store</li> <li>TSM0 does not wait for requests made to the RF2 sit is received</li> <li>on the roll packet for the vector and station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)</li> <li>the control packet and the bedote 10 to the RF2 sit is received</li> <li>on the roll has written all the data to the register file write index for the data to the register file write index for the data to the register file write index for the field data in the tow which we data to the RF2 sit is received</li> <li>an instructions of ALU datase 0 are issued by ASM0 (</li></ul>	<ul> <li>sta</li> <li>sp;</li> <li>als</li> <li>sh</li> <li>Th</li> </ul>	te pointer as well as tag into p ace was allocated in the positio to before the vector is sent to ader program (using the MH e vertex program is assumed t the SEQ then accesses the IS	psition cache is sent along w on cache for transformed pos o the RE, the CP has loade ?) o be loaded when we receiv S base for this shader using	vith vertices sition before the vector was sent of the global instruction store with e the vertex vector. the local state pointer (provided to a	the vertex	
<ul> <li>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</li> <li>SEQ will not send vertex data until space in the register file has been allocated</li> <li>SEQ sends the vector to the SP register file over the RE_SP interface (which has a bandwidth of 2048 bits/cycle)</li> <li>the 64 vertex indices are sent to the 64 register files over 4 cycles</li> <li>RF0 of SU0, SU1, SU2, and SU3 is written the first cycle</li> <li>RF2 of SU0, SU1, SU2, and SU3 is written the first cycle</li> <li>RF2 of SU0, SU1, SU2, and SU3 is written the first cycle</li> <li>RF2 of SU0, SU1, SU2, and SU3 is written the fourth cycle</li> <li>the index is written to the least significant 32 bits (floating point format7) (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)</li> <li>SEC constructs a control packet on the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 PIFO)</li> <li>the control packet contains the state pointer, the tag to the position cache and a register file base pointer.</li> <li>TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store</li> <li>TSM0 was first selected by the TSM arbiter before it could start</li> <li>all instructions of fetch clause 0 are issued by TSM0</li> <li>the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)</li> <li>TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the 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 from the global instructions store</li></ul>	● at the	this point the vector is removed arbiter is not going to select a	I from the Vertex FIFO vector to be transformed if t		-	
<ul> <li>the 64 vertex indices are sent to the 64 register files over 4 cycles <ul> <li>RFD of SU0, SU1, SU2, and SU3 is written the first cycle</li> <li>RF1 of SU0, SU1, SU2, and SU3 is written the second cycle</li> <li>RF2 of SU0, SU1, SU2, and SU3 is written the fourth cycle</li> </ul> </li> <li>the index is written to the least significant 32 bits (floating point format?) (what about compound indices) of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)</li> </ul> <li>SEC constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fatch state machine 0, or TSM0 FIFO)</li> <li>the control packet contains the state pointer, the tag to the position cache and a register file base pointer.</li> <li>TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store <ul> <li>TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store</li> <li>TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store</li> <li>TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the Text 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</li> </ul> </li> <li>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 fetch state machine 1, or TSM1 FIFO)</li> 11. the control packet continues to travel down the path of reservation stations until all clauses have been executed <ul> <li>position can be exported in ALU clause 0 (or 47); the data (and the tag) is sent over a position bus (which</li></ul>	• the sta	number of GPRs required by te pointer that came down with	the program is stored in a lo the vertices	cal state register, which is accessed	using the	
<ul> <li>fetch state machine 0, or TSM0 FIFO)</li> <li>the control packet contains the state pointer, the tag to the position cache and a register file base pointer.</li> <li>TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store</li> <li>TSM0 was first selected by the TSM arbiter before it could stat</li> <li>all instructions of fetch clause 0 are issued by TSM0</li> <li>the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO).</li> <li>TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received</li> <li>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 0 from the global instruction store</li> <li>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)</li> <li>the control fetch state machine 1, or TSM1 FIFO)</li> <li>the control packet continues to travel down the path of reservation stations until all clauses have been executed</li> <li>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</li> <li>A parameter cache pointer is also sent along with the position data before it gets sent back to the PA</li> <li>there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA</li> </ul>	<ul> <li>the</li> <li>•</li> <li>•</li> <li>•</li> <li>the</li> </ul>	e 64 vertex indices are sent to t RF0 of SU0, SU1, SU2, and RF1 of SU0, SU1, SU2, and RF2 of SU0, SU1, SU2, and RF3 of SU0, SU1, SU2, and index is written to the least sig	he 64 register files over 4 cy SU3 is written the first cycle SU3 is written the second cy SU3 is written the third cycle SU3 is written the fourth cyc gnificant 32 bits (floating po	rcles le int format?) (what about compour		
<ul> <li>TSM0 was first selected by the TSM arbiter before it could start</li> <li>7. all instructions of fetch clause 0 are issued by TSM0</li> <li>8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)</li> <li>• TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received</li> <li>• 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</li> <li>9. 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 fetch state machine 1, or TSM1 FIFO)</li> <li>11. the control packet continues to travel down the path of reservation stations until all clauses have been executed</li> <li>• 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</li> <li>• 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.</li> <li>• there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA</li> </ul>	fetch s	tate machine 0, or TSM0 FIFO	)			
<ol> <li>8. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)</li> <li>TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received</li> <li>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</li> <li>9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store</li> <li>10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)</li> <li>11. the control packet continues to travel down the path of reservation stations until all clauses have been executed</li> <li>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</li> <li>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.</li> <li>There is a position export FIFO in the SP that buffers position data before it gets sent back to the PA</li> </ol>					tion store	
<ul> <li>FIFO) <ul> <li>TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received</li> <li>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</li> </ul> </li> <li>9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store</li> <li>10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)</li> <li>11. the control packet continues to travel down the path of reservation stations until all clauses have been executed</li> <li>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</li> <li>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.</li> <li>there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA</li> </ul>	7. all instr	ructions of fetch clause 0 are is	sued by TSM0			
<ul> <li>FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause</li> <li>9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store</li> <li>10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)</li> <li>11. the control packet continues to travel down the path of reservation stations until all clauses have been executed</li> <li>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</li> <li>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.</li> <li>there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA</li> </ul>	FIFO) • TS	M0 does not wait for requests	made to the Fetch Unit to co	omplete; it passes the register file wri		
<ul> <li>clause 0 from the global instruction store</li> <li>10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)</li> <li>11. the control packet continues to travel down the path of reservation stations until all clauses have been executed</li> <li>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</li> <li>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.</li> <li>there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA</li> </ul>	FIF	FO; a count greater than zero i				
<ul> <li>station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)</li> <li>11. the control packet continues to travel down the path of reservation stations until all clauses have been executed</li> <li>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</li> <li>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.</li> <li>there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA</li> </ul>				M arbiter) and gets the instructions fo	or ALU	
<ul> <li>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</li> <li>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.</li> <li>there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA</li> </ul>					ervation	
	• po: sha • A p goi	sition can be exported in ALU o ared with all four shader pipes) parameter cache pointer is also ing to be in the parameter cach	clause 3 (or 4?); the data (ar back to the PA's position ca sent along with the position le.	nd the tag) is sent over a position bus ache n data. This tells to the PA where the	s (which is data is	
	Exhibit 2020.docR44	10_Sequencer.doc 62852 Bytes*** ⓒ АТ			) *** <u>09/04/15 12:56</u>	

ATI Ex. 2106 IPR2023-00922 Page 76 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20155		42 of 43

- 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 clause 7 (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

## 25.1.226.1.2 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
- 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.
- SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
  - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
  - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
  - all other information (such as quad address for example) travels in a separate FIFO
- 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 2020.docR400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:56

- - Formatted: Bullets and Numbering

AMD1044\_0256988

ATI Ex. 2106 IPR2023-00922 Page 77 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE		
	24 September, 2001	4 September, 20155	GEN-CXXXXX-REVA	43 of 43		1487/86
001000				4	Formatted: Bullets and Numbering	12

25.1.326.1.3 Notes

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

## 26.27. Open issues

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

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

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

Saving power?

Exhibit 2020 doc R400\_Sequencer.doc 62852 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:56

AMD1044\_0256989

Formatted: Bullets and Numbering

ATI Ex. 2106 IPR2023-00922 Page 78 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	1 of 48
Author:	Laurent Lefebvre			·
Issue To:		Copy No:		
	R400 S	equencer Spe	ecification	
		SQ		
		Version 1. <u>5</u> 4		
requ		cted uses of the block. It al	cer block (SEQ). It provides an ov lso describes the block interfaces,	
Document Lo		erforce\r400\doc_lib\design\b 0 Sequencer Specification APPROVALS	locks\sq\R400_Sequencer.doc	
	Name/Dept	APPROVALS	Signature/Date	
	NTIALLY DETRIME	NTAL TO THE INTE	NFORMATION THAT CO REST OF ATI TECHNOL SE OR DISCLOSURE.	
work created in unpublished wo confidential, pro	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 co publication has occurred. This w of this document may be used, re sion of ATI Technologies Inc."	oyright in this ork contains
Exhibit 2021.docR400_S4	equencer.doc 64782 Bytes*** ⓒ AT	I Confidential. Reference C PM12/06/01-02:27 PM12/06/01-04:59 P	Copyright Notice on Cover Page © ≝	<b>) ***</b> <u>09/04/15 12:57</u>

# ATI 2021 LG v. ATI IPR2015-00325

AMD1044\_0256990

ATI Ex. 2106 IPR2023-00922 Page 79 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201511		2 of 48

# Table Of Contents

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. SEQUENCER INSTRUCTIONS	
5. CONSTANT STORES	
5.1 Memory organizations	
5.2 Management of the re-mapping tables	
5.2.1 Dirty bits	
5.2.2 Free List Block	
5.2.3 De-allocate Block	
5.2.4 Operation of Incremental model	
5.3 Constant Store Indexing	
5.4 Real Time Commands.	
6. LOOPING AND BRANCHES	2220
6.1 The controlling state	2220
6.2 The Control Flow Program	2220
6.3 Data dependant predicate instructions.	2523
6.4 HW Detection of PV,PS	2523
6.5 Register file indexing	
6.6 Predicated Instruction support for Texture clauses	
6.7 Debugging the Shaders	
6.7.1 Method 1: Debugging registers	
6.7.2 Method 2: Exporting the values in the GPRs (12)	
7. PIXEL KILL MASK	
8. MULTIPASS VERTEX SHADERS (HOS)	
9. REGISTER FILE ALLOCATION	
10. FETCH ARBITRATION	
11. ALU ARBITRATION	
12. HANDLING STALLS	
13. CONTENT OF THE RESERVATION STATION FIFOS	
14. THE OUTPUT FILE	
15. IJ FORMAT	
15.1 Interpolation of constant attributes	
16. THE PARAMETER CACHE	
17. VERTEX POSITION EXPORTING	
18. EXPORTING ARBITRATION	
19. EXPORT TYPES	
19.1 Vertex Shading	
19.2 Pixel Shading	
20. SPECIAL INTERPOLATION MODES	

Exhibit 2021.doc/R400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 00/04/15.12:57

AMD1044\_0256991

ATI Ex. 2106 IPR2023-00922 Page 80 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE		
	24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	3 of 48		
	l time commands					
23. REGI	STERS			3334		
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~				***************************************		
25.1.1						
25.1.2						
25.1.3 SQ to SP: GPR Input Mux select						
25.1.4	SQ to SP: Parameter	Cache Read control bu	JS			
25.1.5	SQ to SX: Parameter	Cache Mux control Bu	\$			
25.1.6	SQ to SP: Staging Re	gister Data				
<u>25.1.7</u>	PA to SQ : Vertex inte	erface				
25.1.8	SQ to CP: State repo	rt				
<u>25.1.9</u>	SQ to SX: Control but	<u>S</u>				
25.1.10	SX to SQ : Output file	control				
<u>25.1.11</u>	SQ to TP: Control but	5				
25.1.12	TP to SQ: Texture sta	<u> 1  </u>				
25.1.13	SQ to SP: Texture sta	all				
25.1.14	SQ to SP: GPR and F	Parameter cache contro	<u>)</u>			
25.1.15	SQ to SPx: Instruction	าร				
25.1.16	SP to SQ: Constant a	ddress load				
25.1.17						
<u>25.1.19</u>	SQ to CP: RBBM bus					
25.1.20						
26.1.1	Sequencer Control of	a Vector of Vertices				
26.1.2	Sequencer Control of	a Vector of Pixels				
26.1.3	Notes					

Exhibit 2021.docR400\_Sequencer.doc 64782 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 00/04/15.12:57

AMD1044\_0256992

ATI Ex. 2106 IPR2023-00922 Page 81 of 223

2 A	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
AND 1	24 September, 2001	4 September, 201511		4 of 48
OPEN	ISSUES	I Doombor 20016 J.		
-OVERV	/IEW			5
				7
Data Fl	ow graph	*****		9
			*****	
0	11 0			
	,			
	~			
	~			
				<del></del>
HMDo	tection of PV PS			
				23
7.1	Vethod 1: Debugging regi	sters		
7.2	Vethod 2: Exporting the va	alues in the GPRs (12)		24
REGIS	TER FILE ALLOCATION	***************************************		
-FETCH	ARBITRATION			
ALU AI	RBITRATION	******		
CONTE	NT OF THE RESERVAT	ION STATION FIFOS		
THE OU	UTPUT FILE			
,				
			*****	
				28
				29
0.3.2				
STATE				
			***************************************	
			*****	
L Conti				
Conte				

Exhibit 2021.doc/R400\_Sequencer.doc 64782 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*
<u>PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM1</u>

AMD1044\_0256993

ATI Ex. 2106 IPR2023-00922 Page 82 of 223

12	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	5 of 48
5. INTER	FACES	Doormbor 20016		
		******		32
25.1.1				
25.1.2	1			
25.1.3				
25.1.4				
25.1.6				
25.1.7				
25.1.8	SQ to CP: State report			
25.1.9		*****	*****	
25.1.10				
25.1.11				
25.1.12				
25.1.13 25.1.14				
<del>25.1.14</del> 25.1.15				
25.1.15				
25.1.17				
25.1.18				
25.1.19				
25.1.20		*****		
26. EXAM	PLES OF PROGRAM EXEC	UTIONS		
26.1.1				
26.1.2				
26.1.3				
4SEQU	ENCER INSTRUCTIONS	***************************************		
5. CONS	TANT STORES			
	0			
			***************************************	
			******	
	,			
	, ,			
6.7.1	Method 1: Debugging regist	ers		
		ues in the GPRs (12)		
7. PIXEL	KILL MASK			
			***************************************	
			*****	
			*****	
14. THE-C		*****	***************************************	
hibit 2021.docR400_1	Sequencer.doc 64782 Bytes*** ⓒ AT	T Confidential. Reference C PM12/06/01-02:27-PM12/05/01-04:58-P	opyright Notice on Cover Page ∉ ≝	<b>0</b> *** <u>09/04/15 12:5</u>

ATI Ex. 2106 IPR2023-00922 Page 83 of 223

"h A	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
AU	24 September, 2001	4 September, 201511		6 of 48
. IJ FOF	MAT	Docombor 20016 J		
.1 Inter	polation of constant attribu	ites		23
THE P	ARAMETER CACHE	*****	*****	
. VERTE	EX POSITION EXPORTIN	G <u></u>	******	
. EXPOI	RTING ARBITRATION	*****	******	
	<u> </u>			
			******	
		FB information		
		******		
20.3.1				25
20.3.2	Pixel shaders			25
			***************************************	
	ex indexes imports TERS		*******	
	rol		*****	
	ext	*******	***************************************	<u>2</u> 0 07
		*****		<u>-</u>
1 Cont	<del>o neolo i eno</del> ovi	***************************************	***************************************	
		*****		<u>-</u>
		******		£0 28
25.1.1		******		
25.1.1				28
25.1.2				
25.1.4				
25.1.5				
25.1.6				
25.1.7	SQ to SP: Parameter C	ache Read control bus	*****	
25.1.8		sche Mux control Bus		
25.1.9		ta		
25.1.10	SX to Interpolators: Par	ameter Cache Return bus		
25.1.11	-SQ to SP0: Staging Reg	jister Data		
25.1.12				
25.1.13				
25.1.14				
25.1.15		****		
25.1.16		ontrol		
25.1.17	0			
25.1.18		t bus		
25.1.19	8	control		
25.1.20		uctions		
25.1.21		ant address load		
25.1.22				
25.1.23 25.1.24		ector load		
25.1.24 25.1.25		******		-/ۆ
			*****	
			***************************************	
26.1.1 26.1.2			•••••••••••••••••••••••••••••••••••••••	
	1			<del>کېکې</del>
26.1.3				

Exhibit 2021.doc/R400\_Sequencer.doc 64782 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*
<u>PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/06/01-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM12/00-02:27-PM1</u>

AMD1044\_0256995

ATI Ex. 2106 IPR2023-00922 Page 84 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	7 of 48
Povision	Changes	December 20016		
REVISION	<u>Changes:</u>			
<b>Rev 0.1 (Laure</b> Date: May 7, 20		First dra	ift.	
Rev 0.2 (Laure	nt Lefebvre)	Change	d the interfaces to reflect the changes	s in the
Date : July 9, 2			led some details in the arbitration sect	
Rev 0.3 (Laure Date : August 6			ed the Sequencer spec after the mee 3, 2001.	ting on
Rev 0.4 (Laure	,		the dynamic allocation method for r	
Date : August 2	24, 2001		an example (written in part by Vic) pixels/vertices in the sequencer.	of the
Rev 0.5 (Laure			iming diagrams (Vic)	
Date : Septemb Rev 0.6 (Laure		Change	d the spec to reflect the new	R400
Date : Septemb	per 24, 2001	archited	ture. Added interfaces.	
Rev 0.7 (Laure Date : October			constant store management, inst anagement, control flow manageme	
		data de	pendant predication.	
Rev 0.8 (Laure Date : October			d the control flow method to be Also updated the external interfaces.	more
Rev 0.9 (Laure	nt Lefebvre)	Incorpo	rated changes made in the 10/18/01	
Date : October	17, 2001		eeting. Added a NOP instruction, re nditional execute or jump. Added	
5 4 6 4		<i>y</i>	s <u>registers</u> .	
Rev 1.0 (Laure Date : October		Refined register	interfaces to RB. Added sregisters.	state
Rev 1.1 (Laure	nt Lefebvre)	Added	SEQ→SP0 interfaces. Changed	
Date : October	26, 2001		n. Changed VGT→SP0 interface. s added.	Debug
Rev 1.2 (Laure			es greatly refined. Cleaned up the spe	C.
Date : Novemb Rev 1.3 (Laure		Added t	he different interpolation modes.	
Date : Novemb	er 26, 2001		·	
Rev 1.4 (Laure Date : Decemb			the auto incrementing counters. Ch Γ→SQ interface. Added content on co	
		manage	ment. Updated registersGPRs.	
Rev 1.5 (Laure Date : Decemb			ed from the spec all interfaces that y tied to the SQ. Added explanation	
		constan	t management. Added P	A→SQ
		synchro	nization fields and explanation.	
Exhibit 2021.docR400_Sec	uencer.doc 64782 Bytes*** © AT	1 Confidential. Reference PM12/06/01_02:27_PM12/05/01_04:69	Copyright Notice on Cover Page ©	*** <u>09/04/15 12:5</u>
		<u>rw+zrowu+ uziz/-rw</u> +z/00/01-04:58	<del></del>	

ATI Ex. 2106 IPR2023-00922 Page 85 of 223

AMD1044\_0256996

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 201511		8 of 48

## 1. Overview

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

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

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

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

Exhibit 2021.doc R400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09/04/15 12:57

AMD1044\_0256997

ATI Ex. 2106 IPR2023-00922 Page 86 of 223

# PROTECTIVE ORDER MATERIAL

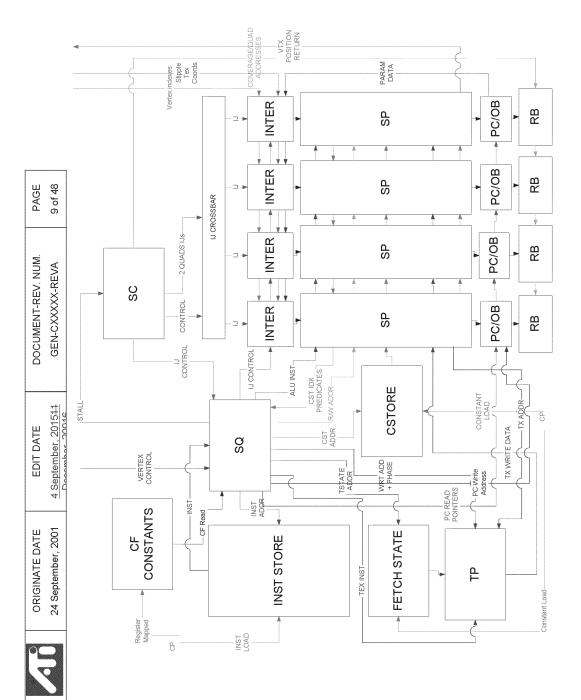


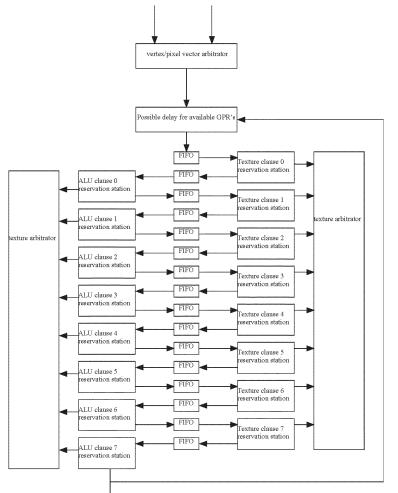
Exhibit 2021.400-R4400\_Sequencercies 64782 Bytes 444 © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*\*0004161.1257 PM1206001.0222 PM1206501.0458 PM

AMD1044\_0256998

ATI Ex. 2106 IPR2023-00922 Page 87 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September, 2001	4 September, 201511		10 of 48	

1.1 Top Level Block Diagram



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

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

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

Exhibit 2021 doc R400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:57

AMD1044\_0256999

ATI Ex. 2106 IPR2023-00922 Page 88 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	11 of 48

XY screen position if needed) are sent to the interpolator buffers which are going to the barycentric coordinates them to interpolate the parameters and place the interpolated valueresults into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

On receipt of a command, the level 0 fetch machine issues a <u>texture-fetch</u> request to the <u>TP</u> and corresponding register-<u>GPR</u> 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 <u>register-GPR</u> write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

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

On receipt of a command, the level 0 ALU machine first decrements the input FIFO <u>1</u>\_counter and then issues a complete set of level 0 shader instructions. For each instruction, the <u>ALU</u>\_state machine generates 3 source addresses, one destination address (<u>3 cycles later</u>) and an instruction. Once the last instruction <u>h</u>as been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

{ISSUE: How do we handle parameter cache pointers (computed, semi-computed or not computed)?}

A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

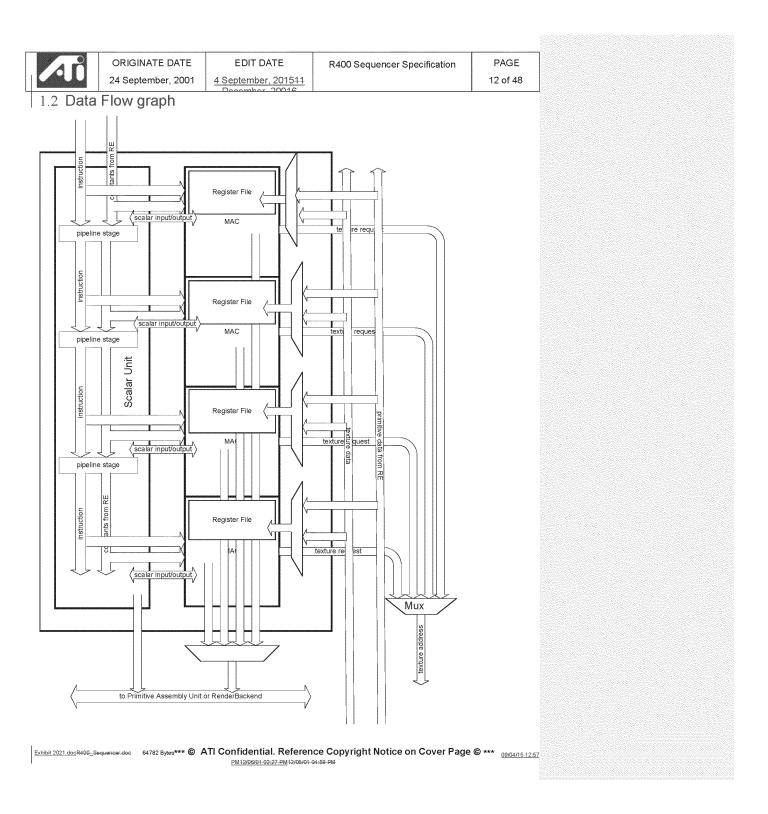
All other clauseslevel process in the same way until the packet finally reaches the last ALU machine (7).

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

Exhibit 2021 docR400\_Sequencer.doc 64782 Bytes\*\*\* 
C ATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\*
O9/04/15 12:57
PM12/06/01 02:27 PM12/06/01 04:69-PM

AMD1044\_0257000

ATI Ex. 2106 IPR2023-00922 Page 89 of 223

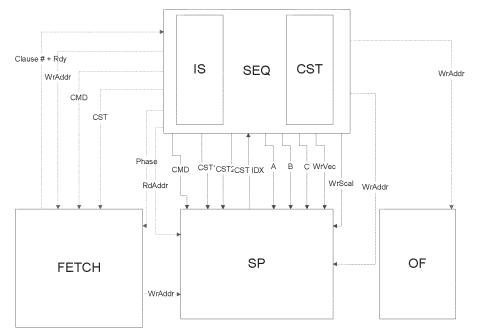


ATI Ex. 2106 IPR2023-00922 Page 90 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	13 of 48

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.

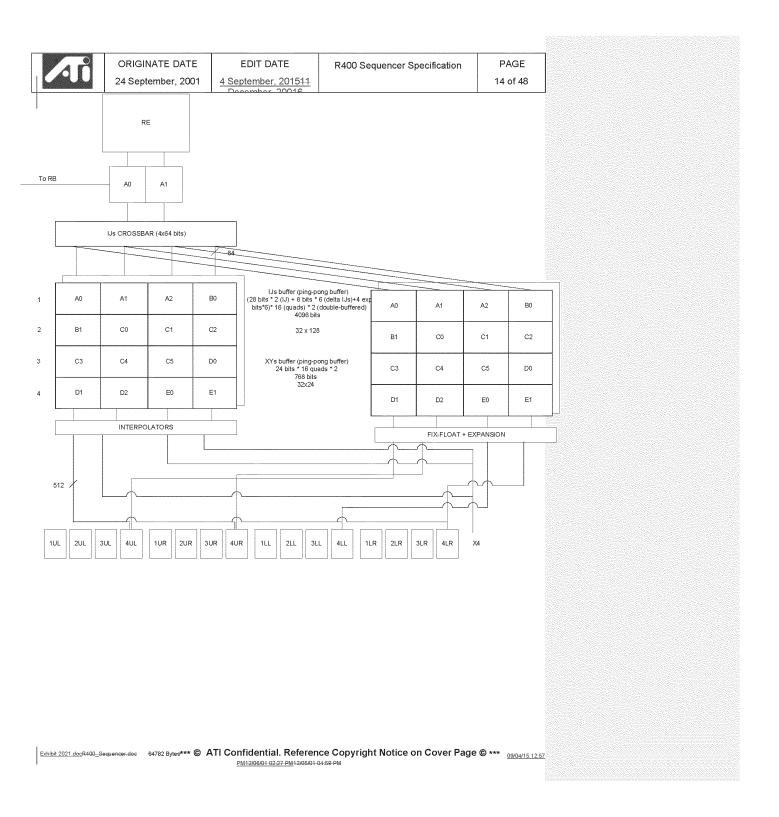
## 2. Interpolated data bus

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

Exhibit 2021 doc R400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:57

AMD1044\_0257002

ATI Ex. 2106 IPR2023-00922 Page 91 of 223



ATI Ex. 2106 IPR2023-00922 Page 92 of 223 PROTECTIVE ORDER MATERIAL

	T15 T16 T17 T18 T19 T20 T21 T22 T23					V V 32- 48- 35 51	V V 36-52- 39 55	V V 40-56- 43 59	V V 44-60- 47 63	~
	T22					35 32 <	1		1	
	121					2 <del>1</del> 6 19 −	≥°, 2°	24- 27	3 <sup>2</sup> 8 <	
	) T20					> 🖔	> ~4	⇒ ∽ £	> 년 £	
	T15							Ш	Ш	
	T18								8	
	T17			≿ ଘ	×Ξ		8	5	ß	
PAGE 15 of 48	T16			ы	ш				BO	
μţ	T15			ы	ш	5	D2			Δ
÷	T14	₹₽	≿8		Хã	3 C	5 2	C5		
'. NUN REVA	T13	ō	5		8	B1				
T-REV XXX-F	T12 T13	δ	D2		8	AO	A1	A2		
DOCUMENT-REV. NUM. GEN-CXXXX-REVA		b E	52	C5 X				Ê	μ	
GEN	T10 T11		<b>C</b> 4	C5	u O				0	
	- 19		<b>Q</b>	C5			8	5	C2	
2 <u>11</u>	T8	≿ຮ	≿ છ	לא מ	×Ω				BO	00111000
EDIT DATE 4 September, 201544	1	ଅ	8	δ	8	5	DZ			Ď
EDIT DATE	T6	ទ	8	5	3	ů.	5	C5		
4 Set	T5	×₽			∑ 8	8				
	T4	Ξ			Bg	AO	A1	A2		
DATE r, 200	13	Ξ			BO	X 48- 51-	XY 52- 55	56- 56-	% ¢≾	
VATE	12	l	۶¥	≽ श्व		XY 32- 2	39 <del>2</del> ×	× 4 43 × 4	XY XY 44- 60- 47 63	>
ORIGINATE DATE 24 September, 2001	Ê	Ao	A1	A2		76 19 19	% % %	24- 27-24-	33 <sup>2</sup> 8 ×	Ś
Ň ~	01	A0	A1	A2		×°.	XX 7-4	Z % E	54 15 15	
00		fammen			ი ი		- SP	N SP	ы СР	

Exhibit 2021 doe R400\_Sequencer doe 64782 Bytes 444 © ATI Confidential. Reference Copyright Notice on Cover Page © 444 0004115 1257 PM1206001 02227 PM1206501 0458 PM

AMD1044\_0257004

ATI Ex. 2106 IPR2023-00922 Page 93 of 223

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 201511		16 of 48

Above is an example of a tile we the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 24 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the register GPRs to write the valid data in.

## 3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the INST\_DATA, INST\_INDEX\_PORT control registers register. The INST\_INDEX\_PORT is auto-incremented on both reads and writes to the INST\_DATA register.

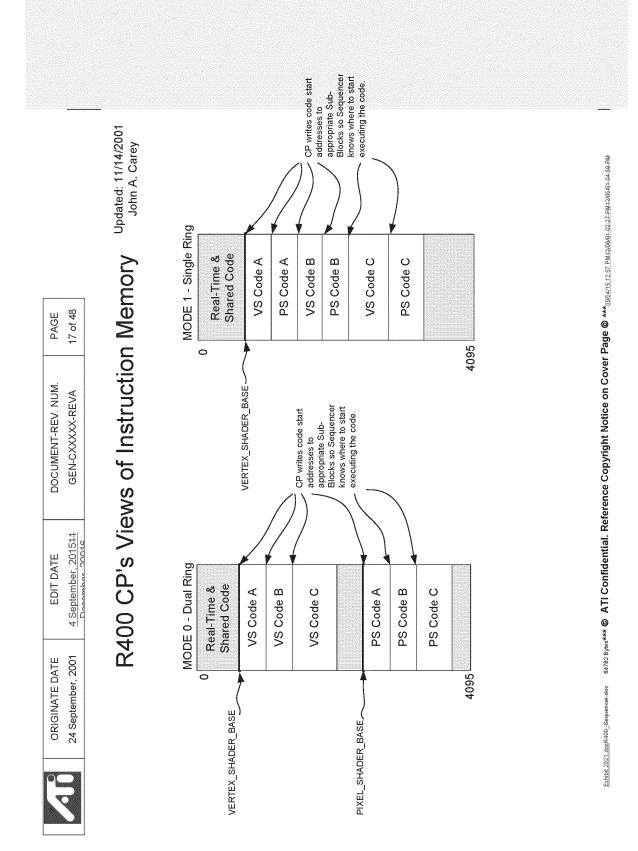
The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The MSB of the INST\_INDEX\_PORT register contains the packet type for the sequencer to know where it must wrap around. The wrap around points are arbitrary and they are specified in the VS\_BASE and PIX\_BASE registersregisters.

For the Real time commands the story is quite the same but for some small differences. The CP will use the INST\_INDEX\_PORT\_RT and INST\_DATA\_RT register pair instead of the regular ones and there are no wrap around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

Exhibit 2021.docR400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15.12:57

AMD1044\_0257005

ATI Ex. 2106 IPR2023-00922 Page 94 of 223



PROTECTIVE ORDER MATERIAL

AMD1044\_0257006

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	1
24 September, 2001	4 September, 201511		18 of 48	Statistics

## 4. Sequencer Instructions

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

## 5. Constant Stores

## 5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the remapingre-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants.

The texture state is also kept in a similar memory. The size of this memory is 192x128. The memory thus holds 128 texture states (192 bits per state). The logical size <u>exposed\_exposes\_32</u> different states total, which are going to be shared between the pixel and the vertex shader. The size of the <u>remapingre-mapping</u> table to for the texture state memory is 16 lines (each line addresses 2 texture state lines in the real memory). The write granularity is 2 texture state lines (or 384 bits). The driver sends 512 bits but the CP ignores the top 128 bits. It thus takes 12 clocks to write the two texture states.

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change. Its size is 256320\*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The CP is loading the constant store using the CONST\_DATA and CONST\_ADDR registersregisters. It does so by writing to the CONST\_ADDR register the logical address for the constant block it wants to update and then writes 16 times to the CONST\_DATA register. The CONST\_ADDR is auto-incremented on both reads and writes to the CONST\_DATA register.

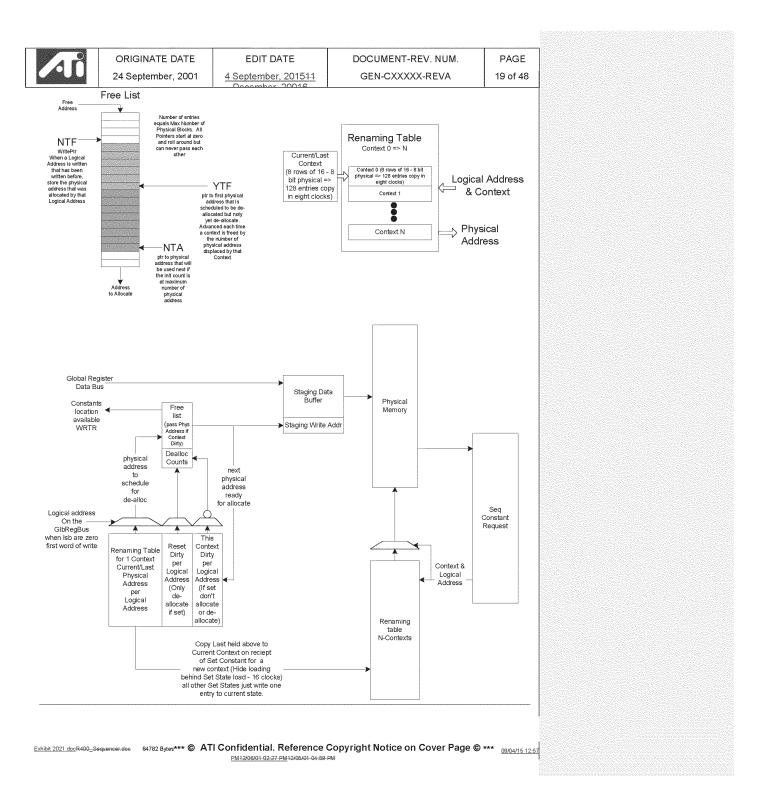
#### 5.2 Management of the remapingre-mapping tables

The sequencer is responsible to manage two remapingre-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its remapingre-mapping tables to a new one. We have 8 different remapingre-mapping tables we can use concurrently. More details and a diagram to come....

Exhibit 2021 docR400\_Sequencer.dec 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:57

AMD1044\_0257007

ATI Ex. 2106 IPR2023-00922 Page 96 of 223



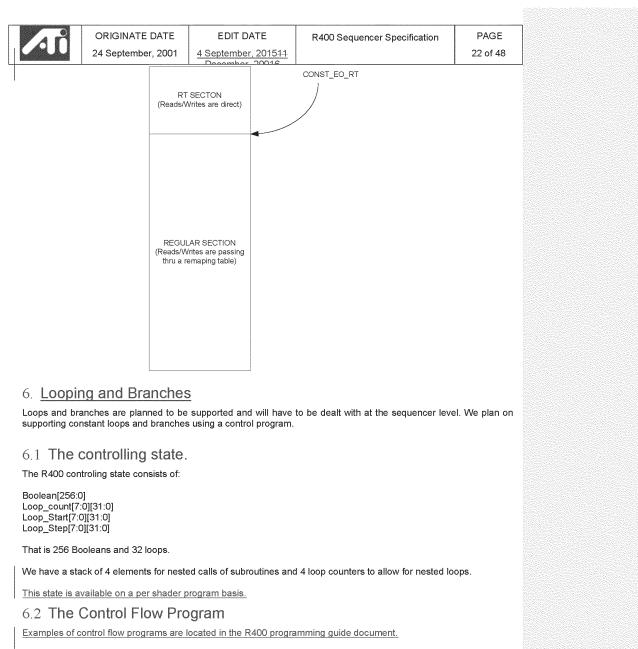
ATI Ex. 2106 IPR2023-00922 Page 97 of 223

AA	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 2001	4 September, 201511		20 of 48	
.2.1 Dirty	y bits			4	- Formatted: Bullets and Numbering
ne logical add ach address equire de-allo ne physical ad ata. If they a ogical address vice to the sa o do it will al	dress is addressed. The written while in this con ocation of whatever addres ddress store needs to be are both set, then the data s. No de-allocation or a mme logical address betwee llow multiple writes to all	second one will be set to text. The reset dirty is n ss stored in the renaming de-allocated and a new p a will be written into the pl illocation takes place. Th sen context changes. NO ocate all physical memor	first one will be set to zero on reset zero when ever a new context is writt ot set, then writing to that logical ad able. If it is set and the context dirty i hysical address is necessary to store nysical address held in the renaming is will happen when the driver does a FE: It is important to detect and prevery and thus hang because a context	en and set for dress will not s not set, then the incoming for the current a set constant ent this, failure	
	tart and thus free up space e <i>List Block</i>	<u>e.</u>		*	
free list bloc cremented e e checked ea heck the free hunk from the torage of a fr laintain three ointer will ide ee more physi- te free list like he second p- hunks de-alid hey are still in he third point	kk that would consist of a every time a chunk of phy- ach time a physical block i a list for an available phy a counter. ree list big enough to store a pointers for the free list entify the next location to sical memory locations th a a ring. ointer will be called YTF pocates when a context fir use. But as soon as the ter will be called NTA (Net	sical memory is used until is needed, and if the origin rsical block address. The all physical block address that are reset to zero. T write the physical address an we have. Once record (Yet To Free). The YTF hishes. The address betw context using then is dism ext To Allocate). This point	r Initial Free Counter) that would res- they have all been used once. This ial ones have not been used up, us a e count is the physical address for w ses. he first one we will call NTF (Next T of a block to be de-allocated. Note: ing address the pointer will be increm pointer will be advanced by the numb veen the YTF and NTF cannot be re- issed the YTF will be advanced. ther will point will point to the next address and the IFC is at its maximum count.	counter would new one, else then getting a o Free). This we can never tented to walk wer of address used because	
	allocate Block	•		*	Formatted: Bullets and Numbering
e maintained pon reset or locks in the p lake available	specifying how many blo when this context is activ previous context that will	cks were written into the e on the back and differer no longer be used. This <s freed="" previou<="" td="" the="" when=""><td>ach context. While in current context ree list at the NTF pointer. This cour t than the previous context. It is actu- count will be used to advance the N s context was done. This allows the</td><td>t will be reset ally a count of ITF pointer to</td><td></td></s>	ach context. While in current context ree list at the NTF pointer. This cour t than the previous context. It is actu- count will be used to advance the N s context was done. This allows the	t will be reset ally a count of ITF pointer to	
.2.4 Ope	eration of Incremer	ntal model		*	
st counter is a postants hap ecause its no enaming table y set start wi he context ch ould be set, /hen a set cc uble will be co ddress with I ddress it repl ill be increme	set to zero. Also all the open, the reset dirty bit wi out at the max value. The open is a set of the physical address of 0. hanges. If a logical address of 0. hanges. If a logical address of the new data will be onstant comes with a diffe opied to the larger renam be loaded with a new phalaced in the renaming tab	lirty bits and the previous II not be set, so we will al data will be written into ph the big renaming table wi This process will be rep ess is hit that has its dirty over-written to the last p erent than last context, the ing table in the current (n nysical address during the le would be entered at th counter for the previous co	NTA pointers in the free list set to zer context will be initialized to zero. Whe locate a physical location from the fri ysical address zero. Both the addition I be updated for the logical address that are v bits set while in the same context, hysical address assigned for this log- previous context stored in the addition ew) context location. Then the set of a copy and if the reset dirty was set e NTF pointer location on the free list intext (zero) will be incremented. This	en the first set be list counter al copy of the at was written not dirty until both dirty bits gical address. onal renaming onstant logical the physical and the NTF	
	ty bits are set for the logi ee list at NTA pointer if NT		d. A line will be allocated of the free-	<u>list counter or</u> ← -	- <b>Formatted:</b> Bullets and Numbering
bit 2021.docR400_Se	equencer.doc 64782 Bytes*** ⓒ 🖡	TI Confidential. Referen PM12/06/01-02:27-PM12/05/01-0	ce Copyright Notice on Cover Page	© *** <u>09/04/15 12:57</u>	

ATI Ex. 2106 IPR2023-00922 Page 98 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	21 of 48	
		not set. A new physical a	ddress is allocated, the physical ad ented along with the de-allocate co		
last co	ontext.		cal address specified by the logical a		
has not free l	ist entries available (counte	er at max and YTF == NT/	I provide back pressure to the CP w A). The command stream will keep		
Now preferab	le when the constant contex		e it will be sent to this block and co		
			er context will no longer be referenc y adding the number of blocks freed		
to the YTF po pointer for futu		he physical addresses use	d by this context available to the N	ITA allocate	
This device al	lows representation of multi	ple contexts of constants da	ata with N copies of the logical addre	ss space. It	
			us some new additional data by jus nstants updates are small it can st		
context. How		e, less contexts will be store	ed and potentially performance will b		
Futilougitite wit	radii periorin da weir da d ili	ig could in this case.			
5.3 Cons	stant Store Indexi	ng			
			ded first with the indexes (that cor r (9 bits pointers x 16 vertexes/clock		
data must par	ss thru the Shader pipe for	the float to fixed conversion	on, there is a latency of 4 clocks (1	instruction)	
like this	me the sequencer is loaded	and the time one can inde	x into the constant store. The assem	idiy wili look	
MOVA R1.X,			2.X, also copies the content of R2.X i	into R1.X	
NOP ADD R3,R4		oat to fixed conversion from the sequencer to add F	R4 to C0[R2.X] into R3		
	don't really care about wha ritten again for the sake of s		e we use the state from the MOVA	instruction.	
5.4The storag	e needed in the sequencer	in order to support this featu	ıre is 2*64*9 bits = 1152 bits.	4	Formatted: Bullets and Numbering
<del>5.5</del> 5.4_Re	eal Time Commar	nds			
registersGPR not sending a mapping table	s. It works is the same way logical address but rather	r than when dealing with rea r a physical address and t	he CONST_DATA_RT and CONST gular constant loads BUT in this cas he reads are not passing thru the lary between the two zones is def	se the CP is remapingre-	
Exhibit 2021.docR400_S	equencer.doc 64782 Bytes*** ⓒ AT	1 Confidential. Reference ( PM12/06/01-02:27 PM12/06/01-04:69	Copyright Notice on Cover Page ©	) *** 09/04/15 12:57	

ATI Ex. 2106 IPR2023-00922 Page 99 of 223



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

Exhibit 2021 docR400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09/04/15 12:57

AMD1044\_0257011

ATI Ex. 2106 IPR2023-00922 Page 100 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	23 of 48

Pixel\_shader\_fetch[7:0][7:0] Pixel\_shader\_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located // eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

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

The control program has eleven basic instructions:

Execute Conditional\_execute\_Predicates Conditional\_jump Call Return Loop\_start Loop\_end End\_of\_clause Conditional\_End\_of\_clause NOP

Execute, causes the specified number of instructions in instruction store to be executed.

Conditional\_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.

Loop\_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.

Loop\_end increments (decrements?) the loop counter and jumps back the specified number of instructions.

Call jumps to an address and pushes the IP counter on the stack. On the return instruction, the IP is popped from the stack.

Conditional\_execute\_or\_Jump executes a block of instructions or jumps to an address is the condition is not met. Conditional\_execute\_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition. End\_of\_clause marks the end of a clause.

Conditional\_End\_of\_clause marks the end of a clause if the condition is met.

Conditional\_jumps jumps to an address if the condition is met.

NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader\_cntl\_size (or vshader\_cntl\_size) we break the program (clause) and set the debug registersregisters. 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 registersregisters.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

	Execute										
47	46 42	23 12	11 0								
Addressing	Addressing 00001 RESERVED Instruc										

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

		NOP
47	46 42	41 0

Exhibit 2021 doc R400\_Sequencer.doc 64782 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/1512:57 PM12/06/01-02:27 PM12/06/01-02:27 PM

AMD1044\_0257012

ATI Ex. 2106 IPR2023-00922 Page 101 of 223

			ATE DATE mber, 2001	EDIT DATE 4 September, 201511	R400 Sequencer Specification	PAGE 24 of 48
-	Addressing	00010		December 20016	RESERVED	I

This is a regular NOP.

Conditionnal_Execute								
47 46 42 41 40 33 32 31 24 23 12 11 0								
Addressing	00011	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address	

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

Conditionnal_Execute_Predicates									
47	46 42	41 35	34 33	32	31 24	23 12	11 0		
Addressing	00100	RESERVED	Predicate	Condition	RESERVED	Instruction count	Exec Address		
			vector						

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid.

	Loop_Start									
47	46 42	41 17	16 12	11 0						
	00101	RESERVED	loop ID	Jump address						
Addressing										

Loop Start. Compares the loop <u>countiterator</u> with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index <u>value-value</u>. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

	Loop_End							
47	46 42	41 17	16 12	11 0				
	00111	RESERVED	loop ID	start address				
Addressing								

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, 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	Jump address			
Addressing						

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

	Return					
47 46 42 41 0						
	01001	RESERVED	1000000			
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 40 33 32				32	31	30 12	11 0	
	01010	RESERVED	Boolean	Condition	FW only	RESERVED	Jump address	
Addressing								

Exhibit 2021.doc R400\_Sequencer.doc 64782 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257013

ATI Ex. 2106 IPR2023-00922 Page 102 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	25 of 48

If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.

Conditional_End_of_Clause							
47	46 42	41	40 33	32	31 0		
	01011	RESERVED	Boolean	Condition	RESERVED		
Addressing			address				

This is an optimization in the case of very short shaders (where the control flow instruction can't be hidden anymore and thus are not free. In this case, if the condition is met, the clause is ended, else we continue the execution of the clause.

End_of_Clause					
47	46 42	41 0			
Addressing	01011	RESERVED			

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop\_end or the loop\_start instruction is going to break the loop and set the debug registersGPRs.

# 6.3 Data dependant predicate instructions

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

PRED\_SETE\_# - similar to SETE except that the result is 'exported' to the sequencer. PRED\_SETNE\_# - similar to SETNE except that the result is 'exported' to the sequencer. PRED\_SETGT\_# - similar to SETGT except that the result is 'exported' to the sequencer PRED\_SETGTE\_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions:

PRED\_SETEO\_# - SETE0

PRED\_SETE1\_# - SETE1

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

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

#### P0\_ADD\_# R0,R1,R2

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

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

## 6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of nonmasked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

Exhibit 2021 docR400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09/04/15 12:57 PM12/06/01 02:27 PM12/06/01 04:68-PM

AMD1044\_0257014

ATI Ex. 2106 IPR2023-00922 Page 103 of 223

2	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
<u> </u>	24 September, 2001	4 September, 201511		26 of 48

The sequencer will also have to insert NOPs between PRED\_SET and MOVA instructions and their uses.

## 6.5 Register file indexing

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

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

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

The sequencer is going to keep a loop index computed as such:

Index = Loop\_counteriterator\*Loop\_iteratorstep + Loop\_initstart.

The index is going to return 0 if it is out of the range. We loop until loop iterator = loop count. Loop step is a signed value [-128...127].

## 6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector. A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. We have to make sure the invalid pixels aren't considered with this optimization.

# 6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

#### 6.7.1 Method 1: Debugging registers registers

Current plans are to expose 2 debugging, or error notification, registersregisters:

- 1. address register where the first error occurred
- 2. count of the number of errors

The sequencer will detect the following groups of errors:

- count overflow
- jump error
- relative jump address > size of the control flow program relative jump address > length of the shader program
- constant overflow
- register overflow
- call stack
- call with stack full
- return with stack empty

With two of the errors, a jump error or a register overflow will cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With the other errors, program can continue to run, potentially to worst-case limits.

Exhibit 2021 doc R400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* <u>PM12/06/01\_02:37 PM12/06/01\_04:59 PM</u>

AMD1044\_0257015

ATI Ex. 2106 IPR2023-00922 Page 104 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	27 of 48	

If indexing outside of the constant range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

6.7.2 Method 2: Exporting the values in the GPRs (12)

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

- 1) Normal
- 2) Debug Kill
- 3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug\_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

#### 7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK\_SETE MASK\_SETNE MASK\_SETGT MASK\_SETGTE

#### 8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

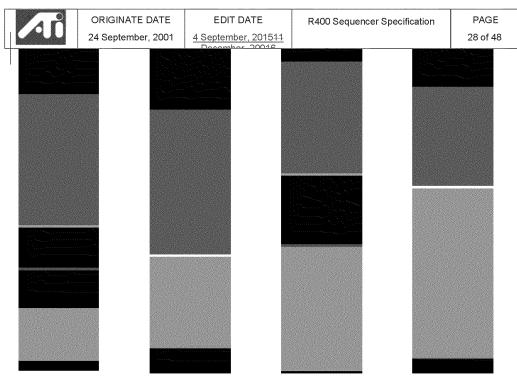
#### 9. Register file allocation

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

Exhibit 2021.docR400\_Sequencer.doc 64782 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257016

ATI Ex. 2106 IPR2023-00922 Page 105 of 223



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 arbitrersarbiters, one for the even clocks and one for the odd clocks. For exaemple, 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. <u>Also note that the interleaving also occurs across</u> clause boundaries.

Exhibit 2021.docR400\_Sequencer.doc 64782 Bytes\*\*\* 
ATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\*
ONCOMPT 12:05/01-02:27 PM12/05/01-04:59 PM

AMD1044\_0257017

ATI Ex. 2106 IPR2023-00922 Page 106 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	29 of 48	

## 12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic to <u>selectfrom selecting</u> 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 <u>enterfrom entering</u> the exporting clause (3?). The sequencer will set the OUT\_FILE\_FULL signal n clocks before the output file is actually full and thus the ALU arbiter will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

# 13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address and 1 bit to specify if the vector is of pixels or vertices. Since pixels and vertices are kept in order in the shader pipe, we only need two fifos (one for vertices and one for pixels) deep enough to cover the shader pipe latency. This size will be determined later when we will know the size of the small fifos between the reservation stations.

## 14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registersregisters with write BW 512 bits/clock and read BW 256 bits/clock. The staging registersregisters are 4x128 (and there are 16 of those on the whole chip).

## 15. IJ Format

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

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

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

P1

P3

$\Delta 01I = I(1) - I(0)$	
$\Delta 01J = J(1) - J(0)$	50
$\Delta 02I = I(2) - I(0)$ $\Delta 02J = J(2) - J(0)$	P0
$\Delta 03I = I(3) - I(0)$	
$\Delta 03J = J(3) - J(0)$	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 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

Exhibit 2021 docR400\_Sequencer.doc 64782 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257018

ATI Ex. 2106 IPR2023-00922 Page 107 of 223

_						
		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
		24 September, 2001	4 September, 201511		30 of 48	
Multiplies (Full Precision): 2 Multiplies (Reduced precision): 6 Subtracts 19x24 (Parameters): 2 Adds: 8						
	FORMAT OF P0's IJ: Mantissa 20 Exp 4 for I + Sign Mantissa 20 Exp 4 for J + Sign					
FORMAT of Deltas (x3):Mantissa 8 Exp 4 for I + Sign Mantissa 8 Exp 4 for J + Sign						

Total number of bits : 20\*2 + 8\*6 + 4\*8 + 4\*2 = 128

The Deltas have a leading 1, the Full precision IJs don't. This means that in the case of the deltas we MUST be able to shift 8 right (exponent value of 0 means number = 0, exponent value of 1 means shift right 8).<u>All numbers are kept</u> using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. <u>This means that tT</u>he maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

#### 15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
  P0,1,2,3 = A;
else if ((I = 0) or (J = 0)) and
      ((J = 0) \text{ or } (1-I-J = 0)) and
       ((1-J-I = 0) or (I = 0))) {
          if(| != 0) {
             P0 = A
          } else if(J != 0) {
             P0 = B;
          } else {
             P0 = C;
        //rest of the quad interpolated normally
3
else
{
         normal interpolation
}
```

# 16. The parameter cache

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

#### 17. Vertex position exporting

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT\_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

Exhibit 2021 docR400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:67

AMD1044\_0257019

ATI Ex. 2106 IPR2023-00922 Page 108 of 223

		T T			
Δň	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	31 of 48	
18. Expo	orting Arbitration				
1) F 2) F 3) F 4) M 5) M 6) Z 7) F 8) F	Position exports and memory Position exports and Z/Color Memory exports and Z/Color Memory exports and Memory Z/color exports and Z/color e Parameter exports and Z/Co Parameter exports and Z/Co	exports cannot be co-issued. vexports cannot be co-issued exports cannot be co-issued. exports cannot be co-issued. vexports cannot be co-issued.	ed.		
19. Exp	ort Types				
The export ty			cified using the destination addres	ss field in the	
19.1 Ver	tex Shading				
	44:47         - Empty           48:59         - 12 debug expoi           60         - export address           61         - Empty           62         - sprite size expo	ed?) ts to the frame buffer and inde rt (interpret as normal vertex e	export)		
19.2 <b>Pi</b>	kel Shading				
	0       - Color for buffer         1       - Color for buffer         2       - Color for buffer         3       - Color for buffer         3       - Color for buffer         4:7       - Empty         8       - Buffer 0 Color/f         9       - Buffer 1 Color/f         10       - Buffer 3 Color/f         11       - Buffer 3 Color/f         12:15       - Empty         16:31       - Empty (Reserv.         32:43       - 12 exports for r         44:47       - Empty         48:59       - 12 debug expoid         60       - export address         61:62       - Empty	1 2 3 Fog (primary) Fog Fog ed?) nultipass pixel shaders. rts (interpret as normal pixel e			
20. <u>Spe</u>	cial Interpolation r	nodes			
20.1 Rea	al time commands	3			
We are unab			r a command stream to write into i 16 interpolants). These will be map		
need to add t	inee toxizo memories (one	s for cach of three vehices x	re interpolatice): Theee this be thap		

ATI Ex. 2106 IPR2023-00922 Page 109 of 223

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 2001	4 September, 201511		32 of 48	

register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the reatime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME.

# 20.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen\_I0 register (in SQ) in conjunction with the SND\_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control registersregister. Here is a list of all the modes and how they interact together:

Gen\_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param\_Gen\_I0 disable, snd\_xy disable, no gen\_st - I0 = No modification Param\_Gen\_I0 disable, snd\_xy disable, gen\_st - I0 = No modification Param\_Gen\_I0 disable, snd\_xy enable, no gen\_st - I0 = No modification Param\_Gen\_I0 disable, snd\_xy enable, gen\_st - I0 = No modification Param\_Gen\_I0 enable, snd\_xy disable, no gen\_st - I0 = garbage, garbage, garbage, faceness Param\_Gen\_I0 enable, snd\_xy disable, gen\_st - I0 = garbage, garbage, s, t Param\_Gen\_I0 enable, snd\_xy enable, no gen\_st - I0 = screen x, screen y, garbage, faceness Param\_Gen\_I0 enable, snd\_xy enable, gen\_st - I0 = screen x, screen y, s, t

### 20.3 Auto generated counters

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1<sup>st</sup> pass data to memory and then use the count to retrieve the data on the 2<sup>nd</sup> pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN\_INDEX register. While there is only one count broadcast to the registers<u>GPRs</u>, the LSB are hardwired to specific values making the index different for all elements in the vector.

### 20.3.1 Vertex shaders

In the case of vertex shaders, if GEN\_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

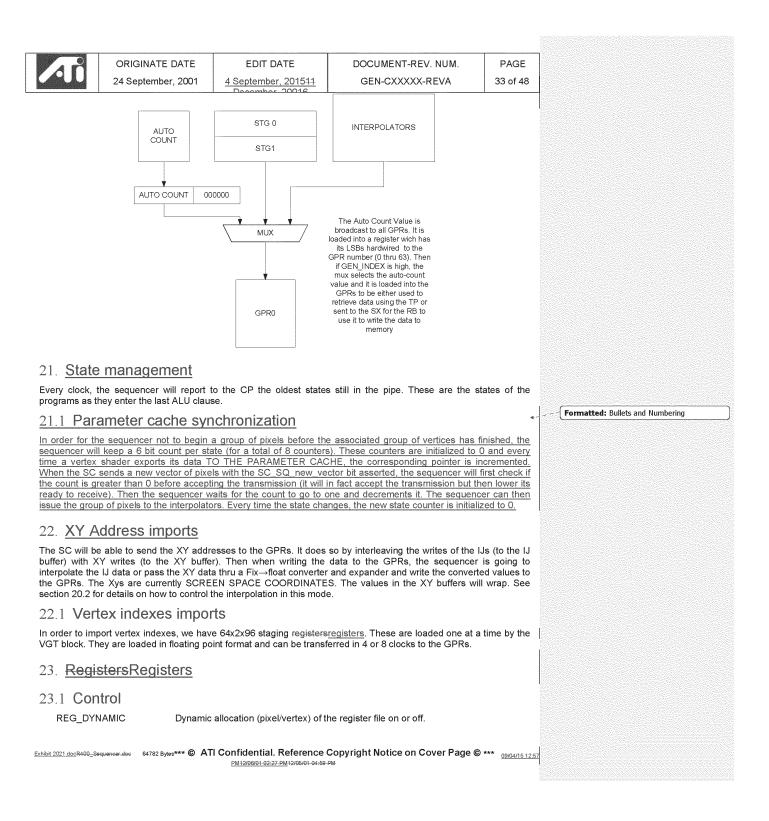
### 20.3.2 Pixel shaders

In the case of pixel shaders, if GEN\_INDEX is set, the data will be put in the x field of the 2<sup>nd</sup> register (I1.x).

Exhibit 2021 docR400\_Sequencer.doc 64782 Bytes\*\*\* ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09/04/15 12:57

AMD1044\_0257021

ATI Ex. 2106 IPR2023-00922 Page 110 of 223



ATI Ex. 2106 IPR2023-00922 Page 111 of 223

					·	
	<b>AP</b>	ORIGINATE E	DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	<u>AU</u>	24 September,	2001	4 September, 201511 December, 20016		34 of 48
	REG_SIZE	E_PIX	Size of on)		tion (minimal size when dynamic alloc	ation turned
	REG_SIZE	E_VTX		the register file's vertex p	ortion (minimal size when dynamic allo	ocation turned
		FION_POLICY DRE_ALLOC BE_VTX	policy c interlea		vertexes and pixels on store (RT always ends at vertex_ba	se and
	INST_BAS ONE_THR ONE_ALU	EAD	start po debug s	int for the pixel shader ins state register. Only allows	struction store one program at a time into the GPRs one ALU program at a time to be exe	cuted (instead
	INSTRUC	TION_INDEX		where the CP puts the ba ented on reads/writes)	ase address of the instruction writes a	nd type (auto-
	CONSTAN CONSTAN	IT_ADDR TION_INDEX	This is This is	where the CP puts consta where the CP puts the log	tual data going to the instruction memo int data (32 bits) jical constant address (9 bits) base address of the instruction writer	
	INSTRUC	TION_DATA_RT	Real Ti	me (auto-incremented on where the CP puts the act		
	CONSTAN	IT_DATA_RT IT_ADDR_RT IT_EO_RT	This is This is This is	where the CP puts consta where the CP puts the log the size of the space res ANT_EO_RT). The remain	nt data for Real Time (32 bits) gical constant address for Real Time (S served for real time in the constant st <del>apingre-mapping</del> table operates on t	ore (from 0 to
	EXPORT_	LATE	Control		e exporting position from clause 3. I	f set, position
2	3.2 Con	text				
	VS_FETC VS_ALU_ PS_FETC PS_ALU_ PS_BASE VS_CF_S PS_CF_S PS_SIZE VS_SIZE VS_SIZE PS_NUM_ VS_NUM_ PARAM_S	(07) H_{07} (07) ZE ZE REG REG	eight 8 eight 8 base po base po size of size of size of number number One 16	bit pointers to the location bit pointers to the location bit pointers to the location pointer for the pixel shader inter for the vertex shade the vertex shader (# of inst the pixel shader (and inst the pixel shader (and inst the vertex shader (and inst the vertex shader (and inst the vertex shader (and inst of registers <u>GPRs</u> to alloor bit register specifying wh	r in the instruction store structions in control program/2) ructions in control program/2) ructions)	is located is located is located
	PARAM_V	VRAP			d channels (xyzw)) do we do the cyl w	rapping
	PS_EXPC	RT_MODE	Oxxxx : 1xxxx : If norma	Normal mode Multipass mode	r many colors (0-4) and z is export z or r.	not
		RT_MASK RT_MODE RT	which c		s exporting (multipass only)	
	_COUNT_		(located	I in VS_EXPORT_COUN	e # of interpolated parameters exporte T_6) OR per clause in multipass mode (per cla	
Exhit	it 2021.docR400_Se	quencer.doc 64782 Byte	.s***© A	TI Confidential. Referen	nce Copyright Notice on Cover Page	© *** 09/04/15 12:5

ATI Ex. 2106 IPR2023-00922 Page 112 of 223

AR	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	35 of 48
CONST_E CONST_S CONST_S INST_PRI CF_BOOL	DEX Auto gen and R3 f BASE_VTX (9 bits) Logical E BASE_PIX (9 bits) Logical E SIZE_PIX (8 bits) Size of th SIZE_VTX (8 bits) Size of th ED_OPTIMIZE Turns on always e LEANS 256 bool P_COUNT 32x8 bit of P_START 32x8 bit of	herates an address from 0 to or vertex shaders Base address for the constar Base address for the constar he logical constant store for he logical constant store for the predicate bit optimization (xecuted).	nts of the Pixel shader pixel shaders vertex shaders on (if of, conditional_execute_predicat /e traverse the loop) ndex computation)	shaders
4. <u>DEB</u>	UG registersRegi	sters		
24.1 Cor	ntext			
DB_P DB_I DB_B DB_C _MOL DB_C	PROB_COUNT number of NST_COUNT instruction BREAK_ADDR break ad CLAUSE DE_ALU_{07} clause m CLAUSE	n counter for debug method dress for method number 2 node for debug method 2 (0:	ring the execution of the program 2	
25. Interf	faces			
25.1 Exte	ernal Interfaces			
			nits of the same name. For example, ormation to all SP instances.	if a bus is
	C to SP : IJ bus			4
		tion to the IJ fifos on the top re are 4 of these buses over t	of each shader pipe. At the same time he whole chip (SP0 thru 3)	-the-control
<u>5.1.2</u> 25.	1.1_SC to SQ : IJ Co	ontrol bus		4-
execute a sha control packel	ader program on the sent pi t is going to be ignored and	ixels. This information is ser d XY information is going to	ol the IJ fifos and all other information it over 2 clocks, if SENDXY is assert be sent on the IJ bus (for the quads mitive, all quads of a vector are fron	ed the next that where
Exhibit 2021.docR400_S	Requences.doc 64782 Bytes*** ⓒ AT	TI Confidential. Reference ( PM12/06/01-02:27 PM12/06/01-04:58-	Copyright Notice on Cover Page ©	*** <u>09/04/15 12:57</u>

ATI Ex. 2106 IPR2023-00922 Page 113 of 223

				PAGE	
Name	ATE EDIT E	DATE	R400 Sequencer Specification	PAGE	
	2001 <u>4 Septembe</u>		11	36 of 48	
	Direction	Bits	Description		
SC_SQ_q_wr_mask	SC→SQ	4	Quad Write mask left to right		
	SC→SQ	24	LOD correction per quad (6 bits per quad)		
	SC→SQ	2	Provoking vertex for flat shading		
	SC→SQ	11	P Store pointer for vertex 0		
SC_SQ_param_ptr1	SC→SQ	11	P Store pointer for vertex 1		
SC_SQ_param_ptr2	SC→SQ	11	P Store pointer for vertex 2		
SC_SQ_end_of_vect	SC→SQ	1	End of the vector		
SC_SQ_store_dealloc	SC→SQ	1	Deallocation token for the P Store		
SC_SQ_state	SC→SQ	3	State/constant pointer (6*3+3)		
SC_SQ_valid_pixel	SC→SQ	16	Valid bits for all pixels		
SC_SQ_null_prim	SC→SQ	1	Null Primitive (for PC deallocation purposes	)	
	SC→SQ	1	End Of the primitive		
SC_SQ_fbface	SC→SQ	1	Front face = 1, back face = 0		
SC_SQ_send_xy	SC→SQ	1	Sending XY information [XY information is sent on the next clock]	going to be	
SC_SQ_prim_type	SC→SQ	3	Real time command need to load tex alternate buffer. Line AA, Point AA and S their parameters from GEN_T an registersGPRs. 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA	Sprite reads	
SC_SQ_new_vector	SC→SQ	1	110 : Sprite This primitive comes from a new vector Make sure that the corresponding vertex finished before starting the group of pixels.		
SC SQ RTRn	SQ→SC	1	Stalls the PA in n clocks		
	SC→SQ	1	SC ready to send data		
25.1.3 <u>25.1.2</u> SQ to SF	י∶ Interpolator b	us		4-	
Name	Direction	Bits	Description		
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive 000 : Normal 011 : Real Time 100 : Line AA 101 : Point AA 110 : Sprite		
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading		
	SQ→SPx	1	Flat or gouraud shading		
	SQ→SPx	4	Wich parameter needs to be cylindrical wra	pped	
	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate		
	SQ→SPx	1	Swap the IJ/XY buffers at the end of the inte		
		1	Generate IO or not. This tells the interpol		
SQ_SPx_interp_buff_swap	SQ→SPx				
SQ_SPx_interp_buff_swap	SQ→SPx		use the parameter cache but rather overwit with interpolated 1 and 0. Overwrite if gen_I	rite the data	

ATI Ex. 2106 IPR2023-00922 Page 114 of 223

gy _ang ne n	ORIGINATI	E DATE	EDF	T DATE	Ξ	DOCUMENT-REV. NUM.	PAGE		
	24 Septemb	er, 2001	4 Septerr	nber, 20	01511				
1		Dissetia	Docom			A:	1	-	
Vame	As see as	Direction	-	Bits	Descrip	rpolated data			
SQ_SPx_data_	_туре	SQ→SPx	ι.	2		n rolated data			
					1x: Cou				
SQ SPx index	count	SQ→SPx		12?		ount, common for all shader pipes			
SQ_SPx_stage		SQ→SPx		1		register address			
						staging register			
					1: secor	nd staging register			
	<del>Q to SPx: Pi</del> 1.4 SQ to S					control hus	4		<u>a</u>
***********						erpolators) are all SYNCHRONIZED	) together.		
Name		Direction		Bits	Descrip	otion	]		
SQ_SPx_ptr0		SQ→SPx		9	Pointer				
SQ_SPx_ptr1		SQ→SPx		9	Pointer				
SQ_SPx_ptr2		SQ→SPx		9	Pointer				
SQ_SP0_read		SQ→SP0		4		nables for the 4 memories in the SP0			
SQ_SP1_read		SQ→SP1		4		nables for the 4 memories in the SP1			
SQ_SP2_read	mma .	SQ→SP2		4		hables for the 4 memories in the SP2			
SQ_SP3_read	_ena	SQ→SP3	5	4	Read er	nables for the 4 memories in the SP3			
<u>25.1.725.</u>	<u>1.5_</u> SQ to S	SX: Para	meter Ca	ache	Mux co	ontrol Bus		Formatted: Bullets and Numbering	g
Vame		Direction		Bits	Descrip				
		SQ→SXx		4		ntrol for PC (4 MSbs of Pointer)			
SQ_SXx_mux0 SQ_SXx_mux1	1	SQ→SXx	;	4	Mux cor	ntrol for PC (4 MSbs of Pointer)			
	1		;		Mux cor				
SQ_SXx_mux1 SQ_SXx_mux2	1	SQ→SXx SQ→SXx		4	Mux cor	ntrol for PC (4 MSbs of Pointer)		<b>Formatted:</b> Bullets and Numberin	g
SQ_SXx_mux1 SQ_SXx_mux2 25.1.8_SF	1 2 P to SX: Pai	SQ→SXx SQ→SXx rameter (	data	4	Mux cor Mux cor	ntrol for PC (4 MSbs of Pointer) ntrol for PC (4 MSbs of Pointer)	•	(Formatted: Bullets and Numbering	
5Q_SXx_mux1 SQ_SXx_mux2 25.1.8- <i>SF</i> 25.1.9- <i>S</i> >	1 2 2 to SX: Par < to Interpol	sq→sxx sq→sxx rameter d lators: Pa	data arameter	4 4 Cac	Mux cor Mux cor	ntrol for PC (4 MSbs of Pointer) ntrol for PC (4 MSbs of Pointer)			
3Q_SXx_mux1 3Q_SXx_mux2 25.1.8- <i>SF</i> 25.1.9- <i>S</i> >	1 2 P to SX: Pai	sq→sxx sq→sxx rameter d lators: Pa	data arameter	4 4 Cac	Mux cor Mux cor	ntrol for PC (4 MSbs of Pointer) ntrol for PC (4 MSbs of Pointer)			
50_5Xx_mux1 50_5Xx_mux2 25.1.8 <i>SF</i> 25.1.9 <i>S</i> 25.1.1025	1 2 2 to SX: Par < to Interpol	sq⊸sxx sq⊸sxx rameter c lators: Pa	data arameter aging Re	4 4 Cac	Mux cor Mux cor he Ret	ntrol for PC (4 MSbs of Pointer) ntrol for PC (4 MSbs of Pointer)	4		
30_SXx_mux1 30_SXx_mux2 25.1.8 <i>SF</i> 25.1.9 <i>SX</i> 25.1.10 <u>25</u> Vame	<u>2</u> 2 <i>to SX: Pai</i> 6 <i>to Interpol</i> 5 1.6 SQ to	sq→sxx sq→sxx rameter d lators: Pa	data arameter aging Re	4 4 Caci	Mux cor Mux cor he Ret er Data Descrip	ntrol for PC (4 MSbs of Pointer) ntrol for PC (4 MSbs of Pointer)			
30_SX_mux1 30_SXx_mux2 25.1.9_SA 25.1.1025 25.1.1025 Vame 30_SP0_vgt_v	1 2 2 7 to SX: Pai 5 to Interpol 5.1.6_SQ to vsisr_data	SQ→SXx SQ→SXx rameter C lators: Pa SP0: Sta Direction	data arameter aging Re	4 4 Cac giste Bits	Mux cor Mux cor he Ret er Data Descrip Pointers	ntrol for PC (4 MSbs of Pointer) ntrol for PC (4 MSbs of Pointer) <i>urn-bus</i>			
iq_SXx_mux1 iq_SXx_mux2 25.1.9 SF 25.1.1025 25.1.1025 Jame 30_SP0_vgt_v	1 2 2 2 2 2 2 3 4 5 5 1.6 SQ to 5 5 5 2 5 2 5 2 5 2 5 2 5 2 5 2 5 2 5	SQ→SXx SQ→SXx rameter C lators: Pa SP0: Sta Direction SQ→SP0	data arameter aging Re	4 4 <i>Cac</i> <i>giste</i> Bits 96	Mux cor Mux cor he Ret er Data Descrip Pointers	ntrol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) <i>urn-bus</i> sof indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p			
SQ_SXX_mux1 SQ_SXX_mux2 25.1.9-SA 25.1.1025 25.1.1025 Vame SQ_SP0_vgt SQ_SP0_vgt SQ_SP0_data SQ_SP0_data	1 2 2 2 2 2 2 2 3 3 3 5 5 5 7 6 5 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7	SQ→SXx SQ→SXx rameter C lators: Pa Direction SQ→SPO SQ→SPO SQ→SPO	data arameter aging Re	4 4 <i>Cac</i> <i>giste</i> Bits 96 1 1 96	Mux cor Mux cor he Ret pr Data Descrip O: Norrer O: Norrer Data is Pointers	ntrol for PC (4 MSbs of Pointer) ntrol for PC (4 MSbs of Pointer) <i>urn bus</i> s of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid s of indexes or HOS surface informatio	er vert		
30_SXX_mux1 30_SXX_mux2 25.1.8_SF 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25.1.1025 25	1 2 2 2 2 2 2 3 4 4 5 5 5 7 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	SQ→SXx SQ→SXx rameter C lators: Pa SP0: Sta Direction SQ→SP0 SQ→SP0 SQ→SP1 SQ→SP1	data arameter aging Re	4 4 <i>egiste</i> 96 1 1 96 1	Mux cor Mux cor he Ret er Data Descrip Pointers 0: Norm Data is Pointers 0: Norm	ntrol for PC (4 MSbs of Pointer) ntrol for PC (4 MSbs of Pointer) <i>urn-bus</i> of indexes or HOS surface information al 96 bits per vert 1: double 192 bits per valid s of indexes or HOS surface information al 96 bits per vert 1: double 192 bits per valid	er vert		
SQ_SXx_mux1           SQ_SXx_mux2           SQ_SXx_mux2           25.1.8           SQ_S1.9           SQ_SP0_vgt_v           SQ_SP0_vgt_v           SQ_SP0_vgt_v           SQ_SP0_vgt_v           SQ_SP1_vgt_v           SQ_SP1_vgt_v           SQ_SP1_vgt_v           SQ_SP1_vgt_v           SQ_SP1_vgt_v           SQ_SP1_vgt_v           SQ_SP1_vgt_v           SQ_SP1_vgt_v	1 2 2 2 2 2 2 2 3 4 5 5 5 5 7 6 5 7 6 7 7 7 7 7 7 7 7 7 7 7	SQ→SXx SQ→SXx rameter c lators: Pa SP0: Sta Direction SQ→SP0 SQ→SP0 SQ→SP1 SQ→SP1 SQ→SP1	data arameter aging Re	4 4 <i>egiste</i> 96 1 1 96 1	Mux cor Mux cor he Ret r Data Descrip Pointers 0: Norm Data is Pointers 0: Norm Data is	ntrol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) <i>urn-bus</i> of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid	er vert on er vert		
SQ_SXx_mux1 SQ_SXx_mux2 25.1.8 SF 25.1.9 S> 25.1.1025 Name SQ_SP0_vgt_v SQ_SP0_vgt_v SQ_SP0_vgt_v SQ_SP0_data SQ_SP1_vgt_v SQ_SP1_vgt_v SQ_SP1_vgt_v SQ_SP1_vgt_v SQ_SP1_vgt_v	1 2 2 2 2 2 2 2 4 4 5 1.6 SQ to 5 1.6 SQ to 5 SQ to 5 S 5 SQ to 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S 5 S S S S S S S S S S S S S	SQ→SXx SQ→SXx rameter C lators: Pa SP0: Sta Direction SQ→SP0 SQ→SP0 SQ→SP1 SQ→SP1 SQ→SP1 SQ→SP1	data arameter aging Re	4 4 <i>egiste</i> 96 1 1 96 1 1 96	Mux cor Mux cor Mux cor he Ret r Data Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers	ntrol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) <i>urn-bus</i> tion a of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid o f indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid s of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid s of indexes or HOS surface informatio	er vert		
SQ_SXx_mux1 SQ_SXx_mux2 25.1.8 <i>SF</i> 25.1.9 <i>S</i> 25.1.1025 Name SQ_SP0_vgt_v SQ_SP0_vgt_v SQ_SP0_vgt_v SQ_SP1_vgt SQ_SP1_vgt SQ_SP1_vgt SQ_SP1_vgt SQ_SP1_data SQ_SP2_vgt_v SQ_SP2_vgt_v	1 2 2 2 2 2 2 2 3 3 4 5 5 5 4 5 5 7 5 7 5 7 5 7 7 7 7 7 7 7	SQ→SXx SQ→SXx rameter C bators: Pa bSP0: Sta Direction SQ→SP0 SQ→SP0 SQ→SP1 SQ→SP1 SQ→SP1 SQ→SP2 SQ→SP2	data arameter aging Re	4 4 9 96 1 1 9 6 1 1 9 6 1	Mux cor Mux cor Mux cor he Ref. er Data Descrip Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm	ntrol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) <i>urn-bus</i> e of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid s of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid s of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid s of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p	er vert		
SQ_SXx_mux1           SQ_SXx_mux2           SQ_SXx_mux2           25.1.8           SQ_SX_102           SQ_SP0_vgt_v           SQ_SP0_vgt_v           SQ_SP0_vgt_v           SQ_SP0_vgt_v           SQ_SP0_vgt_v           SQ_SP0_vgt_v           SQ_SP0_vgt_v           SQ_SP1_vgt_v           SQ_SP1_vgt_v           SQ_SP1_vgt_v           SQ_SP2_vgt_v           SQ_SP2_vgt_v           SQ_SP2_vgt_v	1 2 2 2 2 2 2 2 2 2 2 2 2 2	SQ→SXx SQ→SXx rameter C b SP0: Sta Direction SQ→SP0 SQ→SP0 SQ→SP1 SQ→SP1 SQ→SP1 SQ→SP2 SQ→SP2 SQ→SP2	data arameter aging Re	4 4 96 1 1 96 1 1 96 1	Mux cor Mux cor Mux cor he Ref. er Data Descrip Pointers 0: Norm Data is Pointers 0: Norm Data is 0: Norm Data is	ntrol for PC (4 MSbs of Pointer) ntrol for PC (4 MSbs of Pointer) <i>urn bus</i> of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid	er vert		
3Q_SXx_mux1           3Q_SXx_mux2           3Q_SXx_mux2           25.1.8           25.1.9           3Q_SP1.9           3Q_SP0_vgt_v           3Q_SP0_vgt_v           3Q_SP0_vgt_v           3Q_SP1_vgt_v           3Q_SP1_vgt_v           3Q_SP1_vgt_v           3Q_SP1_vgt_v           3Q_SP1_vgt_v           3Q_SP2_vgt_v           3Q_SP2_vgt_v           3Q_SP2_vgt_v           3Q_SP2_vgt_v           3Q_SP2_vgt_v           3Q_SP2_vgt_v           3Q_SP2_vgt_v           3Q_SP3_vgt_v	1 2 2 2 2 2 2 2 2 2 4 4 5 1.6 SQ to 5 1.6 SQ to 5 4 5 4 4 4 4 5 4 4 4 4 5 4 4 4 5 4 4 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5	SQ→SXx SQ→SXx rameter C lators: Pa SP0: Sta SQ→SP0 SQ→SP0 SQ→SP0 SQ→SP1 SQ→SP1 SQ→SP1 SQ→SP2 SQ→SP2 SQ→SP2	data arameter aging Re	4 4 96 1 1 96 1 1 1 96 1 1 1 96	Mux cor Mux cor Mux cor he Ret. er Data Descrip Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm	ntrol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) urn bus of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p	ver vert		
SQ_SXx_mux1           SQ_SXx_mux2           SQ_SXx_mux2           25.1.8           SQ_SX_102           SQ_SP0_vgt_           SQ_SP0_vgt_           SQ_SP0_vgt_           SQ_SP0_vgt_           SQ_SP0_vgt_           SQ_SP1_data           SQ_SP1_data           SQ_SP2_vgt_           SQ_SP2_vgt_           SQ_SP2_vgt_           SQ_SP2_vgt_           SQ_SP3_vgt_           SQ_SP3_vgt_	1 2 2 2 2 2 2 2 4 4 5 1.6 SQ to 5 1.6 SQ to 5 1.6 SQ to 5 4 2 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4	SQ→SXx SQ→SXx rameter C b SP0: Sta Direction SQ→SP0 SQ→SP0 SQ→SP1 SQ→SP1 SQ→SP1 SQ→SP2 SQ→SP2 SQ→SP2 SQ→SP3 SQ→SP3	data arameter aging Re	4 4 9 9 1 1 1 9 6 1 1 1 9 6 1 1 1 9 6 1	Mux cor Mux cor Mux cor he Ret r Data Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm	ntrol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) urn-bus of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p	ver vert		
SQ_SXx_mux1           SQ_SXx_mux2           SQ_SXx_mux2           25.1.8           SQ_SX_102           SQ_SP0_vgt_           SQ_SP0_vgt_           SQ_SP0_vgt_           SQ_SP0_vgt_           SQ_SP0_vgt_           SQ_SP1_data           SQ_SP1_data           SQ_SP2_vgt_           SQ_SP2_vgt_           SQ_SP2_vgt_           SQ_SP2_vgt_           SQ_SP3_vgt_           SQ_SP3_vgt_	1 2 2 2 2 2 2 2 4 4 5 1.6 SQ to 5 1.6 SQ to 5 1.6 SQ to 5 4 2 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4	SQ→SXx SQ→SXx rameter C lators: Pa SP0: Sta SQ→SP0 SQ→SP0 SQ→SP0 SQ→SP1 SQ→SP1 SQ→SP1 SQ→SP2 SQ→SP2 SQ→SP2	data arameter aging Re	4 4 96 1 1 96 1 1 1 96 1 1 1 96	Mux cor Mux cor Mux cor he Ret. er Data Descrip Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm	ntrol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) urn-bus of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p	ver vert	(Formatted: Bullets and Numberin	9
SQ_SXx_mux1 SQ_SXx_mux2 25.1.8 SF 25.1.9 S> 25.1.1025 Name SQ_SP0_vgt_V SQ_SP0_vgt_V SQ_SP0_vgt_V SQ_SP0_vgt_V SQ_SP1_vgt_V SQ_SP1_vgt_V SQ_SP1_vgt_V SQ_SP2_vgt_V SQ_SP2_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V	to SX: Pariprice to SX: Pariprice to SX: Pariprice to SQ to S.1.6 SQ to SSI data vsisr_double valid vsisr_data vsisr_data vsisr_data vsisr_data vsisr_data vsisr_data vsisr_data SSI data SSI d	SQ→SXx           SQ→SXx           rameter (           /ators: Pa           >SP0: Sta           Direction           SQ→SP0           SQ→SP0           SQ→SP1           SQ→SP1           SQ→SP2           SQ→SP2           SQ→SP3           SQ→SP3	data arameter aging Re	4 4 4 9 1 1 1 9 6 1 1 1 9 6 1 1 1 9 6 1 1 1 9 6 1 1	Mux cor Mux cor Mux cor he Ret r Data Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm	ntrol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) urn-bus of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p	ver vert		9
SQ_SXx_mux1 SQ_SXx_mux2 25.1.8 SF 25.1.9 S> 25.1.1025 Name SQ_SP0_vgt_V SQ_SP0_vgt_V SQ_SP0_vgt_V SQ_SP0_vgt_V SQ_SP1_vgt_V SQ_SP1_vgt_V SQ_SP1_vgt_V SQ_SP2_vgt_V SQ_SP2_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V SQ_SP3_vgt_V	2 2 2 2 2 2 2 2 2 2 2 2 2 2	SQ→SXx           SQ→SXx           rameter (           /ators: Pa           >SP0: Sta           Direction           SQ→SP0           SQ→SP0           SQ→SP1           SQ→SP1           SQ→SP2           SQ→SP2           SQ→SP3           SQ→SP3	data arameter aging Re	4 4 4 9 1 1 1 9 6 1 1 1 9 6 1 1 1 9 6 1 1 1 9 6 1 1	Mux cor Mux cor Mux cor he Ret r Data Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm Data is Pointers 0: Norm	ntrol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) urn-bus of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p	ver vert	(Formatted: Bullets and Numberin	9
SQ_SXx_mux1 SQ_SXx_mux2 25.1.8 <i>SF</i> 25.1.9 <i>S</i> 25.1.1025 25.1.1025 25.1.1025 25.1.1025 SQ_SP0_vgt_v SQ_SP0_vgt_v SQ_SP0_vgt_v SQ_SP1_vgt SQ_SP1_vgt_v SQ_SP1_vgt_v SQ_SP2_vgt_v SQ_SP2_vgt_v SQ_SP3_vgt_v SQ_SP3_vgt_v SQ_SP3_vgt_v SQ_SP3_data 25.1.1125 25.1.11.125	2 2 2 2 2 2 2 2 2 2 2 2 2 2	SQ→SXx SQ→SXx rameter C P P P P P P P P P P	data arameter aging Re b b b b b b b b b b b b b b b b b b b	4 4 4 9 1 1 1 9 6 1 1 1 9 6 1 1 1 1 9 6 1 1 1 1	Mux cor Mux cor Mux cor Anter Data Pointers O: Norm Data is Pointers O: Norm	ntrol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) trol for PC (4 MSbs of Pointer) urn-bus of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p valid of indexes or HOS surface informatio al 96 bits per vert 1: double 192 bits p	er vert	(Formatted: Bullets and Numberin	9

ATI Ex. 2106 IPR2023-00922 Page 115 of 223

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 2001	4 September, 201511		38 of 48	
VSISRs (via t	he Shader Sequencer)	in full, 32-bit floating-po	pint format. The VGT can transmit u	p to six 32-bit	

floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Name	Bits	Description
PA_SQ_vgt_vsisr_data	96	Pointers of indexes or HOS surface information
PA_SQ_vgt_vsisr_double	1	0: Normal 96 bits per vert 1: double 192 bits per vert
PA_SQ_vgt_end_of_vector	1	Indicates the last VSISR data set for the current process vector (for double vector
		data, "end_of_vector" is set on the second vector)
PA_SQ_vgt_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when
		"PA_SQ_vgt_end_of_vector" is high.
PA_SQ_vgt_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR
		interface handshaking)
SQ_PA_vgt_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface
		handshaking)

25.1.11.225.1.7.2 Interface Diagrams

----- (Formatted: Bullets and Numbering

Exhibit 2021 docR400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:57

AMD1044\_0257027

ATI Ex. 2106 IPR2023-00922 Page 116 of 223

# PROTECTIVE ORDER MATERIAL

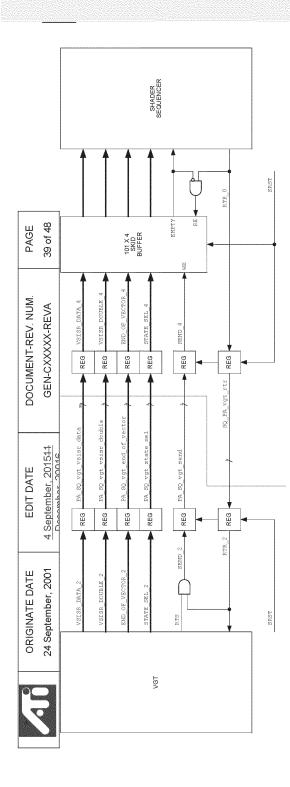
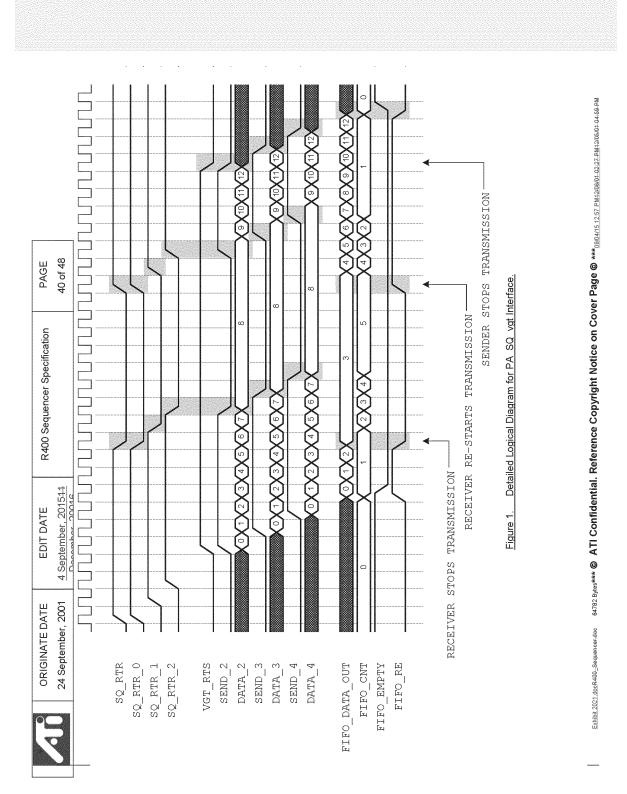


Exhibit 2021 doc R400\_Sequencer doc 64782 Bytes 444 © ATI Confidential. Reference Copyright Notice on Cover Page © 444 0004115 1257 PM1206001 02227 PM1206501 0458 PM

AMD1044\_0257028

ATI Ex. 2106 IPR2023-00922 Page 117 of 223





	INATE DATE	EDIT DA	TE	DOCUMENT-REV. NUM.	PAGE	
24 Se	ptember, 2001	4 September,	201511	GEN-CXXXXX-REVA	41 of 48	
2 <u>5.1.12</u> 25.1.8 s	Q to CP: Stat	e report			4-	
lame	Direction	,	s Descrij	ation		
SQ_CP_vrtx_state	SEQ→CF			/ertex state still in the pipe		
SQ_CP_pix_state	SEQ→CF	2 3		pixel state still in the pipe		
25.1.13 SP to S	<del>X : Pixel/Verte</del>	əx writə to S	\$X		*-	
2 <u>5.1.1425.1.9</u> S	Q to SX: Con	trol bus			4-	
Name	Direction		s Descrij	otion		
SQ_SXx_exp_Pixel	SQ→SXx	1	1: Pixel			
SQ_SXx_exp_start	SQ→SXx	1	0: Verte Raised	to indicate that the SQ is starting an	export	
SQ_SXx_exp_Clause	SQ→SXx			number, which is needed for vertex		
SQ_SXx_exp_State	SQ→SXx	3		D, which is needed for vertex clauses		
These fields are sent sy	nchronously with 9	Pernort data	lescribed in	SP0→SX0 interface		
ISSUE: Where are the		. onpon uala, (				
2 <u>5.1.15</u> 25.1.10	ev to en vo	utnut fila ca	ntrol		*-	
		· · · · · · · · · · · · · · · · · · ·				
Name	rdv SXx→SQ			otion by SX0 to indicate that the following	two fields	
SXx_SQ_Export_count		l l		he result of the most recent export	I WAO IIGIUS	
SXx_SQ_Export_Position			Specifi	es whether there is room for another		
SXx_SQ_Export_Buffer	SXx→SQ	7		es the space available in the output I	buffers.	
				rs are full its available (32-bits for each of the i	64	
				n a clause)	~ .	
			64: 128 64 pixe	K-bits available (16 128-bit entries f ls)	or each of	
				RESERVED		
					,	(
25.1.16 Shader I	Engine to Fete	h Unit Bus			4	Formatted: Bullets and Numbering
			nit avanu ch	ck. These are sourced from a differer	t nivel within	
each of the sub-engi	nes repeating every	4 clocks. The re	gister file in	lex to read must procede the data by i	2 clocks. The	
Read address assoc 3 clocks after the Ins		ust be sent 1 ck	ock after the	Instruction Start signal is sent, so that	t data is read	
a crocka arter the INS	austun otan.					
				every clock. These are directed to a		
of the sub-engines re associated with the C				o write must accompany the data. Da Start signal is sent.	ita and index	
				~		
25 + 1725 + 11	Sequencer to	Fetch Unit	busSO	to TP: Control bus	*-	Formatted: Bullets and Numbering
	•				data in the	
				clause it is now working and if the the fetch counters for the reservation		
The sequencer also pro	vides the instruction			ch to execute and the address in the		
vhere to write the fetch	return data.					
Name	Direction	Bit	s Descrij	otion		
INGINE	TPx→ SC		Data re			
TPx_SQ_data_rdy	11 × / 00	• • • •		-		
		I	Reference	Copyright Notice on Cover Page @	) *** 09/04/15 12:57	

ATI Ex. 2106 IPR2023-00922 Page 119 of 223

	<u>A</u>	ORIGINATE D	ATE	EDIT D	ATE		R400 Sequencer Specification	PAGE
4	400	24 September,	2001	4 Septembe				42 of 48
	TPx_SQ_c	lause num	TPx→		3		use number	
	SQ_TPx_c	onst	SQ→T	Px	64	Fet	ch state sent over 4 clocks	
	SQ_TPx_i	nstuct	SQ→T	Px	24	Fet	ch instruction sent over 4 clocks	
	SQ_TPx_e	end_of_clause	SQ→T	Px	1	Las	t instruction of the clause	
	SQ_TPx_p	hase	SQ→T	Px	2	Wri	te phase signal	
	SQ_TPO_I	od_correct	SQ→T	P0	6	LOI	O correct 3 bits per comp 2 components	per quad
	SQ_TP0_p	omask	SQ→T	P0	4	Pixe	el mask 1 bit per pixel	
	SQ_TP1_k	od_correct	SQ→T	P1	6	LOI	D correct 3 bits per comp 2 components	per quad
	SQ_TP1_p	omask	SQ→T	P1	4	Pix	el mask 1 bit per pixel	
	SQ_TP2_I	od_correct	SQ→T	P2	6	LO	D correct 3 bits per comp 2 components	per quad
	SQ_TP2_p	omask	SQ→T	P2	4	Pixe	el mask 1 bit per pixel	
	SQ_TP3_k	od_correct	SQ→T	P3	6	LO	D correct 3 bits per comp 2 components	per quad
	SQ_TP3_p	omask	SQ→T	P3	4	Pixe	el mask 1 bit per pixel	
	SQ_TPx_c	lause_num	SQ→T	Px	3	Cla	use number	
	SQ TPx v	vrite gpr index	SQ->TI	⊃ <sub>X</sub>	7	Ind	ex into Register file for write of returned	Fetch Data

### 25.1.12 TP to SQ: Texture stall

The TP sends this signal to the SQ when its input buffer is full. The SQ is going to send it to the SP X clocks after reception (maximum of 3 clocks of pipeline delay).

ľ	lame	Direction	Bits	Description
1	P_SQ_fetch_stall	<u>TP→ SQ</u>	1	Do not send more texture request if asserted

# 25.1.13 SQ to SP: Texture stall

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	<u>SQ→SPx</u>	1	Do not send more texture request if asserted

# 25.1.1825.1.14 SequencerQ to SP: GPR and Parameter cache control

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_re_addr	SQ→SPx	1	Read Enable
SQ_SPx_gpr_we_addr	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux
SQ_SPx_gpr_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_gpr_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_gpr_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_gpr_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_gpr_pixel_mask	SQ→SP3	4	The pixel mask
SQ SPx pc we addr	SQ→SPx	1	Write Enable for the parameter caches

Exhibit 2021 doc R400\_Sequencer.doc 64782 Bytes\*\*\* S ATI Confidential. Reference Copyright Notice on Cover Page S \*\*\* PM12/06/01-02:27 PM12/05/01-04:59 PM

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0257031

ATI Ex. 2106 IPR2023-00922 Page 120 of 223

	RIGINATE DATE	EDIT D	ATE	DC	CUMENT	REV. NUM.	PAGE	
24 5	September, 2001	4 Septembe	r, 201511	C	SEN-CXXX	XX-REVA	43 of 48	E
1.19 <u>25.1.15</u>	Sequencer S	<u>SQ to SPx:</u> I	Instructio	ns	-		4-	Formatted: Bullets and Numbering
Name Name			DirectionD	rection	BitsBits	DescriptionDes	cription	
Q SPx instruct	startSQ_SPx_instruc	xt_start	<u>SQ→SPx</u> S0	Q→SPx	<u>1</u> 1	Instruction st start	artInstruction	
SQ SP instructS4	Q_SP_instruct		<u>SQ→SPx</u> S0	2→SPx	2020	Instruction ser		
						clocksInstruction	i sent over 4	
SQ SPx stallSQ	SPx_stall		<u>SQ</u> →SPxS(	⊋SP×	14	Stall signalStall-	signal	
Q SPx export	countSQ_SPx_Shad	er_Count	<u>SQ→SPx</u> S0	⊋→SPx	33	Each set of fo		
						vectors is exp eight clocks.		
						specifies where	the SP is in	
						that sequence. four pixels or		
						exported over o		
						This field spec	sifies where	
SO SPY event	lastSQ_SPx_Shader	Last	SQ→SPx <del>S</del> Q	J_SPV	14	the SP is in that Asserted on the		
	I WE ARE ARE AND	and here here he		m indition	<u> </u>	count of the la	st export of	
						the clauseAsse first_shader_co		
						last export of th		
SQ_SP0_export_	pvalidSQ_SP0_Sha	der_PixelValid	SQ→SP0S	2→SP0	44	Result of pixe	I kill in the	
						shader pipe, who output for all p		
						(depth and		
						buffers). 4x4 I	because 16	
						pixels are cor clockResult of		
						the shader p		
						must-be-output		
						exports (depth buffers). 4x4 I		
						pixels are coi		
CO CDO overant	wvalidSQ_SP0_Sha	dor Mard (ali	SQ→SP0S	0.000	22	clock Specifies whet	har to write	
1	wvaliu302_3F0_3Ha	ueivvoiu vaii	00-000	a(010	has this	low and/or high		
						of the 64-bit		
						from each of the or vector	ne 16 pixels	
						whether to write	e low and/or	
						high 32-bit wor bit export data f		
						the 16 pixels or		
Q_SP1_export	_pvalidSQ_SP1_Sha	der_PixelValid	SQ→SP1S	2→SP1	44	Result of pixe	I kill in the	
						shader pipe, who output for all p		
						(depth and		
						buffers). 4x4 I		
						pixels are con clockResult of		
						the shader p	pipe, which	
						must be output exports (depth-		
						buffers). 4x4		
						pixels are col		
SQ SP1			SQ→SP1S	2→SP1	22	clock Specifies whet	her to write	
r -ws. 3671 3				~ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	des dan non	speames witch	iner to write	

AMD1044\_0257032

ATI Ex. 2106 IPR2023-00922 Page 121 of 223

ORIGINATE DATE					
	EDIT DAT	E R40	0 Sequencer Specific	ation PAGE	
24 September, 2001	4 September, 2			44 of 48	
xport_wvalidSQ_SP1_Shader_Wo Q_SP2_export_pvalidSQ_SP2_S	ordValid	<u>SQ</u> →SP2SQ→S	of th from or wheth high- bit ex the 1 \$P2 44 Resu shad outpu	nd/or high 32-bit word e 64-bit export data each of the 16 pixels vectorsSpecifies her to write low and/or 32-bit word of the 64- port data from each of 6 pixels or vectors It of pixel kill in the er pipe, which must be it for all pixel exports	
			buffe pixels clock the must expoi buffe pixels clock		
Q_SP2_ xport_wvalidSQ_SP2_Shader_Wo		<u>SQ→SP2</u> SQ→S	of th from or wheth high bit ex the 1	ifies whether to write nd/or high 32-bit word e 64-bit export data each of the 16 pixels vectorsSpecifies her to write low and/or 32-bit word of the 64- port data from each of 6 pixels or vectors	
<u>Q_SP3_export_pvalid</u> SQ_SP3_S	hader_PixelValid	<u>SQ</u> →SP3SQ→S	- shade outpu (dept buffe pixels clock the must experience buffe	It of pixel kill in the er pipe, which must be it for all pixel exports h and all color rs). 4x4 because 16 6 are computed per Result of pixel kill in shader pipe, which be output for all pixel rts (depth and all color rs). 4x4 because 16 b are computed per	
<u>Q_SP3_</u> xport_wvalidSQ_SP3_Shader_Wo	ordValid	<u>SQ→SP3</u> SQ→S	3P3 <u>2</u> 2 <u>Spec</u> low a of th from or wheth high	ifies whether to write nd/or high 32-bit word e 64-bit export data each of the 16 pixels vectorsSpecifies her to write low and/or 32-bit word of the 64- port data from each of	
				6 pixels or vectors	
<del>.20</del> 25.1.16 SP to Segu		mada ut - d-t-	the 1	6 pixels or vectors	<b>Formatted:</b> Bullets and Numberi

Name	Direction	Bits	Description	٦
SP0_SQ_const_addr	SP0→SQ	36	Constant address load to the sequencer	٦
SP0_SQ_valid	SP0→SQ	1	Data valid	٦
SP1_SQ_const_addr	SP1→SQ	36	Constant address load to the sequencer	

Exhibit 2021 doc R400\_Sequencer.doc 64782 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257033

ATI Ex. 2106 IPR2023-00922 Page 122 of 223

R	ORIGINATE	E DATE	EDIT	DATE	=	DOCUMENT-REV. NUM.	PAGE	
	24 Septemb	er, 2001	4 Septemb			GEN-CXXXXX-REVA	45 of 48	
SP1_SQ_V	/alid	SP1→SC	Decembri	<u></u>	Data va	lid		
	const_addr	SP2→SC		36		nt address load to the sequencer		
SP2_SQ_	/alid const_addr	SP2→SC SP3→SC		1 36	Data va	lid nt address load to the sequencer		
SP3 SQ		SP3→SC		1	Data va			
	5. <u>1.17</u> Soqi						<b></b>	> Formatted: Bullets and Numbering
Name SQ_SPx_c	onstant	Direction SQ→SP×		Bits 128	Descri	otion nt broadcast		
	5.1.18_SP0							Formatted: Bullets and Numbering
Name		Direction		Bits	Descri			
SP0_SQ_I		SP0→SC		4	Kill vec			
SP1_SQ_I		SP1→SC SP2→SC		4 4	Kill vec	tor load		
SP3 SQ 1		SP2→3C		4	Kill vec			
	5.1.19_SQ t				1			Formatted: Bullets and Numbering
Name		Directior	1	Bits	Descri	otion		
SQ_RBB_	~	SQ→CP		1	Read S			
SQ_RBB_	*	SQ→CP		32	Read [			
SQ_RBB		SQ→CP		1	Option			
SQ_RBB	И_rtr	SQ→CP		1	Real-I	ime (Optional)		(
<u>25.1.2425</u>	5.1.20 CP to	o SQ: RE	3BM bus				4~	Formatted: Bullets and Numbering
Name		Direction		Bits	Descri			
rbbm_we		CP→SQ		1	Write E			
rbbm_a		CP→SQ CP→SQ		18 32		s Upper Extent is TBD		
rbbm_wd rbbm_be		CP→SQ CP→SQ		52 4	Data Byte E	nahlae		
rbbm_be		CP→SQ		1	Read			
rbb rs0		CP→SQ		1		Return Strobe 0		
rbb_rs1		CP→SQ		1		Return Strobe 1		
rbb_rd0		CP→SQ		32	Read [			
rbb_rd1		CP→SQ		32	Read I	Data O		
RBBM_SC	Q_soft_reset	CP→SQ		1	Soft Re	eset		
26. <u>Exan</u>	nples of pr	ogram	executio	<u>ns</u>				
26.1.1 Se	equencer Co	ontrol of	a Vector c	of Ve	ertices			
<ul> <li>state  </li> <li>space</li> <li>also b</li> <li>shade</li> <li>The ventor of the second sec</li></ul>	pointer as well a was allocated in pefore the vecto er program (usi ertex program is e SEQ then acce equencers by the	s tag into po n the positio or is sent to ng the MH? assumed to esses the IS e RBBM whe	esition cache i n cache for tra the RE, the b be loaded w b base for this en the CP is c	s sen ansfo CP h hen v shad lone l	It along v rmed po as loade ve receiv ler using oading th	sition before the vector was sent of the global instruction store with e the vertex vector. the local state pointer (provided to a ne program)	h the vertex	
<ul><li>at this</li><li>the ar</li></ul>	point the vector	is removed to select a	from the Ver vector to be t	ex Fl	FO	sically the Vertex FIFO always has the parameter cache is full unless th		
			-			Copyright Notice on Cover Page @		

ATI Ex. 2106 IPR2023-00922 Page 123 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201511		46 of 48

- 3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
- the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
  - SEQ will not send vertex data until space in the register file has been allocated
- 4. SEQ sends the vector to the SP register file over the RE\_SP interface (which has a bandwidth of 2048 bits/cycle)
  - the 64 vertex indices are sent to the 64 register files over 4 cycles
  - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
  - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
  - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
  - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
  - the index is written to the least significant 32 bits (floating point format?) (what about compound indices)
    of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
- SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
  - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
- TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   TSM0 was first selected by the TSM arbiter before it could start
- 7. all instructions of fetch clause 0 are issued by TSM0
- the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
  - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
  - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
- ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU
  clause 0 from the global instruction store
- 10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
- 11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
  - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
    - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
      - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
    - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
    - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
    - parameter data is sent to the Parameter Cache over a dedicated bus
    - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
    - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full
- 12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

### 26.1.2 Sequencer Control of a Vector of Pixels

- 1. As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP
  - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
- 2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
  - the state pointer and the LOD correction bits are also placed in the Pixel FIFO
  - · the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle

Exhibit 2021.docR400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09004/15.12:57

AMD1044\_0257035

ATI Ex. 2106 IPR2023-00922 Page 124 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201511	GEN-CXXXXX-REVA	47 of 48	

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

### 26.1.3 Notes

- 14. The state machines and arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.
- 15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer.

16. Waterfalling still needs to be specked out.

### 27. Open issues

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

1)Let the compiler handle the case and put those instructions in a fetch clause so we can use the bandwidth there to operate. This requires a significant amount of temporary storage in the register store.

Exhibit 2021.docR400\_Sequencer.doc 64782 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12

•- -<	<b>Formatted:</b> Bullets and Numbering	$\square$
•	<b>Formatted:</b> Bullets and Numbering	
:57		

AMD1044\_0257036

ATI Ex. 2106 IPR2023-00922 Page 125 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201511		48 of 48

2)Waterfall down the pipe allowing only at a given time the vertices having the same constants to operate in parallel. This might in the worst case slow us down by a factor of 16.

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

Saving power?

Exhibit 2021 doc R400\_Sequencer.doc 64782 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257037

ATI Ex. 2106 IPR2023-00922 Page 126 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	1 of 47
Author:	Laurent Lefebvre	L		
Issue To:		Сору №		
	R400 S	equencer Spe	ecification	
		SQ		
		Version 1. <u>6</u> 5		
req	uired capabilities and expe	cted uses of the block. It a		
Document L	24 September, 2001       4 September, 20157       GEN-CXXXXX-REVA       1 of 4         hor:       Laurent Lefebvre       Copy No:         re To:       Copy No:         R400 Sequencer Specification         SQ         Version 1.65         rview: This is an architectural specification for the R400 Sequencer block (SEQ). It provides an overview of 1 required capabilities and expected uses of the block. It also describes the block interfaces, internal state diagrams.         JTOMATICALLY UPDATED FIELDS:         Duront Location:         C:/perforce/r400/doc_lib/design/blocks/sq/R400_Sequencer.doc         JTOMATICALLY UPDATED FIELDS:         Duront Location:         C:/perforce/r400/doc_lib/design/blocks/sq/R400_Sequencer.doc         JTOMATICALLY UPDATED FIELDS:         Duront Location:         APPROVALS         Name/Dept         Signature/Date			
Remarks:				
COULD E	BE SUBSTANTIALLY	DETRIMENTAread	ptrL TO THE INTEREST	OF ATI
work created ir unpublished w confidential, pr	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	provide notice that ATI owns a cop publication has occurred. This w of this document may be used, re	oyright in this ork contains
Exhibit 2022.docR400_4	Sequencer.doc 67798 Bytes*** © AT	1 Confidential. Reference ( PM01/04/02 02:67 PM12/06/01 02:27 (		*** 09/04/15 12:57

ATI 2022 LG v. ATI IPR2015-00325

AMD1044\_0257038

ATI Ex. 2106 IPR2023-00922 Page 127 of 223

OR	IGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 5	September, 2001	4 September, 20157		2 of 47
		Januany 200211		1
Table Of Conte	<u>ents</u>			
1. OVERVIEW	1			
1.2 Data Flow g	Jraph			
,				
	,			
-				
	list Block			<u>19</u> 18
5.2.3 De-all	ocate Block			<u>19</u> 18
5.2.4 Opera	tion of Incremer	ntal model		
5.3 Constant St	ore Indexing			<u>21</u> 19
5.4 Real Time (	Commands	··· <i>·</i> ·····		<u>21</u> 19
			\$	
	, <b>v</b>		; (12)	and the second se
				advariant/
				bledetwoetch
				automation and a second automation and a second automation and a second automation and a second automation automation and a second automation auto
13. CONTENT	OF THE RESER	<b>RVATION STATION F</b>	IFOS	
14. THE OUTP	UT FILE	*****		
15. IJ FORMAT	******	*****		<u>29</u> 27
15.1 Interpolat	ion of constant a	attributes		<u>30</u> 28
				and a second a second s
				NAMES OF A DESCRIPTION OF A DESCRIPTION OF A DESCRIPTIONO
				504560000000000000000000000000000000000
				040040000000000000000
	<u> </u>			
I MU. OF LOIML II	MENTOLATIO	1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1		

PM01/04/02 02:57 PM12/06/01 02:27 PM

AMD1044\_0257039

ATI Ex. 2106 IPR2023-00922 Page 128 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	3 of 47
	-			2.2
20.3.1				
				22
			********	
23. REGIS	STERS	* * * * * * * * * * * * * * * * * * * *		<u>3533</u> 34
			*****	000000000000000000000000000000000000000
25.1.1				
25.1.2				
25.1.3	SQ to SP: GPR Input	Mux select		<u>3835</u> 34
25.1.4	SQ to SP: Parameter	Cache Read control bu	s	<u>3836</u> 35
25.1.5	SQ to SX: Parameter	Cache Mux control Bus	;	<u>3836</u> 35
25.1.6	SQ to SP: Staging Re	egister Data		<u>3836</u> 35
25.1.7	PA to SQ : Vertex int	erface		<u>3836</u> 35
25.1.8	SQ to CP: State repo	rt		<u>4240</u> 39
25.1.9	SQ to SX: Control bu	s		<u>4240</u> 39
25.1.10	SX to SQ : Output file	e control		<u>4240</u> 39
25.1.11	SQ to TP: Control bu	s		<u>4240</u> 39
25.1.12	TP to SQ: Texture sta	all		<u>4341</u> 40
25.1.13	SQ to SP: Texture sta	all		<u>4341</u> 40
25.1.14	SQ to SP: GPR and I	Parameter cache contro		<u>4341</u> 40
25.1.15	SQ to SPx: Instructio	ns		<u>4442</u> 41
25.1.16	SP to SQ: Constant a	address load		<u>4442</u> 41
25.1.17	SQ to SPx: constant	broadcast		<u>4442</u> 41
25.1.18	SP0 to SQ: Kill vecto	r load		<u>4543</u> 42
25.1.19	SQ to CP: RBBM bus	s		<u>4543</u> 42
	IPLES OF PROGRAM	EXECUTIONS		<u>4543</u> 42
26.1.1	Sequencer Control of	a Vector of Vertices		<u>4543</u> 42
26.1.2	,			
26.1.3	Notes			<u>4745</u> 44
Exhibit 2022.docR400_Se	ациепсег.doc 67798 Bytes*** ⓒ АТ		Copyright Notice on Cover Page	© *** 09/04/15 12:57
		PM01/04/02-02:57-PM12/06/01-02:27-P	4A	

ATI Ex. 2106 IPR2023-00922 Page 129 of 223

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20157		4 of 47
27. OPEN	ISSUES			474544

Exhibit 2022.docR400\_Sequencer.doc 67798 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 00/04/15.12:57

AMD1044\_0257041

ATI Ex. 2106 IPR2023-00922 Page 130 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	5 of 47
Revision	Changes:			
		· · · ·		
Date: May 7, 2	rent Lefebvre) 2001	First dra	iπ.	
Rev 0.2 (Laur Date : July 9,			d the interfaces to reflect the change led some details in the arbitration sec	
Rev 0.3 (Laur	ent Lefebvre)		ed the Sequencer spec after the me	
Date : August			3, 2001.	
Rev 0.4 (Laur Date : August			the dynamic allocation method for an example (written in part by Vic	
-		flow of p	pixels/vertices in the sequencer.	,
Rev 0.5 (Laur		Added t	iming diagrams (Vic)	
Date : Septen Rev 0.6 (Laur		Change	d the spec to reflect the new	R400
Date : Septen	1ber 24, 2001	architec	ture. Added interfaces.	
Rev 0.7 (Laur Date : Octobe			constant store management, ins anagement, control flow management	
Date . Octobe	1 0, 2001		pendant predication.	
Rev 0.8 (Laur			d the control flow method to be	
Date : Octobe Rev 0.9 (Laur			Also updated the external interfaces. rated changes made in the 10/18/01	
Date : Octobe		flow me	eeting. Added a NOP instruction, re	emoved
			nditional_execute_or_jump. Added	debug
Rev 1.0 (Laur	ent Lefebvre)	registers Refined	s. interfaces to RB. Added state registe	ers
Date : Octobe	r 19, 2001		_	
Rev 1.1 (Laur Date : Octobe			SEQ→SP0 interfaces. Changed n. Changed VGT→SP0 interface.	
Date . Octobe	1 20, 2001		s added.	Debug
Rev 1.2 (Laur			es greatly refined. Cleaned up the sp	ec.
Date : Novem Rev 1.3 (Laur		Addad t	he different interpolation modes.	
Date : Novem		Audeu i	ne unerent interpolation modes.	
Rev 1.4 (Laur			the auto incrementing counters. C	
Date : Decem	ber 6, 2001		「→SQ interface. Added content on c ment. Updated GPRs.	onstant
Rev 1.5 (Laur	ent Lefebvre)	Remove	ed from the spec all interfaces that	
Date : Decem	ber 11, 2001		tied to the SQ. Added explanati	
		constan synchro	t management. Added F nization fields and explanation.	PA→SQ
Rev 1.6 (Laur		Added I	more details on the staging register.	
<u>Date : Januar</u>	y 7, 2002		bout the parameter caches. Chang truction to a Conditionnal call inst	
			details on constant manageme	
			the diagram.	

Exhibit 2022.doc/R400\_Sequencer.doc 67758 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257042

ATI Ex. 2106 IPR2023-00922 Page 131 of 223

4 8	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
-UU	24 September, 2001	4 September, 20157		6 of 47

## 1. Overview

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

To support the shader pipe the sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

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

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

Exhibit 2022.doc R400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257043

ATI Ex. 2106 IPR2023-00922 Page 132 of 223

# PROTECTIVE ORDER MATERIAL

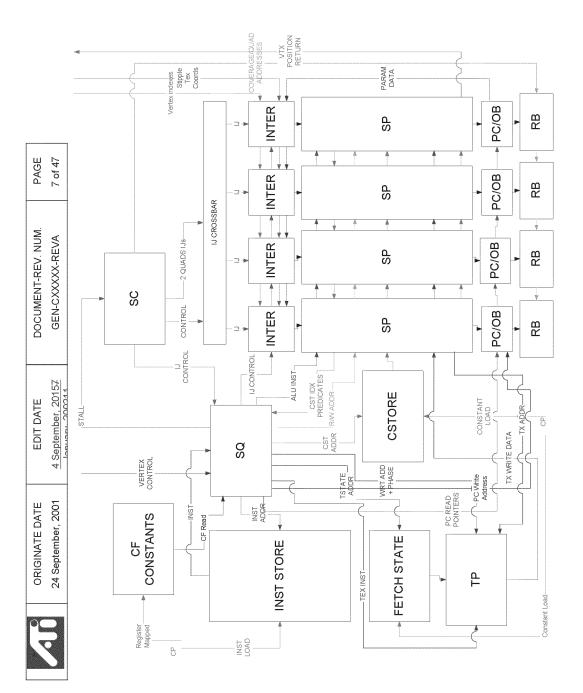


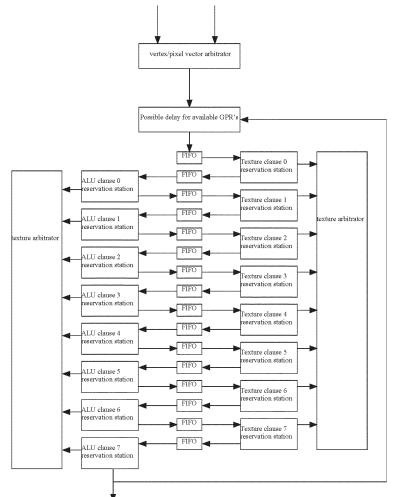
Exhibit 2022 dop R400\_Sequencerates 67789 Bytes 444 
Confidential. Reference Copyright Notice on Cover Page 
\*\*\*\*
S004/15 1257 PM010402 0257 PM1206001 0227 PM

AMD1044\_0257044

ATI Ex. 2106 IPR2023-00922 Page 133 of 223

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20157		8 of 47

1.1 Top Level Block Diagram



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

Depending on the arbitration state, the sequencer will either choose a vertex or a pixel packet. The control packet consists of 3 bits of state, 7 bits for the base address of the Shader program and some information on the coverage to determine fetch LOD plus other various small state bits.

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

Exhibit 2022 docR400\_Sequencer.doc 67798 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:57

AMD1044\_0257045

ATI Ex. 2106 IPR2023-00922 Page 134 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	9 of 47

screen position if needed) are sent to the interpolator which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

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

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

On receipt of a command, the level 0 ALU machine first decrements the input FIFO 1 counter and then issues a complete set of level 0 shader instructions. For each instruction, the ALU state machine generates 3 source addresses, one destination address and an instruction. Once the last instruction has been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

[ISSUE: How do we handle parameter cache pointers (computed, semi-computed or not computed)?]

A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

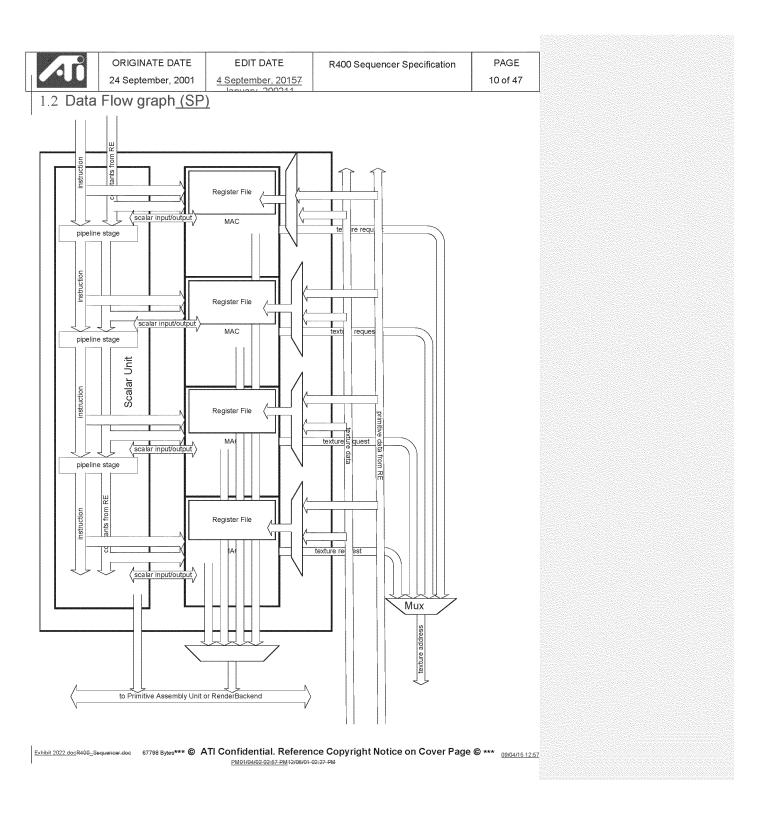
All other clauses process in the same way until the packet finally reaches the last ALU machine (7).

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

Exhibit 2022 docR400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257046

ATI Ex. 2106 IPR2023-00922 Page 135 of 223

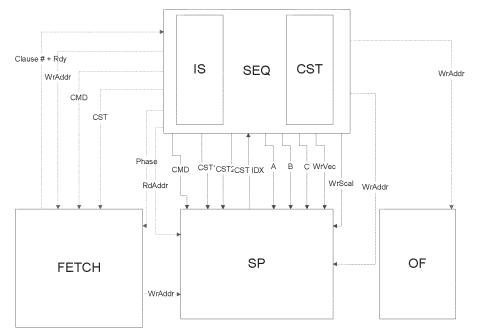


ATI Ex. 2106 IPR2023-00922 Page 136 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	11 of 47

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.

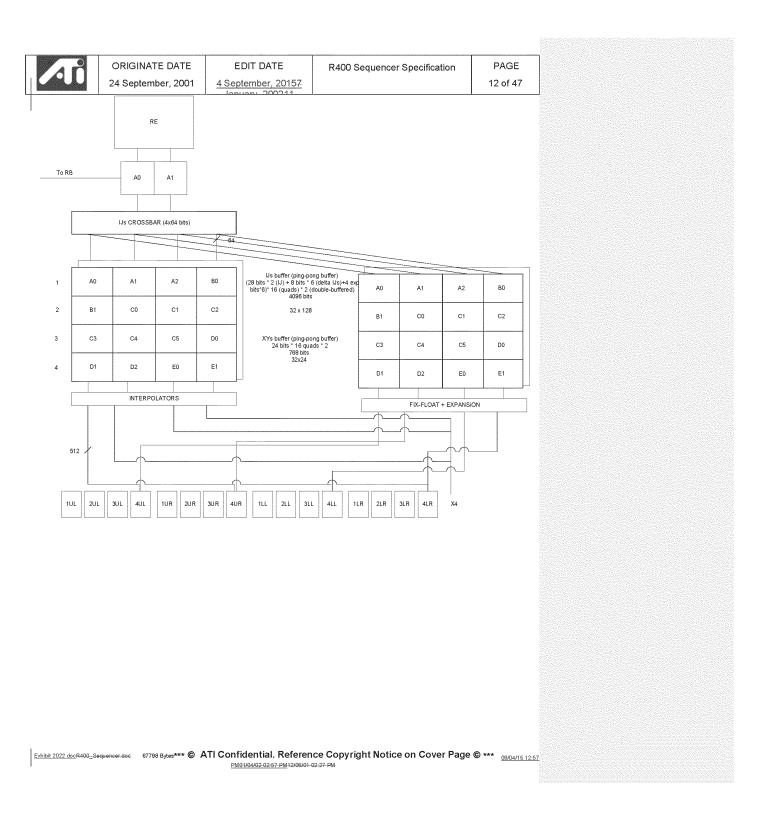
# 2. Interpolated data bus

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

Exhibit 2022.docR4400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257048

ATI Ex. 2106 IPR2023-00922 Page 137 of 223



ATI Ex. 2106 IPR2023-00922 Page 138 of 223 PROTECTIVE ORDER MATERIAL

		53				1	202	245	> 6 g	120	
		22 T					V V 32-48- 35 51	V V 36-52- 39 55	<pre>40-56- 43 59</pre>	<pre>&lt; &lt; &lt; &lt; &lt; &lt; &lt; 44-60- 47 63</pre>	$\rightarrow$
		[21] T					-16 < 1 19 €	33 ¦2 <	24-2	33 <mark>-</mark> 8 <	
		T20 1					> °-3	> 4-7	⇒ φ <i>⊂</i>	> 4 5 7 1 3	
		T19 <sup>-</sup>						X	о Ш	ũ	
		T16 T17 T18 T19 T20 T21 T22 T23								0	
щ	47	T17			ÿ₿	×Ω		8	5	ß	
PAGE	13 of 47	T16			ß	Ē				BO	P1 VTX
					ы	Ш	5	D2			Ω
IUM.	Ą	T14	גֿצ	≿8		≳৪	ប	5 2	C5		
REV. N	X-RE	T13	Б	D2		8	B1				
ENT-F	DOCUMENT-REV. NUM. GEN-CXXXX-REVA	T10 T11 T12 T13 T14 T15	δ	5		8	AO	A1	A2		
CUM		T11		≿2	×Ω				С Ш	μ	
ă		T10	18.0	5	C5	d	ų			8	
	EDIT DATE 4 September, 20157	Т9		2	C5			8	<del>ن</del>	ß	
ШĽ		Т8	≿ິ	≿ ວ	≿ວ	≿ ິ				BO	
IT DA		1	ខ	8	ۍ	ß	5	D2			È
		T6	ទ	8	ۍ	8	ប	2	C5		
	41	T5	×₽			≿ 8	B1				
ATE	2001	T4	B1			B	AO	A1	A2		
ORIGINATE DATE	September, 2001	13	<u>B</u>			BO			******	≳ ္ జ	
SIGINA	Septer	T2	۶X	žξ	Z X		XY XY 32- 48- 35 51	XY XY 36-52- 39 55	XY XY 40-56- 43 59	× 44 24 24 24 24 24 24 24 24 24 24 24 24 2	$\succ$
0 HO	4	11	AO	A1	A2		19 <sup>†</sup> 5 ×	Σģ≈	27 24- 27	3 % ≾	$\times$
<b>P</b> C		10	Ao	A1	A2		9.3 X	¥7 4-7	≍ ° ≿	5 7 ×	
k	2	///////////////////////////////////////	d o	dS –	SP SP	ი ი	с о	д <i>–</i>	N SP	ы ср	
		I		1		1		I	I		

Exhibit 2022 dog R400\_Sequencerates 67798 Bytes 444 
Confidential. Reference Copyright Notice on Cover Page 
\*\*\*\*
BID 4155 PM0140402 0257 PM1206001 0227 PM

AMD1044\_0257050

ATI Ex. 2106 IPR2023-00922 Page 139 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20157		14 of 47

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

# 3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1 port memory; we use 1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the INST\_DATA, INST\_INDEX\_PORT\_control register. The INST\_INDEX\_PORT is auto-incremented on both reads and writes to the INST\_DATA register.register mapped registers.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries.\_<u>The MSB of the INST\_INDEX\_PORT register</u> contains the packet type for the sequencer to know where it must wrap around. The wrap around points are <u>wrap</u> around points are arbitrary and they are specified in the VS\_BASE and PIX\_BASE registers.

For the Real time commands the story is quite the same but for some small differences. The CP will use the INST\_INDEX\_PORT\_RT\_and INST\_DATA\_RT register pair instead of the regular ones and T there are no wrap around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

Exhibit 2022.doc R400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257051

ATI Ex. 2106 IPR2023-00922 Page 140 of 223 PROTECTIVE ORDER MATERIAL

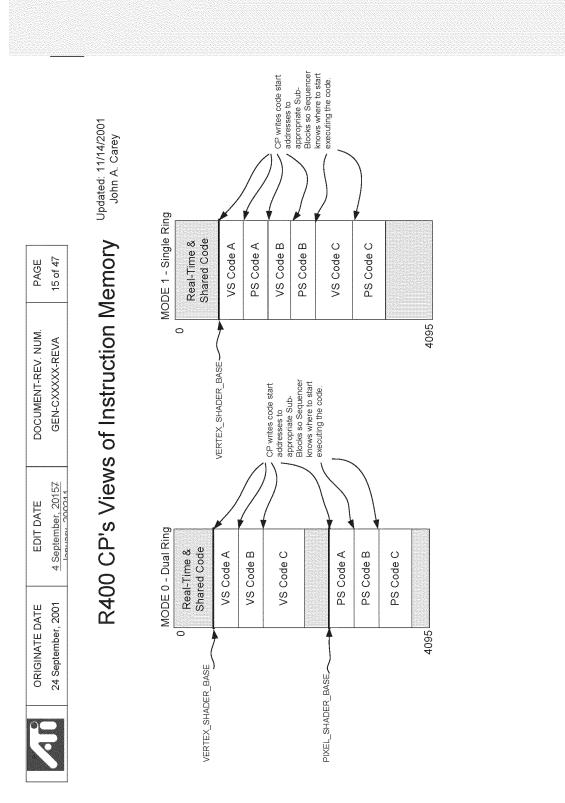


Exhibit 2022 doo R400\_Sequencedee 67789 Bytes \*\*\* 
Confidential. Reference Copyright Notice on Cover Page \*\*\*\* 
Sequence 12.57 PM0410402 20:57 PM1206604 02:27 PM

AMD1044\_0257052

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20157		16 of 47

# 4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV, PV, PS, PS) if they have nothing else to do.

# 5. Constant Stores

## 5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants.

The texture state is also kept in a similar memory. The size of this memory is 192x128. The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 16 lines (each line addresses 2 texture state lines in the real memory). The write granularity is 2 texture state lines (or 384 bits). The driver sends 512 bits but the CP ignores the top 128 bits. It thus takes 12 clocks to write the two texture states.

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change. Its size is 320\*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The CP is loading the constant store using the CONST\_DATA and CONST\_ADDR registers. It does so by writing to the CONST\_ADDR register the logical address for the constant block it wants to update and then writes 16 times to the CONST\_DATA register. The CONST\_ADDR is auto-incremented on both reads and writes to the CONST\_DATA register. The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

# 5.2 Management of the re-mapping tables

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

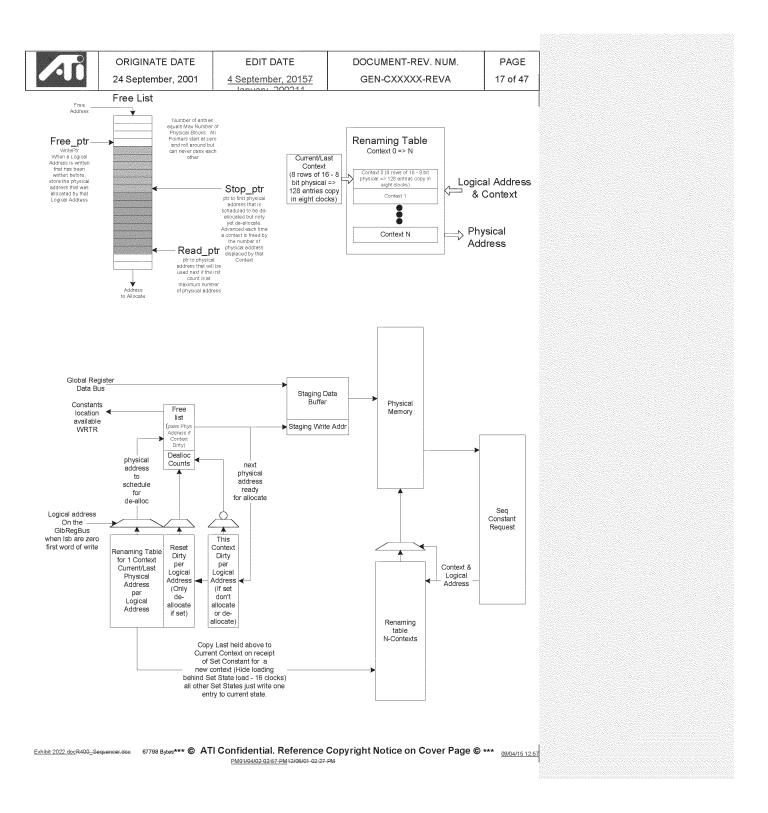
The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work, the requirement is that the physical memory MUST be at least twice as large as the logical address space. In our case, since the logical address space is 512, the memory must be of sizes 1024 and above.

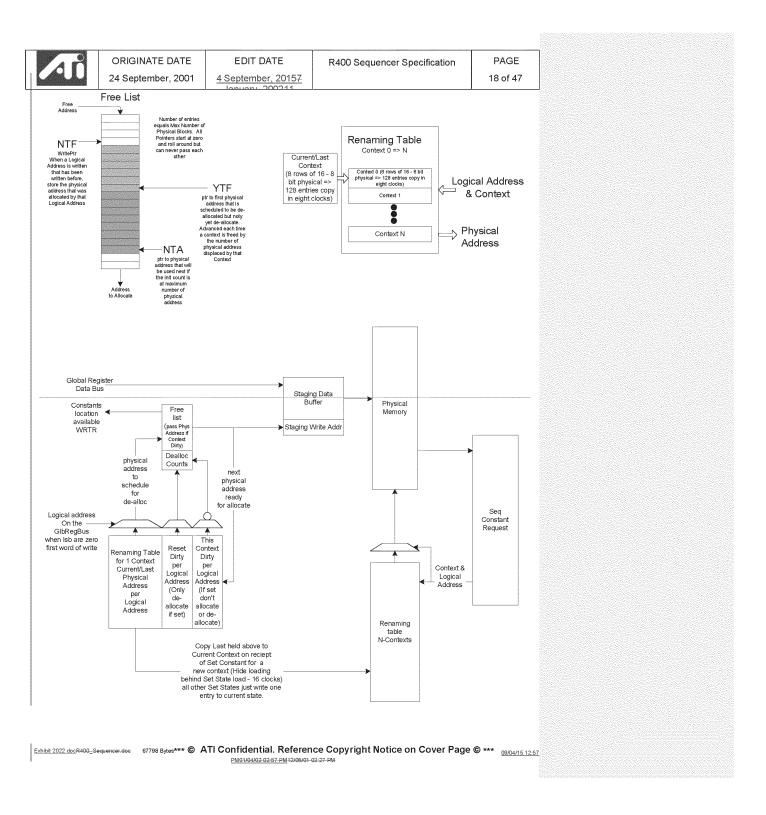
Exhibit 2022 doc R400\_Sequencer.doc 67798 Bytes\*\*\* 
CATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\* 
00/04/16 12:57 PM01/04/02 02:57 PM12/06/01 02:27 PM

AMD1044\_0257053

ATI Ex. 2106 IPR2023-00922 Page 142 of 223



ATI Ex. 2106 IPR2023-00922 Page 143 of 223



ATI Ex. 2106 IPR2023-00922 Page 144 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	19 of 47

#### 5.2.1 Dirty bits

Two sets of dirty bits will be maintained per logical address. The first one will be set to zero on reset and set when the logical address is addressed. The second one will be set to zero when ever a new context is written and set for each address written while in this context. The reset dirty is not set, then writing to that logical address will not require de-allocation of whatever address stored in the renaming table. If it is set and the context dirty is not set, then the physical address store needs to be de-allocated and a new physical address is necessary to store the incoming data. If they are both set, then the data will be written into the physical address held in the renaming for the current logical address. No de-allocation or allocation takes place. This will happen when the driver does a set constant twice to the same logical address between context changes. NOTE: It is important to detect and prevent this, failure to do it will allow multiple writes to allocate all physical memory and thus hang because a context will not fit for rendering to start and thus free up space.

#### 5.2.2 Free List Block

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented every time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call write <u>ptr</u>. This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called stop ptr. The stop ptr pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the stop ptr and write ptr cannot be reused because they are still in use. But as soon as the context using then is dismissed the stop ptr will be advanced.

The third pointer will be called read ptr. This pointer will point will point to the next address that can be used for allocation as long as the read ptr does not equal the stop ptr and the IFC is at its maximum count.

A free list block that would consist of a counter (called the IFC or Initial Free Counter) that would reset to zero and incremented eveny time a chunk of physical memory is used until they have all been used once. This counter would be checked each time a physical block is needed, and if the original ones have not been used up, us a new one, else check the free list for an available physical block address. The count is the physical address for when getting a chunk from the counter.

Storage of a free list big enough to store all physical block addresses.

Maintain three pointers for the free list that are reset to zero. The first one we will call NTF (Next To Free). This pointer will identify the next location to write the physical address of a block to be de-allocated. Note: we can never free more physical memory locations than we have. Once recording address the pointer will be incremented to walk the free list like a ring.

The second pointer will be called YTF (Yet To Free). The YTF pointer will be advanced by the number of address chunks de-allocates when a context finishes. The address between the YTF and NTF cannot be reused because they are still in use. But as soon as the context using then is dismissed the YTF will be advanced.

The third pointer will be called NTA (Next To Allocate). This pointer will point will point to the next address that can be used for allocation as long as the NTA does not equal the YTF and the IFC is at its maximum count.

#### 5.2.3 De-allocate Block

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write ptr pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de-allocation of any number of blocks in one clock.

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the NTF pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the NTF pointer to make available the set of physical blocks freed when the previous context was done. This allows the discard or de allocation of any number of blocks in one clock.

Exhibit 2022.docR400\_Sequencer.doc 67798 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09/04/15 12:57

AMD1044\_0257056

ATI Ex. 2106 IPR2023-00922 Page 145 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	1
24 September, 2001	4 September, 20157		20 of 47	

#### 5.2.4 Operation of Incremental model

The basic operation of the model would start with the write\_ptr, stop\_ptr, read\_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write\_ptr pointer location on the free list and the write\_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

- No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read\_ptr pointer if read\_ptr != to stop\_ptr.
- 2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write\_ptr and it is incremented along with the de-allocate counter for the last context.
- 3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop\_ptr == read\_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero). If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop ptr pointer. This will make all the physical addresses used by this context available to the read\_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case. The basic operation of the model would start with the NTF. YTF. NTA pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When a set constant comes with a different than last context, the previous context stored in the additional renaming table will be copied to the larger renaming table in the current (new) context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the NTF pointer location on the free list and the NTF will be incremented. The de-allocation counter for the previous context (zero) will be incremented. This as set states come in for this context one of the following will happen:

1.)No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the - - - (Formatted: Bullets and Numbering free list at NTA pointer if NTA != to YTF.

2.)Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at NTF and it is incremented along with the de-allocate counter for the last context.

3.)Context dirty is set then the data will be written into the physical address specified by the logical address.

Exhibit 2022 docR400\_Sequencer.doc 67798 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 00/04/15 12:57

Formatted: Bullets and Numbering

AMD1044\_0257057

ATI Ex. 2106 IPR2023-00922 Page 146 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	21 of 47

This process will continue as long as set states arrive. This block will provide back pressure to the CP when ever he has not free list entries available (counter at max and YTF == NTA). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero). If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the YTF pointer. This will make all the physical addresses used by this context available to the NTA allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

# 5.3 Constant Store Indexing

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

MOVA R1.X,R2.X // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X NOP // latency of the float to fixed conversion

ADD R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3

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

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

## 5.4 Real Time Commands

The real time commands constants are written by the CP using the CONST DATA RT and CONST ADDR RT GPRsregister mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST\_EO\_RT control register.

## 5.5 Constant Waterfalling

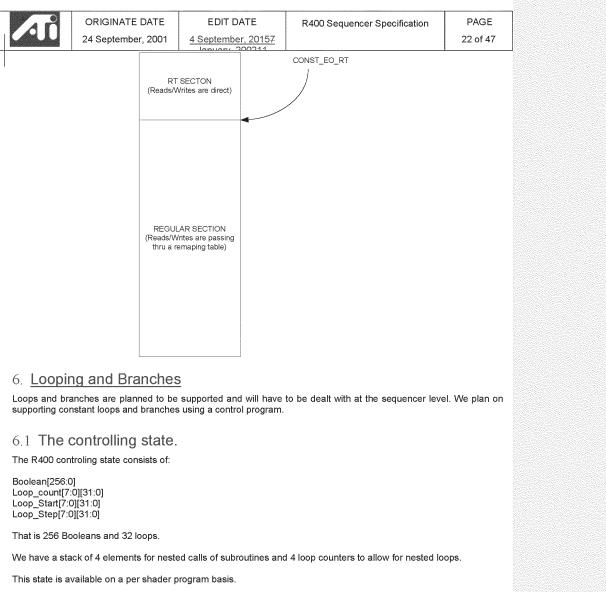
In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.

Exhibit 2022.docR400\_Sequencer.doc 67798 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* PM01/04/02 02:57 PM12/06/01 02:27 PM

Formatted: Bullets and Numbering

AMD1044 0257058

ATI Ex. 2106 IPR2023-00922 Page 147 of 223



## 6.2 The Control Flow Program

Examples of control flow programs are located in the R400 programming guide document.

The basic model is as follows:

The render state defined the clause boundaries: Vertex\_shader\_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located Vertex\_shader\_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located

Exhibit 2022.docR400\_Sequencer.doc 67798 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:57

AMD1044\_0257059

ATI Ex. 2106 IPR2023-00922 Page 148 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	23 of 47

Pixel\_shader\_fetch[7:0][7:0] Pixel\_shader\_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located // eight 8 bit pointers to the location where each clauses control program is located

A pointer value of FF means that the clause doesn't contain any instructions.

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

The control program has eleven basic instructions:

Execute Conditional\_execute\_Predicates Conditional\_jump <u>Conditional\_call</u> Return Loop\_start Loop\_end End\_of\_clause Conditional\_End\_of\_clause NOP

Execute, causes the specified number of instructions in instruction store to be executed. Conditional\_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.

Loop\_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.

Loop\_end increments (decrements?) the loop counter and jumps back the specified number of instructions.

<u>Conditionnal</u> Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.

Conditional\_execute\_or\_Jump executes a block of instructions or jumps to an address is the condition is not met. Conditional\_execute\_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition. End\_of\_clause marks the end of a clause.

Conditional\_End\_of\_clause marks the end of a clause if the condition is met.

Conditional\_jumps jumps to an address if the condition is met.

NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader\_cntl\_size (or vshader\_cntl\_size) we break the program (clause) and set the debug registers. If an execute or conditional\_execute is lower than cntl\_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

	Execute								
47	46 42	41 24	23 12	11 0	1				
Addressing	00001	RESERVED	Instruction count	Exec Address	10000				

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

		NOP
47	46 42	41 0

Exhibit 2022 doc R400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57
PM04/04/02 02:67-PM12/06/01-02:27-PM

AMD1044\_0257060

ATI Ex. 2106 IPR2023-00922 Page 149 of 223

	ORIGII	NATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 Sept	tember, 2001	4 September, 20157		24 of 47
Addressing	00010	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~		RESERVED	

This is a regular NOP.

	Conditional_Execute							
47 46 42 41 40 33 32 31 24 23 12 11 0						11 0		
Addressing	00011	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address	

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions)

	Conditional_Execute_Predicates								
47 46 42 41 35 34 33 32 31 24 23 12 11 0							11 0		
Addressing	00100	RESERVED	Predicate	Condition	RESERVED	Instruction count	Exec Address		
			vector						

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid.

		Loop_Start		
47	46 42	41 17	16 12	11 0
	00101	RESERVED	loop ID	Jump address
Addressing				

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

	Loop_End							
47	46 42	41 17	16 12	11 0				
	00111	RESERVED	loop ID	start address				
Addressing								

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop.

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

	Conditionnal_Call								
47	46 42	41	<u>34 33</u>	<u>32</u>	<u>31 12</u>	11 0			
		<u>35</u> 4112							
	01000	RESERVED	Predicate	Condition	RESERVED	Jump address			
Addressing			vector						

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack.

		Return		
47	46 42	41 0	41 0	
	01001	RESERVED	RESERVED	
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     40 33     32     31     30 12	11 0								

Exhibit 2022 doc R400\_Sequences.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257061

ATI Ex. 2106 IPR2023-00922 Page 150 of 223

	ORIGINATE DATE		EC	EDIT DATE		DOCUMENT-REV. NUM.		
	24 September, 2001		4 September, 20157		GEN-CXXXXX-REVA			25 of 47
	01010	RESERVED	Boolean	Condition	FW only	RESERVED	Jum	p address
Addressing			address					

If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.

	Conditional_End_of_Clause									
47	46 42	41	40 33	32	31 0					
	01011	RESERVED	Boolean	Condition	RESERVED					
Addressing			address							

This is an optimization in the case of very short shaders (where the control flow instruction can't be hidden anymore and thus are not free. In this case, if the condition is met, the clause is ended, else we continue the execution of the clause.

End_of_Clause							
47	46 42	41 0	11111				
Addressing	01011	RESERVED	1 Marine				

Marks the end of a clause.

To prevent infinite loops, we will keep 9 bits loop counters instead of 8 (we are only able to loop 256 times). If the counter goes higher than 255 then the loop\_end or the loop\_start instruction is going to break the loop and set the debug GPRs.

# 6.3 Data dependant predicate instructions

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

PRED\_SETE\_# - similar to SETE except that the result is 'exported' to the sequencer. PRED\_SETNE\_# - similar to SETNE except that the result is 'exported' to the sequencer. PRED\_SETGT\_# - similar to SETGT except that the result is 'exported' to the sequencer PRED\_SETGTE\_# - similar to SETGTE except that the result is 'exported' to the sequencer.

For the scalar operations only we will also support the two following instructions:

PRED\_SETE0\_# - SETE0 PRED\_SETE1\_# - SETE1

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

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

#### P0\_ADD\_# R0,R1,R2

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

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

## 6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of nonmasked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by

Exhibit 2022.docR400\_Sequencer.doc 67798 Bytes\*\*\* ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09/04/15 12:57

AMD1044\_0257062

ATI Ex. 2106 IPR2023-00922 Page 151 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20157		26 of 47

comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED\_SET and MOVA instructions and their uses.

## 6.5 Register file indexing

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

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

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

The sequencer is going to keep a loop index computed as such:

Index = Loop\_iterator\*Loop\_step + Loop\_start.

The index is going to return 0 if it is out of the range. We loop until loop\_iterator = loop\_count. Loop\_step is a signed value [-128...127].

## 6.6 Predicated Instruction support for Texture clauses

For texture clauses, we support the following optimization: we keep 1 bit (thus 4 bits for the four predicate vectors) per predicate vector in the reservation stations. A value of 1 means that one ore more elements in the vector have a value of one (thus we have to do the texture fetches for the whole vector). A value of 0 means that no elements in the vector have his predicate bit set and we can thus skip over the texture fetch. We have to make sure the invalid pixels aren't considered with this optimization.

### 6.7 Debugging the Shaders

In order to be able to debug the pixel/vertex shaders efficiently, we provide 2 methods.

#### 6.7.1 Method 1: Debugging registers

Current plans are to expose 2 debugging, or error notification, registers: 1. address register where the first error occurred

2. count of the number of errors

The sequencer will detect the following groups of errors:

- count overflow
- jump-error
- -relative jump address > size of the control flow program
- -relative jump address > length of the shader program
- ---constant overflow
- register overflow
- register overflow
- call stack
- -call with stack full
- -return with stack empty
- Compiler recognizable errors:
- jump errors

Exhibit 2022 docR400\_Sequencer.doc 67798 Bytes\*\*\* 
C ATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\*
09/04/15 12:57
PM01/04/02 02:67 PM12/06/01 02:27 PM

AMD1044\_0257063

Formatted: Bullets and Numbering

ATI Ex. 2106 IPR2023-00922 Page 152 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1	
	24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	27 of 47		
relative jump address > size of the control flow program						
relative jump address > length of the shader program						
oall stack					12363	

<u>- call stack</u> call with stack full

return with stack empty

With two of the errors, a jump error or a register overflow will cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With the other errors, program can continue to run, potentially to worst-case limits.

If indexing outside of the constant range, causing an overflow error, the hardware is specified to return the value with an index of 0. This could be exploited to generate error tokens, by reserving and initializing the 0th register (or constant) for errors.

{ISSUE : Interrupt to the driver or not?}

#### 6.7.2 Method 2: Exporting the values in the GPRs (12)

The sequencer will have a count register and an address register for this mode and 3 bits per clause specifying the execution mode for each clause. The modes can be :

- 1) Normal
- 2) Debug Kill
- 3) Debug Addr + Count

Under the normal mode execution follows the normal course. Under the kill mode, all control flow instructions are executed but all normal shader instructions of the clause are replaced by NOPs. Only debug\_export instructions of clause 7 will be executed under the debug kill setting. Under the other mode, normal execution is done until we reach an address specified by the address register and instruction count (useful for loops) specified by the count register. After we have hit the instruction n times (n=count) we switch the clause to the kill mode.

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

#### 7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK\_SETE MASK\_SETNE MASK\_SETGT MASK\_SETGTE

#### 8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

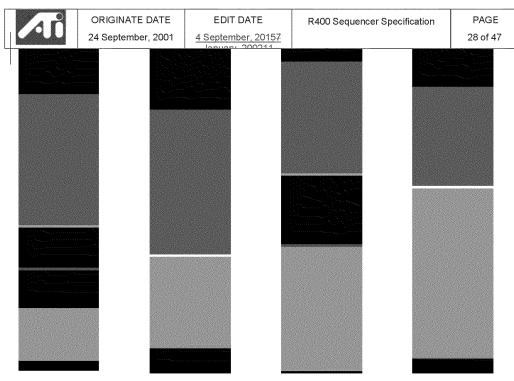
#### 9. Register file allocation

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

Exhibit 2022.docR400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*
<u>PM01/04/02-02-67-PM12/06/01-02:27-PM</u>

AMD1044\_0257064

ATI Ex. 2106 IPR2023-00922 Page 153 of 223



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 arbitres, one for the even clocks and one for the odd clocks. For example, here is the sequencing of two interleaved ALU clauses (E and O stands for Even and Odd sets of 4 clocks):

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0... Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

Exhibit 2022 docR400\_Sequencer.doc 67798 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page \*\*\* 09/04/15 12:57 PM01/04/02 02:57 PM12/06/01 02:27 PM

AMD1044\_0257065

ATI Ex. 2106 IPR2023-00922 Page 154 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	29 of 47	

## 12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS\_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT\_FILE\_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

# 13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch for only valid pixels, the quad address.

## 14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

## 15. IJ Format

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

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

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

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

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

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

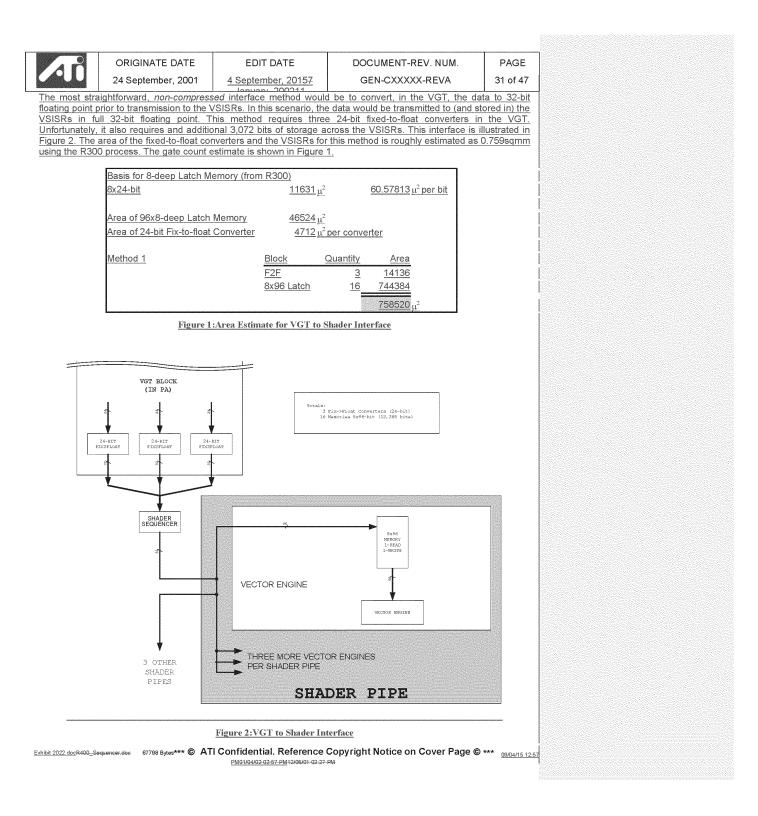
Exhibit 2022.docR400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57
PM04/04/02 02:67-PM12/06/01-02:27-PM

AMD1044\_0257066

ATI Ex. 2106 IPR2023-00922 Page 155 of 223

	ORIGINATE DATE 24 September, 2001	EDIT DATE 4 September, 20157	R400 Sequencer Specification	PAGE 30 of 47	
FORMAT OF	P0's IJ : Mantissa 20 Ex Mantissa 20 Ex	p 4 for I + Sign			
FORMAT of D	)eltas (x3):Mantissa 8 Exp Mantissa 8 Exp	4 for I + Sign			
Total number	of bits : 20*2 + 8*6 + 4*8 +	Ū			
normalized if r			ention: if exponent is different than 0 im range for the IJs (Full precision) is		
15.1 Inte	rpolation of cons	stant attributes			
	e floating point imprecision barycentric coordinates a		provisions if all the interpolated terms	are the same	
	the premise that if A = B a so we extend this to:	nd B = C and C = A, then	P0,1,2,3 = A. Since one or more of the	e IJ terms	
((J = 0) ((1-J-I = if(I ! P } els P } els P } els }	A; or (J = 0)) and or (1-I-J = 0)) and = 0) { 0 = A; 0 = A; we if(J != 0) { 0 = B; we { 0 = C;				
//rest } else	of the quad interpolated no	ormally			
{	al interpolation				
}	·				<b>Formatted:</b> Bullets and Numbering
In order for th			ill have to re-order the data sent IN C ement. Given the following group of ve		
		<u>16 17 18 19 20 21 22 23</u> 52 53 54 55 56 57 58 59 6	24 25 26 27 28 29 30 31    32 33 34 3 0 61 62 63	<u>5 36 37 38 39</u>	
The sequence	er will re-arrange them in t	his fashion:			
		50 51    4 5 6 7 20 21 22 2 28 29 30 31 44 45 46 47 6	23 36 37 38 39 52 53 54 55    8 9 10 1 10 61 62 63	1 24 25 26 27	
the missing pi	pe. For example, if SP1 is	broken, vertices 4 5 6 7 2	is broken, the VGT will send padding 20 21 22 23 36 37 38 39 52 53 54 55 w 1 thus should be considered invalid (b	vill still be sent	Formatted
VGT).		-ของกรรมการสาราชสารสาราชสารสาราชสารสาราชสารสาราชสารสาราชสารสาราชสารสาราชสาร			
Exhibit 2022.doc8400_S	equencer.doc 67798 Bytes*** ⓒ A	ATI Confidential. Referen PM01/04/02-02:67-PM12/06/01-1	ce Copyright Notice on Cover Page	© *** 09/04/15 12:57	

ATI Ex. 2106 IPR2023-00922 Page 156 of 223



ATI Ex. 2106 IPR2023-00922 Page 157 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	1
24 September, 2001	4 September, 20157		32 of 47	

## 16.17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER	ADDRESS
4 bits	7 bits

The PA generates the parameter cache addresses as the positions comes from the SQ. All it needs to do is keep a Current Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current Location by VS EXPORT\_COUNT\_6 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS EXPORT\_COUNT\_6 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 00100000000 and so on up to 11110000000. Then the next position received (the 17<sup>th</sup>) is going to have the address 0000001000, the 18<sup>th</sup> 00010001000, the 19<sup>th</sup> 0010001000 and so on. The Current location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add 1\*VS EXPORT\_COUNT\_6 to Current\_Location and reset the memory count to 0 before the next vector begins).

# 17.18. Vertex position exporting

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT\_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

## 18.19. Exporting Arbitration

Any type of exporting clause can be co-issued. The sequencer will have to make sure back to back memory exports (position/straight memory exports) are interleaved with NOPs as we don't have the bandwidth to service them at full speed.

Here are the rules for co-issuing exporting ALU clauses.

Position exports and position exports cannot be co-issued.
 Position exports and memory exports cannot be co-issued.
 Position exports and Z/Color exports cannot be co-issued.

4)Memory exports and Z/Color exports cannot be co-issued.
5)Memory exports and memory exports cannot be co-issued.
6)Z/color exports and Z/Color exports cannot be co-issued.
7)Parameter exports and Z/Color exports CAN be co-issued.

8)Parameter exports and parameter exports CAN be co-issued. 9)Parameter exports and memory exports CAN be co-issued.

### 19.20. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

Exhibit 2022.docR400\_Sequencer.doc 67798 Bytes\*\*\* 
C ATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\*

09/04/15 12:57
PM01/04/02 02:57 PM12/06/01-02:27 PM

Formatted: Bullets and Numbering

Formatted

### Formatted

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0257069

ATI Ex. 2106 IPR2023-00922 Page 158 of 223

ORIGINATE DATE         EDIT DATE         DOCUMENT-REV. NUM.         PAGE           24 September, 2001         4 September, 2017         GEN-CXCOCX.REVA         33 of 47           19.120.11         Vertex Shading         Selection         Selection         Selection           19.120.11         Vertex Shading         Figure 3000000000000000000000000000000000000					
19-1201       Vertex Shading       • • • • • • • • • • • • • • • • • • •	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
19-1201 Vertex Shading 19-2202 Pixel Shading 0 - Color for buffer 0 (primary) 1 - Color for buffer 0 (primary) 1 - Color for buffer 0 (primary) 1 - Color for buffer 1 2 - Color for buffer 2 3 - Color for buffer 2 3 - Color for buffer 2 3 - Color for buffer 3 4 - T - Empty 6 - Buffer 2 Color/Fog 11 - Buffer 3 Color/Fog 12:15 - Empty 6 - Buffer 2 Color/Fog 12:15 - Empty 16:31 - Empty (Reserved?) 32:43 - 12 exports for multipase pixel shaders. 44:47 - Empty 48:59 - 12 debug exports (interpret as normal pixel export) 6 - export addressing mode 6 : 12 - Empty 10 - Suffer 2 Color/Fog 11 - Buffer 3 Color/Fog 12:15 - Empty 12 debug exports for multipase pixel shaders. 44:47 - Empty 48:59 - 12 debug exports (interpret as normal pixel export) 6 - export addressing mode 6 : 12 debug exports for multipase pixel shaders. 20-12 Special Interpolation modes 20-12 Special Interpolation to the export of the export interpolate interpolat	24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	33 of 47	
16:31 - Empty (Reserved?) 32:43 - 12 vertex sports for the frame buffer and index 44:47 - Empty 48:59 - 12 debug export (interpret as normal vertex export) 60 - export addressing mode 61 - Empty 62 - sprite size export that goes with position export (point), -point, w.edgellag, misc.) 63 - position 19.220.2 Pixel Shading 0 - Color for buffer 0 (primary) 1 - Color for buffer 1 2 - Color for buffer 1 2 - Color for buffer 1 3 - Color for buffer 0 (primary) 1 - Color for buffer 1 2 - Color for buffer 1 3 - Color for buffer 2 4 - Formatted: Buffets and Humbering 0 - Suffer 1 Color/Fog 11 - Buffer 3 Color/Fog 12 - Sports for multipase pixel shaders. 4 - Formatted: Buffets and Humbering 20-21. Special Interpolation modes 20-42.1. Real time commands We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16/x128 memories (one for each of time vertices x 16 interpolarit). These will be mapped on to the realities and humbering 20-21. Special Interpolation modes 20-42.1. Real time commands We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we meed to add three 16/x128 memories (one for each of time vertices x 16 interpolarit). There will be mapped on to the realities the add three 16/x128 memories (one for each of time vertices x 16 interpolarit). There will be to restrict the memory to 16x64 or 32x64 allowing only two interpolates coordinates, one oppon might be to restrict the memory to 16x64 or 32x64 allowing only two interpolates condrates. One oppon might be to restrict the memory t	9.120.1 Vertex Shading		•	*-	Formatted: Bullets and Numbering
19.220.2       Pixel Shading         0       - Color for buffer 0 (primary)         1       - Color for buffer 1         2       - Color for buffer 2         3       - Color for buffer 3         4.7       - Empty         8       - Buffer 1 Color/Fog         10       - Buffer 2 Color/Fog         11       - Buffer 2 Color/Fog         12       - Empty         13       - Empty         1631       - Empty         1633       - Empty (Reserved?)         32.43       - 12 exports for multipass pixel shaders.         44.47       - Empty         4359       - 12 debug exports (interpret as normal pixel export)         60       - export addressing mode         61:62       - Empty         63       - Z for primary buffer (2 exported to 'alpha' component)         20-21_1 Real time commands       - Ender addressing mode         8-unbel to use the parameter memory since there is no way for a command stream to write into it. Instead we the able to leader withern as two banks of 16 and od ouble buffering allowing on to the banks of 16 and od ouble buffering allowing on to the banks of 16 and od ouble buffering allowing on to the banks of 16 and od ouble buffering allowing on the uniterpolated satars per cycle, the only prolibon mise with high sing (if we way thaderes withe med 2 or 4 scalar coordinates, one option	16:31       - Empty (Reserve         32:43       - 12 vertex export         44:47       - Empty         48:59       - 12 debug expor         60       - export addressi         61       - Empty         62       - sprite size expo         (point_h,point_v)	ed?) s to the frame buffer and ind t (interpret as normal vertex ng mode rt that goes with position ex	r export)		
Color for buffer 0 (primary) <ul> <li>- Color for buffer 1</li> <li>- Color for buffer 2</li> <li>- Color for buffer 3</li> <li>- Color for buffer 1 Color/Fog</li> <li>- Buffer 1 Color/Fog</li> <li>- Buffer 1 Color/Fog</li> <li>- Buffer 1 Color/Fog</li> <li>- Buffer 2 Color/Fog</li> <li>- Buffer 2 Color/Fog</li> <li>- Buffer 2 Color/Fog</li> <li>- Buffer 3 Color/Fog</li> <li>- Eurphy</li> <li>- Corport addressing mode</li> <li>- Corport 16 vector-4 interpolating importa</li></ul>	9.220,2 Pixel Shading			*-	Formatted: Bullets and Numbering
20-21. Special Interpolation modes 20-121.1 Real time commands We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the reatime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. 20-221.2 Sprites/XY screen coordinates/ FB information When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back	1- Color for buffer2- Color for buffer3- Color for buffer4:7- Empty8- Buffer 0 Color/F9- Buffer 1 Color/F10- Buffer 2 Color/F11- Buffer 3 Color/F12:15- Empty16:31- Empty (Reserve32:43- 12 exports for n44:47- Empty48:59- 12 debug expor60- export addressi61:62- Empty	1 2 3 Fog (primary) Fog Fog ed?) hultipass pixel shaders. ts (interpret as normal pixel ng mode			
20.121.1 Real time commands We are unable to use the parameter memory since there is no way for a command stream to write into it. Instead we need to add three 16x128 memories (one for each of three vertices x 16 interpolants). These will be mapped onto the register bus and written by type 0 packets, and output to the the parameter busses (the sequencer and/or PA need to be able to address the reatime parameter memory as well as the regular parameter store. For higher performance we should be able able to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. 20.221.2 Sprites/XY screen coordinates/ FB information When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_I0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back	20.21. Special Interpolatio	n modes		*	Formatted: Bullets and Numbering
20.221.2 Sprites/ XY screen coordinates/ FB information When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen_10 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back	We are unable to use the parameter mer need to add three 16x128 memories (one egister bus and written by type 0 packets be able to address the reatime parameter should be able able to view them as two other is rasterized with. Most overlay sha nemory to 16x64 or 32x64 allowing only riew support for 16 vector-4 interpolants stream), then the PA/sequencer need to	nory since there is no way f for each of three vertices > s, and output to the the para memory as well as the reg banks of 16 and do doubl aders will need 2 or 4 scala two interpolated scalars pe important (true only if we m support a realtime-specific	c 16 interpolants). These will be map ameter busses (the sequencer and/o ular parameter store. For higher perf le buffering allowing one to be loade r coordinates, one option might be t er cycle, the only problem I see with ap Microsoft's high priority stream to mode where we need to address 3	ped onto the r PA need to formance we ed, while the o restrict the this is, if we the realtime	
coordinates may be needed in the shader program. This functionality is controlled by the gen_l0 register (in SQ) in conjunction with the SND_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back	20.221.2 Sprites/ XY scree	en coordinates/ Fl	B information	<b>4</b> -	
together:	coordinates may be needed in the shade conjunction with the SND_XY register (in pecial operations) to the shader using th	er program. This functionali SC). Also it is possible to s	ty is controlled by the gen_I0 registent and the faceness information (for OC	er (in SQ) in GL front/back	

Exhibit 2022.doc/R400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257070

ATI Ex. 2106 IPR2023-00922 Page 159 of 223

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20157		34 of 47

Gen\_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param\_Gen\_I0 disable, snd\_xy disable, no gen\_st - I0 = No modification Param\_Gen\_I0 disable, snd\_xy disable, gen\_st - I0 = No modification Param\_Gen\_I0 disable, snd\_xy enable, no gen\_st - I0 = No modification Param\_Gen\_I0 disable, snd\_xy enable, gen\_st - I0 = No modification Param\_Gen\_I0 enable, snd\_xy disable, no gen\_st - I0 = garbage, garbage, garbage, faceness Param\_Gen\_I0 enable, snd\_xy disable, gen\_st - I0 = garbage, garbage, s, t Param\_Gen\_I0 enable, snd\_xy enable, no gen\_st - I0 = screen x, screen y, garbage, faceness Param\_Gen\_I0 enable, snd\_xy enable, gen\_st - I0 = screen x, screen y, s, t

### 20.321.3 Auto generated counters

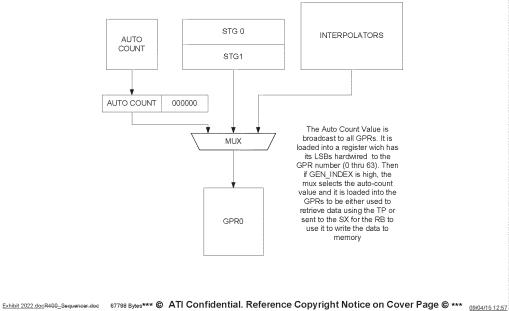
In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1<sup>st</sup> pass data to memory and then use the count to retrieve the data on the 2<sup>nd</sup> pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN\_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

#### 20.3.121.3.1 Vertex shaders

In the case of vertex shaders, if GEN\_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

#### 20.3.221.3.2 Pixel shaders

In the case of pixel shaders, if GEN\_INDEX is set and Param\_Gen\_10 is enabled, the data will be put in the x field of the  $2^{nd}$  register (R1.x), else if GEN\_INDEX is set the data will be put into the x field of the  $1_{\star}^{st}$  register (R0.x).



PM01/04/02 02:67 PM12/06/01-02:27 PM

AMD1044\_0257071

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted

ATI Ex. 2106 IPR2023-00922 Page 160 of 223

2000 001 2000 000 000 000 000 000 000 00					
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	35 of 47	
21.22. St	ate manageme			4	· · · · · · · · · · · · · · · · · · ·
	the sequencer will repo hey enter the last ALU cl		s still in the pipe. These are the s	tates of the	
<del>21.1</del> 22.1	Parameter cacl	ne synchronization			
sequencer wi time a vertex When the SC the count is g ready to rece	Il keep a 6 bit count per shader exports its data sends a new vector of p reater than 0 before acc ive). Then the sequence	state (for a total of 8 counters TO THE PARAMETER CAC ixels with the SC_SQ_new_ve epting the transmission (it will r waits for the count to go to c	e associated group of vertices has f s). These counters are initialized to HE, the corresponding pointer is in ctor bit asserted, the sequencer will in fact accept the transmission but th one and decrements it. The sequencinges, the new state counter is initiali	0 and every ncremented. first check if nen lower its cer can then	
<del>22.</del> 23. X	Y Address impo	rts		4	<b>Formatted:</b> Bullets and Numbering
The SC will b buffer) with X interpolate the the GPRs. Th	e able to send the XY ac XY writes (to the XY bu I J data or pass the XY ne Xys are currently SC	dresses to the GPRs. It does iffer). Then when writing the lata thru a Fix→float converter	so by interleaving the writes of the I data to the GPRs, the sequencer and expander and write the convert S. The values in the XY buffers wi this mode.	is going to ed values to	
<del>22.1</del> 23.1	Vertex indexes	imports		4-	( Formatted: Bullets and Numbering
	,		registers. These are loaded one <u>line</u> can be transferred in 4 or 8 clocks to	~ *	
<del>23.<u>24</u>. R</del>	egisters			4	
<u>23.124.1</u>	Control				
REG_DYI REG_SIZ		nic allocation (pixel/vertex) of t f the register file's pixel portion	he register file on or off. (minimal size when dynamic allocat	ion turned	
REG_SIZ		f the register file's vertex portion	on (minimal size when dynamic alloca	ation turned	
	TION_POLICY policy ORE_ALLOC interle SE_VTX start p	of the arbitration between vert aved, separate oint for the vertex instruction s s at 0)	exes and pixels tore (RT always ends at vertex_base	and	
INST_BA ONE_TH ONE_ALU	SE_PIX start p READ debug	oint for the pixel shader instruct state register. Only allows one	stion store e program at a time into the GPRs e ALU program at a time to be execu	ted (instead	
INSTRUC	TION_ADDR This is	where the CP puts the base nented on reads/writes)	address of the instruction writes and	l type (auto-	
	TION_DATA This is	where the CP puts the actual	data going to the instruction memory	1	
		512*4 ALU constants + 3	ogical constant address (9 bits) <u>C</u> ;2*6 Texture state 32 bits registe		
INSTRUC	TION_ADDR_RT This i		e address of the instruction writes a ds/writes)	and type for	
INSTRUC		where the CP puts the actual	data going to the instruction memory	for	
CONSTA		where the CP puts constant c	lata for Real Time (32-bits)		
Exhibit 2022.docR400_S	equencer.doc 67798 Bytes*** ⓒ	ATI Confidential. Reference PM01/04/02-02:67-PM12/06/01-02:27	Copyright Notice on Cover Page © ™	) *** <u>09/04/15 12:57</u>	

ATI Ex. 2106 IPR2023-00922 Page 161 of 223

ORIGINATE D	ATE EDIT DATE	R400 Sequencer Specification	PAGE	
24 September,	2001 <u>4 September, 20157</u>		36 of 47	
CONSTANT_ADDR_RT CONSTANT_EO_RT	This is where the CP puts bits)CONSTANTS_RT_256*4_A mapped)	the logical constant address for F LU constants + 32*6 texture states served for real time in the constant st	s? (physically	
EXPORT_LATE	CONSTANT_EO_RT). The re-ma	apping table operates on the rest of the exporting position from clause 3. It	e memory	
23.224.2 Context			4	
VS_FETCH_{07} VS_ALU_{07} PS_FETCH_{07} PS_BASE VS_CF_SIZE PS_CF_SIZE PS_SIZE VS_SIZE PS_NUM_REG VS_NUM_REG PARAM_SHADE	eight 8 bit pointers to the location eight 8 bit pointers to the location eight 8 bit pointers to the location base pointer for the pixel shader base pointer for the vertex shade size of the vertex shader (# of ins size of the vixel shader (# of inst size of the vixel shader (cntl+ins size of the vertex shader (cntl+ins number of GPRs to allocate for p One 16 bit register specifying wh = gouraud)	r in the instruction store structions in control program/2) uctions in control program/2) ructions) structions) ixel shader programs ertex shader programs ich parameters are to be gouraud shad	s located is located s located ded (0 = flat, 1	
PARAM_WRAP PS_EXPORT_MODE	(0=linear, 1=cylindrical). 0xxxx : Normal mode 1xxxx : Multipass mode	d channels (xyzw)) do we do the cyl w r many colors (0-4) and z is export z or		
VS_EXPORT_MASK VS_EXPORT_MODE VS_EXPORT _COUNT_{06}	If multipass 1-12 exports for color which of the last 6 ALU clauses is 0: position (1 vector), 1: position	r. s exporting (multipass only)		
PARAM_GEN_I0 GEN_INDEX CONST_BASE_VTX (9 bits CONST_BASE_PIX (9 bits) CONST_SIZE_PIX (8 bits)	(located in VS_EXPORT_COUN # of exported vectors to memory Do we overwrite or not the param Auto generates an address from shaders and R3.R2 for vertex sha )Logical Base address for the con Logical Base address for the con Size of the logical constant store Size of the logical constant store	T_6) OR per clause in multipass mode (per clau leter 0 with XY data and generated T a 0 to XX. Puts the results into R1 <u>R0-1</u> aders stants of the Vertex shader stants of the Pixel shader for pixel shaders for vertex shaders ation (if of, conditional_execute_predic es we traverse the loop) in index computation)	ise) ind S values for pixel	Enmatted: Bullet: and Numbering
24.25. DEBUG Regi	sters		4	Formatted: Bullets and Numbering
24.125.1 Context				
DB_PROB_ADDR DB_PROB_COUNT DB_INST_COUNT DB_BREAK_ADDR	instruction address where the firs number of problems encountered instruction counter for debug met break address for method number	during the execution of the program hod 2		
Exhibit 2022.docR400_Sequencer.doc 67798 Byter	**** © ATI Confidential. Referen PM01/04/02-02:67-PM12/06/01-4	ce Copyright Notice on Cover Page	© *** 09/04/15 12:57	
	<u></u>			

ATI Ex. 2106 IPR2023-00922 Page 162 of 223

ORIGINAT	E DATE	EDIT DATE		NT-REV. NUM.	PAGE	
24 Septemb	er, 2001 🔮	September, 2	0157 GEN-C	XXXXX-REVA	37 of 47	
DB_CLAUSE _MODE_ALU_{07} DB_CLAUSE	clause mode	for debug meth	od 2 (0: normal, 1: addı	, 2: kill)		
_MODE_FETCH_{0	.7} claus	e mode for del	ug method 2 (0: normal	, 1: addr, 2: kill)		
25.26. Interfaces					<b></b> -	Formatted: Bullets and Numbering
<u>25.126.1</u> External l	nterfaces					
Whenever an x is used, it m named SQ→SPx it means tha					e, if a bus is	
<del>25.1.1</del> 26.1.1_SC to S	SQ : IJ Contr	ol bus			4-	Formatted: Bullets and Numbering
This is the control information execute a shader program or control packet is going to be just sent). All pixels from the render state.	the sent pixels. ignored and XY	This informati information is	on is sent over 2 clocks going to be sent on the	, if SENDXY is asse IJ bus (for the quad	rted the next Is that where	
Name	Direction	Bits	Description			
SC_SQ_q_wr_mask	SC→SQ	4	Quad Write mask left f			
SC_SQ_lod_correct	SC→SQ	24	LOD correction per qu			
SC_SQ_flat_vertex	SC→SQ	2	Provoking vertex for fla	X		
SC_SQ_param_ptr0	SC→SQ	11	P Store pointer for ver	tex 0		
SC_SQ_param_ptr1	SC→SQ	11	P Store pointer for ver			
SC_SQ_param_ptr2	SC→SQ	11	P Store pointer for ver	tex 2		
SC_SQ_end_of_vect	SC→SQ	1	End of the vector			
SC SQ store dealloc	SC→SQ	1	Deallocation token for	the P Store		
SC_SQ_state	SC→SQ	3	State/constant pointer	-		
SC_SQ_valid_pixel	SC→SQ	16	Valid bits for all pixels			
SC SQ null prim	SC→SQ	1	Null Primitive (for PC of	leallocation purpose	s)	
SC SQ end of prim	SC→SQ	1	End Of the primitive		-/	
SC_SQ_send_xy	SC→SQ	1	Sending XY information		s going to be	
SC_SQ_prim_type	SC→SQ	3	Real time command alternate buffer. Line their parameters from 000: Normal 011: Real Time 100: Line AA 101: Point AA 110: Sprite	AA, Point AA and	Sprite reads	
	SC→SQ	1	This primitive comes Make sure that the c finished before starting	orresponding vertex the group of pixels.	shader has	
		1	Stalls the PA in n cloc		]	
SC_SQ_RTRn	SQ→SC		SC ready to send data			
SC_SQ_new_vector SC_SQ_RTRn SC_SQ_RTS	SQ→SC SC→SQ	1				
SC_SQ_RTRn SC_SQ_RTS	SC→SQ	1 .	, <b>,</b>		4-	
SC_SQ_RTRn	SC→SQ	1 .	Description		*-	- 1 Formatted: Bullets and Numbering

ATI Ex. 2106 IPR2023-00922 Page 163 of 223

	<b>A</b> Î	ORIGINATE I 24 September		EDIT D	er, 201	57	R400 Sequencer Specification	PAGE 38 of 47
	SQ_SPx_inter	p_flat_vtx	SQ→SF	°x	2	Pro	voking vertex for flat shading	
1	SQ_SPx_inter	p_flat_gouraud	SQ→SI	⊃x	1	Fla	t or gouraud shading	
	SQ_SPx_inter	p_cyl_wrap	SQ→SI	°x	4	Wio	h parameter needs to be cylindrical w	rapped
	SQ_SPx_inter	p_ijline	SQ→SI	⊃x	2	Lin	e in the IJ/XY buffer to use to interpola	te
	SQ_SPx_inter	p_buff_swap	SQ→SI	⊃x	1	Sw	ap the IJ/XY buffers at the end of the in	nterpolation
	SQ_SPx_inter	p_gen_I0	SQ→SF	⊃x	1	Ge	nerate I0 or not. This tells the interp	olators not to
							the parameter cache but rather over	
						with	n interpolated 1 and 0. Overwrite if gen	_10 is high.

#### 25.1.3SQ to SP: GPR Input Mux select

This interface is synchronized with the Interpolator bus. This controls the input mux to the GPRs. The three types of data are: generated index, Interpolated data, vertex index data (coming from the staging registers).

## 25.1.426.1.3 SQ to SP: Parameter Cache Read control bus

The four following interfaces (SQ→SP, SQ→SX,SP→SX and SX→Interpolators) are all SYNCHRONIZED together.

Name	Direction	Bits	Description
SQ_SPx_ptr0	SQ→SPx	9	Pointer of PC
SQ_SPx_ptr1	SQ→SPx	9	Pointer of PC
SQ_SPx_ptr2	SQ→SPx	9	Pointer of PC
SQ_SP0_read_ena	SQ→SP0	4	Read enables for the 4 memories in the SP0
SQ_SP1_read_ena	SQ→SP1	4	Read enables for the 4 memories in the SP1
SQ_SP2_read_ena	SQ→SP2	4	Read enables for the 4 memories in the SP2
SQ_SP3_read_ena	SQ→SP3	4	Read enables for the 4 memories in the SP3

#### 25.1.526.1.4 SQ to SX: Parameter Cache Mux control Bus

'	Name	Direction	Bits	Description
	SQ_SXx_mux0	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)
	SQ_SXx_mux1	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)
	SQ SXx mux2	SQ→SXx	4	Mux control for PC (4 MSbs of Pointer)

## 25.1.626.1.5 SQ to SP: Staging Register Data

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx0_vgt_vsisr_data	SQ→SP <u>x</u> Q	96	Pointers of indexes or HOS surface information
SQ_SP0x_vgt_vsisr_double	SQ→SP <u>x</u> 0	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_data_valid	SQ→SP0	1	Data is valid
SQ_SP1_data_valid	SQ→SP1	1	Data is valid
SQ_SP2_data_valid	SQ→SP2	1	Data is valid
SO SP3 data valid	SQ→SP3	1	Data is valid

25.1.726.1.6 PA to SQ : Vertex interface

#### 25.1.7.126.1.6.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. <u>Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.</u> The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Exhibit 2022 docR400\_Sequencer.doc 67798 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 09/04/15 12:57

--- Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0257075

ATI Ex. 2106 IPR2023-00922 Page 164 of 223

	ORIGINATI	E DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 Septemb	er, 2001	4 September, 20157	GEN-CXXXXX-REVA	39 of 47
Name		Bits	Description		
PA_SQ_vgt_vs	isr_data	96	Pointers of indexes or HOS sur	face information	
PA_SQ_vgt_vs	isr_double	1	0: Normal 96 bits per vert 1: do	uble 192 bits per vert	
PA_SQ_vgt_er	id_of_vector	1	Indicates the last VSISR data s data, "end_of_vector" is set on	set for the current process vector (for the second vector)	double vector
PA_SQ_vgt_st	ate	3	Render State (6*3+3 for consta "PA_SQ_vgt_end_of_vector" is	ants). This signal is guaranteed to be high.	correct when
PA_SQ_vgt_se	nd	1	Data on the VGT_SQ is valid n interface handshaking)	eceive (see write-up for standard R40	0 SEND/RTR
SQ_PA_vgt_rtr		1	Ready to receive (see writ handshaking)	e-up for standard R400 SEND/R	TR interface

# 25.1.7.226.1.6.2 Interface Diagrams

Formatted: Bullets and Numbering

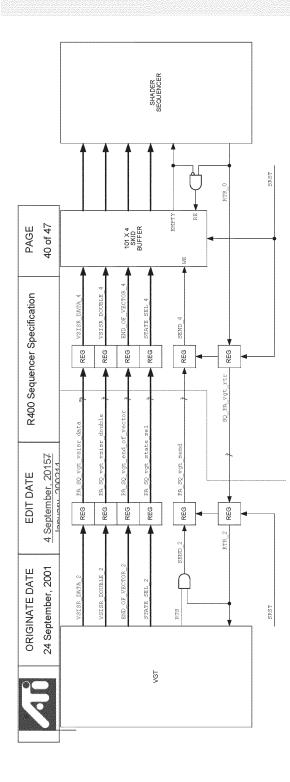
**«**...

Exhibit 2022 doc R400\_Sequencer.doc 67798 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 00/04/15 12:57

AMD1044\_0257076

ATI Ex. 2106 IPR2023-00922 Page 165 of 223

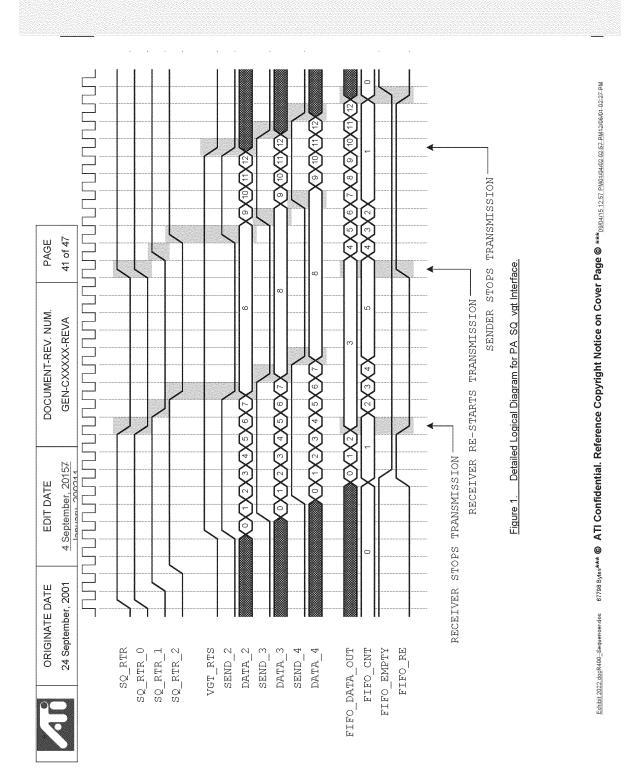
# PROTECTIVE ORDER MATERIAL



67788 Bytes\*\*\* 
 ATI Confidential. Reference Copyright Notice on Cover Page 
 \*\*\*\*
 \*\*\*\*
 asserting 12.57 PMOTADAD20257 PM1206001-02277 PM Exhibit 2022.docR400\_Sequencer.doc

AMD1044\_0257077

ATI Ex. 2106 IPR2023-00922 Page 166 of 223 PROTECTIVE ORDER MATERIAL



AMD1044\_0257078

ORIGINATE	DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 Septembe	r, 2001 <u>4</u>	September, 201		42 of 47	
. <u>1.826.1.7</u> SQ to 0	CP: State i		1 1	4-	
me	Direction	Bits	Description		
_CP_vrtx_state	SEQ→CP	3	Oldest vertex state still in the pipe		
CP_pix_state	SEQ→CP	3	Oldest vertex state still in the pipe		
5.1.926.1.8 SQ to S		· · · · · ·			Formatted: Bullets and Numbering
ame			Description		
Q SXx_exp_Pixel	Direction SQ→SXx	Bits 1	Description 1: Pixel		
aTovyTeyhTuyei		1	0: Vertex		
Q SXx exp start	SQ→SXx	1	Raised to indicate that the SQ is starting a	an export	
Q SXx exp Clause	SQ→SXx	3	Clause number, which is needed for verte		
Q SXx exp State	SQ→SXx	3	State ID, which is needed for vertex claus		
2 <u>5.1.1026.1.9</u> SX to	SQ : Outp	out file contro	0/	*	Formatted: Bullets and Numbering
					Formatted: Bullets and Numbering
lame	SQ : Outp Direction SXx→SQ	out file contro Bits 1	Description Raised by SX0 to indicate that the following		Formatted: Bullets and Numbering
lame SXx_SQ_Export_count_rdy	Direction SXx→SQ	Bits 1	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export		Formatted: Bullets and Numbering
Vame SXx_SQ_Export_count_rdy SXx_SQ_Export_Position	Direction SXx→SQ SXx→SQ	Bits 1 1	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth	er position.	Formatted: Bullets and Numbering
Vame SXx_SQ_Export_count_rdy SXx_SQ_Export_Position	Direction SXx→SQ	Bits 1	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the output	er position.	Formatted: Bullets and Numbering
Vame SXx_SQ_Export_count_rdy SXx_SQ_Export_Position	Direction SXx→SQ SXx→SQ	Bits 1 1	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the output 0: buffers are full	er position. t buffers.	Formatted: Bullets and Numbering
Vame SXx_SQ_Export_count_rdy SXx_SQ_Export_Position	Direction SXx→SQ SXx→SQ	Bits 1 1	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the output	er position. t buffers.	Formatted: Bullets and Numbering
Vame SXx_SQ_Export_count_rdy SXx_SQ_Export_Position	Direction SXx→SQ SXx→SQ	Bits 1 1	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the output 0: buffers are full 1: 2K-bits available (32-bits for each of the	er position. t buffers. e 64	Formatted: Bullets and Numbering
Vame SXx_SQ_Export_count_rdy SXx_SQ_Export_Position	Direction SXx→SQ SXx→SQ	Bits 1 1	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the outpu 0: buffers are full 1: 2K-bits available (32-bits for each of the pixels in a clause)  64: 128K-bits available (16 128-bit entries 64 pixels)	er position. t buffers. e 64	Formatted: Bullets and Numbering
Jame SXx_SQ_Export_count_rdy SXx_SQ_Export_Position	Direction SXx→SQ SXx→SQ	Bits 1 1	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the outpu 0: buffers are full 1: 2K-bits available (32-bits for each of the pixels in a clause)  64: 128K-bits available (16 128-bit entries	er position. t buffers. e 64	
ame Xx_SQ_Export_count_rdy Xx_SQ_Export_Position Xx_SQ_Export_Buffer	Direction SXx→SQ SXx→SQ SXx→SQ	Bits 1 1 7	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the outpu 0: buffers are full 1: 2K-bits available (32-bits for each of the pixels in a clause)  64: 128K-bits available (16 128-bit entries 64 pixels)	er position. t buffers. e 64	Formatted: Bullets and Numbering
Iame IXx_SQ_Export_count_rdy IXx_SQ_Export_Position IXx_SQ_Export_Buffer IXx_SQ_Export_Buffer	o <i>TP:</i> Con	Bits 1 7 7	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the outpu 0: buffers are full 1: 2K-bits available (32-bits for each of the pixels in a clause)  64: 128K-bits available (16 128-bit entries 64 pixels) 65-127: RESERVED	er position. t buffers. e 64 for each of	
Iame Xx_SQ_Export_count_rdy Xx_SQ_Export_Position Xx_SQ_Export_Buffer Xx_SQ_Export_Buffer 25.1.1126.1.10_SQ to Donce every clock, the fetch ur	o <i>TP: Con</i>	Bits 1 1 7 strol bus e sequencer on v	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the outpu 0: buffers are full 1: 2K-bits available (32-bits for each of the pixels in a clause)  64: 128K-bits available (16 128-bit entries 64 pixels)	er position. t buffers. e 64 for each of a in the GPRs	
ame Xx_SQ_Export_count_rdy Xx_SQ_Export_Position Xx_SQ_Export_Buffer 5.1.1126.1.10_SQ to nce every clock, the fetch ur ready or not. This way the s	o <i>TP: Con</i> bit sends to the	Bits 1 1 1 r r r r r r r r r r r r r r r r	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the output 0: buffers are full 1: 2K-bits available (32-bits for each of the pixels in a clause)  64: 128K-bits available (16 128-bit entries 64 pixels) 65-127: RESERVED	er position. t buffers. e 64 for each of for each of a in the GPRs he sequencer	
lame Xx_SQ_Export_count_rdy Xx_SQ_Export_Position Xx_SQ_Export_Buffer Xx_SQ_Export_Buffer Xx_SQ_Export_Buffer Conce every clock, the fetch ur ready or not. This way the s lso provides the instruction a	o <i>TP: Con</i> bit sends to the	Bits 1 1 1 r r r r r r r r r r r r r r r r	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the outpu 0: buffers are full 1: 2K-bits available (32-bits for each of the pixels in a clause)  64: 128K-bits available (16 128-bit entries 64 pixels) 65-127: RESERVED	er position. t buffers. e 64 for each of a in the GPRs he sequencer	
lame iXx_SQ_Export_count_rdy iXx_SQ_Export_Position iXx_SQ_Export_Buffer iXx_SQ_Export_Buffer iXx_SQ_Export_Buffer iXx_SQ_Export_Buffer iXx_SQ_Export_Buffer iXx_SQ_Export_Buffer iXx_SQ_Export_Support iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export iXx_SQ_Export_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXx_SQ_Export iXX_SQ_Export iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPORT iXX_SQ_EXPO	o <i>TP: Con</i> bit sends to the	Bits 1 1 1 r r r r r r r r r r r r r r r r	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the outpu 0: buffers are full 1: 2K-bits available (32-bits for each of the pixels in a clause)  64: 128K-bits available (16 128-bit entries 64 pixels) 65-127: RESERVED	er position. t buffers. e 64 for each of a in the GPRs he sequencer	
Ame Xx_SQ_Export_count_rdy Xx_SQ_Export_Position Xx_SQ_Export_Buffer Xx_SQ_Export_Buffer 25.1.1126.1.10_SQ to Droce every clock, the fetch ur is ready or not. This way the so lso provides the instruction a the fetch return data.	Direction SXx→SQ SXx→SQ SXx→SQ O $TP: Con bit sends to the equencer can nd constants Direction TPx→SQ$	Bits 1 1 r r r r r r r r r r r r r r r r r	Description Raised by SX0 to indicate that the followin reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the outpu 0: buffers are full 1: 2K-bits available (32-bits for each of the pixels in a clause)  64: 128K-bits available (16 128-bit entries 64 pixels) 65-127: RESERVED	er position. t buffers. e 64 for each of a in the GPRs he sequencer	
Ame Xx_SQ_Export_count_rdy Xx_SQ_Export_Position Xx_SQ_Export_Buffer Xx_SQ_Export_Buffer 25.1.1126.1.10 SQ to Conce every clock, the fetch ures are ready or not. This way the s Iso provides the instruction a the fetch return data. Name TPX_SQ_data_rdy TPX_SQ_clause_num	Direction         SXx→SQ         SXx→SQ         SXx→SQ         SXx→SQ         ist sends to the equencer can nd constants         Direction         TPx→SQ         TPx→SQ	Bits 1 1 7 find bus e sequencer on v update the fetch for the fetch to e Bits 1 3	Description         Raised by SX0 to indicate that the following reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the output 0: buffers are full         1: 2K-bits available (32-bits for each of the pixels in a clause)            64: 128K-bits available (16 128-bit entries 64 pixels)         65-127: RESERVED         which clause it is now working and if the date to counters for the reservation station fifos. Texecute and the address in the register file         Description         Data ready         Clause number	a in the GPRs he sequencer where to write	
s ready or not. This way the s also provides the instruction a the fetch return data. Name TPx_SQ_data_rdy	Direction SXx→SQ SXx→SQ SXx→SQ O $TP: Con bit sends to the equencer can nd constants Direction TPx→SQ$	Bits 1 1 7 strol bus e sequencer on v n update the fetch for the fetch to e Bits 1	Description         Raised by SX0 to indicate that the following reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the output 0: buffers are full         1: 2K-bits available (32-bits for each of the pixels in a clause)            64: 128K-bits available (16 128-bit entries 64 pixels)         65-127: RESERVED         which clause it is now working and if the date to counters for the reservation station fifos. Texecute and the address in the register file         Description         Data ready	a in the GPRs he sequencer where to write	

Fetch instruction sent over 4 clocks

LOD correct 3 bits per comp 2 components per quad

LOD correct 3 bits per comp 2 components per quad

Last instruction of the clause

Pixel mask 1 bit per pixel

Pixel mask 1 bit per pixel

Write phase signal

SQ\_TP0\_lod\_correct SQ\_TP0\_pmask SQ\_TP1\_lod\_correct SQ\_TP1\_lod\_correct SQ\_TP1\_pmask SQ\_TP2\_lod\_correct SQ→TP2 6 LOD correct 3 bits per comp 2 components per quad SQ\_TP2\_pmask SQ\_TP3\_lod\_correct SQ→TP2 4 Pixel mask 1 bit per pixel SQ→TP3 6 LOD correct 3 bits per comp 2 components per quad SQ\_TP3\_pmask SQ\_TPx\_clause\_num  $SQ \rightarrow TP3$ Pixel mask 1 bit per pixel 4  $SQ \rightarrow TPx$ 3 Clause number SQ\_TPx\_write\_gpr\_index Index into Register file for write of returned Fetch Data SQ->TPx 7

24

1

2

6

4

6

4

SQ\_TPx\_instuct

SQ\_TPx\_end\_of\_clause SQ\_TPx\_phase

 $\mathsf{SQ}{\rightarrow}\mathsf{TPx}$ 

SQ→TPx

SQ→TPx

SQ→TP0

SQ→TP0

SQ→TP1

SQ→TP1

Exhibit 2022 doc R400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* PM01/04/02 02:67 PM12/06/01 02:27 PM

AMD1044\_0257079

ATI Ex. 2106 IPR2023-00922 Page 168 of 223

	ORIGINATE [	DATE	EDIT D	DATE		DOCUMENT-REV. N	UM.	PAGE	1	
	24 September		4 Septemb		157	GEN-CXXXXX-REV		43 of 47		
25 1 1 2 2 6	5.1.11 TP to		vturo otoll	20021	11			<b>4</b> -	1	Formatted: Bullets and Numbering
	s this signal to the ximum of 3 clocks			er is tu	uli. The	SQ is going to send it to t	ne SP X d	clocks after		
Name		Discution			Densis	- 4 <b>:</b>				
TP_SQ_fe	tch_stall	Direction TP→ SQ	I C		Descrip Do not s	send more texture request if a	asserted			
			· · · ·							
<del>25.1.13</del> 26	5.1.12 SQ to	SP: Te	xture stall					4	150	Formatted: Bullets and Numbering
Name		Direction		Bits I	Descrip	otion				
SQ_SPx_f	etch_stall	SQ→SPx				send more texture request if a	asserted			
										Formatted: Bullets and Numbering
<del>25.1.14</del> 26	5.1.13 SQ to	SP: GF	PR, and Pa	aram	neter d	cache control <u>and a</u>	uto cou	nter *		Formatted: Builets and Numbering
Name		Directio	n I	Bits	Descri	ption				
SQ_SPx_v		SQ→SP	x .	7	Write a	iddress				
	pr_rd_addr pr_re_addr	SQ→SP SQ→SP			Read a Read E	address Fnable				
SQ_SPx_c	pr_we_addr	SQ→SP	x	1	Write E	Enable for the GPRs				
SQ_SPx_g	gpr_phase_mux	SQ→SP	x   :			hase mux (arbitrates betwe and writes)	en inputs,	ALU SRC		
SQ_SPx_c	hannel_mask	SQ→SP	x			annel mask				
SQ_SP0_p		SQ→SP				kel mask				
	pixel_mask pixel_mask	SQ→SP SQ→SP		4		kel mask kel mask				
SQ_SP3_p	pixel_mask	SQ→SP	3 4	4	The pix	kel mask				
	pc_we_addr ]pr_input_mux	SQ→SP SQ→SP		1 2		Enable for the parameter cack the phase mux selects the		e talle from		
	pr input mux	00-01	^   f		which	source to read from: Inter				
SO SPy i	ndex_count	SQ→SP	×			autogen counter. count, common for all shader	ninos			
JQ_JFX_I	ndex_count	<u>oq</u> →or	<u>^</u> .	121	index c	count, common for an shader	pipes			
				<b>n</b> -		· · · · · · · · · · · · ·				
Exhibit 2022.docR400_Se	aquencer.doc 67798 Bytes	*** © ATI	Confidential	. Ketei 57 PM12/06	erence ( 16/01-02:27 F	Copyright Notice on Cove	r Page ©	*** 09/04/15 12:51	4	

ATI Ex. 2106 IPR2023-00922 Page 169 of 223

		ORIGINATE DAT	E EDIT D	ATE	R400 Sequencer Specification	PAGE	
	~~UU	24 September, 200	01 <u>4 Septembe</u>			44 of 47	
4	25 1 1526	114 00 4- 00					Formatted: Bullets and Numbering
	<del>29.1.19</del> 20	<u>.1.14_</u> SQ to SF	-x: Instruction	S			
1	Name		Direction	Bits	Description		7
	SQ_SPx_in	nstruct_start	SQ→SPx	1	Instruction start		
	SQ_SP_ins	struct	SQ→SPx	20	Instruction sent over 4 clocks		

SQ_SP_instruct	SQ→SPx	20	Instruction sent over 4 clocks
SQ_SPx_stall	SQ→SPx	1	Stall signal
SQ_SPx_export_count	SQ→SPx	3	Each set of four pixels or vectors is exported over eight clocks. This field specifies where the SP is in that sequence.
SQ_SPx_export_last	SQ→SPx	1	Asserted on the first shader count of the last export of the clause
SQ_SP0_export_pvalid	SQ→SP0	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP0_export_wvalid	SQ→SP0	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP1_ export_pvalid	SQ→SP1	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP1_ export_wvalid	SQ→SP1	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP2_ export_pvalid	SQ→SP2	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP2_ export_wvalid	SQ→SP2	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors
SQ_SP3_ export_pvalid	SQ→SP3	4	Result of pixel kill in the shader pipe, which must be output for all pixel exports (depth and all color buffers). 4x4 because 16 pixels are computed per clock
SQ_SP3_ export_wvalid	SQ→SP3	2	Specifies whether to write low and/or high 32-bit word of the 64-bit export data from each of the 16 pixels or vectors

# 25.1.1626.1.15\_SP to SQ: Constant address load

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

	<del>25.1.17</del> 26.1.16  SQ to	SPx: constant	broadcast
--	-----------------------------------	---------------	-----------

Name		Direction	Bits	Description	22520
SQ_SP	x_constant	SQ→SPx	128	Constant broadcast	0.0254.05

Exhibit 2022 doc R400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 09/04/15 12:57

AMD1044\_0257081

ATI Ex. 2106 IPR2023-00922 Page 170 of 223

۰.

Formatted: Bullets and Numbering

AN	ORIGINATE DA	TE	EDIT DAT	E	DOCUMENT-REV. NUM.	PAGE		
	24 September, 2001 <u>4 September, 20157</u>		GEN-CXXXXX-REVA	45 of 47		C		
<del>5.1.18</del> 26.	1.17 SP0 to 3	SQ: Kill ve	ector load	1		*	-	Formatted: Bullets and Numbering
Name		irection	Bits	Descrip	otion			
SP0_SQ_kil		P0→SQ	4	Kill vect	tor load			
SP1_SQ_kil		P1→SQ	4	Kill vect				
SP2_SQ_kil		P2→SQ P3→SQ	4	Kill vect				
SP3_SQ_kil				Kill vect	oriodu			Formatted: Bullets and Numbering
5 <u>.1.1926.</u>	<u>1.18_</u> SQ to C	P: RBBM	l bus			*		
Name		irection	Bits	Descrip				
SQ_RBB_r	-	Q→CP	1	Read S				
SQ_RBB_r		Q→CP	32	Read D				
SQ_RBBM SQ_RBBM		Q→CP Q→CP	1	Optiona	ai ime (Optional)			
			i	Real-11	ine (Optional)			Formatted: Bullets and Numbering
<u>5.1.2026.</u>	<u>1.19_</u> CP to S	Q: RBBM	1 bus			4	1	
Name	D	irection	Bits	Descrip	otion			
rbbm_we		P→SQ	1	Write E				
rbbm_a		P→SQ	18		s Upper Extent is TBD			
rbbm_wd		P→SQ	32	Data Data				
rbbm_be rbbm_re		P→SQ P→SQ	4	Byte Er Read E				
rbb rs0		r⊸sQ P→SQ	1		Return Strobe 0			
rbb_rb1		P→SQ	1		Return Strobe 1			
THE			32				1002102/02/02	
rbb_rd0	C	P→SQ	32	Read D	Data O			
rbb_rd0 rbb_rd1		P⊸SQ P→SQ	32	Read L				
rbb_rd1 RBBM_SQ	C	P→SQ P→SQ	32 1	Read E Soft Re	Data 0	4	1	Formatted: Bullets and Numbering
rbb_rd1 RBBM_SQ 6.27. Exa 6.1.127.1 PA sends a • state po • space v • also be shader • The ver • the	soft_reset C amples of pr .1 Sequenced vector of 64 vertice binter as well as tag vas allocated in the effore the vector is program (using the tex program is asso SEQ then accessed	$P \rightarrow SQ$ $P \rightarrow SQ$ Ogram e r Control ( r c	32 1 execution of a Vector rertex indices in cache is set cache is set cache for transfor RE, the CP r oaded when e for this share	Read L Soft Re Soft Re Or of Vo - 32 bits nt along w prmed pos nas loade we receiv der using	Pata 0 eset Pata 0 Pata	h the vertex		Formatted: Bullets and Numbering
rbb_rd1 RBBM_SQ 6.27. Exa 6.1.127.1 PA sends a state po space v also be shader The ver the seq SEQ arbitra at this p the arbit	amples of pr <u>.1</u> Sequenced vector of 64 vertice inter as well as tag vas allocated in the fore the vector is program (using the tex program is ass SEQ then accesses juencers by the RB attes between the Pi point the vector is re-	$P \rightarrow SQ$ $P \rightarrow SQ$ Ogram e r Control ( r c	32 1 execution of a Vector vertex indices in cache is ser- che for transfor RE, the CP H oaded when h e for this share e CP is done d the Vertex F or to be transfor	Read E Soft Re Soft Re Or of Vi - 32 bits nt along w prmed pos nas loade we receiv der using loading th FIFO – ba IFO	Pata 0 eset Pata 0 Pata	h the vertex all priority		Formatted: Bullets and Numbering
rbb_rd1 RBBM_SQ 6.27. Exa 6.1.127.1 PA sends a • state po • space v • also be shader • The ver • the seq SEQ arbitra • at this p • the arbitron othing SEQ allocat • the num state po	amples of pr <u>amples of pr</u> <u>1</u> Sequenced vector of 64 vertice binter as well as tag vas allocated in the effore the vector is program (using the tex program is ass SEQ then accesses uencers by the RB attes between the Pi boint the vector is re- ter is not going to se else to do (ie no pi tes space in the SF	P→SQ P→SQ Control ( es (actually v) into position cacl sent to the F me MH?) uned to be lo s the IS base BM when the exel FIFO and emoved from veloct a vector xels are in th register file f red by the province of the red by the province of the velocity of the province of the province of the velocity of the province of the provinc	32 1 execution of a Vector vertex indices in cache is see the for transfor RE, the CP f oaded when e for this share e CP is done d the Vertex F to the Vertex F to the Vertex F to the Vertex F to the Vertex F or to be transfor to pixel fifo). for index data ogram is stor vertices	Read L Soft Re Soft Re - 32 bits. - 32 bits.	Pata 0 eset ertices findex for 2048 bit total) to the RE's with vertices sition before the vector was sent id the global instruction store with e the vertex vector. the local state pointer (provided to a ne program) sically the Vertex FIFO always has the parameter cache is full unless the PRs used by the program cal state register, which is accessed	h the vertex all priority ne pipe as		Formatted: Bullets and Numbering
rbb_rd1 RBBM_SQ 6-27. Exa 6.1.127.1 PA sends a state po space v also be shader The ver the seq SEQ arbitra at this p the arbitra state po SEQ allocal the num state po SEQ sends EQ sends the 64 v RF	amples of pr <u>amples of pr</u> <u>amples of pr}</u> <u>amples of pr}</u> <u>amples of pr}</u> <u>amples of pr}</u> <u>amples of pr}</u> <u>amples of pr} <u>amples of pr} <u>am</u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u></u>	$P \rightarrow SQ$ $P \rightarrow SQ$ $P \rightarrow SQ$ Control (a) Control (a) Cont	32 1 of a Vection of a Vection rertex indices in cache is series the for transformer for the series the for transformer for the series e CP is done of the Vertex F or to be transformer for the Vertex F or to be transformer the Vertex F or index data rogram is stor- vertices ce in the regi e over the RE is written the f s written the f	Read L Soft Re Soft Re - 32 bits. nt along w ormed pos has loade we receiv der using loading the FIFO – ba IFO formed if the a plus GP red in a lo ster file has SP inter over 4 cy irst cycle hird cycle	Pata 0 Pata 10 Pata 10	h the vertex all priority ne pipe as d using the		Formatted: Bullets and Numbering

ATI Ex. 2106 IPR2023-00922 Page 171 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20157		46 of 47

- the index is written to the least significant 32 bits (floating point format?) (what about compound indices) of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
- 5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
  - the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
- TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   TSM0 was first selected by the TSM arbiter before it could start
- 7. all instructions of fetch clause 0 are issued by TSM0
- the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
  - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
  - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
- 9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
- 10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
- the control packet continues to travel down the path of reservation stations until all clauses have been executed
   position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
  - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
    - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA
  - the ASM arbiter will prevent a packet from starting an exporting clause if the position export FIFO is full
  - parameter data is exported in clause 7 (as well as position data if it was not exported earlier)
  - parameter data is sent to the Parameter Cache over a dedicated bus
  - the SEQ allocates storage in the Parameter Cache, and the SEQ deallocates that space when there is no longer a need for the parameters (it is told by the PA when using a token).
  - the ASM arbiter will prevent a packet from starting on ASM7 if the parameter cache (or the position buffer if position is being exported) is full
- 12. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

## 26.1.227.1.2 Sequencer Control of a Vector of Pixels

- 1. As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP
- At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
- 2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
  - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
  - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
- 3. SEQ arbitrates between Pixel FIFO and Vertex FIFO when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
- 4. SEQ allocates space in the SP register file for all the GPRs used by the program
  - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
  - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
- 5. SEQ controls the transfer of interpolated data to the SP register file over the RE\_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.

Exhibit 2022 doc R400\_Sequencer.doc 67798 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*
<u>09/04/15 12:57</u>
<u>PM01/04/02 02:67-PM</u>1206/01-02:27-PM

Formatted: Bullets and Numbering

AMD1044\_0257083

ATI Ex. 2106 IPR2023-00922 Page 172 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 2001	4 September, 20157	GEN-CXXXXX-REVA	47 of 47	
6. SEQ constructs a control packet for the	e vector and sends it to the	first reservation station (the FIFO in	front of	
-		s/state machines for vertices and fo	-	
<ul> <li>the control packet contains the sta</li> <li>all other information (such as guad</li> </ul>		ase pointer, and the LOD correctior els in a separate FIFO	n bits	
<ul> <li>TSM0 accepts the control packet and t</li> <li>TSM0 was first selected by the TS</li> </ul>	fetches the instructions for t	etch clause 0 from the global instru	ction store	
8. all instructions of fetch clause 0 are iss				
9. the control packet is passed to the nex	t reservation station (the FI	FO in front of ALU state machine 0,	or ASM0	
<ul><li>index for the fetch data to the TU,</li><li>once the TU has written all the dat</li></ul>	which will write the data to t a for a particular clause to t count greater than zero inc	to complete; it passes the register i the RF as it is received the register files, it increments a cou licates that the ALU state machine o	inter that is	
<ol> <li>ASM0 accepts the control packet (afte clause 0 from the global instruction stop</li> </ol>		<i>I</i> arbiter) and gets the instructions for	or ALU	
11. all instructions of ALU clause 0 are iss station (the FIFO in front of fetch state			ervation	
<ul> <li>12. the control packet continues to travel d</li> <li>pixel data is exported in the last Al</li> <li>it is sent to an output FIFO wh</li> <li>the ASM arbiter will prevent a</li> </ul>	U clause (clause 7) here it will be picked up by t	he render backend	n executed	
<ol> <li>after the shader program has complete shader program</li> </ol>	ed, the SEQ will free up the	GPRs so that they can be used by	another	
26.1.327.1.3 Notes			«	Formatted: Bullets and Numbering
14. The state machines and arbiters will o threads or stall.	perate ahead of time so tha	t they will be able to immediately sta	art the real	
15. The register file base pointer for a vect instruction store base pointer does not pointer is only different for each state a	- this is because the RF pe	pinter is different for all threads, but		
			4-	Formatted: Bullets and Numbering
27.28. Open issues				
Need to do some testing on the size of the static).	e register file as well as on	the register file allocation method	(dynamic VS	
Saving power?				
	Confidential Deferences	Converget Nation on Onver Dama	S	
Exhibit 2022.docR400_Sequencer.doc 67798 Bytes*** ⓒ AII	Confidential. Reference ( PM01/04/02-02:67-PM12/06/01-02:27-f	Copyright Notice on Cover Page © ™	9 *** <u>09/04/15 12:57</u>	

ATI Ex. 2106 IPR2023-00922 Page 173 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	4 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	1 of 50
Author: Laur	rent Lefebvre			•
1 197				
Issue To:		Copy No:		
	R400 S	equencer Spe	ecification	
		8 8		
		SQ		
		Version 1. <u>7</u> 6		
		α του ει του ει του κ.κ. κ. κ. κ.κ.κ. ποποκ		
			cer block (SEQ). It provides an ov	
	nd provides internal st		lso describes the block interfaces,	Internal Sub-
	UPDATED FIELDS:			
Document Locatio Current Intranet S		erforce\r400\doc_lib\design\b D Sequencer Specification	olocks\sq\R400_Sequencer.doc	
	Neme/Dent	APPROVALS	Circulture (Date	
	Name/Dept		Signature/Date	
Demender				
Remarks:				
				тилт
			DENTIAL INFORMATION	
			RIZED USE OR DISCLOS	
			rial in this document constitutes ar provide notice that ATI owns a co	
unpublished work. T	The copyright notice	is not an admission that	publication has occurred. This we of this document may be used, re	vork contains
			sion of ATI Technologies Inc."	eproduceu, or
Exhibit 2023.docR400_Sequencer.c	doc 71499 Bytes*** © ATI	Confidential. Reference Co	opyright Notice on Cover Page ©	*** 01/04/02-02:57-PM

ATI 2023 LG v. ATI IPR2015-00325

AMD1044\_0257085

ATI Ex. 2106 IPR2023-00922 Page 174 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		2 of 50

Table Of Contents	
1. OVERVIEW	
1.1 Top Level Block Diagram	
1.2 Data Flow graph (SP)	
1.3 Control Graph	
2. INTERPOLATED DATA BUS	
3. INSTRUCTION STORE	
4. SEQUENCER INSTRUCTIONS	
5. CONSTANT STORES	
5.1 Memory organizations	<u> 1746</u>
5.2 Management of the re-mapping tables	
5.2.1 Dirty bits	
5.2.2 Free List Block	
5.2.3 De-allocate Block	
5.2.4 Operation of Incremental model	
5.3 Constant Store Indexing	
5.4 Real Time Commands	
5.5 Constant Waterfalling	<u>2219</u>
6. LOOPING AND BRANCHES	2320
6.1 The controlling state	
6.2 The Control Flow Program	
6.3 Data dependant predicate instructions.	
6.4 HW Detection of PV,PS	
6.5         Register file indexing           6.6         Predicated Instruction support for Texture clauses	2724
6.7 Debugging the Shaders	
6.7.1 Method 1: Debugging registers	
6.7.2 Method 2: Exporting the values in the GPRs (12)	
7. PIXEL KILL MASK	2020
9. REGISTER FILE ALLOCATION	
10. FETCH ARBITRATION.	
11. ALU ARBITRATION	
12. HANDLING STALLS	***************************************
13. CONTENT OF THE RESERVATION STATION FIFOS	3027
14. THE OUTPUT FILE.	
15. IJ FORMAT	
15.1 Interpolation of constant attributes	
16. STAGING REGISTERS	3128
17. THE PARAMETER CACHE	
18. VERTEX POSITION EXPORTING	
19. EXPORTING ARBITRATION	3430
20. EXPORT TYPES	
20.1 Vertex Shading	

Exhibit 2023 docR400\_Sequencer.dec 71499 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 01/04/02-02:57-PM

AMD1044\_0257086

ATI Ex. 2106 IPR2023-00922 Page 175 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE		
	24 September, 2001	4 September 20154	GEN-CXXXXX-REVA	3 of 50		
20.2 Pixe	ixel Shading					
21. SPECIAL INTERPOLATION MODES						
21.1 Real time commands						
21.2 Sprites/ XY screen coordinates/ FB information						
	21.3     Auto generated counters.       21.3.1     Vertex shaders					
21.3.2						
***************************************						
	***************************************			~~~~~~~		
26.1.1						
26.1.2						
26.1.3		s				
26.1.4						
26.1.5						
26.1.6						
26.1.7						
26.1.8	SQ to SX: Control bu					
26.1.9	SX to SQ : Output file					
26.1.10	SQ to TP: Control bu					
26.1.11	TP to SQ: Texture sta					
26.1.12	SQ to SP: Texture sta					
26.1.13	SQ to SP: GPR, Para	meter cache control and	d auto counter			
26.1.14	SQ to SPx: Instructio	ns				
26.1.15						
26.1.16						
26.1.17						
26.1.18						
26.1.19						
			*****			
27.1.1						
27.1.2						
27.1.3						

Exhibit 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 01/04/02-02:67-PM

AMD1044\_0257087

ATI Ex. 2106 IPR2023-00922 Page 176 of 223

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20154		4 of 50
		*****		
. OVER	VIEW	******	**********	6
1 Top Le	evel Block Diagram	*****		8
.2 Data F	low graph	*****	*****	
.3 Contro	Graph			
INTER	POLATED DATA BU	JS	******	
SEQU	ENCER INSTRUCTION	ONS	*******	
			*****	
2 Manac	ement of the re-man	ning tables		16
-	· ·			
5.2.3	De-allocate Block			
5.2.4 (	Operation of Increment	ntal model		
-3 Consta	ant Store Indexing	****		
.4 Real T	ime Commands			
LOOP	ING AND BRANCHE	S	*****	
				20
				20
3 Data d	lenendant predicate i	nstructions		
	atection of PV PS			
5 Regist	er file indexing			23
6 Predic	ated Instruction sunn	ort for Texture clause	S	24
7 Debug	aina the Shaders		~	24
			s (12)	
			5 (12)	
			IFOS	
5. JJ FOF	RMAT	*****		
			*****	
	RTING ARBITRATIC			
			*****	
			*****	
			*****	
0.2 Sprit	es/ XY screen coordi	nates/ FB information		

Exhibit 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 01/04/02-02:67-PM

AMD1044\_0257088

ATI Ex. 2106 IPR2023-00922 Page 177 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
200	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	5 of 50
20.3.1	Vertex shaders	Estrumy 20020		
20.3.2	Pixel shaders			
		*******	*****	
	ameter cache synchron			
	DDRESS IMPORTS			32
	STERS			
3.1 Con	ntrol			
	itext			
N * 500* MIGH MINP *60	JG REGISTERS	******		<b>34</b> 34
000		*****		
		******		
25 1 1-	-SC to SQ : IJ Contr	ol bus		
25.1.2	-SQ to SP: Interpolato	r bus		
25.1.3		Mux select		
25.1.4	-SQ to SP: Parameter Cache Read control bus			
25.1.5	SQ to SX: Parameter	Cache Mux control Bus		
25.1.6		gister Data		
25.1.7-	PA to SQ : Vertex interface			
25.1.8	-SQ to CP: State report			
25.1.9	-SQ to SX: Control but	S		
25.1.10		-control		
25.1.11		\$		
25.1.12		<u>،اا</u>		
25.1.13	SQ to SP: Texture sta			
25.1.14		Parameter cache contro		41
25.1.15	SQ to SPx: Instruction	n <del>s</del>		
25.1.16		ddress load		42
25.1.17	-SQ to SPx: constant I	proadcast		42
25 1 18	SP0 to SQ: Kill vector	· load		43
25 1 19				43
25 1 20	CP to SQ: RBBM bus			43
		EXECUTIONS		
26.1.1-	-Sequencer Control of	a Vector of Vertices		
26.1.2	-Sequencer Control of	a Vector of Pixels		
26-1-3_	Notes			

Exhibit 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257089

ATI Ex. 2106 IPR2023-00922 Page 178 of 223

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20154		6 of 50
Revision	Changes:			
<b>Rev 0.1 (Laur</b> Date: May 7, 2		First	draft.	
Rev 0.2 (Laure Date : July 9, 2 Rev 0.3 (Laure Date : August Rev 0.4 (Laure Date : August Rev 0.5 (Laure Date : Septem Rev 0.6 (Laure Date : Septem	2001 ent Lefebvre) 6, 2001 ent Lefebvre) 24, 2001 ent Lefebvre) ber 7, 2001 ent Lefebvre)	SP. / Revi Augu Adde file a flow Adde Chai	nged the interfaces to reflect the chan Added some details in the arbitration s ewed the Sequencer spec after the m ist 3, 2001. ed the dynamic allocation method for and an example (written in part by V of pixels/vertices in the sequencer. ed timing diagrams (Vic) nged the spec to reflect the no itecture. Added interfaces.	ection. neeting on or register /ic) of the
Rev 0.7 (Laure Date : October Rev 0.8 (Laure Date : October Rev 0.9 (Laure	ent Lefebvre) 5, 2001 ent Lefebvre) <sup>c</sup> 8, 2001	Adde store data Chai flexit	ed constant store management, i e management, control flow manage dependant predication. nged the control flow method to ole. Also updated the external interface prorated changes made in the 10/18/	ment and be more es.
Date : October Rev 1.0 (Laure Date : October Rev 1.1 (Laure Date : October	ent Lefebvre) 19, 2001 ent Lefebvre)	the regis Refir Adde	meeting. Added a NOP instruction, conditional_execute_or_jump. Adde sters. red interfaces to RB. Added state regi- ed SEQ→SP0 interfaces. Chang ision. Changed VGT→SP0 interfac	ed debug sters. ed delta
Rev 1.2 (Laure Date : Novemt Rev 1.3 (Laure Date : Novemt Rev 1.4 (Laure Date : Decemt	ent Lefebvre) ber 16, 2001 ent Lefebvre) ber 26, 2001 ent Lefebvre) ber 6, 2001	Adde Adde Adde the V man	nods added. faces greatly refined. Cleaned up the s ed the different interpolation modes. ed the auto incrementing counters. /GT→SQ interface. Added content or agement. Updated GPRs.	changed
Rev 1.5 (Laure Date : Decemt Rev 1.6 (Laure Date : January	per 11, 2001 ent Lefebvre)	direc cons sync Adde deta call Adde	oved from the spec all interfaces the ty tied to the SQ. Added explan- tant management. Added hronization fields and explanation. ed more details on the staging regist il about the parameter caches. Cha instruction to a Conditionnal_call in ed details on constant managen ted the diagram.	ations on PA→SQ er. Added anged the nstruction.
Rev 1.7 (Laure Date : Februar		Adde	a Real Time parameter control in face. Updated the control flow section.	

Exhibit 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 01/04/02-02:67 PM

AMD1044\_0257090

ATI Ex. 2106 IPR2023-00922 Page 179 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	7 of 50	

#### 1. Overview

The sequencer is based on the R300 design. It chooses two ALU clauses and a fetch clause to execute, and executes all of the instructions in a clause before looking for a new clause of the same type. Two ALU clauses are executed interleaved to hide the ALU latency. Each vector will have eight fetch and eight ALU clauses, but clauses do not need to contain instructions. A vector of pixels or vertices ping-pongs along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation 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 sequencer also contains the shader instruction cache, constant store, control flow constants and texture state. The four shader pipes also execute the same instruction thus there is only one sequencer for the whole chip.

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

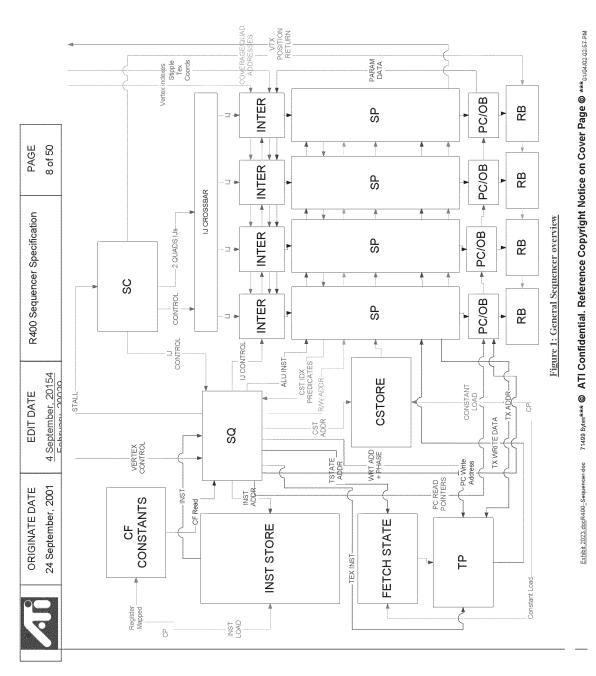
The vertex or pixel program specifies how many GPRs it needs to execute. The sequencer will not start the next vector until the needed space is available in the GPRs.

Exhibit 2023.doc R400\_Sequencer.doc 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 04/04/02/02/07-07

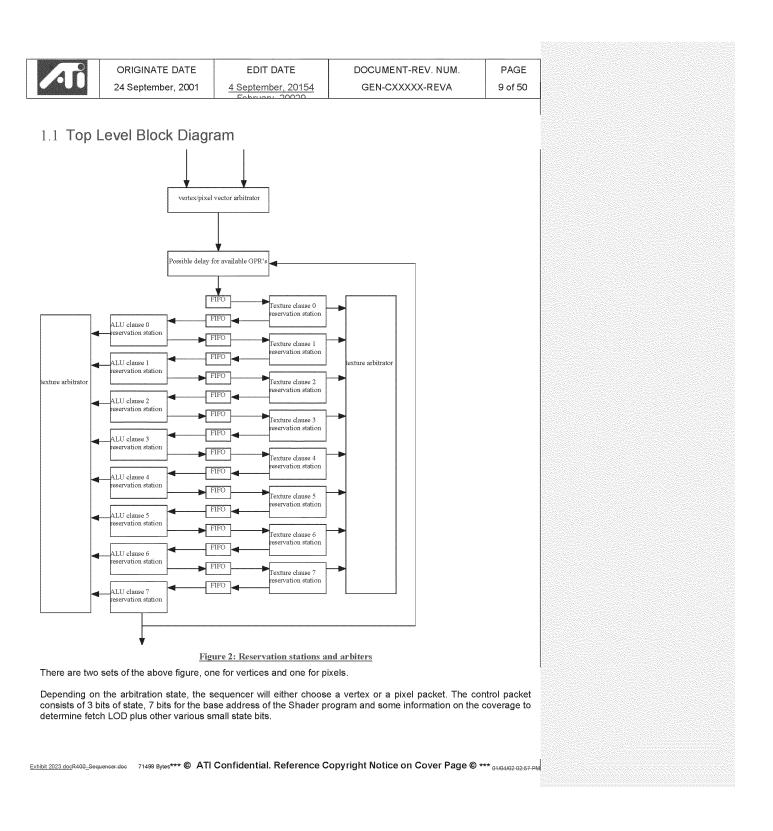
AMD1044\_0257091

ATI Ex. 2106 IPR2023-00922 Page 180 of 223





ATI Ex. 2106 IPR2023-00922 Page 181 of 223



ATI Ex. 2106 IPR2023-00922 Page 182 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		10 of 50

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator which interpolator, which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

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

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

On receipt of a command, the level 0 ALU machine first decrements the input FIFO 1 counter and then issues a complete set of level 0 shader instructions. For each instruction, the ALU state machine generates 3 source addresses, one destination address -and an instruction. Once the last instruction has been issued, the packet is put into FIFO 2.

There will always be two active ALU clauses at any given time (and two arbiters). One arbiter will arbitrate over the odd instructions (4 clocks cycles) and the other one will arbitrate over the even instructions (4 clocks cycles). The only constraints between the two arbiters is that they are not allowed to pick the same clause number as the other one is currently working on if the packet is not of the same type (render state).

If the packet is a vertex packet, upon reaching ALU clause 3, it can export the position if the position is ready. So the arbiter must prevent ALU clause 3 to be selected if the positional buffer is full (or can't be accessed). Along with the positional data, if needed the sprite size and/or edge flags can also be sent.

A special case is for multipass vertex shaders, which can export 12 parameters per last 6 clauses to the output buffer. If the output buffer is full or doesn't have enough space the sequencer will prevent such a vertex group to enter an exporting clause.

Multipass pixel shaders can export 12 parameters to memory from the last clause only (7).

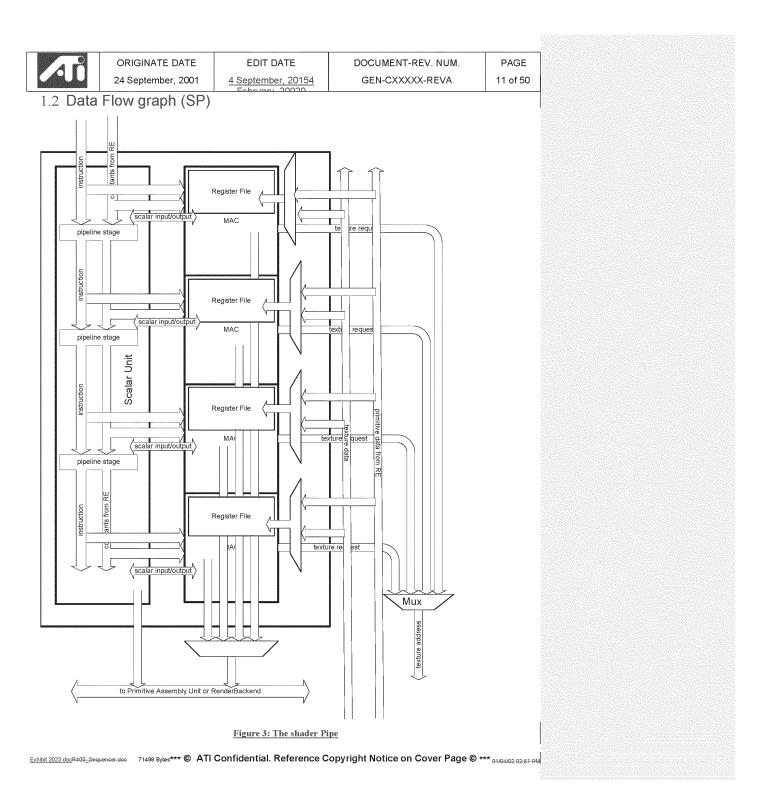
All other clauses process in the same way until the packet finally reaches the last ALU machine (7).

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

Atlb 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01/04/02-02:67-PM

AMD1044\_0257094

ATI Ex. 2106 IPR2023-00922 Page 183 of 223

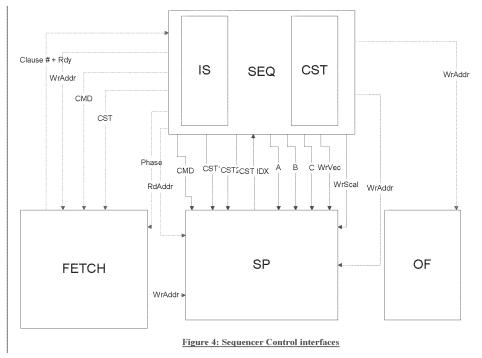


ATI Ex. 2106 IPR2023-00922 Page 184 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		12 of 50

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.

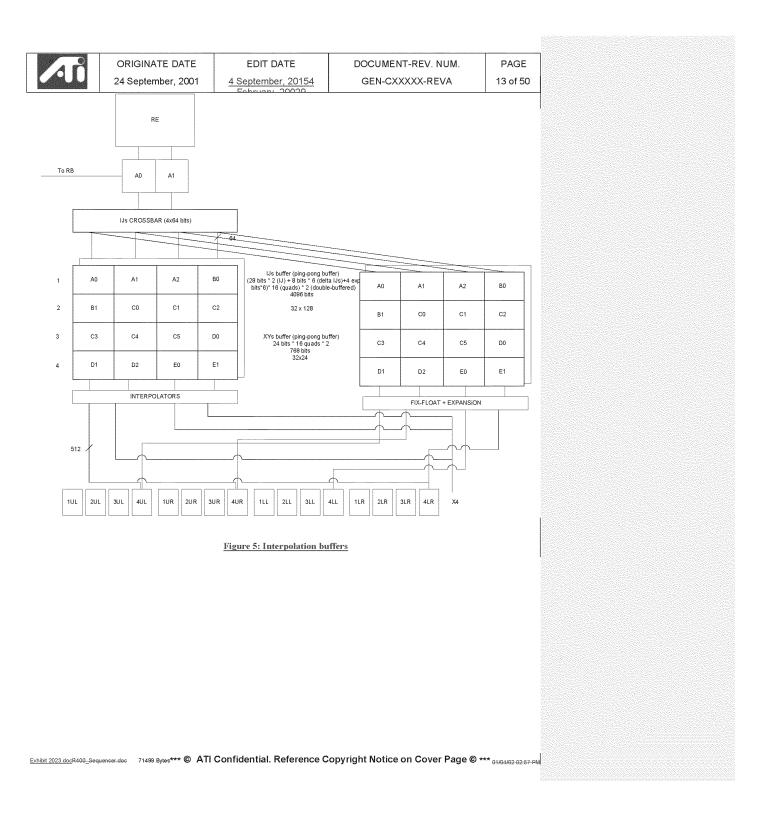
### 2. Interpolated data bus

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

the state of the s

AMD1044\_0257096

ATI Ex. 2106 IPR2023-00922 Page 185 of 223



ATI Ex. 2106 IPR2023-00922 Page 186 of 223 PROTECTIVE ORDER MATERIAL

		~									
		T23					V V 32- 48- 35 51	V V 36-52- 39 55	<pre>40- 56- 43 59</pre>	V V 44-60- 47 63	$\sim$
		T22					35 <	3 % <		> 44 4	
		T21					> 4 6	23 <sup>2</sup> 0 <	24- 27	31 <mark>2</mark> 8 <	
		T20					> ;	> 4-7	⇒ ∽ £	> 54 fc	
		T19							Ш	Ē	
		T18								8	
<b></b>	0	T17			ÿ₩	ХШ		ទ	ۍ ۲	ß	
PAGE	14 of 50	T9 T10 T11 T12 T13 T14 T15 T16 T17 T18 T19 T20 T21 T22 T23			ОШ	Ш				BO	Р2
		T15			С	Ш.	5	D2			Δ
ation		T14	≳Ճ	×8		≿8	ប៊	5	C5		
ecifica		T13	ā	D2		8	<u>B</u>				
R400 Sequencer Specification		T12	Б	D2		8	AO	A1	A2		
equen		T11		≿2	≿ રડ	000	É		Ш	ш	
2400 S		T10	Ē	5	C5	ŭ	ŕ			8	
		19		Q	C5			8	ۍ ۲	G	
	0154	Т8	≿ຬ	≳ 8	≿ ວ	×Ω				BO	Annema
EDIT DATE	iber, 2	17	S	S	5	8	5	D2			Ď
EDIT	4 September, 20154 Exheritory 20030	T6	ទ	8	5	C2	ប៊	5	C5		
	4	T5	×₽			×8	8				
ATE	24 September, 2001	Τ4	12			BO	Ao	A1	A2		
TE D/	her, 2	T3	<u>B</u>			BO	51 48 ×∕	55 <sup>2</sup> 2 ≺	26 ¦2 ≺	SöX	
IGINA	Septen	12	Χð	Υž	≿ସ୍		XY XY 32- 48- 35 51	XY XY 36-52- 39 55	XY 40-56- 59	XY XY 44- 60- 47 63	$\succ$
OR	24 8	È	AO	A1	A2		7 <del>1</del> 6 19		24- 27		$\times$
6		10 T	Ao	A1	A2		×°		≍ ∞ ≍	35 42 ₹	
	$\overline{\mathbf{x}}$	62940303030303030	dS 0	dS –	∿ SP	ч S р	dS o	<del>с</del> –	S ∽	ч С С	
			•	I	I	I		I	1	1 I	

Figure 6: Interpolation timing diagram

71499 Byses\*\*\* 
 ATI Confidential. Reference Copyright Notice on Cover Page 
 \*\*\*\*
 \*\*\*\*
 \*\*\*\*
 \*\*\*\*

# AMD1044\_0257098

Exhibit 2023.docR400\_Sequencer.doc

ATI Ex. 2106 IPR2023-00922 Page 187 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	15 of 50

Above is an example of a tile the sequencer might receive from the SC. The write side is how the data get stacked into the XY and IJ buffers, the read side is how the data is passed to the GPRs. The IJ information is packed in the IJ buffer 4 quads at a time or two clocks. The sequencer allows at any given time as many as four quads to interpolate a parameter. They all have to come from the same primitive. Then the sequencer controls the write mask to the GPRs to write the valid data in.

{ISSUE : Do we do the center + centroid approach using both IJ buffers?}

### 3. Instruction Store

There is going to be only one instruction store for the whole chip. It will contain 4096 instructions of 96 bits each.

It is likely to be a 1\_port memory; we use -1 clock to load the ALU instruction, 1 clocks to load the Fetch instruction, 1 clock to load 2 control flow instructions and 1 clock to write instructions.

The instruction store is loaded by the CP thru the register mapped registers.

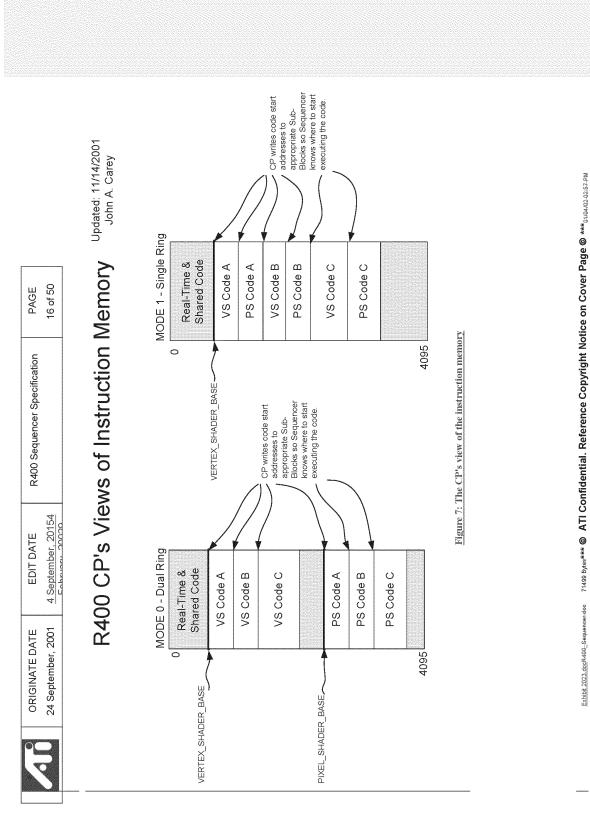
The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The wrap-around points are arbitrary and they are specified in the VS\_BASE and PIX\_BASE control registers. The VS\_BASE and PS\_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

Exhibit 2023.docR400\_Sequencer.doc 71498 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 04/04/02/02/07-07

AMD1044\_0257099

ATI Ex. 2106 IPR2023-00922 Page 188 of 223 PROTECTIVE ORDER MATERIAL



AMD1044\_0257100

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 2001	<u>4 September, 20154</u>	GEN-CXXXXX-REVA	17 of 50	

### 4. Sequencer Instructions

All control flow instructions and move instructions are handled by the sequencer only. The ALUs will perform NOPs during this time (MOV PV,PV, PS,PS) if they have nothing else to do.

## 5. Constant Stores

### 5.1 Memory organizations

A likely size for the ALU constant store is 1024x128 bits. The read BW from the ALU constant store is 128 bits/clock and the write bandwidth is 32 bits/clock (directed by the CP bus size not by memory ports).

The maximum logical size of the constant store for a given shader is 256 constants. Or 512 for the pixel/vertex shader pair. The size of the re-mapping table is 128 lines (each line addresses 4 constants). The write granularity is 4 constants or 512 bits. It takes 16 clocks to write the four constants. Real time requires 256 lines in the physical memory (this is physically register mapped).

The texture state is also kept in a similar memory. The size of this memory is 492x128128x192 bits. The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 46-32 lines (each line addresses 2-1 texture state lines in the real memory). The <u>CP</u> write granularity is 2-1 texture state lines (or 384-192 bits). The driver sends 512 bits but the CP ignores the top 428-320 bits. It thus takes 42-6 clocks to write the two texture states. Real time requires 32 lines in the physical memory (this is physically register mapped).

The control flow constant memory doesn't sit behind a renaming table. It is register mapped and thus the driver must reload its content each time there is a state change. Its size is 320\*32 because it must hold 8 copies of the 32 dwords of control flow constants and the loop construct constants must be aligned.

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

### 5.2 Management of the re-mapping tables

### 5.2.1 R400 Constant management

The sequencer is responsible to manage two re-mapping tables (one for the constant store and one for the texture state). On a state change (by the driver), the sequencer will broadside copy the contents of its re-mapping tables to a new one. We have 8 different re-mapping tables we can use concurrently.

The constant memory update will be incremental, the driver only need to update the constants that actually changed between the two state changes.

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space <u>+ the space allocated for Real Time</u>. In our case, since the logical address space is 512<u>and the reserved RT space can be up to 256 entries</u>, the memory must be of sizes  $\frac{1024-1280}{2}$  and above. Similarly the size of the texture store must be of 32\*2+32 = 96 entries and above.

### 5.2.15.2.2 Proposal for R400LE constant management

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ\_IDLE and PA\_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 9: De-allocation mechanismFigure 9: De-allocation mechanism, Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

Exhibit 2023.docR400\_Sequencer.dec 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01/04/02-02:67-PM

Formatted: Bullets and Numbering

AMD1044\_0257101

ATI Ex. 2106 IPR2023-00922 Page 190 of 223

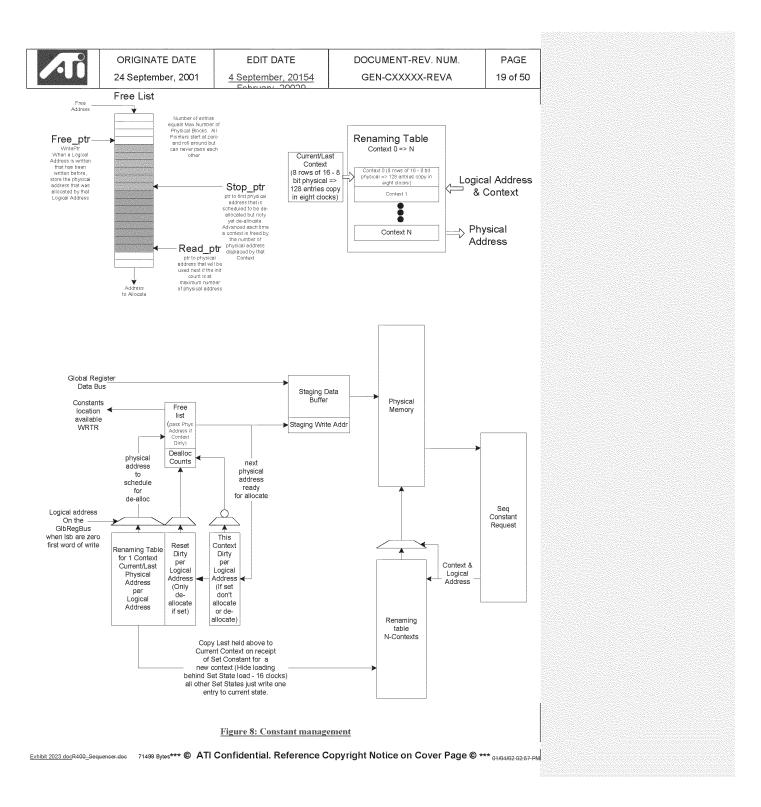
ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		18 of 50

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed). Be careful to set only those bits that the CURENT STATE IS USING (if for example the current state uses only 64 constants we set only lines 0 thru 15 to 1). This is ok to do so because the blocks are idle and thus the context has finished drawing.

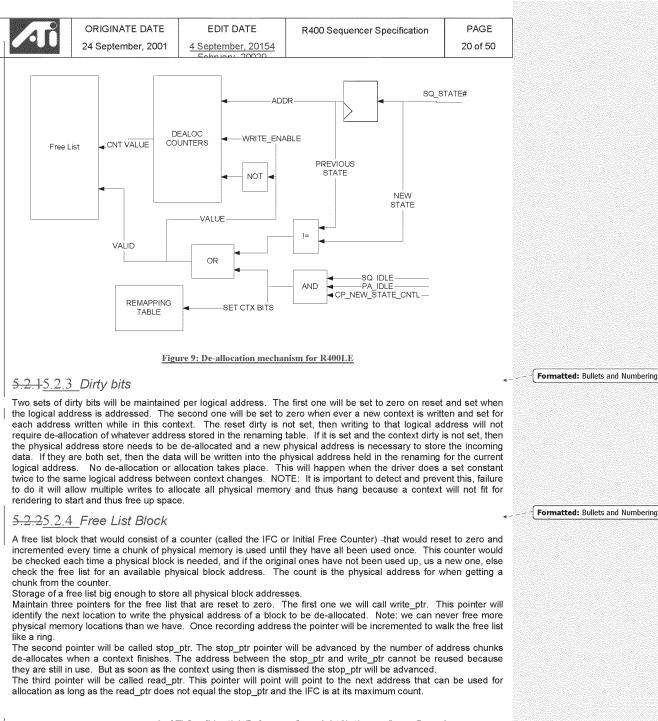
Exhibit 2023.doc R400\_Sequencer.doc 71499 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 01/04/02-02:67 PM

AMD1044\_0257102

ATI Ex. 2106 IPR2023-00922 Page 191 of 223



ATI Ex. 2106 IPR2023-00922 Page 192 of 223



thibit 2023.docR400\_Sequences.do 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 04.04/92-02-67-PM

AMD1044\_0257104

ATI Ex. 2106 IPR2023-00922 Page 193 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	]
	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	21 of 50	( <u></u>
					Formatted: Bullets and Numbering

5.2.35.2.5 De-allocate Block

This block will maintain a free physical address block count for each context. While in current context, a count shall be maintained specifying how many blocks were written into the free list at the write\_ptr pointer. This count will be reset upon reset or when this context is active on the back and different than the previous context. It is actually a count of blocks in the previous context that will no longer be used. This count will be used to advance the write\_ptr pointer to make available the set of physical blocks in one clock.

### 5.2.45.2.6 Operation of Incremental model

The basic operation of the model would start with the write\_ptr, stop\_ptr, read\_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context location. Then the set constant logical address with be loaded with a new physical address during the copy and if the reset dirty was set, the physical address it replaced in the renaming table would be entered at the write\_ptr pointer location on the free list and the write\_ptr will be incremented. The de-allocation counter for the previous context (eight) will be incremented. This as set states come in for this context one of the following will happen:

- 1.) No dirty bits are set for the logical address being updated. A line will be allocated of the free-list counter or the free list at read\_ptr pointer if read\_ptr != to stop\_ptr .
- 2.) Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write\_ptr and it is incremented along with the de-allocate counter for the last context.
- 3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

This process will continue as long as set states arrive. This block will provide backpressure to the CP whenever he has not free list entries available (counter at max and stop\_ptr == read\_ptr). The command stream will keep a count of contexts of constants in use and prevent more than max constants contexts from being sent.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

Now preferable when the constant context leaves the last ALU clause it will be sent to this block and compared with the previous context that left. (Init to zero) If they differ than the older context will no longer be referenced and thus can be de-allocated in the physical memory. This is accomplished by adding the number of blocks freed this context to the stop\_ptr pointer. This will make all the physical addresses used by this context available to the read\_ptr allocate pointer for future allocation.

This device allows representation of multiple contexts of constants data with N copies of the logical address space. It also allows the second context to be represented as the first set plus some new additional data by just storing the delta's. It allows memory to be efficiently used and when the constants updates are small it can store multiple context. However, if the updates are large, less contexts will be stored and potentially performance will be degraded. Although it will still perform as well as a ring could in this case.

# 5.3 Constant Store Indexing

In order to do constant store indexing, the sequencer must be loaded first with the indexes (that come from the GPRs). There are 144 wires from the exit of the SP to the sequencer (9 bits pointers x 16 vertexes/clock). Since the data must pass thru the Shader pipe for the float to fixed conversion, there is a latency of 4 clocks (1 instruction)

Exhibit 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01/04/02 92:67 PM

AMD1044\_0257105

Formatted: Bullets and Numbering

ATI Ex. 2106 IPR2023-00922 Page 194 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		22 of 50

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

MOVA R1.X,R2.X // Loads the sequencer with the content of R2.X, also copies the content of R2.X into R1.X

NOP // latency of the float to fixed conversion ADD R3,R4,C0[R2.X]// Uses the state from the sequencer to add R4 to C0[R2.X] into R3

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

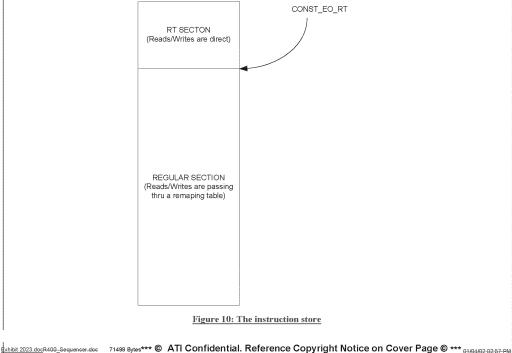
The storage needed in the sequencer in order to support this feature is 2\*64\*9 bits = 1152 bits.

# 5.4 Real Time Commands

The real time commands constants are written by the CP using the register mapped registers allocated for RT. It works is the same way than when dealing with regular constant loads BUT in this case the CP is not sending a logical address but rather a physical address and the reads are not passing thru the re-mapping table but are directly read from the memory. The boundary between the two zones is defined by the CONST\_EO\_RT control register. Similarly, for the fetch state, the boundary between the two zones is defined by the TSTATE\_EO\_RT control register.

# 5.5 Constant Waterfalling

In order to have a reasonable performance in the case of constant store indexing using the address register, we are going to have the possibility of using the physical memory port for read only. This way we can read 1 constant per clock and thus have a worst-case waterfall mode of 1 vertex per clock. There is a small synchronization issue related with this as we need for the SQ to make sure that the constants where actually written to memory (not only sent to the sequencer) before it can allow the first vector of pixels or vertices of the state to go thru the ALUs. To do so, the sequencer keeps 8 bits (one per render state) and sets the bits whenever the last render state is written to memory and clears the bit whenever a state is freed.



AMD1044\_0257106

ATI Ex. 2106 IPR2023-00922 Page 195 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
<u>C900</u>	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	23 of 50

6. Looping and Branches

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

# 6.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0] Loop\_count[7:0][31:0] Loop\_Start[7:0][31:0] Loop\_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

We have a stack of 4 elements for nested calls of subroutines and 4 loop counters to allow for nested loops.

This state is available on a per shader program basis.

### 6.2 The Control Flow Program

Examples of control flow programs are located in the R400 programming guide document.

The basic model is as follows:

### The render state defined the clause boundaries:

Vertex\_shader\_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located Vertex\_shader\_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located Pixel\_shader\_fetch[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located Pixel\_shader\_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located Pixel\_shader\_alu[7:0][7:0] // eight 8 bit pointers to the location where each clauses control program is located

### A pointer value of FF means that the clause doesn't contain any instructions.

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

The control program has eleven basic instructions:

Execute Conditional\_execute\_Predicates Conditional\_jump Conditionnal\_Call Return Loop\_start Loop\_end End\_of\_clause Conditional\_End\_of\_clause NOP

Execute, causes the specified number of instructions in instruction store to be executed.

Conditional\_execute checks a condition first, and if true, causes the specified number of instructions in instruction store to be executed.

Loop\_start resets the corresponding loop counter to the start value on the first pass after it checks for the end condition and if met jumps over to a specified address.

Loop\_end increments (decrements?) the loop counter and jumps back the specified number of instructions.

Exhibit 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01/04/02-02:67.PM

AMD1044\_0257107

ATI Ex. 2106 IPR2023-00922 Page 196 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		24 of 50

Conditionnal\_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.

Conditional\_execute\_or\_Jump executes a block of instructions or jumps to an address is the condition is not met. Conditional\_execute\_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition. End\_of\_clause marks the end of a clause.

Conditional\_End\_of\_clause marks the end of a clause if the condition is met.

Conditional\_jumps jumps to an address if the condition is met.

NOP is a regular NOP

NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.

Also if the jump is logically bigger than pshader\_cntl\_size (or vshader\_cntl\_size) we break the program (clause) and set the debug registers. If an execute or conditional\_execute is lower than cntl\_size or bigger than size we also break the program (clause) and set the debug registers.

We have to fit instructions into 48 bits in order to be able to put two control flow instruction per line in the instruction store.

Note that whenever a field is marked as RESERVED, it is assumed that all the bits of the field are cleared (0).

Execute				
47	46 42	41 24	23 12	11 0
Addressing	00001	RESERVED	Instruction count	Exec Address

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

		NOP	
47	46 42	41 0	1000
Addressing	00010	RESERVED	

This is a regular NOP.

			Con	iditional_Exec	ute		
47	46 42	41	40 33	32	31 24	23 12	11 0
Addressing	00011	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions)

	Conditional_Execute_Predicates						
47	46 42	41 35	34 33	32	31 24	23 12	11 0
Addressing	00100	RESERVED	Predicate	Condition	RESERVED	Instruction count	Exec Address
			vector				

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid.

		Loop_Start		
47	46 42	41 17	16 12	11 0
	00101	RESERVED	loop ID	Jump address
Addressing				

Loop Start. Compares the loop iterator with the end value. If loop condition not met jump to the address. Forward jump only. Also computes the index value. The loop id must match between the start to end, and also indicates which control flow constants should be used with the loop.

Libible 2023.docR400\_Sequencer.dec 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 010/0/02.02-67-PM

AMD1044\_0257108

ATI Ex. 2106 IPR2023-00922 Page 197 of 223

<b>/</b> 1î	ORIGINATE DATE 24 September, 2001		EDIT DATE <u>4 September, 20154</u> <u>February</u> 20020	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA			PAGE 25 of 50
	Loop End						
47	46 42		41 17		16 12	-	11 0
	001104		RESERVED		loop ID	star	t address
Addressing							

Loop end. Increments the counter by one, compares the loop count with the end value. If loop condition met, continue, else, jump BACK to the start of the loop.

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

	Conditionnal_Call					
47	46 42	41 35	34 33	32	31 12	11 0
	01 <u>0111</u> 0	RESERVED	Predicate	Condition	RESERVED	Jump address
Addressing	00		vector			

If the condition is met, jumps to the specified address and pushes the control flow program counter on the stack.

		Return	20102	
47	46 42	41 0	1000	
	0100000	RESERVED		
Addressing	1			

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

	Conditionnal_Jump						
47	46 42	41	40 33	32	31	30 12	11 0
	0100140	RESERVED	Boolean	Condition	FW only	RESERVED	Jump address
Addressing			address		_		

If condition met, jumps to the address. FORWARD jump only allowed if bit 31 set. Bit 31 is only an optimization for the compiler and should NOT be exposed to the API.

			Condit	onal_End_of_	Clause	
47	46 42	41	40 33	32	31 0	
	010104	RESERVED	Boolean	Condition	RESERVED	
Addressing			address			

This is an optimization in the case of very short shaders (where the control flow instruction can't be hidden anymore and thus are not free. In this case, if the condition is met, the clause is ended, else we continue the execution of the clause.

End_of_Clause						
47	46 42	41 0				
Addressing	01 <u>011</u> 0	RESERVED	00///00			

Marks the end of a clause.

To prevent infinite loops, we will keep  $\underline{9}$ -<u>9</u> bits loop <u>counters-iterators instead of 8</u> (we are only able to loop 256 times). If the counter goes higher than 255 then the loop\_end or the loop\_start instruction is going to break the loop and set the debug GPRs.

### 6.3 Data dependant predicate instructions

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

PRED\_SETE\_# - similar to SETE except that the result is 'exported' to the sequencer.

Exhibit 2023 docR400\_Sequencer.doc 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01.04/62/20267-M

AMD1044\_0257109

ATI Ex. 2106 IPR2023-00922 Page 198 of 223

<b>F</b> A	ORI
	24 S
	PRE

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE			
24 September, 2001	4 September, 20154		26 of 50			
PRED_SETNE_# - similar to SETNE except that the result is 'exported' to the sequencer.						
PRED_SETGT_# - similar to SETGT except that the result is 'exported' to the sequencer						

PRED\_SETGTE\_# - similar to SETGTE except that the result is 'exported' to the sequencer PRED\_SETGTE\_# - similar to SETGTE except that the result is 'exported' to the sequencer

For the scalar operations only we will also support the two following instructions: PRED\_SETE0\_# – SETE0 PRED\_SETE1\_# – SETE1

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

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

### P0\_ADD\_# R0,R1,R2

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

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

### 6.4 HW Detection of PV,PS

Because of the control program, the compiler cannot detect statically dependant instructions. In the case of nonmasked writes and subsequent reads the sequencer will insert uses of PV,PS as needed. This will be done by comparing the read address and the write address of consecutive instructions. For masked writes, the sequencer will insert NOPs wherever there is a dependant read/write.

The sequencer will also have to insert NOPs between PRED\_SET and MOVA instructions and their uses.

### 6.5 Register file indexing

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

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

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

The sequencer is going to keep a loop index computed as such:

Index = Loop\_iterator\*Loop\_step + Loop\_start.

The index is going to return 0 if it is out of the range.

We loop until loop\_iterator = loop\_count. Loop\_step is a signed value [-128...127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the "indexing logic" so that it knows when the provided index is out of range and thus can make the necessary arrangements.

Athbit 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01/04/02-02-57-PM

AMD1044\_0257110

ATI Ex. 2106 IPR2023-00922 Page 199 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
$\langle \mathcal{A} \mathcal{U} \rangle$	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	27 of 50	
6.6 Pred	icated Instruction	support for Textu	ire clauses		
For texture cla per predicate value of one ( vector have h pixels aren't					
6.7 <b>Deb</b> u	ugging the Shader	ſS			
In order to be	able to debug the pixel/verte	ex shaders efficiently, we pr	ovide 2 methods.		
671 Mot	thod 1: Debugging r	onictore			
	are to expose 2 debugging in	*	rs:		
1. address reg	gister where the first error of number of errors				
The sequence - count overflo	er will detect the following gr	oups of errors:			
	lexing overflow			-	Formatted: Bullets and Numbering
- jump errors relativ call stack call wi	gnizable errors: re jump address > size of th re jump address > length of ith stack full with stack empty				
	<del>ne errors, a</del> <u>A</u> jump error or that a clause will halt execut		ays cause the program to break. In thuses to be executed.	his case, a	
	ther errors, program can co 3_BREAK register is set.	ntinue to run, potentially to	worst-case limits. The program will o	nly break if	
to return the v			ising an overflow error, the hardware i nerate error tokens, by reserving and		
{ISSUE : Inter	rupt to the driver or not?}				
6.7.2 Met	thod 2: Exporting the	e values in the GPR	s (12)		
Under the not executed but clause 7 will b an address sp	de for each clause. The mod 1) Normal 2) Debug Kill 3) Debug Addr + Count rmal mode execution follow all normal shader instruction be executed under the debug	les can be : vs the normal course. Under ns of the clause are replace g kill setting. Under the othe ster and instruction count (	r this mode and 3 bits per clause spe er the kill mode, all control flow instru- ted by NOPs. Only debug_export inst er mode, normal execution is done unt useful for loops) specified by the cou se to the kill mode.	uctions are tructions of il we reach	
<u>Exhibit 2023.doc</u> R400_Seq	uuencer.doc 71499 Bytes*** ⓒ ATI	Confidential. Reference C	opyright Notice on Cover Page © **	** 01/04/02-02:67 PM	

ATI Ex. 2106 IPR2023-00922 Page 200 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		28 of 50

Under the debug mode (debug kill OR debug Addr + count), it is assumed that clause 7 is always exporting 12 debug vectors and that all other exports to the SX block (position, color, *z*, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

### 7. Pixel Kill Mask

A vector of 64 bits is kept by the sequencer per group of pixels/vertices. Its purpose is to optimize the texture fetch requests and allow the shader pipe to kill pixels using the following instructions:

MASK\_SETE MASK\_SETNE MASK\_SETGT MASK\_SETGTE

### 8. Multipass vertex shaders (HOS)

Multipass vertex shaders are able to export from the 6 last clauses but to memory ONLY.

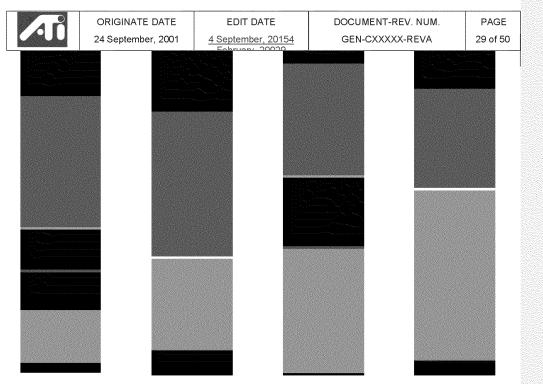
### 9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to <u>128-VERTEX\_REG\_SIZE</u> for vertices and <u>256-VERTEX\_PIXEL\_REG\_SIZE</u> for pixels.

the state of the s

AMD1044\_0257112

ATI Ex. 2106 IPR2023-00922 Page 201 of 223



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

### 10. Fetch Arbitration

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

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

### 11. ALU Arbitration

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

Einst0 Oinst0 Einst1 Oinst1 Einst2 Oinst2 Einst0 Oinst3 Einst1 Oinst4 Einst2 Oinst0... Proceeding this way hides the latency of 8 clocks of the ALUs. Also note that the interleaving also occurs across clause boundaries.

Exhibit 2023.docR400\_Sequencer.dec 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 0140402-02:67-PA

AMD1044\_0257113

ATI Ex. 2106 IPR2023-00922 Page 202 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		30 of 50

### 12. Handling Stalls

When the output file is full, the sequencer prevents the ALU arbitration logic from selecting the last clause (this way nothing can exit the shader pipe until there is place in the output file. If the packet is a vertex packet and the position buffer is full (POS\_FULL) then the sequencer also prevents a thread from entering the exporting clause (3?). The sequencer will set the OUT\_FILE\_FULL signal n clocks before the output file is actually full and thus the ALU arbitrer will be able read this signal and act accordingly by not preventing exporting clauses to proceed.

# 13. Content of the reservation station FIFOs

The reservation FIFOs contain the state of the vector of pixels and vertices. We have two sets of those: one for pixels, and one for vertices. They contain 3 bits of Render State 7 bits for the base address of the GPRs, some bits for LOD correction and coverage mask information in order to fetch fetch for only valid pixels, the quad address.

### 14. The Output File

The output file is where pixels are put before they go to the RBs. The write BW to this store is 256 bits/clock. Just before this output file are staging registers with write BW 512 bits/clock and read BW 256 bits/clock. The staging registers are 4x128 (and there are 16 of those on the whole chip).

### 15. IJ Format

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

We have a vector of IJ's (one IJ per pixel at the centroid of the fragment or at the center of the pixel depending on the mode bit). The interpolation is done at a different precision across the 2x2. The upper left pixel's parameters are always interpolated at full 20x24 mantissa precision. Then the result of the interpolation along with the difference in IJ in reduced precision is used to interpolate the parameter for the other three pixels of the 2x2. Here is how we do it:

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

$\Delta 01I = I(1) - I(0)$		<b>p</b> -0.1
$\Delta 01J = J(1) - J(0)$		
$\Delta 02I = I(2) - I(0)$	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)  $P1 = P0 + \Delta 01I*(A - C) + \Delta 01J*(B - C)$   $P2 = P0 + \Delta 02I*(A - C) + \Delta 02J*(B - C)$  $P3 = P0 + \Delta 03I*(A - C) + \Delta 03J*(B - C)$ 

P0 is computed at 20x24 mantissa precision and P1 to P3 are computed at 8X24 mantissa precision. So far no visual degradation of the image was seen using this scheme.

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

Ath Bit 2023.doc R400\_Sequencer.dec 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 0104/02-02-57-PM

AMD1044\_0257114

ATI Ex. 2106 IPR2023-00922 Page 203 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	31 of 50	

Adds: 8

FORMAT OF P0's IJ : Mantissa 20 Exp 4 for I + Sign Mantissa 20 Exp 4 for J + Sign

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

Total number of bits : 20\*2 + 8\*6 + 4\*8 + 4\*2 = 128

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

### 15.1 Interpolation of constant attributes

Because of the floating point imprecision, we need to take special provisions if all the interpolated terms are the same or if two of the barycentric coordinates are the same.

We start with the premise that if A = B and B = C and C = A, then P0,1,2,3 = A. Since one or more of the IJ terms may be zero, so we extend this to:

```
if (A=B and B=C and C=A)
  P0,1,2,3 = A;
else if ((I = 0) \text{ or } (J = 0)) and
       ((J = 0) or (1-I-J = 0)) and
       ((1-J-I = 0) \text{ or } (I = 0)))
           if(l != 0) {
              P0 = A:
           } else if(J != 0) {
              P0 = B;
           } else {
              P0 = C:
         //rest of the quad interpolated normally
}
else
{
         normal interpolation
}
```

### 16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

Exhibit 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 0104/02.92-67-PM

AMD1044\_0257115

ATI Ex. 2106 IPR2023-00922 Page 204 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		32 of 50

The most straightforward, *non-compressed* interface method would be to convert, in the VGT, the data to 32-bit floating point prior to transmission to the VSISRs. In this scenario, the data would be transmitted to (and stored in) the VSISRs in full 32-bit floating point. This method requires three 24-bit fixed-to-float converters in the VGT. Unfortunately, it also requires and additional 3,072 bits of storage across the VSISRs. This interface is illustrated in Figure 12Figure 12Figure 2. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 11Figure 11Figure 1.

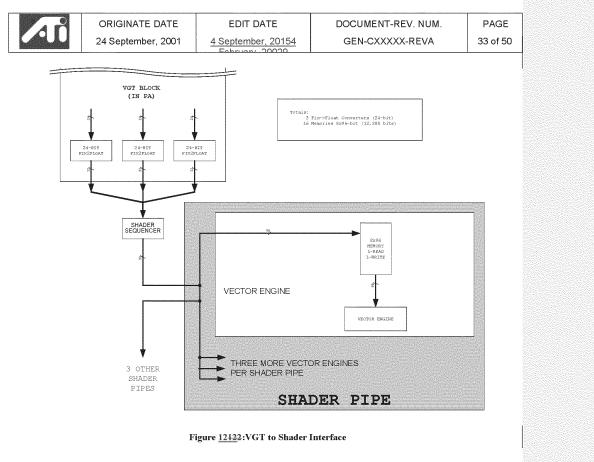
8x24-bit	11631	$\mu^2$	$60.57813\mu^2$ per b
Area of 96x8-deep Latch Memory	46524	$\mu^2$	
Area of 24-bit Fix-to-float Converter	4712	$\mu^2$ per conve	erter
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 µ <sup>2</sup>

Figure 11111: Area Estimate for VGT to Shader Interface

thibs 2023.doc.P400\_Sequences.doc 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01.04/02/02.67.PM

AMD1044\_0257116

ATI Ex. 2106 IPR2023-00922 Page 205 of 223



# 17. The parameter cache

The parameter cache is where the vertex shaders export their data. It consists of 16 128x128 memories (1R/1W). The reuse engine will make it so that all vertexes of a given primitive will hit different memories. The allocation method for these memories is a simple round robin. The parameter cache pointers are mapped in the following way: 4MSBs are the memory number and the 7 LSBs are the address within this memory.

MEMORY NUMBER	ADDRESS
4 bits	7 bits

The PA generates the parameter cache addresses as the positions comes from the SQ. All it needs to do is keep a Current\_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current\_Location by VS\_EXPORT\_COUNT\_67\_\_\_\_ (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS\_EXPORT\_COUNT\_6-7\_= 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 0010000000 and so on up to 11110000000. Then the next position received (the 17<sup>th</sup>) is going to have the address 0000001000, the 18<sup>th</sup> 00010001000, the 19<sup>th</sup> 00100001000 and so on. The Current\_location is NEVER reset BUT on chip resets. The only thing to be careful about is that if the SX doesn't send you a full group of positions (<64) then you need to fill the address space so that the next group starts correctly aligned (for example if you receive only 33 positions then you need to add  $42^*VS_EXPORT_COUNT_6-7$  to Current\_Location and reset the memory count to 0 before the next vector begins).

Exhibit 2023.docR400\_Sequencer.dec 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 04/04/02-02:67-PM

AMD1044\_0257117

ATI Ex. 2106 IPR2023-00922 Page 206 of 223

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
<u>an</u>	24 September, 2001	4 September, 20154		34 of 50

### 18. Vertex position exporting

On clause 3 the vertex shader can export to the PA both the vertex position and the point sprite. It can also do so at clause 7 if not done at clause 3. The storage needed to perform the position export is at least 64x128 memories for the position and 64x32 memories for the sprite size. It is going to be taken in the pixel output fifo from the SX blocks. The clause where the position export occurs is specified by the EXPORT\_LATE register. If turned on, it means that the export is going to occur at ALU clause 7 if unset position export occurs at clause 3.

### 19. Exporting Arbitration

Here are the rules for co-issuing exporting ALU clauses.

1) Position exports and position exports cannot be co-issued.

All other types of exports can be co-issued as long as there is place in the receiving buffer.

{ISSUE: Do we move the parameter caches to the SX?}

Any type of exporting clause can be co-issued. The sequencer will have to make sure back to back memory exports (position/straight memory exports) are interleaved with NOPs as we don't have the bandwidth to service them at full speed.

### 20. Export Types

The export type (or the location where the data should be put) is specified using the destination address field in the ALU instruction. Here is a list of all possible export modes:

### 20.1 Vertex Shading

- 0:15 16 parameter cache
- 16:31 Empty (Reserved?)
- 32:43 12 vertex exports to the frame buffer and index
- 44:47 Empty
- 48:59 12 debug export (interpret as normal vertex export)
- 60 export addressing mode

61	- Empty
62	<ul> <li>position sprite size export that goes with position export</li> </ul>
	(point_h,point_w,edgeflag,misc)
63	- positionsprite size export that goes with position export
	(point_h,point_w,edgeflag,misc)

# 20.2 Pixel Shading

1

2

- 0 Color for buffer 0 (primary)
- Color for buffer 1
- Color for buffer 2
- 3 Color for buffer 3
- 4:7 Empty
- 8 Buffer 0 Color/Fog (primary)
- 9 Buffer 1 Color/Fog
- 10 Buffer 2 Color/Fog
- 11 Buffer 3 Color/Fog
- 12:15 Empty
- 16:31 Empty (Reserved?)
- 32:43 12 exports for multipass pixel shaders.
- 44:47 Empty
- 48:59 12 debug exports (interpret as normal pixel export)
- 60 export addressing mode
- 61:62 Empty
- 63 Z for primary buffer (Z exported to 'alpha' component)

Laber 2023. doc PA400\_Bequencer.doc 71499 Bytes\*\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 04.04/02-02-67. PM

AMD1044\_0257118

ATI Ex. 2106 IPR2023-00922 Page 207 of 223

Formatted: Bullets and Numbering

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	35 of 50	

### 21. Special Interpolation modes

## 21.1 Real time commands

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

# 21.2 Sprites/ XY screen coordinates/ FB information

When working with sprites, one may want to overwrite the parameter 0 with SC generated data. Also, XY screen coordinates may be needed in the shader program. This functionality is controlled by the gen\_I0 register (in SQ) in conjunction with the SND\_XY register (in SC). Also it is possible to send the faceness information (for OGL front/back special operations) to the shader using the same control register. Here is a list of all the modes and how they interact together:

Gen\_st is a bit taken from the interface between the SC and the SQ. This is the MSB of the primitive type. If the bit is set, it means we are dealing with Point AA, Line AA or sprite and in this case the vertex values are going to generated between 0 and 1.

Param\_Gen\_I0 disable, snd\_xy disable, no gen\_st - I0 = No modification Param\_Gen\_I0 disable, snd\_xy disable, gen\_st - I0 = No modification Param\_Gen\_I0 disable, snd\_xy enable, no gen\_st - I0 = No modification Param\_Gen\_I0 disable, snd\_xy enable, gen\_st - I0 = No modification Param\_Gen\_I0 enable, snd\_xy disable, no gen\_st - I0 = garbage, garbage, garbage, faceness Param\_Gen\_I0 enable, snd\_xy disable, gen\_st - I0 = garbage, garbage, s, t Param\_Gen\_I0 enable, snd\_xy enable, no gen\_st - I0 = screen x, screen y, garbage, faceness Param\_Gen\_I0 enable, snd\_xy enable, gen\_st - I0 = screen x, screen y, s, t

### 21.3 Auto generated counters

In the cases we are dealing with multipass shaders, the sequencer is going to generate a vector count to be able to both use this count to write the 1<sup>st</sup> pass data to memory and then use the count to retrieve the data on the 2<sup>nd</sup> pass. The count is always generated in the same way but it is passed to the shader in a slightly different way depending on the shader type (pixel or vertex). This is toggled on and off using the GEN\_INDEX register. The sequencer is going to keep two counters, one for pixels and one for vertices. Every time a full vector of vertices or pixels is written to the GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

### 21.3.1 Vertex shaders

In the case of vertex shaders, if GEN\_INDEX is set, the data will be put into the x field of the third register (it means that the compiler must allocate 3 GPRs in all multipass vertex shader modes).

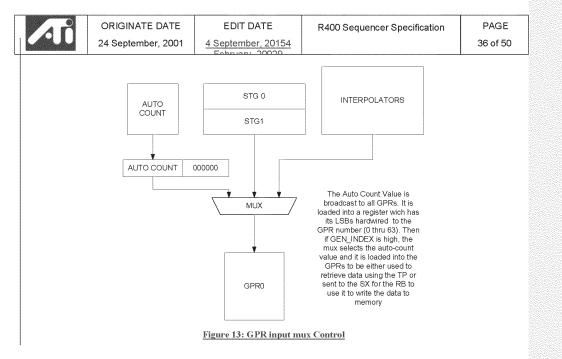
### 21.3.2 Pixel shaders

In the case of pixel shaders, if GEN\_INDEX is set and Param\_Gen\_I0 is enabled, the data will be put in the x field of the  $2^{nd}$  register (R1.x), else if GEN\_INDEX is set the data will be put into the x field of the  $1^{st}$  register (R0.x).

Exhibit 2023.docR400\_Sequencer.dec 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 0140402-02:67-PA

AMD1044\_0257119

ATI Ex. 2106 IPR2023-00922 Page 208 of 223



### 22. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

### 22.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC\_SQ\_new\_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

### 23. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix—float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 21.2 for details on how to control the interpolation in this mode.

### 23.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

AthBit 2023.docR400\_Sequencer.dec 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01/04/02-02:67-PM

AMD1044\_0257120

ATI Ex. 2106 IPR2023-00922 Page 209 of 223

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	37 of 50

# 24. Registers

24.1 Control

REG_DYNAMIC	Dynamic allocation (pixel/vertex) of the register file on or off.
REG_SIZE_PIX	Size of the register file's pixel portion (minimal size when dynamic allocation turned on)
REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
ARBITRATION_POLICY	policy of the arbitration between vertexes and pixels
INST_STORE_ALLOC	interleaved, separate
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and
	Begins at 0)
INST_BASE_PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION_ADDR	This is where the CP puts the base address of the instruction writes and type (auto- incremented on reads/writes) Register mapped
INSTRUCTION DATA	This is where the CP puts the actual data going to the instruction memory
CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
	This is where the CP puts the base address of the instruction writes and type for
morrison _nsbbn_ht	Real Time (auto-incremented on reads/writes)
INSTRUCTION DATA RT	This is where the CP puts the actual data going to the instruction memory for
40000 ABB97	Real Time
CONSTANTS RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT EO RT	This is the size of the space reserved for real time in the constant store (from 0 to
· · · · · · · · <b>_</b> - · <b>_</b> · · ·	CONSTANT EO RT). The re-mapping table operates on the rest of the memory
TSTATE EO RT	This is the size of the space reserved for real time in the fetch state store (from 0 to
	TSTATE EO RT). The re-mapping table operates on the rest of the memory
EXPORT LATE	Controls whether or not we are exporting position from clause 3. If set, position
	exports occur at clause 7.
24.2 Context	
VS FETCH {07}	eight 8 bit pointers to the location where each clauses control program is located
	sight of hit pointers to the location where each clause control program is located

VS_FEIGH_{07}	eight 8 bit pointers to the location where each clauses control program is located
VS_ALU_{07}	eight 8 bit pointers to the location where each clauses control program is located
PS_FETCH_{07}	eight 8 bit pointers to the location where each clauses control program is located
PS_ALU_{07}	eight 8 bit pointers to the location where each clauses control program is located
PS_BASE	base pointer for the pixel shader in the instruction store
VSBASE	base pointer for the vertex shader in the instruction store
VS_CF_SIZE	size of the vertex shader (# of instructions in control program/2)
PS_CF_SIZE	size of the pixel shader (# of instructions in control program/2)
PS_SIZE	size of the pixel shader (cntl+instructions)
VS_SIZE	size of the vertex shader (cntl+instructions)
PS_NUM_REG	number of GPRs to allocate for pixel shader programs
VS_NUM_REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1
	= gouraud)
PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical).
PS EXPORT MODE	Oxxxx : Normal mode
· · · <u> </u>	1xxxx : Multipass mode
	If normal, bbbz where bbb is how many colors (0-4) and z is export z or not
	If multipass 1-12 exports for color.
VS EXPORT MASK	which of the last 6 ALU clauses is exporting (multipass only)
VS EXPORT MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS EXPORT	

Exhibit 2023 doc R400\_Sequencer.doc 71498 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 01/04/02 02:67 PM

AMD1044\_0257121

ATI Ex. 2106 IPR2023-00922 Page 210 of 223

	DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September,		4 September, 20154		38 of 50	
COUNT_{06} PARAM_GEN_I0 GEN_INDEX CONST_BASE_VTX (9 bits) CONST_SIZE_PIX (9 bits) CONST_SIZE_PIX (8 bits) CONST_SIZE_VTX (8 bits) INST_PRED_OPTIMIZE CF_BOOLEANS CF_LOOP_COUNT CF_LOOP_START CF_LOOP_STEP	(located # of exp Do we of Auto ge and R2 s)Logical Size of 1 Size of 1 Turns o always 256 boo 32x8 bit 32x8 bit	counters representing the in VS_EXPORT_COUN orted vectors to memory overwrite or not the param nerates an address from for vertex shaders Base address for the con Base address for the con the logical constant store the logical constant store	per clause in multipass mode (per clau leter 0 with XY data and generated T a 0 to XX. Puts the results into R0-1 for stants of the Vertex shader stants of the Pixel shader for pixel shaders for vertex shaders ation (if of, conditional_execute_predi es we traverse the loop) in index computation)	use) and S values pixel shaders	
25. DEBUG Registe	ers				
25.1 Context					
DB_PROB_ADDR DB_PROB_COUNT DB_PROB_BREAK DB_INST_COUNT DB_BREAK_ADDR DB_CLAUSE _MODE_ALU_{07} DB_CLAUSE MODE FETCH {07	number break th instructi break a clause r	e clause if an error is fou on counter for debug met ddress for method numbe node for debug method 2	during the execution of the program nd. hod 2		
		-			
25.2 Control DB_ALUCST_MEMSIZ DB_TSTATE_MEMSIZ		Size of the physical ALU Size of the physical textu			
26. Interfaces					
26.1 External Interfa	ices				
			II units of the same name. For examp information to all SP instances.	ole, if a bus is	
26.1.1 SC to SQ : IJ C	ontrol	bus			
execute a shader program on t control packet is going to be ig	he sent p nored an	oixels. This information is d XY information is going	ontrol the IJ fifos and all other informal sent over 2 clocks, if SENDXY is ass to be sent on the IJ bus (for the qua primitive, all quads of a vector are f	erted the next ids that where	
<u>  ⊈xhibit 2023.doc</u> R400_Sequencer.doc 71499 Bytes <sup>1</sup>	*** © AT	l Confidential. Referenc	e Copyright Notice on Cover Page @	€ *** 01/04/02-02:57 FM	

ATI Ex. 2106 IPR2023-00922 Page 211 of 223

	ORIGINATE	DATE	EDF	T DATE		DOCUMENT-REV. NUM.	PAGE
CQU	24 September	r, 2001	4 Septer	nber, 20		GEN-CXXXXX-REVA	39 of 50
Name		Direction		Bits	Descri	otion	
SC SQ q wr	mask	SC→SQ		4		Vrite mask left to right	
SC SQ lod c	orrect	SC→SQ		24	LOD co	prrection per quad (6 bits per quad)	
SC SQ flat v	ertex	SC→SQ		2		ing vertex for flat shading	
SC_SQ_paran	n_ptr0	SC→SQ		11	P Store	e pointer for vertex 0	
SC_SQ_paran	n_ptr1	SC→SQ		11	P Store	pointer for vertex 1	
SC SQ paran	n_ptr2	SC→SQ		11	P Store	pointer for vertex 2	
SC_SQ_end_c	of_vect	SC→SQ		1	End of	the vector	
SC_SQ_store_	dealloc	SC→SQ		1	Dealloo	ation token for the P Store	
SC_SQ_state		SC→SQ		3	State/c	onstant pointer	
SC_SQ_valid_	pixel	SC→SQ		16	Valid b	its for all pixels	
SC_SQ_null_p	prim	SC→SQ		1	Null Pr	imitive (for PC deallocation purposes	)
SC_SQ_end_c	of_prim	SC→SQ		1	End Of	the primitive	
SC_SQ_send_	_ху	SC→SQ		1		g XY information [XY information is the next clock]	going to be
SC_SQ_prim_	type	SC→SQ		3	Real t alterna their pa 000 : N 011 : R 100 : L	ime command need to load tex te buffer. Line AA, Point AA and S arameters from GEN_T and GEN_S ( lormal leal Time ine AA loint AA	Sprite reads
SC_SQ_new_	vector	SC→SQ		1	This p Make	imitive comes from a new vector sure that the corresponding vertex before starting the group of pixels.	
SC SQ RTRn		SQ→SC		1		he PA in n clocks	
SC SQ RTS		SC→SQ		1	SC rea	dy to send data	

## 26.1.2 SQ to SP: Interpolator bus

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive
			000 : Normal
			011 : Real Time
			100 : Line AA
			101 : Point AA
			110 : Sprite
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ SPx interp gen I0	SQ→SPx	1	Generate I0 or not. This tells the interpolators not to
			use the parameter cache but rather overwrite the data
			with interpolated 1 and 0. Overwrite if gen, 10 is high

### 26.1.3 SQ to SX: Interpolator bus

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	<u>SQ→SPx</u>	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	<u>SQ→SPx</u>	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	<u>SQ→SPx</u>	4	Wich channel needs to be cylindrical wrapped

# <u>26.1.326.1.4</u> SQ to SP: Parameter Cache Read control bus

The four following interfaces (SQ $\rightarrow$ SP, SQ $\rightarrow$ SX, SP $\rightarrow$ SX and SX $\rightarrow$ Interpolators) are all SYNCHRONIZED together.

Name	Direction	Bits	Description
SQ_SPx_ptr0SQ_SPx_ptr0	<u>SQ→SPx</u> SQ→SPx	79	Parameter Pointer into PC Pointer of PC
SQ_SPx_ptr1SQ_SPx_ptr1	<u>SQ→SPx</u> SQ→SPx	<u>7</u> 9	Parameter Pointer into PC Pointer of PC

Exhibit 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 01.04/02-02:57 PM

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

٠

AMD1044\_0257123

ATI Ex. 2106 IPR2023-00922 Page 212 of 223

<b>A</b> î	ORIGINATE DATE 24 September, 2001	EDIT DATE 4 September, 20154	R	400 Sequencer Specification	PAGE 40 of 50
SQ_SPx_ptr2SQ	_SPx_ptr2	<u>SQ→SPx</u> SQ→SPx	79	Parameter Pointer into CachePointer of PC	Parameter
SQ_SPx_pc0_ad	dr_selSQ_SP0_read_ena	<u>SQ→SPx</u> SQ→SP0	<u>2</u> 4	Selection one of the pointers cache 0Read enables for the 4 m SP0	
SQ_SPx_pc1_ad	dr_selSQ_SP1_read_ena	<u>SQ→SPx</u> SQ→SP1	<u>2</u> 4	Selection one of the pointers cache 1Read enables for the 4 m SP1	
SQ SPx pc2 ad	<u>dr_sel</u> SQ_SP2_read_ena	<u>SQ→SPx</u> SQ→SP2	<u>2</u> 4	Selection one of the pointers cache 2Read enables for the 4 m SP2	
SQ_SPx_pc3_ad	dr_selSQ_SP3_read_ena	<u>SQ→SPx</u> SQ→SP3	<u>2</u> 4	Selection one of the pointers cache 3Read enables for the 4 m SP3	
SQ_SP0_read_e	na	SQ→SP0	4	Read enables for the 4 memories	s in the SP0
SQ_SP1_read_e	na	SQ→SP1	4	Read enables for the 4 memories	s in the SP1
SQ_SP2_read_e	na	SQ→SP2	4	Read enables for the 4 memories	s in the SP2
SQ_SP3_read_e	na	SQ→SP3	4	Read enables for the 4 memories	s in the SP3

### <u>26.1.426.1.5</u> SQ to SX: Parameter Cache Mux control Bus

Name	Direction	Bits	Description
SQ_SXx_mux0	SQ→SXx	4	Mux control for PC or RT (4 MSbs of Pointer in the PC case)
SQ_SXx_mux1	SQ→SXx	4	Mux control for PC or RT (4 MSbs of Pointer in the PC case)
SQ_SXx_mux2	SQ→SXx	4	Mux control for PC or RT (4 MSbs of Pointer in the PC case)
SQ SXx RT switch	SQ→SXx	1	Selects between RT and Normal data

### 26.1.526.1.6 SQ to SP: Staging Register Data

This is a broadcast bus that sends the VSISR information to the staging registers of the shader pipes.

Name	Direction	Bits	Description
SQ_SPx_vgt_vsisr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vgt_vsisr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_data_valid	SQ→SP0	1	Data is valid
SQ_SP1_data_valid	SQ→SP1	1	Data is valid
SQ_SP2_data_valid	SQ→SP2	1	Data is valid
SQ_SP3_data_valid	SQ→SP3	1	Data is valid

# 26.1.626.1.7 PA to SQ : Vertex interface

26.1.6.126.1.7.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. <u>Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs (via the Shader Sequencer) in full, 32-bit floating-point format.</u> The VGT can transmit up to six 32-bit floating-point values to each VSISR where four or more values require two transmission clocks. The data bus is 96 bits wide.

Link 2023 doc R400\_Sequencer.doc 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01.04/02/02-67-PM

Formatted: Bullets and Numbering

AMD1044\_0257124

ATI Ex. 2106 IPR2023-00922 Page 213 of 223

- Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

	ORIGINATI	E DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 Septemb	er, 2001	4 September, 20154	GEN-CXXXXX-REVA	41 of 50
Name		Bits	Description		
PA_SQ_vgt_vs	isr_data	9966	Pointers of indexes or HOS sur	face information	
PA_SQ_vgt_vs	isr_double	1	0: Normal 96-96 bits per vert 1:	double 192-192 bits per vert	
PA_SQ_vgt_en	d_of_vector	1	Indicates the last VSISR data s	set for the current process vector (for	double vector
			data, "end_of_vector" is set on	the second vector)	
PA SQ vgt vs	<u>isr_valid</u>	1	Vsisr data is valid		
PA_SQ_vgt_sta	ate	3	Render State (6*3+3 for consta	ants). This signal is guaranteed to be	correct when
			"PA_SQ_vgt_end_of_vector" is	high.	
PA_SQ_vgt_se	nd	1		eceive (see write-up for standard R40	0 SEND/RTR
			interface handshaking)		
SQ_PA_vgt_rtr		1		te-up for standard R400 SEND/R	TR interface
			handshaking)		

26.1.6.226.1.7.2 Interface Diagrams

---- Formatted: Bullets and Numbering

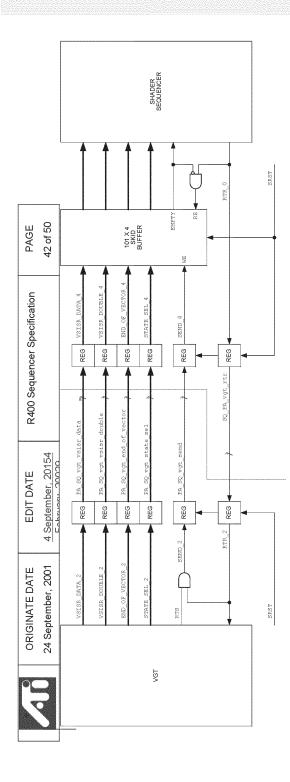
4.--

Exhibit 2023.doc R400\_Sequencer.doc 71498 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01/04/02-02:67 PM

AMD1044\_0257125

ATI Ex. 2106 IPR2023-00922 Page 214 of 223

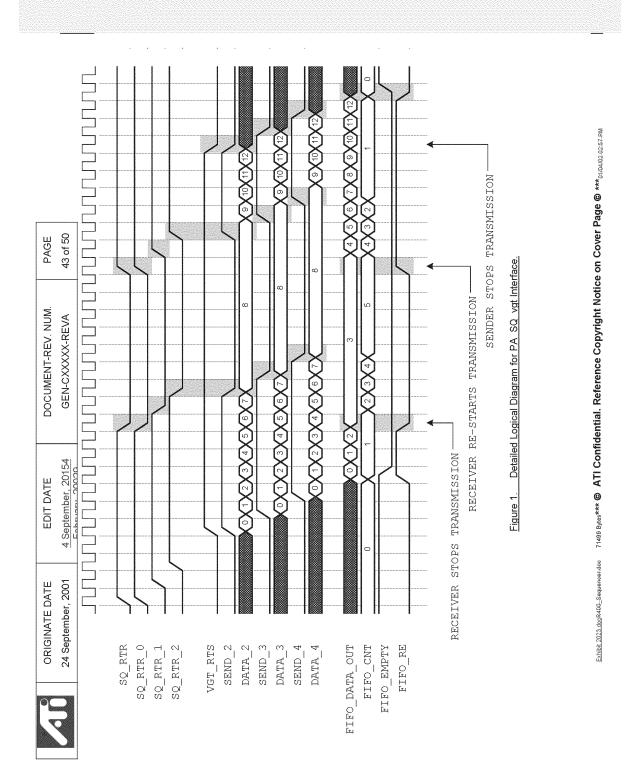
# PROTECTIVE ORDER MATERIAL



71498 Bytes\*\*\* 
Confidential. Reference Copyright Notice on Cover Page 
\*\*\*\*
GRIGHORD 025257PM Exhibit 2023.docR400\_Sequencer.doc

AMD1044\_0257126

ATI Ex. 2106 IPR2023-00922 Page 215 of 223 PROTECTIVE ORDER MATERIAL



AMD1044\_0257127

ORIGINATE 24 September		EDIT I <u>4 Septemb</u>	er, 201		PAGE 44 of 50	
<del>26.1.7</del> 26.1.8 SQ to (	D. Cta	Echrunn to roport	<u>, 2002(</u>		4	<b>Formatted:</b> Bullets and Numbering
20.1.720.1.0_30 10 0	5F. Sla	le report				
Name	Directi		Bits	Description		
SQ_CP_vrtx_ state	SEQ→		3	Oldest vertex state still in the pipe		
SQ_CP_pix_state	SEQ→	CP	3	Oldest pixel state still in the pipe		
26.1.8 <u>26.1.9</u> SQ to 3	SX: Cor	ntrol bus			*	<b>Formatted:</b> Bullets and Numbering
Name	Directi	on	Bits	Description		
SQ_SXx_exp_Pixel	SQ→S		1	1: Pixel		
			<u> </u>	0: Vertex		
SQ_SXx_exp_start	SQ→S		1	Raised to indicate that the SQ is starting		
SQ_SXx_exp_Clause	SQ→S		3	Clause number, which is needed for verte		
SQ_SXx_exp_State	SQ→S	XX	3	State ID, which is needed for vertex claus	ses	
	ointers}	·		scribed in SP0→SX0 interface D/	*	
<u>26.1.9</u> 26.1.10_SX to Name	ointers} SQ : O Directi	utput file		D/ Description	4	
<u>26.1.9</u> 26.1.10_SX to Name	SQ : O	utput file	contro	ol		
2 <u>6.1.926.1.10</u> SX to Name SXx_SQ_Export_count_rdy	ointers} SQ : O Directi	utput file on SQ	CONTRO Bits	) Description Raised by SX0 to indicate that the followi	rt	<b>Formatted:</b> Bullets and Numbering
{ISSUE: Where are the PC pc 26.1.926.1.10_SX to Name SXx_SQ_Export_count_rdy <u>SXx_SQ_Export_Position</u> SXx_SQ_Export_Buffer	SQ : O Directi SXx→S	utput file on SQ	CONTRO Bits	D/ Description Raised by SX0 to indicate that the following reflect the result of the most recent exponent Specifies whether there is room for anoth Specifies the space available in the output 0: buffers are full 1: 2K-bits available (32-bits for each of the second se	rt ner position. ut buffers.	<b>Formatted:</b> Bullets and Numbering
26.1.926.1.10_SX to Name SXx_SQ_Export_count_rdy SXx_SQ_Export_Position	SQ : O Directi SXx→S	utput file on SQ	CONTRO Bits 1	D/ Description Raised by SX0 to indicate that the followi reflect the result of the most recent expor Specifies whether there is room for anoth Specifies the space available in the output 0: buffers are full	rt ner position. ut buffers. ne 64	(Formatted: Bullets and Numbering
26.1.926.1.10_SX to Name SXx_SQ_Export_count_rdy SXx_SQ_Export_Position	SQ : O Directi SXx→S SXx→S	utput file	Bits 1 1 7	D Description Raised by SX0 to indicate that the followi reflect the result of the most recent export Specifies whether there is room for anoth Specifies the space available in the output 0: buffers are full 1: 2K-bits available (32-bits for each of the pixels in a clause)  64: 128K-bits available (16 128-bit entries 64 pixels)	rt ner position. ut buffers. ne 64	Formatted: Bullets and Numbering

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_clause_num	<u>TPx→ SQ</u>	3	Clause number
TPx_SQ_TypeTPx_SQ_clause_num	$\frac{\text{TPx} \rightarrow \text{SQ}}{\text{SQ}} \xrightarrow{\text{TPx}} \rightarrow$	13	Type of data sent (0:PIXEL, 1:VERTEX)Clause
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instuct	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause
SQ TPx Type	<u>SQ→TPx</u>	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ TP0 pmask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad

Athibit 2023 doc R400\_Sequencer.doc 71499 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 01/04/02-02:57 PM

AMD1044\_0257128

ATI Ex. 2106 IPR2023-00922 Page 217 of 223

<u>A</u>	ORIGINATE DATE	EDIT DAT	E	DOCUMENT-REV. NUM.	PAGE
24 September, 2001		4 September, 20154		GEN-CXXXXX-REVA	45 of 50
SQ_TP2_	omask	SQ→TP2	4	Pixel mask 1 bit per pixel	
SQ_TP3_I	od_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 comp	onents per
				quad	
SQ TP3 pmask		SQ→TP3	4	Pixel mask 1 bit per pixel	
SQ TPx clause num		SQ→TPx	3	Clause number	
SQ_TPx_v	vrite_gpr_index	SQ->TPx	7	Index into Register file for write of retu Data	urned Fetch

### 26.1.1126.1.12 TP to SQ: Texture stall

The TP sends this signal to the SQ when its input buffer is full. The SQ is going to send it to the SP X clocks after reception (maximum of 3 clocks of pipeline delay).

Name	Direction	Bits	Description
TP_SQ_fetch_stall	$TP \rightarrow SQ$	1	Do not send more texture request if asserted

# 26.1.1226.1.13 SQ to SP: Texture stall

Name	Direction	Bits	Description
SQ_SPx_fetch_stall	SQ→SPx	1	Do not send more texture request if asserted

### 26.1.1326.1.14 SQ to SP: GPR, Parameter cache control and auto counter

Name	Direction	Bits	Description
SQ_SPx_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_re_addr	SQ→SPx	1	Read Enable
SQ_SPx_gpr_we_addr	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_pixel_mask	SQ→SP3	4	The pixel mask
SQ_SPx_pc_we_addr	SQ→SPx	1	Write Enable for the parameter caches
SQ_SPx_gpr_input_mux	SQ→SPx	2	When the phase mux selects the inputs this tells from which source to read from: Interpolated data, VTX0, VTX1, autogen counter.
SQ SPx index count	SQ→SPx	12?	Index count, common for all shader pipes

Exhibit 2023.docR400\_Sequencer.doc 71499 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\* 01404/02-02:57 PM

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0257129

ATI Ex. 2106 IPR2023-00922 Page 218 of 223

ORIGIN/	TE DATE	EDIT	DATE	R400 Sequencer Specification	PAGE	
24 Septer	nber, 2001	<u>4 Septem</u>	ber, 20154		46 of 50	
1.1426.1.15 S	to CDV	Echruqu	~ 20020			Formatted: Bullets and Numbering
Name SQ_SPx_instruct_start		ction →SPx	Bits 1	Description Instruction start		
SQ_SPX_Instruct_start		→SPx →SPx	2120	Transferred over 4 cycles		
			And the for	SRC A Select         2:0           SRC A Argument Modifier         3:3           SRC A swizzle         11:4           Unused         20:12		
				1: SRC B Select         2:0           SRC B Argument Modifier         3:3           SRC B swizzle         11:4           Unused         20:12		
				2: SRC C Select         2:0           SRC C Argument Modifier         3:3           SRC C swizzle         11:4           Unused         20:12		
				3:         Vector Opcode         4:0           Scalar Opcode         10:5           Vector Clamp         11:11           Scalar Clamp         12:12           Vector Write Mask         16:13           Scalar Write Mask         20:17Instruct           4-clocks         4	tion sent over	
SQ_SPx_stall		→SPx	1	Stall signal		
SQ_SPx_export_count	SQ-	→SPx	3	Each set of four pixels or vectors is e eight clocks. This field specifies where that sequence.		
SQ_SPx_export_last	SQ-	→SPx	1	Asserted on the first shader count of th of the clause	ne last export	
SQ_SP0_export_pvalio	I SQ-	→SP0	4	Result of pixel kill in the shader pipe, w output for all pixel exports (depth a buffers). 4x4 because 16 pixels are c clock	ind all color	
SQ_SP0_export_wvali	1 SQ-	→SP0	2	Specifies whether to write low and/or hig of the 64-bit export data from each of th vectors		
GSP1_ export_pvali	d SQ-	→SP1	4	Result of pixel kill in the shader pipe, w output for all pixel exports (depth a buffers). 4x4 because 16 pixels are o clock	ind all color	
Q_SP1_ export_wval	d SQ-	→SP1	2	Specifies whether to write low and/or hig of the 64-bit export data from each of th vectors		
SQ_SP2_ export_pvali	d SQ-	→SP2	4	Result of pixel kill in the shader pipe, w output for all pixel exports (depth a buffers). 4x4 because 16 pixels are c clock	ind all color	
Q_SP2_ export_wval	d SQ-	→SP2	2	Specifies whether to write low and/or hig of the 64-bit export data from each of th vectors		
SQ_SP3_ export_pvali	d SQ-	→SP3	4	Result of pixel kill in the shader pipe, w output for all pixel exports (depth a buffers). 4x4 because 16 pixels are of	ind all color	

ATI Ex. 2106 IPR2023-00922 Page 219 of 223

Q: Constant addrectionBi $20 \rightarrow SQ$ 36 $20 \rightarrow SQ$ 1 $21 \rightarrow SQ$ 36 $21 \rightarrow SQ$ 1 $22 \rightarrow SQ$ 1 $22 \rightarrow SQ$ 1 $23 \rightarrow SQ$ 36 $23 \rightarrow SQ$ 1PX: constant brorectionBi $21 \rightarrow SPx$ 12	Clock Clock Spec of the vecto Constar bata va Constar to the si Data va	iffies whether to write low and/or high 32-bit wo e 64-bit export data from each of the 16 pixels ors d/ <u>Predicate Set</u> ption nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid	rd or *-	- <b>Formatted:</b> Bullets and Numbering
SQ $\rightarrow$ SP3Q: Constant addrectionPiPi $\rightarrow$ SQPi $\rightarrow$ SQ	clock       2     Spec of the vecto       dress load       dress load       its     Descrip       6     Constar to the sr       Data va       0       Data va	iffies whether to write low and/or high 32-bit wo e 64-bit export data from each of the 16 pixels ors d/ <u>Predicate Set</u> ption nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid	or	
Q: Constant addrectionBi $20 \rightarrow SQ$ 36 $20 \rightarrow SQ$ 1 $21 \rightarrow SQ$ 36 $21 \rightarrow SQ$ 1 $22 \rightarrow SQ$ 1 $22 \rightarrow SQ$ 1 $23 \rightarrow SQ$ 1 $23 \rightarrow SQ$ 1 $Px: constant brownrectionBi21 \rightarrow SPx12$	of the vecto dress load tits Descrip 6 Constar to the si Data va 6 Constar to the si Data va	e 64-bit export data from each of the 16 pixels ors <u>d/ Predicate Set</u> <u>ption</u> nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid	or	
rectionBi $^{20} \rightarrow SQ$ 36 $^{20} \rightarrow SQ$ 1 $^{21} \rightarrow SQ$ 36 $^{21} \rightarrow SQ$ 1 $^{22} \rightarrow SQ$ 36 $^{22} \rightarrow SQ$ 1 $^{23} \rightarrow SQ$ 1	its Descrip 6 Constar to the si Data va 6 Constar to the si Data va 6 Constar to the si Data va 6 Constar to the si Data va 6 Constar to the si Data va	ption nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid		
$20 \rightarrow SQ$ $36$ $20 \rightarrow SQ$ 1 $21 \rightarrow SQ$ 36 $21 \rightarrow SQ$ 1 $22 \rightarrow SQ$ 36 $22 \rightarrow SQ$ 1 $23 \rightarrow SQ$ 1 $23 \rightarrow SQ$ 1 $Px: constant brownerrectionBi2 \rightarrow SPx12$	6 Constar to the si Data va 6 Constar to the si Data va 6 Constar to the si Data va 6 Constar to the si Data va 9 Data va 9 Data va	nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid		<b>Formatted:</b> Bullets and Numbering
$20 \rightarrow SQ$ 1 $21 \rightarrow SQ$ 36 $21 \rightarrow SQ$ 1 $22 \rightarrow SQ$ 36 $22 \rightarrow SQ$ 1 $23 \rightarrow SQ$ 36 $23 \rightarrow SQ$ 1 $PX: constant brownerrectionBiQ \rightarrow SPx12$	to the si Data va Constar to the si Constar to the si to the s	sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid		<b>Formatted:</b> Bullets and Numbering
$21 \rightarrow SQ$ $36$ $21 \rightarrow SQ$ 1 $22 \rightarrow SQ$ 36 $22 \rightarrow SQ$ 1 $23 \rightarrow SQ$ 36 $23 \rightarrow SQ$ 1Px: constant brorectionBi $2 \rightarrow SPx$ 12	6 Constar to the s Data va 6 Constar to the s Data va 6 Constar to the s Data va Data va	nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid	X)	<b>Formatted:</b> Bullets and Numbering
$22 \rightarrow SQ$ $36$ $22 \rightarrow SQ$ 1 $23 \rightarrow SQ$ 36 $23 \rightarrow SQ$ 1 $Px: constant brorectionBiQ \rightarrow SPx12$	Data va Constar to the s Data va Constar to the s Data va Data va Data va	alid nt address load / predicate vector load (4 bits or sequencer alid nt address load / predicate vector load (4 bits or sequencer alid		<b>Formatted:</b> Bullets and Numbering
$P_2 \rightarrow SQ$ 1 $P_3 \rightarrow SQ$ 36 $P_3 \rightarrow SQ$ 1 $P_X: constant brown rection Bi Q \rightarrow SP_X 12$	to the su Data va Constar to the su Data va	sequencer alid nt address load / predicate vector load (4 bits or sequencer alid		<b>Formatted:</b> Bullets and Numbering
P3→SQ         36           P3→SQ         1           Px: constant bro         rection           Bi         2→SPx         12	6 Constar to the so Data va Dadcast its Descrip	nt address load <u>/ predicate vector load (4 bits or</u> equencer alid	▲ 1 - - - - - - - - - - - - -	<b>Formatted:</b> Bullets and Numbering
Px: constant bro rection Bi ⊇→SPx 12	Data va Dadcast its Descrip	alid		
Px: constant bro rection Bi ⊇→SPx 12	oadcast its Descrip		*	- Formatted: Bullets and Numbering
rection Bi ⊋→SPx 12	its Descrip	ntion	•-	Formatted: Bullets and Numbering
Q→SPx 12		ntion		
	28 Constar	nt broadcast		· · · · · · · · · · · · · · · · · · ·
SQ: Kill vector lo	bad		4-10	Formatted: Bullets and Numbering
	its Descrip			
P0→SQ 4				
2→3Q 4				
P: RBBM bus	i		•	Formatted: Bullets and Numbering
	Neur H		•	Formatted: Bullets and Numbering
	ite Docorir	ntion		
			196626	
$P \rightarrow SQ \qquad + G$ $P \rightarrow SQ \qquad 32$	2 Data			
P→SQ 32 P→SQ 4	Byte Er			
P→SQ 32	Byte Er			
P→SQ 32 P→SQ 4 P→SQ 1 P→SQ 1	Byte Er Read E Read R	Enable Return Strobe 0		
D-SQ         32           D-SQ         4           D-SQ         1           D-SQ         1           D-SQ         1	Byte Er Read E Read R Read R	Enable Return Strobe 0 Return Strobe 1		
P→SQ 32 P→SQ 4 P→SQ 1 P→SQ 1	Byte Er Read E Read R Read R Read R 2 Read D	Enable Return Strobe 0 Return Strobe 1 Data 0		
	$I \rightarrow SQ$ 4 $2 \rightarrow SQ$ 4 $3 \rightarrow SQ$ 4 $2 \rightarrow SQ$ 4 $2 \rightarrow SQ$ 4 $2 \rightarrow SQ$ 1 $2 \rightarrow CP$ 1 $2 \rightarrow SQ$ 1	I→SQ       4       Kill vec         2→SQ       4       Kill vec         3→SQ       4       Kill vec $P: RBBM bus$ Section       Bits       Descrition         →CP       1       Read S         →CP       32       Read S         →CP       1       Option         →CP       1       Read S         →SQ       1       Write S         →SQ       1       Write S	I $\rightarrow$ SQ       4       Kill vector load         2 $\rightarrow$ SQ       4       Kill vector load $3\rightarrow$ SQ       4       Kill vector load $3\rightarrow$ SQ       4       Kill vector load $B\rightarrow$ SQ       4       Kill vector load $B\rightarrow$ SQ       4       Kill vector load $P: RBBM bus$ Exercise       Exercise $\rightarrow$ CP       1       Read Strobe $\rightarrow$ CP       1       Optional $\rightarrow$ CP       1       Optional $\rightarrow$ CP       1       Real-Time (Optional) $P: RBBM bus$ Exerction       Bits         Description       SQ       1	$I \rightarrow SQ 4 Kill vector load 2→SQ 4 Kill vector load 3→SQ 4 Kill vector load 3→SQ 4 Kill vector load 3→SQ 4 Kill vector load 3→CP 1 Read Strobe →CP 32 Read Data →CP 1 Optional →CP 1 Optional →CP 1 Real-Time (Optional) 7: RBBM bus ection Bits Description →SQ 1 Write Enable →SQ 1 Write Enable →SQ 1 4815 Address Upper Extent is TBD (16:2)$

ATI Ex. 2106 IPR2023-00922 Page 220 of 223

Γ	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20154		48 of 50
	 	12		

### 27. Examples of program executions

27.1.1 Sequencer Control of a Vector of Vertices

- 1. PA sends a vector of 64 vertices (actually vertex indices 32 bits/index for 2048 bit total) to the RE's Vertex FIFO
  - state pointer as well as tag into position cache is sent along with vertices
  - · space was allocated in the position cache for transformed position before the vector was sent
  - also before the vector is sent to the RE, the CP has loaded the global instruction store with the vertex shader program (using the MH?)
  - The vertex program is assumed to be loaded when we receive the vertex vector.
    - the SEQ then accesses the IS base for this shader using the local state pointer (provided to all sequencers by the RBBM when the CP is done loading the program)
- 2. SEQ arbitrates between the Pixel FIFO and the Vertex FIFO basically the Vertex FIFO always has priority
  - at this point the vector is removed from the Vertex FIFO
  - the arbiter is not going to select a vector to be transformed if the parameter cache is full unless the pipe as nothing else to do (ie no pixels are in the pixel fifo).
- 3. SEQ allocates space in the SP register file for index data plus GPRs used by the program
  - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer that came down with the vertices
    - SEQ will not send vertex data until space in the register file has been allocated
- 4. SEQ sends the vector to the SP register file over the RE\_SP interface (which has a bandwidth of 2048 bits/cycle)
  - the 64 vertex indices are sent to the 64 register files over 4 cycles
    - RF0 of SU0, SU1, SU2, and SU3 is written the first cycle
    - RF1 of SU0, SU1, SU2, and SU3 is written the second cycle
    - RF2 of SU0, SU1, SU2, and SU3 is written the third cycle
    - RF3 of SU0, SU1, SU2, and SU3 is written the fourth cycle
  - the index is written to the least significant 32 bits (floating point format?) (what about compound indices) of the 128-bit location within the register file (w); the remaining data bits are set to zero (x, y, z)
- 5. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
- the control packet contains the state pointer, the tag to the position cache and a register file base pointer.
- TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
   TSM0 was first selected by the TSM arbiter before it could start
- 7. all instructions of fetch clause 0 are issued by TSM0
- the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
  - TSM0 does not wait for requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
  - once the TU has written all the data to the register files, it increments a counter that is associated with ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go ahead start to execute the ALU clause
- 9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
- 10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
- 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

Albibit 2023.docR400\_Sequencer.dec 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 01/04/02-02:67.PM

AMD1044\_0257132

ATI Ex. 2106 IPR2023-00922 Page 221 of 223

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
<u>2400</u>	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	49 of 50

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

### 27.1.2 Sequencer Control of a Vector of Pixels

- 1. As with vertex shader programs, pixel shaders are loaded into the global instruction store by the CP
  - At this point it is assumed that the pixel program is loaded into the instruction store and thus ready to be read.
- 2. the RE's Pixel FIFO is loaded with the barycentric coordinates for pixel quads by the detailed walker
  - the state pointer and the LOD correction bits are also placed in the Pixel FIF0
  - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
- 3. SEQ arbitrates between Pixel FIFO and Vertex FIFO when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
- 4. SEQ allocates space in the SP register file for all the GPRs used by the program
  - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
  - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
- 5. SEQ controls the transfer of interpolated data to the SP register file over the RE\_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
- SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
  - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
  - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
  - all other information (such as quad address for example) travels in a separate FIFO
- 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 2023.docR409\_Sequencer.dec 71499 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\* 0104/02-02:67.PM

AMD1044\_0257133

ATI Ex. 2106 IPR2023-00922 Page 222 of 223

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		50 of 50

27.1.3 Notes

- 14. The state machines and arbiters will operate ahead of time so that they will be able to immediately start the real threads or stall.
- 15. The register file base pointer for a vector needs to travel with the vector through the reservation stations, but the instruction store base pointer does not this is because the RF pointer is different for all threads, but the IS pointer is only different for each state and thus can be accessed via the state pointer.

### 28. Open issues

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

Saving power?

Parameter caches in SX?

Using both IJ buffers for center + centroid interpolation?

the state of the s

AMD1044\_0257134

ATI Ex. 2106 IPR2023-00922 Page 223 of 223