Manufactures and a second secon		· · · · · · · · · · · · · · · · · · ·		
An	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
<u>Mun</u>	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	1 of 48
Author:	Laurent Lefebvre			
Issue To:		Copy No:		
	R400 S	equencer Spe	ecification	
		SQ		
		Version 1. <u>8</u> 7		
Overview: This req blo	s is an architectural specifi uired capabilities and expe cks, and provides internal st	cation for the R400 Sequen cted uses of the block. It a fate diagrams.	cer block (SEQ). It provides an ov lso describes the block interfaces,	erview of the internal sub-
AUTOMATIC Document L Current Intra	ALLY UPDATED FIELDS: ocation: C:\p anet Search Title: R40 Name/Dept	erforce\r400\doc_lib\design\b 0 Sequencer Specification APPROVALS	olocks\sq\R400_Sequencer.doc Signature/Date	
Pomarke:				
Remarks.				
THIS DO SUBSTA	CUMENT CONTAIN NTIALLY DETRIME INC. THROUGH (	IS CONFIDENTIAL I ENTAL TO THE INTE UNAUTHORIZED US	NFORMATION THAT CO REST OF ATI TECHNOL E OR DISCLOSURE.	ULD BE .OGIES
"Copyright 200 work created ir unpublished w confidential, pr transmitted in a	1, ATI Technologies Inc. An 2001. The use of this cop ork. The copyright notice oprietary information and tr ny form or by any means wi	Il rights reserved. The mate byright notice is intended to is not an admission that rade secrets of ATI. No part ithout the prior written permis	rial in this document constitutes an provide notice that ATI owns a cop publication has occurred. This w of this document may be used, re sion of ATI Technologies Inc."	unpublished byright in this york contains produced, or
Exhibit 2024.doc	R400_Sequencer.doc 71269 Bytes*** 🔇	D ATI Confidential. Referer	nce Copyright Notice on Cover Pa	ge © ***

ATI 2024 LG v. ATI IPR2015-00325

AMD1044\_0257135

ATI Ex. 2107 IPR2023-00922 Page 1 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20154		2 of 48
<u>Table Of</u>	Contents			
1. OVE	RVIEW			6
1.1 Top	_evel Block Diagram			8
1.2 Data	Flow graph (SP)			
2 INTE	RPOLATED DATA RI	IS		
3. INST	RUCTION STORE	***************************************		
4. SEQ	UENCER INSTRUCTION	ONS		16
5. CON	STANT STORES	*****		
5.1 Mem	ory organizations	ning tables		
52 Man	Dirty bite			1018
522	Eroo List Plock			1010
502	De ellesete Dieck			004040
5.2.3	De-allocate Block	······		
52.4	Operation of Increment	ntal model		<u>2019</u> 18 2010
5.3 Cons	Time Commands			212019
5.5 Cons	tant Waterfalling			
6. LOO	PING AND BRANCHE	S		<u>2221</u> 20
6.1 The	controlling state			
6.2 The	dependant predicate i	nstructions		<u>ZZ<del>Z   20</del></u> 2423
6.4 HW I	Detection of PV,PS			
6.5 Regi	ster file indexing			
6.6 Pred	icated Instruction supp	ort for Texture clause	S	
6.7 Debu	Agging the Shaders	·····		<u>20<del>20</del>24</u>
670	Method I. Debugging		- (40)	<u>202</u> 724
0./.2 7 DIVE	wethoa ∠: Exporting ti	ne values in the GPRs	5 (12)	<u>26</u> 20 2725
8. MUL	TIPASS VERTEX SHA	DERS (HOS)		
9. REG	ISTER FILE ALLOCA	TION		
10. FET(	CH ARBITRATION	*****		<u>28<del>27</del>26</u>
11. ALU	ARBITRATION	*****		<u>2827</u> 26
12. HAN	TENT OF THE RESER		IFOS	<u>2928</u> 27
13. CON 14. THE	OUTPUT FILE			292827
15. IJ FC	RMAT	*****		<u>2928</u> 27
15.1 Inte	erpolation of constant a	attributes		<u>3029</u> 28
16. STA	GING REGISTERS	***************************************	••••••	<u>302928</u>
17. THE 18. VER	TEX POSITION EXPO	RTING	*****	333130
19. EXP	ORTING ARBITRATIC	N		333130
20. EXP	ORT TYPES	*****		<u>3331</u> 30
20.1 Ve	rtex Shading			<u>3331</u> 30

Exhibit 2024.doc R409\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257136

ATI Ex. 2107 IPR2023-00922 Page 2 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
<b>ZUU</b>	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	3 of 48
0.2 Pixe	I Shading			<u>3332</u> 30
1. SPEC	IAL INTERPOLATION	MODES	*****	<u>3432</u> 31
.1.1 Rea /1.2 Spri	tes/ XY screen coordin	ates/ FB information		<u>3432</u> 31 343231
1.3 Auto	generated counters			343331
21.3.1	Vertex shaders			<u>3433</u> 32
21.3.2	Pixel shaders			<u>3433</u> 32
2. STAT	E MANAGEMENT	*****		<u>3533</u> 32
2.1 Para	ameter cache synchron	ization		<u>3533</u> 32
3.1 Vert	ex indexes imports		* * * * * * * * * * * * * * * * * * * *	<u>3534</u> 32 353433
4. REGIS	STERS	*****	****	<u>3634</u> 33
.4.1 Con	trol			<u>3634</u> 33
4.2 Con	text			<u>3634</u> 33 373534
.5. DEBU	text		******	373534
6. INTE	RFACES	*****		373534
.6.1 Ext	ernal Interfaces			<u>3735</u> 34
26.1.1	SC to SQ : IJ Contr	ol bus		<u>3736</u> 34
26.1.2	SQ to SP: Interpolato	r bus	****	<u>3836</u> 35
26.1.3	SQ to SP: Parameter	Cache Read control bu	IS	<u>3836</u> 35
26.1.4	SQ to SX: Parameter	Cache Mux control Bus	\$	<u>3937</u> 36
26.1.5	SQ to SP: Staging Re	egister Data		<u>3937</u> 36
26.1.6	PA to SQ : Vertex inte	erface		<u>3937</u> 36
26.1.7	SQ to CP: State repo	rt		<u>4241</u> 39
26.1.8	SQ to SX: Control bu	s		<u>4241</u> 39
26.1.9	SX to SQ : Output file	control		<u>4241</u> 39
26.1.10	SQ to TP: Control bu	S		<u>4241</u> 39
26.1.11	TP to SQ: Texture sta			<u>4342</u> 40
26.1.12	SQ to SP: Texture sta	all		<u>4342</u> 40
26.1.13	SQ to SP: GPR, Para	meter cache control an	d auto counter	<u>4342</u> 40
26.1.14	SQ to SPx: Instructio	าร		<u>4443</u> 41
26.1.15	SP to SQ: Constant a	ddress load		<u>4544</u> 41
26.1.16	SQ to SPx: constant	proadcast		<u>4544</u> 41
26.1.17	SP0 to SQ: Kill vecto	load		<u>4544</u> 42
26.1.18	SQ to CP: RBBM bus			<u>4544</u> 42
26.1.19	CP to SQ: RBBM bus			<u>4544</u> 42
7. EXAM	IPLES OF PROGRAM	EXECUTIONS		<u>4644</u> 42
27.1.1	Sequencer Control of	a Vector of Vertices		<u>4644</u> 42
27.1.2	Sequencer Control of	a Vector of Pixels		<u>4746</u> 43
2713	Notes			484644

AMD1044\_0257137

ATI Ex. 2107 IPR2023-00922 Page 3 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE			
	24 September, 2001	4 September, 20154		4 of 48			
28. OPEN ISSUES							

Exhibit 2024.doc R400\_Sequencer.doc 71269 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257138

ATI Ex. 2107 IPR2023-00922 Page 4 of 260

000000000000000000000000000000000000000			1	
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	5 of 48
Revision	Changes:			
<b>Rev 0.1 (Laur</b> Date: May 7, 2	ent Lefebvre) 1001	First dr	aft.	
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 Rev 0.7 (Laure Date : October Rev 0.8 (Laure Date : October Rev 0.9 (Laure Date : October	ent Lefebvre) 2001 ent Lefebvre) 6, 2001 ent Lefebvre) 24, 2001 ent Lefebvre) ber 7, 2001 ent Lefebvre) ber 24, 2001 ent Lefebvre) 5, 2001 ent Lefebvre) 8, 2001 ent Lefebvre) - 17, 2001	Change SP. Ad Review August Added file and flow of Added Change archited Store n data de Change flexible Incorpo flow m	ed the interfaces to reflect the changes ded some details in the arbitration sect ed the Sequencer spec after the mee 3, 2001. the dynamic allocation method for r I an example (written in part by Vic) pixels/vertices in the sequencer. timing diagrams (Vic) ed the spec to reflect the new cture. Added interfaces. constant store management, inst nanagement, control flow manageme pendant predication. ed the control flow method to be . Also updated the external interfaces. rated changes made in the 10/18/01 eeting. Added a NOP instruction re	in the ion. ting on egister of the R400 ruction nt and more control moved
Rev 1.0 (Laure Date : October Rev 1.1 (Laure Date : October	ent Lefebvre) - 19, 2001 ent Lefebvre) - 26, 2001	now m the cc register Refined Added precisid	eeting. Added a NOP instruction, re onditional_execute_or_jump. Added s. I interfaces to RB. Added state register SEQ→SP0 interfaces. Changed on. Changed VGT→SP0 interface. is added	debug rs. delta Debug
Rev 1.2 (Laure Date : Novemb Rev 1.3 (Laure Date : Novemb	ent Lefebvre) per 16, 2001 ent Lefebvre) per 26, 2001	Interfac	the different interpolation modes.	C.
Rev 1.4 (Laure Date : Decemb Rev 1.5 (Laure Date : Decemb	ent Lefebvre) ber 6, 2001 ent Lefebvre) ber 11, 2001	Added the VG manag Remov directly consta	the auto incrementing counters. Ch T→SQ interface. Added content on co ement. Updated GPRs. ed from the spec all interfaces that v tied to the SQ. Added explanation the magagement Added P	nanged onstant weren't ons on
Rev 1.6 (Laure Date : January	ent Lefebvre) r 7, 2002	Synchra Added detail a call ins Added undate.	management. Added Fi onization fields and explanation. more details on the staging register. about the parameter caches. Chang struction to a Conditionnal_call instr details on constant managemen d the diagram.	Added ed the uction. t and
Rev 1.7 (Laure Date : Februar <u>Rev 1.8 (Laure</u> Date : March 4	ent Lefebvre) y 4, 2002 ent Lefebvre) I, 2002	Added interfac <u>New in</u> <u>clause</u> instruct	Real Time parameter control in the Updated the control flow section. terfaces to the SX block. Added the modifier, removed the end of ions.	ne SX <u>end of</u> clause
Exhibit 2024.docR	400_Sequencer.doc 71269 Bytes*1	** © ATI Confidential. Refere	ence Copyright Notice on Cover Pag	e © ***

AMD1044\_0257139

ATI Ex. 2107 IPR2023-00922 Page 5 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20154		6 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 2024.docR400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257140

ATI Ex. 2107 IPR2023-00922 Page 6 of 260

# PROTECTIVE ORDER MATERIAL



Exhibit 2024 doe R400\_Sequenceer doe 71289 Byres\*\*\*\* 
ATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\*

AMD1044\_0257141

ATI Ex. 2107 IPR2023-00922 Page 7 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 20154		8 of 48

# 1.1 Top Level Block Diagram



#### Figure 2: Reservation stations and arbiters

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

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

Exhibit 2024.docR400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257142

ATI Ex. 2107 IPR2023-00922 Page 8 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	9 of 48

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator, which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

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

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

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

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

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

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

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

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

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

Exhibit 2024.doc R400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257143

ATI Ex. 2107 IPR2023-00922 Page 9 of 260



AMD1044\_0257144

ATI Ex. 2107 IPR2023-00922 Page 10 of 260

<b>A</b> li	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1
	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	11 of 48	
					19909909

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

# 1.3 Control Graph



Figure 4: Sequencer Control interfaces

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

# 2. Interpolated data bus

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

Exhibit 2024.docR400\_Sequencer.doc 71269 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257145

ATI Ex. 2107 IPR2023-00922 Page 11 of 260



AMD1044\_0257146

ATI Ex. 2107 IPR2023-00922 Page 12 of 260 PROTECTIVE ORDER MATERIAL

		23					<u>ک</u> ہم ک	2 4 5	> ဖ် ဂ	<u>&gt; 6 0</u>	
		-22 T					32-4	39.5	2 4 4 3 5 5 7	V \ 44-6( 47 6	$-\times$
		[21] T					> å 6	> ഗ്റ്റ് റ്റ്റ്റ്റ്റ്റ്റ്റ്റ്റ്റ്റ്റ്റ്റ്	24- 1	> % E	
		L20					> ~	>1	> ∞	245	
		T19 <sup>-</sup>						¥	С Ш	μ	
		T18								å	
щ	48	T17			≿ ଘ	≿⊡		8	5	S	
PAC	13 of	Т16			ЮШ	E1				BO	2
		T15			ЕО	Ш Ц	5	D2			Ω
NUM.	NA	T14	≳Б	≿8		≿8	ប៊	5 C	C5		
REV. I	XX-RE	T13	5	5		8	<u>8</u>				
IENT-	XXX	T12	Б	5		å	AO	A1	A2		
ocuN	GEN-(	T11		≿2	ç S	000	2		о Ш	Ξ	
ā		T10		2	C5	ŭ	2			8	
	4	<u>Т</u> 9		5	C5			පි	ស	5 C	
TE	, 2015	Т8	≿ ເິ	≿ ଚ	≿Շ	≳ છ				BO	Gamman
DT D/	embel	17	ប៊	8	5	8	D	D2			٩
Ш	4 Sept	Тб	ទ	8	5	8	ទ	5 2	C5		
	P.V.	T5	×₽			∑ 8	<u>B</u>				
DATE	2001	Т4	B1			BO	AO	A1	A2		
ATE D	mber,	T3	81			Bo	51 <del>2</del>	52, 52,	56 - X	3 ° ×	
RIGIN	Septe	Т2	۶X	۶¥	≿ &		35 ⊰ 35 -	⊁ ဗို ဗိ	∑ 4 &	X 4 4	$\geq$
ō	24	11	AO	A1	A2		16 <sup>†</sup> X	3 2 ×	24- 27	3 % ×	×
PC		Р С	Ao	A1	A2		×°¦	¥ 7-4	≍ ∾ ≿	24 2 ≾	
	र्		<sup>с</sup> о	₽ G	2 SP	ი ი	С O	Ч <del>С</del>	S C	ია	

Figure 6: Interpolation timing diagram

Exhibit 2024 doe R400\_Sequences 71289 Bytes\*\*\*\* @ ATI Confidential. Reference Copyright Notice on Cover Page @ \*\*\*

AMD1044\_0257147

ATI Ex. 2107 IPR2023-00922 Page 13 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	1
24 September, 2001	4 September, 20154		14 of 48	

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 2024.docR490\_Sequences.doc 71269 Bytes\*\*\* <sup>©</sup> ATI Confidential. Reference Copyright Notice on Cover Page <sup>©</sup> \*\*\*

AMD1044\_0257148

ATI Ex. 2107 IPR2023-00922 Page 14 of 260

# PROTECTIVE ORDER MATERIAL



# R400 CP's Views of Instruction Memory







Exhibit 2024 doo R400\_Gequencer doo 71289 Byres\*\*\*\* @ ATI Confidential. Reference Copyright Notice on Cover Page @ \*\*\*

AMD1044\_0257149

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		16 of 48

# 4. Sequencer Instructions

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

# 5. Constant Stores

# 5.1 Memory organizations

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

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

The texture state is also kept in a similar memory. The size of this memory is 128x192 bits. The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

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

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

# 5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF\_WR\_BASE). On the read side, one level of indirection is used. A register (SQ\_CONTEXT\_MISC.CF\_RD\_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

# 5.25.3 Management of the re-mapping tables

#### 5.2.15.3.1 R400 Constant management

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

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

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space

Exhibit 2024.doc R400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

Formatted: Bullets and Numbering

AMD1044\_0257150

ATI Ex. 2107 IPR2023-00922 Page 16 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	17 of 48
5 mm / m				

is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32\*2+32 = 96 entries and above.

#### 5.2.25.3.2 Proposal for R400LE constant management

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ\_IDLE and PA\_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 9: De-allocation mechanismFigure 9: De-allocation mechanism

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).

Exhibit 2024.doc R400\_Sequencer.doc 71269 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

Formatted: Bullets and Numbering

AMD1044\_0257151

ATI Ex. 2107 IPR2023-00922 Page 17 of 260



AMD1044\_0257152

ATI Ex. 2107 IPR2023-00922 Page 18 of 260



#### Figure 9: De-allocation mechanism for R400LE

# 5.2.35.3.3 Dirty bits

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

#### 5.2.45.3.4 Free List Block

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

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

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

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

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

Exhibit 2024.doc R409\_Sequencer.doc 71269 Bytes\*\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

#### Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0257153

ATI Ex. 2107 IPR2023-00922 Page 19 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	]
	24 September, 2001	4 September, 20154		20 of 48	
_					Formatted: Bullets and Numbering

5.2.55.3.5 De-allocate Block

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

# 5.2.65.3.6 Operation of Incremental model

The basic operation of the model would start with the write\_ptr, stop\_ptr, read\_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context stored in the additional renaming table will be 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.

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.

# 5.35.4 Constant Store Indexing

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

Exhibit 2024.docR400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0257154

ATI Ex. 2107 IPR2023-00922 Page 20 of 260

AP	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	21 of 48

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.

## 5.45.5 Real Time Commands

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

# 5.55.6 Constant Waterfalling

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



Exhibit 2024 docR409\_Sequencer.doc 71269 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

Formatted: Bullets and Numbering Formatted: Bullets and Numbering

AMD1044\_0257155

ATI Ex. 2107 IPR2023-00922 Page 21 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		22 of 48

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 alu[7:0][7:0] Pixel\_shader\_fetch[7:0][7:0] Pixel\_shader\_alu[7:0][7:0]

Vertex\_shader\_fetch[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 // 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 nine basic instructions:

Execute Conditional\_execute Conditional\_Execute\_Predicates Conditional jump Conditionnal\_Call Return Loop\_start Loop\_end End\_of\_clause Conditional\_End\_of\_clause NOP

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

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

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

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

Exhibit 2024.doc R400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257156

ATI Ex. 2107 IPR2023-00922 Page 22 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	23 of 48

Conditionnal\_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.

Conditional\_execute\_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition. End\_of\_clause marks the end of a clause.

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

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

NOP is a regular NOP

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

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

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

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

	Execute							
47	46 42	4141 24	40 24	23 12	11 0			
Addressing	00001	LastRESERVE	RESERVED	Instruction	Exec Address			
_		Ð		count				

Execute up to 4k instructions at the specified address in the instruction memory. If Last is set, this is the last group of instructions of the clause.

	NOP					
47	46 42	<u>41</u> 410	40 0			
Addressing	00010	LastRESERVE	RESERVED			

This is a regular NOP. If Last is set, this is the last instruction of the clause.

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

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions). If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Conditional_Execute_Predicates								
47	46 42	41	<u>40 35</u>	34 33	32	31 24	23 12	11 0
		35						
Addressing	00100	Last	RESERVED	Predicate	Condition	RESERVED	Instruction	Exec Address
		RES		vector			count	
		ER₩						
		ED						

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Exhibit 2024.doc R400\_Sequencer.doc 71269 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257157

ATI Ex. 2107 IPR2023-00922 Page 23 of 260

<b>A</b> î	ORIGII 24 Sept	NATE DATE ember, 2001	EDIT DATE <u>4 September, 20154</u> Mareb, 20024	R400 Sequencer Specification	PAGE 24 of 48		
Loop Start							
47	46 42		41 17	16 12	11 0		
Addressing	00101		RESERVED	loop ID	Jump address		

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

Loop_End						
47	46 42	41 17	16 12	11 0		
	00110	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 35	34 33	32	31 12	11 0	
	00111	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	1
	01000	RESERVED	1
Addressing			

Pops the topmost address from the stack and jumps to that address. If nothing is on the stack, the program will just continue to the next instruction.

Conditionnal_Jump							
47	46 42	41	40 33	32	31	30 12	11 0
	01001	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.

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

Marks the end of a clause.

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

# 6.3 Data dependant predicate instructions

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

Exhibit 2024.docR490\_Sequences-dec 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257158

ATI Ex. 2107 IPR2023-00922 Page 24 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	]
24 September, 2001	4 September, 20154 March, 20024 Eabruary	GEN-CXXXXX-REVA	25 of 48	

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\_SETÉO\_# - SETEO

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.

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.

Exhibit 2024.doc\_R409\_Sequencer.doc 71269 Bytes\*\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257159

ATI Ex. 2107 IPR2023-00922 Page 25 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		26 of 48

6.6 Predicated Instruction support for Texture clauses

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

# 6.7 Debugging the Shaders

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

#### 6.7.1 Method 1: Debugging registers

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

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

The sequencer will detect the following groups of errors:

- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:

jump errors

- relative jump address > size of the control flow program
- call stack
  - call with stack full return with stack empty

return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB\_PROB\_BREAK register is set.

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

{ISSUE : Interrupt to the driver or not?}

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

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

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

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

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

Exhibit 2024.docR400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257160

ATI Ex. 2107 IPR2023-00922 Page 26 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	27 of 48

# 7. Pixel Kill Mask

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

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

# 8. Multipass vertex shaders (HOS)

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

# 9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX\_REG\_SIZE for vertices and PIXEL\_REG\_SIZE for pixels.

Exhibit 2024.docR400\_Sequencer.doc 71269 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257161

ATI Ex. 2107 IPR2023-00922 Page 27 of 260



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 2024.doc R400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257162

ATI Ex. 2107 IPR2023-00922 Page 28 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	]
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	29 of 48	

# 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).

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

PO	P1
P2	P3

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

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

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

Exhibit 2024.docR400\_Sequencer.dec 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257163

ATI Ex. 2107 IPR2023-00922 Page 29 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		30 of 48

Adds: 8

FORMAT OF PO'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) or (I = 0))) {
           if(| != 0) {
             P0 = A
           } else if(J != 0) {
              P0 = B;
           } else {
              PO = 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 2024.docR409\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257164

ATI Ex. 2107 IPR2023-00922 Page 30 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	31 of 48

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 12. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 11Figure 11Figure 11.

Basis for 8-deep Latch Memory (fror	n R300)		
8x24-bit	11631	$\mu^2$	$60.57813\mu^2\text{per}$ bit
Area of 96x8-deep Latch Memory	46524	$\mu^2$	
Area of 24-bit Fix-to-float Converter	4712	μ <sup>2</sup> per conv	erter
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 u <sup>2</sup>

Figure 11:Area Estimate for VGT to Shader Interface

Exhibit 2024.doc R400\_Sequencer.doc 71269 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257165

ATI Ex. 2107 IPR2023-00922 Page 31 of 260



#### 17. The parameter cache

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

MEMORY NUMBER	ADDRESS
4 bits	7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current\_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current\_Location by VS\_EXPORT\_COUNT\_7 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS\_EXPORT\_COUNT\_7 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 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> 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 2\*VS\_EXPORT\_COUNT\_7 to Current\_Location and reset the memory count to 0 before the next vector begins).

Exhibit 2024.docR400\_Sequencer.doc 71289 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257166

ATI Ex. 2107 IPR2023-00922 Page 32 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	33 of 48

# 18. Vertex position exporting

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

# 19. Exporting Arbitration

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

1) Position exports and position exports cannot be co-issued.

All other types of exports can be co-issued as long as there is place in the receiving buffer.

{ISSUE: Do we move the parameter caches to the SX?}

# 20. Export Types

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

## 20.1 Vertex Shading

- 0:15 16 parameter cache
- 16:31 Empty (Reserved?)
- 32:43 12 vertex exports to the frame buffer and index
- 44:47 Empty
- 48:59 12 debug export (interpret as normal vertex export)
- 60 - export addressing mode
- 61 - Empty
- 62 - position 63
  - sprite size export that goes with position export (point\_h,point\_w,edgeflag,misc)

# 20.2 Pixel Shading 0

- Color for buffer 0 (primary)
- Color for buffer 1 1
- Color for buffer 2 2
- 3 - Color for buffer 3
- 4:7 - Empty
- 8 - Buffer 0 Color/Fog (primary)
- Buffer 1 Color/Fog 9
- 10 - Buffer 2 Color/Fog
- Buffer 3 Color/Fog 11
- 12:15 Empty
- 16:31 Empty (Reserved?) - 12 exports for multipass pixel shaders.
- 32:43 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)

Exhibit 2024.doc R400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257167

ATI Ex. 2107 IPR2023-00922 Page 33 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		34 of 48

# 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, no gen\_st - I0 = screen x, screen y, garbage, faceness Param\_Gen\_I0 enable, snd\_xy enable, no 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 2024.docR400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257168

ATI Ex. 2107 IPR2023-00922 Page 34 of 260



Figure 13: GPR input mux Control

#### 22. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

# 22.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC\_SQ\_new\_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

#### 23. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix→float converter and expander and write the converted values to the GPRs. The Xys are currently SCREEN SPACE COORDINATES. The values in the XY buffers will wrap. See section 21.2 for details on how to control the interpolation in this mode.

#### 23.1 Vertex indexes imports

In order to import vertex indexes, we have 16 8x96 staging registers. These are loaded one line at a time by the VGT block (96 bits). They are loaded in floating point format and can be transferred in 4 or 8 clocks to the GPRs.

Exhibit 2024.docR409\_Sequencer.doc 71259 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257169

ATI Ex. 2107 IPR2023-00922 Page 35 of 260



# 24. Registers

# 24.1 Control

	REG_DYNAMIC REG_SIZE_PIX	Dynamic allocation (pixel/vertex) of the register file on or off. Size of the register file's pixel portion (minimal size when dynamic allocation turned
	REG_SIZE_VTX	Size of the register file's vertex portion (minimal size when dynamic allocation turned on)
	ARBITRATION_POLICY INST_STORE_ALLOC	policy of the arbitration between vertexes and pixels interleaved, separate
	INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
	INST_BASE_PIX	start point for the pixel shader instruction store
	ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
	ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
	INSTRUCTION	This is where the CP puts the base address of the instruction writes and type (auto- incremented on reads/writes) Register mapped
	CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
	CONSTANTS_RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
	CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT_EO_RT). The re-mapping table operates on the rest of the memory
	TSTATE_EO_RT	This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE EO RT). The re-mapping table operates on the rest of the memory
	EXPORT_LATE	Controls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.
24	1.2 Context	
	VS_FETCH_{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
	VS_BASE	base pointer for the vertex shader in the instruction store
		sing of the substant she show (the first methods are tool and show means (2))

VS_BASE	base pointer for the vertex shader in the instruction store
VS CF SIZE	size of the vertex shader (# of instructions in control program/2)
PS CF SIZE	size of the pixel shader (# of instructions in control program/2)
PS SIZE	size of the pixel shader (cntl+instructions)
VS SIZE	size of the vertex shader (cntl+instructions)
PS NUM REG	number of GPRs to allocate for pixel shader programs
VS NUM REG	number of GPRs to allocate for vertex shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1
PROVO VERT	0 · vertex 0 1· vertex 1 2· vertex 2 3· Last vertex of the primitive
PARAM_WRAP	64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear 1=cylindrical)
PS EXPORT MODE	Oxxxx · Normal mode
	1xxxx : Multipass mode
	If normal, bbbz where bbb is how many colors $(0-4)$ and z is export z or not
	If multipass 1-12 exports for color.
VS EXPORT MASK	which of the last 6 ALU clauses is exporting (multipass only)
VS EXPORT MODE	0: position (1 vector), 1: position (2 vectors), 3:multipass
VS EXPORT	
_COUNT_{06}	Six 4 bit counters representing the # of interpolated parameters exported in clause 7
	(located in VS_EXPORT_COUNT_6) OR
	# of exported vectors to memory per clause in multipass mode (per clause)
PARAM_GEN_I0	Do we overwrite or not the parameter 0 with XY data and generated T and S values
E-bible 2024 A-state 0.0 Commence day	21220 Date: *** @ ATI Confidential Reference Convright Notice on Cover Page @ ***

AMD1044\_0257170

ATI Ex. 2107 IPR2023-00922 Page 36 of 260
	ORIGINATE [	DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September,	2001	4 September, 20154	GEN-CXXXXX-REVA	37 of 48
GEN_IND	)EX	Auto gen and R2 fo	erates an address from 0 to or vertex shaders	XX. Puts the results into R0-1 for pi	xel shaders
CONST_ CONST_ CONST_ INST_PR CF_BOO CF_LOOI CF_LOOI CF_LOOI	BASE_VTX (9 bits) BASE_PIX (9 bits) SIZE_PIX (8 bits) SIZE_VTX (8 bits) ED_OPTIMIZE LEANS P_COUNT P_START P_START P_STEP	Logical B Logical B Size of th Size of th Turns on always e 256 boole 32x8 bit o 32x8 bit o 32x8 bit o	ase address for the constan ase address for the constan le logical constant store for v the predicate bit optimizatio vecuted). ean bits counters (number of times w counters (init value used in ir counters (step value used in ir	ts of the Vertex shader ts of the Pixel shader pixel shaders n (if of, conditional_execute_predica e traverse the loop) ndex computation) index computation)	ates is
25. <u>DEB</u>	UG Registe	<u>rs</u>			
25.1 Cor	ntext				
DB_F DB_F DB_II DB_II DB_E DB_C MOI DB_C	ROB_ADDR ROB_COUNT ROB_BREAK NST_COUNT REAK_ADDR LAUSE DE_ALU_{07} LAUSE	instructio number o break the instructio break ad clause m	n address where the first pro of problems encountered dur clause if an error is found. n counter for debug method dress for method number 2 ode for debug method 2 (0: 1	blem occurred ing the execution of the program 2 normal, 1: addr, 2: kill)	
	JE_FEICH_{07}	· .	lause mode for debug metri	ou 2 (0. normai, 1. adur, 2. kiii)	
25.2 Cor	ntrol				
DB_A DB_T	LUCST_MEMSIZE		size of the physical ALU cons Size of the physical texture si	stant memory tate memory	
26. Inter	faces				
26.1 Exte	ernal Interfa	ces			
Whenever an named SQ→S	x is used, it mear SPx it means that S	ns that th SQ is goin	e bus is broadcast to all un g to broadcast the same info	its of the same name. For example ormation to all SP instances.	e, if a bus is
26.1.1 SC	C to SQ : IJ Co	ontrol b	us		
This is the co	ntrol information se ader program on th t is going to be igr	ent to the le sent pi lored and oup of a	sequencer in order to contro xels. This information is sen XY information is going to uads are from the same prin	of the IJ fifos and all other information t over 2 clocks, if SENDXY is asse be sent on the IJ bus (for the quad mitive, all quads of a vector are fro	on needed to rted the next s that where on the same

Exhibit 2024.doc.R400\_Sequences.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257171

ATI Ex. 2107 IPR2023-00922 Page 37 of 260

	ORIGINATE [	DATE	EDIT D	DATE		R400 Sequencer Specification	PAGE			
CAUL	24 September	2001	001 4 September, 20154		54		38 of 48			
	·		Marob '	ab 20024			<u> </u>			
Name		Directio	on	Bits	Des	scription				
SC_SQ_q_wr	_mask	SC→S	Q	4	Qu	ad Write mask left to right				
SC_SQ_lod_c	orrect	SC→S	Q	24	LO	D correction per quad (6 bits per quad	)			
SC_SQ_parar	n_ptr0	SC→S	Q	11	PS	tore pointer for vertex 0				
SC_SQ_parar	n_ptr1	SC→S	Q	11	PS	tore pointer for vertex 1				
SC SQ parar	n ptr2	SC→S	Q	11	PS	tore pointer for vertex 2				
SC SQ end	of vect	SC→S	Q	1	End	I of the vector				
SC SQ store	dealloc	SC→S	Q	1	Dea	allocation token for the P Store				
SC SQ state		SC→S	Q	3	Sta	te/constant pointer				
SC SQ valid	pixel	SC→SQ		16	Valid bits for all pixels					
SC SQ null p	orim	SC→SQ		1	Nul	Null Primitive (for PC deallocation purposes)				
SC SQ end	of prim	SC→SQ		1	End Of the primitive					
SC SQ send	XV	SC→S	Q	1	Ser	inding XY information [XY information	is going to be			
	_ ,				sen	t on the next clock]				
SC_SQ_prim_	type	SC→S	Q	3	Rea	al time command need to load te	x cords from			
					alte	rnate buffer. Line AA, Point AA and	Sprite reads			
					the	r parameters from GEN_T and GEN_	S GPRs.			
					000	) : Normal				
					011	: Real Time				
					100	) : Line AA				
					101	: Point AA				
					110	) : Sprite				
SC_SQ_new_	vector	SC→S	Q	1	Thi	s primitive comes from a new vector	or of vertices.			
					Ma	ke sure that the corresponding verte	ex shader has			
					finis	shed before starting the group of pixel	S.			
SC_SQ_RTRr	1	SQ→S	с	1	Sta	lls the PA in n clocks				
SC SQ RTS		SC→S	0	1	SC	ready to send data				

## 26.1.2 SQ to SP: Interpolator bus

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive
			000 : Normal
			011 : Real Time
			100 : Line AA
			101 : Point AA
			110 : Sprite
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_gen_I0	SQ→SPx	1	Generate I0 or not. This tells the interpolators not to
			use the parameter cache but rather overwrite the data
			with interpolated 1 and 0. Overwrite if gen 10 is high.

#### 26.1.3 SQ to SX: Interpolator bus

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_mux0	<u>SQ→SXx</u>	11	Parameter Cache Pointer
SQ_SXx_mux1	<u>SQ→SXx</u>	11	Parameter Cache Pointer
<u>SQ SXx mux2</u>	<u>SQ→SXx</u>	11	Parameter Cache Pointer
SQ_SXx_RT_switch	<u>SQ→SXx</u>	1	Selects between RT and Normal data

#### 26.1.4SQ to SP: Parameter Cache Read control bus

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

Exhibit 2024.docR400\_Sequencer.doc 71269 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

#### - Formatted: Bullets and Numbering

AMD1044\_0257172

ATI Ex. 2107 IPR2023-00922 Page 38 of 260

	ODICINIA				.		DAGE	1	
	24 Senter	te DATE	4 Senten	nher 2	0154	GEN-CXXXXX-REVA	39 of 48		
26.1.4.00			March 200	1001, 2 124 Eal			00 01 40		
<del>20.1.4-3</del> 6	<del>LIO SA: P</del> I	агатнек	H Gache M	UX GO	miron	<del>sus</del>			(
<del>26.1.6</del> 26.	<u>1.4_</u> SQ to	SP: Sta	aging Regis	ster D	Pata			Y.	Formatted: Bullets and Numbering
This is a broad	dcast bus that	sends the	VSISR informa	tion to	he stagi:	ng registers of the shader pipes.			
Name	reier data	Direct	tion	Bits	Descrip	otion s of indexes or HOS surface informati	<u></u>		
SQ_SPX_VgL_V	sisi_uata	<u>SQ</u> →.		1	0: Norm	al 96 hits per vert 1: double 192 hits r	on vert		
SQ SP0 data	valid	SQ→S	SP0	1	Data is	valid			
SQ_SP1_data	valid	SQ→S	SP1	1	Data is	valid			
SQ_SP2_data	valid	SQ→S	SP2	1	Data is	valid			
SQ_SP3_data_	_valid	SQ→S	SP3	1	Data is	valid			
<u>26.1.726.</u>	<u>1.5_</u> PA to	SQ : Ve	ertex interfa	ice			4	Y	Formatted: Bullets and Numbering
<del>26.1.7.1</del> 26.	1.5.1 Inter	face Sig	nal Table						
The area diffe	erence betwee	en the two	methods is n	ot suffi	cient to	warrant complicating the interface	or the state		
requirements	of the VSISRs	. Therefo	re, the POR fo	r this	nterface	is that the VGT will transmit the	data to the		
VSISRs (via t	the Shader S	equencer	in full, 32-bit	floatir	ig-point	format. The VGT can transmit up	to six 32-bit		
floating-point v	values to each	VSISR w	here four or mo	ore val	ues requ	ire two transmission clocks. The da	ata bus is 96		
bits wide.									
Name		Bits	Description				]		
PA SQ vat vs	sisr data	96	Pointers of inde	exes or	HOS sur	face information			
PA_SQ_vgt_vs	sisr_double	1	0: Normal 96 b	its per	/ert 1: do	uble 192 bits per vert			
PA_SQ_vgt_er	nd_of_vector	1	Indicates the la data, "end_of_	st VSI vector"	SR data s is set on	et for the current process vector (for the second vector)	double vector		
PA_SQ_vgt_vs	sisr_valid	1	Vsisr data is va	ılid					
PA_SQ_vgt_st	ate	3	Render State( "PA_SQ_vgt_e	6*3+3 nd_of_	or consta vector" is	ants). This signal is guaranteed to be high.	correct when		
PA_SQ_vgt_se	ənd	1	Data on the VC interface hands	GT_SQ shaking	is valid re )	eceive (see write-up for standard R40	0 SEND/RTR		
SQ_PA_vgt_rti	r	1	Ready to rec handshaking)	ceive (	see writ	e-up for standard R400 SEND/R	TR interface		
2617226	1.5.2. Inter	face Dia	orams						Formatted: Bullets and Numbering
and V. & . I . and find V.	1.0.12 111001	1400 1914	<u>Branno</u>						
Exhibit 2024.docR	400-Sequencer.doc	71269 Bytes**	* © ATI Confid	lential	Referen	nce Copyright Notice on Cover Pa	ige © ***		
		-,>				., .	<b>-</b>		

ATI Ex. 2107 IPR2023-00922 Page 39 of 260

# PROTECTIVE ORDER MATERIAL



Exhibit 2024 doc R400\_Sequencer doo 71209 Byes\*\*\*\* (Confidential. Reference Copyright Notice on Cover Page (Cover Page (Cover

AMD1044\_0257174

ATI Ex. 2107 IPR2023-00922 Page 40 of 260 PROTECTIVE ORDER MATERIAL



	ORIGINATE [	DATE	EDIT D	ATE	R400 Sequencer Specification	PAGE	
	24 September	, 2001	4 September	er, 201	54	42 of 48	
5 <u>.1.8</u> 26.1. <sup>,</sup>	6 SQ to C	P: State	ə report			4-	
ime		Directio	n	Bits	Description		
2_CP_vrtx_ sta	ite	SEQ→C	P	3	Oldest vertex state still in the pipe		
2_CP_pix_state	9	SEQ→C	P	3	Oldest pixel state still in the pipe		
5 <u>.1.926.1.</u>	7_SQ to S.	X: Cont	trol bus			*-	Formatted: Bullets and Numbering
ime		Directio	n	Bits	Description		
⊋_SXx_exp_P	ixel	SQ→SX	x	1	1: Pixel		
	tart	20. ev		4	U: Vertex		
2_SAX_exp_st	tan	SQASY	x	1	clause	an exporting	
SXx exp C	lause	SQ→SX	x	3	Clause number, which is needed for verte	x clauses	
SXx_exp_S	tate	SQ→SX	x	3	State ID, which is needed for vertex clause	36	
⊋ SXx exp V	'Dest	<u>SQ</u> →SX	x	6	Export Destination		
SXx exp e	xportID	SQ→SX	X	1	ALUID		
_							
ese fields are	sent synchron	ously with	SP export da	ata, des	scribed in SP0→SX0 interface		
SUE: VVnere (	are the PC poir	<del>11016}</del>					Competted: Dullets and Numbering
5 <u>.1.1026</u> .1	1.8_SX to 5	SQ : Ou	ıtput file c	contro	5/	4-	(rormatted: buildts and Numbering
		Directio	n	Rite	Description		
(x SQ Export	t count rdv	SXx-S	2	1	Raised by SX0 to indicate that the followin	a two fields	
ovev_mxhere	,				reflect the result of the most recent export		
(x_SQ_Export	_Position	SXx→S0	2	1	Specifies whether there is room for anothe	er position.	
(x_SQ_Export	Buffer	SXx→S0	2	7	Specifies the space available in the output	buffers.	
					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)	e 64 for each of	
					65-127: RESERVED		
							(
<del>5     </del> 26	9 SQ to	TP: Cor	ntrol bus			4-	
5.1.1126.1 nce every cloci ready or not. T io provides the effech return c	L9_SQ to k, the fetch unii This way the se e instruction ar data.	TP: Cor t sends to quencer c id constan	ntrol bus the sequence an update th its for the feto	er on v e fetch ch to e	which clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file w	a in the GPRs he sequencer vhere to write	Formatted: Bullets and Numbering
5.1.1126.1 nce every cloci ready or not. T io provides the effech return c Name	L.9_SQ to k, the fetch unii Chis way the se e instruction an data.	TP: Coi t sends to equencer c id constan	ntrol bus the sequence an update th its for the feto	er on v e fetch ch to e Bits	which clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file v Description	a in the GPRs he sequencer vhere to write	
5.1.1126.1 nce every cloci ready or not. T io provides the error fetch return c Name TPx_SQ_data TPx_SQ_clau	L.9_SQ to k, the fetch unii Chis way the se e instruction ar data. a_rdy use num	TP: Coi t sends to equencer c nd constan Directio $TPx \rightarrow S$ $TPx \rightarrow S$	ntrol bus the sequencian update th its for the feto n Q Q	er on v e fetch ch to e Bits 1 3	which clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file w Description Data ready Clause number	a in the GPRs he sequencer vhere to write	Formatted: Bullets and Numbering
5.1.1126.1 nce every clocl ready or not. T so provides the etch return c Name TPx_SQ_dat TPx_SQ_clau	L.9_SQ to k, the fetch unii Chis way the se e instruction ar data. a_rdy use_num	TP: Coi t sends to equencer c nd constan Directio $TPx \rightarrow S^{+}$ $TPx \rightarrow S^{+}$	ntrol bus the sequence an update th its for the fete n Q Q	er on v e fetch ch to e Bits 1 3	which clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file w Description Data ready Clause number	a in the GPRs he sequencer vhere to write	Formatted: Bullets and Numbering
5.1.1126.1 nce every cloci ready or not. T so provides the etch return c Name TPx_SQ_dat TPx_SQ_clau TPx_SQ_clau	L.9_SQ to k, the fetch unii Chis way the se e instruction an data. a_rdy use_num	TP: Coi t sends to equencer c nd constant Directio TPx $\rightarrow$ S: TPx $\rightarrow$ S:	ntrol bus the sequence an update th its for the fete Q Q	er on v e fetch ch to e Bits 1 3 1	which clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file w Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX)	a in the GPRs he sequencer where to write	<b>Formatted:</b> Bullets and Numbering
5.1.1126.1 nce every clocl ready or not. T so provides the efech return c TPx_SQ_data TPx_SQ_clau TPx_SQ_clau TPx_SQ_TPx_con	L.9_SQ to k, the fetch unit Chis way the set e instruction and data. a_rdy use_num De sst	TP: Coi t sends to equencer c d constan Directio TPx $\rightarrow$ S TPx $\rightarrow$ S TPx $\rightarrow$ S	ntrol bus the sequence an update th the for the feter Q Q Q Q X	er on v e fetch ch to e Bits 1 3 1 48	/hich clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file v Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Fetch state sent over 4 clocks (192 bits total	a in the GPRs he sequencer where to write	<b>Formatted:</b> Bullets and Numbering
5.1.1126.1 nce every clocl ready or not. T so provides the e fetch return c TPx_SQ_date TPx_SQ_clau TPx_SQ_clau TPx_SQ_TPx_con SQ_TPx_con SQ_TPx_con	L.9_SQ to k, the fetch unit This way the set e instruction and data. a_rdy use_num be tuct tot clause	TP: Coi t sends to equencer c d constant Directio TPx $\rightarrow$ S TPx $\rightarrow$ S SQ $\rightarrow$ TP SQ $\rightarrow$ TP	ntrol bus the sequence an update th the for the feto Q Q Q x x x	er on v e fetch ch to e Bits 1 3 1 48 24 1	/hich clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file v Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Fetch state sent over 4 clocks (192 bits total Fetch instruction sent over 4 clocks Last instruction sent over 4 clocks	a in the GPRs he sequencer where to write	Formatted: Bullets and Numbering
5.1.1126.1 rece every clocd ready or not. T to provides the effecth return control TPX_SQ_data TPX_SQ_clau TPX_SQ_TPX_cont SQ_TPX_cont SQ_TPX_inst SQ_TPX_tor	L.9_SQ to k, the fetch unit fhis way the se e instruction and data. a_rdy use_num be nuct tuct d_of_clause be	TP: Coi t sends to equencer c ind constant TPx $\rightarrow$ S TPx $\rightarrow$ S SQ $\rightarrow$ TP SQ $\rightarrow$ TP SQ $\rightarrow$ TP	ntrol bus the sequence an update th this for the feto Q Q Q X X X X X	er on v e fetch ch to e Bits 1 3 1 48 24 1 1	/hich clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file v Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Fetch instruction sent over 4 clocks (192 bits total Fetch instruction sent over 4 clocks Last instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX)	a in the GPRs he sequencer where to write	<b>Formatted:</b> Bullets and Numbering
5.1.1126.1 ready or not. T to provides the to provides the fetch return of TPx_SQ_data TPx_SQ_clau TPx_SQ_TPx_con SQ_TPx_con SQ_TPx_enc SQ_TPx_enc SQ_TPx_pha	L.9_SQ to k, the fetch unit This way the se e instruction ar data. a_rdy use_num be nst tuct d_of_clause be sse	TP: Coi t sends to equencer c id constant TPx $\rightarrow$ S TPx $\rightarrow$ S SQ $\rightarrow$ TP SQ $\rightarrow$ TP SQ $\rightarrow$ TP SQ $\rightarrow$ TP	ntrol bus the sequence an update th this for the feto Q Q Q X X X X X X	er on v e fetch ch to e Bits 1 3 1 48 24 1 1 2	hich clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file v Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Fetch state sent over 4 clocks (192 bits total Fetch instruction sent over 4 clocks Last instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal	a in the GPRs he sequencer where to write	<b>Formatted:</b> Bullets and Numbering
5.1.1126.1 ready or not. T to provides the to provides the fetch return of TPx_SQ_data TPx_SQ_clau TPx_SQ_TPx_con SQ_TPx_con SQ_TPx_inst SQ_TPx_inst SQ_TPx_pha SQ_TP0 lod	L.9_SQ to k, the fetch unit l'his way the see e instruction ar data. a_rdy use_num be nut tuct d_of_clause be correct	TP: Coi t sends to equencer c ind constant TPx $\rightarrow$ S TPx $\rightarrow$ S SQ $\rightarrow$ TP SQ $\rightarrow$ TP SQ $\rightarrow$ TP SQ $\rightarrow$ TP SQ $\rightarrow$ TP	ntrol bus the sequence an update th tts for the feto Q Q Q X X X X X X X X X Z	er on v e fetch ch to e Bits 1 3 1 48 24 1 1 2 6	hich clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file v Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Fetch state sent over 4 clocks (192 bits total Fetch instruction sent over 4 clocks Last instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components of	a in the GPRs he sequencer where to write	<b>Formatted:</b> Bullets and Numbering
5.1.1126.1 ince every clocl ready or not. T iso provides the iso fetch return c Name TPx_SQ_data TPx_SQ_clau TPx_SQ_TPx_con SQ_TPx_on SQ_TPx_inst SQ_TPx_ond SQ_TPx_ond SQ_TPx_ond SQ_TP0_lod SQ_TP0_pm	L.9_SQ to k, the fetch unit l'his way the see e instruction ar data. a_rdy use_num be nst tuct d_of_clause be 	TP: Coi t sends to equencer c nd constan TPx $\rightarrow$ Si TPx $\rightarrow$ Si SQ $\rightarrow$ TP SQ $\rightarrow$ TP SQ $\rightarrow$ TP SQ $\rightarrow$ TP SQ $\rightarrow$ TP	ntrol bus the sequence an update th tits for the feto Q Q Q X X X X X X X X D D	er on v e fetch ch to e Bits 1 3 1 48 24 1 1 2 6 4	Arich clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file w Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Fetch state sent over 4 clocks (192 bits total Fetch instruction sent over 4 clocks Last instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components p Pixel mask 1 bit per pixel	a in the GPRs he sequencer where to write	<b>Formatted:</b> Bullets and Numbering
5.1.1126.1 ncc every cloci ready or not. 1 so provides the fetch return c Name TPx_SQ_data TPx_SQ_clau TPx_SQ_TPx_con SQ_TPx_inst SQ_TPx_opta SQ_TPx_pha SQ_TP0_pm SQ_TP1_lod	L.9_SQ to k, the fetch unit Chis way the se e instruction and data. a_rdy use_num be correct ask _correct	TP: Coi t sends to equencer c nd constan $TPx \rightarrow S$ $TPx \rightarrow S$ $TPx \rightarrow S$ $SQ \rightarrow TP$ $SQ \rightarrow TP$ $SQ \rightarrow TP$ $SQ \rightarrow TP$ $SQ \rightarrow TP$	ntrol bus the sequenc can update th tits for the feto Q Q Q X X X X X X X D D 1	er on v e fetch ch to e Bits 1 3 1 48 24 1 1 2 6 4 6 4 6	/hich clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file w Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Fetch state sent over 4 clocks (192 bits total Fetch instruction sent over 4 clocks Last instruction sent over 4 clocks Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components p Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components p	a in the GPRs he sequencer where to write	<b>Formatted:</b> Bullets and Numbering
5.1.1126.1 ready or not. T to provides the effecth return of TPX_SQ_date TPX_SQ_date TPX_SQ_clau TPX_SQ_TPX_con SQ_TPX_con SQ_TPX_inst SQ_TPX_inst SQ_TPX_pha SQ_TP0_lod SQ_TP1_lod SQ_TP1_pm	L.9_SQ to k, the fetch unii Chis way the se e instruction and data. a_rdy use_num be st tuct d_of_clause be correct ask _correct ask	TP: Coi t sends to equencer c nd constan Directio TPx $\rightarrow$ S: TPx $\rightarrow$ S: SQ $\rightarrow$ TP: SQ \rightarrowTP: SQ \rightarrowTP: TP: SQ \rightarrowTP: SQ \rightarrowTP: SQ \rightarrowTP: SQ \rightarrowTP: TP: TP: SQ \rightarrowTP: SQ \rightarrowTP: TP: SQ \rightarrowTP: SQ \rightarrowT	ntrol bus the sequenc an update th the sequenc the sequenc the sequenc an update th the sequence an update the an	er on v e fetch ch to e Bits 1 3 1 48 24 1 1 2 4 6 4 6 4 4	/hich clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file v Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Fetch state sent over 4 clocks (192 bits total Fetch instruction sent over 4 clocks Last instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components p Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components p	a in the GPRs he sequencer where to write	<b>Formatted:</b> Bullets and Numbering
5.1.1126.1 rece every cloci ready or not. 1 o provides the effecth return of TPX_SQ_data TPX_SQ_clau TPX_SQ_TPX_con SQ_TPX_con SQ_TPX_inst SQ_TPX_opta SQ_TPX_opta SQ_TP2_opta SQ_TP0_lod SQ_TP1_pm SQ_TP2_lod	L.9_SQ to k, the fetch unit This way the set e instruction and data. a_rdy use_num be tuct d_of_clause be ase _correct ask _correct ask _correct	TP: Coi t sends to equencer c id constant TPx $\rightarrow$ S TPx $\rightarrow$ S TPx $\rightarrow$ S SQ $\rightarrow$ TP SQ $\rightarrow$ TP	ntrol bus the sequenc an update th this for the feto Q Q Q X X X X X X X X 0 0 0 1 1 2	er on v e fetch ch to e Bits 1 3 1 48 24 1 1 2 6 4 6 4 6 6	/hich clause it is now working and if the data counters for the reservation station fifos. T xecute and the address in the register file v Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Fetch state sent over 4 clocks (192 bits total Fetch instruction sent over 4 clocks Last instruction sent over 4 clocks Last instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components p Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components p Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components p	a in the GPRs he sequencer where to write	• Formatted: Bullets and Numbering

ATI Ex. 2107 IPR2023-00922 Page 42 of 260

เป็	ORIGINATE	DATE	EDI	t date	Ξ	DOCUMENT-REV. NUM.	PAGE	
	24 Septembe	r, 2001	4 Septer	mber, 2	0154	GEN-CXXXXX-REVA	43 of 48	
SQ TP3 la	od correct	SQ→TP?	Marah 20	024 E	LOD co	rrect 3 bits per comp 2 components per	quad	4
SQ_TP3_p	omask	SQ→TP3	3	4	Pixel m	ask 1 bit per pixel	,	
SQ_TPx_cl	lause_num	SQ→TPx	(	3	Clause	number		
SQ_TPx_w	vrite_gpr_index	SQ->TPx		7	Index in	to Register file for write of returned Fetc	ch Data	
1 1 2 2 6	110 TP to	SO' To	vtura eta	a//			ج	
1.12.0	<u>.1.10</u> // 10	50, 10	ALUIG SLC	<i>a11</i>				
TP sends ption (max	this signal to the kimum of 3 clocks	e SQ wher s of pipeline	n its input b e delay).	uffer is	full. The	SQ is going to send it to the SP X cl	locks after	
SQ_SP_fet	tch_Stall							
	*							
) SP wr add	ir l							
a_or _m_aaa			Ļ					
			<b>v</b> ]					
			SU1					
					¥			
					SU2			
						¥		
						SU3		
						SU3		
						SU3		
B			-	T P5://		SU3		
Name TB SO 6-1	teh stoll	Direction	1	Bits	Descrip	SU3		
Name TP_SQ_fet	tch_stall	Directior TP→ SQ	1	Bits 1	Descrip Do not	► SU3		
Name TP_SQ_fet	tch_stall	Directior TP→ SQ	1	Bits 1	Descrij Do not	► SU3		
Name TP_SQ_fet	tch_stall	Direction TP→ SQ SP: Te	n exture sta	Bits 1	Descrip Do not	► SU3		<b>Formatted:</b> Bullets and Numbering
Name TP_SQ_fet	tch_stall	Direction TP→ SQ SP: Te	xture sta	Bits 1 a//	Descrip Do not	► SU3		<b>Formatted:</b> Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPy_fe	tch_stall	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$	n exture sta	Bits 1 a// Bits	Descrip Do not	send more texture request if asserted		<b>Formatted:</b> Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe	tch_stall 5.1.11_SQ to etch_stall	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$	n exture sta	Bits 1 3// Bits 1	Descrip Do not Descrip Do not	SU3		
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe	tch_stall 5.1.11_SQ to etch_stall	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$	xture sta	Bits 1 3// Bits 1	Descrij Do not Descrij Do not	SU3		Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426	tch_stall 5.1.11_SQ to etch_stall 5.1.12_SQ to	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$ SP: GP	exture sta	Bits 1 Bits 1 mete	Descrij Do not Descrij Do not r cach	SU3 Send more texture request if asserted		
Name TP_SQ_fet .1.1326 Name SQ_SPX_fe .1.1426 Name	tch_stall 5.1.11_SQ <i>to</i> etch_stall 5.1.12_SQ <i>to</i>	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$ SP: GP Direction	xture sta	Bits 1 Bits 1 mete	Descrip Do not Descrip Do not r cach	SU3 Send more texture request if asserted Send more texture request i		
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_m	tch_stall 5.1.11_SQ to etch_stall 5.1.12_SQ to vr_addr	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$ SP: GF Direction $SQ \rightarrow SP$	n xture sta PR, Para	Bits 1 Bits 1 mete Bits 7	Descrij Do not Descrij Do not r cach Write s	SU3 Send more texture request if asserted Send more texture request i		Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_w SQ_SPx_w	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr upr rd addr	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$ SP: GP Direction $SQ \rightarrow SP$	n n C PR, Para	Bits 1 Bits 1 mete Bits 7 7	Descrij Do not Descrij Do not r cach Write a Read a	SU3 tion send more texture request if asserted e control and auto counter ption ddress ddress		<b>Formatted:</b> Bullets and Numbering
Name TP_SQ_fet 0.1.1326 Name SQ_SPx_fe 0.1.1426 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr upr_rd_addr upr re_addr	Direction TP $\rightarrow$ SQ SP: Te Direction SQ $\rightarrow$ SP Direction SQ $\rightarrow$ SP SQ $\rightarrow$ SF SQ $\rightarrow$ SF SQ $\rightarrow$ SF	PR, Para	Bits 1 Bits 1 Bits 1 Bits 7 7 1	Descrig Do not Descrig Do not r cach Write a Read a Read a	sub send more texture request if asserted ation send more texture request if asserted e control and auto counter ption ddress ddress inable		<b>Formatted:</b> Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_w SQ_SPx_w SQ_SPx_g SQ_SPx_g SQ_SPx_g	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr pp_rd_addr pp_re_addr pp_rwe_addr	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$ SP: GF Direction $SQ \rightarrow SF$ $SQ \rightarrow SF$ $SQ \rightarrow SF$ $SQ \rightarrow SF$ $SQ \rightarrow SF$	PR, Para	Bits 1 Bits 1 Bits 7 7 7 1 1	Descrip Do not Descrip Do not r CaCh Write a Read a Read B Write B			Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr pr_rd_addr pr_re_addr pr_we_addr pr_phase_mux	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$	PR, Para	Bits 1 Bits 1 mete Bits 7 7 1 1 2	Descrij Do not Descrij Do not <i>r cach</i> Write a Read a Read B Write F The p reads a	SU3     SU3     Su3     Su3     Send more texture request if asserted     send more texture request if asserted     e control and auto counter     ption     ddress     ddress     inable     inable for the GPRs     inase mux (arbitrates between inputs,     ind writes)		Formatted: Bullets and Numbering
Name TP_SQ_fet Name SQ_SPx_fe .1.1426 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr pp_rd_addr pp_re_addr pp_phase_mux thannel_mask	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$ SP: GA Direction $SQ \rightarrow SF$ $SQ \rightarrow SF$	PR, Para	Bits 1 Bits 1 Bits 7 7 1 1 2 4	Descrij Do not Descrij Do not <i>r cach</i> Write a Read a Read a Write E The p reads a The ch	SU3     SU3     tion     send more texture request if asserted     e control and auto counter     ption     ddress     inable     inable for the GPRs     nase mux (arbitrates between inputs,     ind writes)     annel mask		Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr upr_rd_addr upr_rd_addr upr_we_addr upr_phase_mux thannel_mask bixel_mask	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$ SP: GI Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$	n xture sta PR, Para PR, Para xx xx xx xx xx xx xx xx xx x	Bits 1 Bits 1 Bits 7 7 7 1 1 2 4 4	Descrig Do not Descrig Do not <i>r CaCh</i> Descri Write a Read a Read a Read a The pri The pri The pri	SU3     SU3     tion     send more texture request if asserted     e control and auto counter     ption     ddress     inable for the GPRs     nase mux (arbitrates between inputs,     und writes)     annel mask     tel mask		<b>Formatted:</b> Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g	tch_stall .1.11_SQ to etch_stall .1.12_SQ to yr_addr ypr_re_addr ppr_re_addr ppr_we_addr ppr_phase_mux hannel_mask bixel_mask	Direction TP→ SQ SP: Te Direction SQ→SP SP: GF Direction SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF	PR, Para	Bits 1 Bits 1 Bits 7 7 1 1 2 4 4 4	Descrip Do not Descrip Do not <i>r CaCh</i> Descri Write a Read a Read a Read a The p reads a The ch The pi The pi The pi	SU3  tion  send more texture request if asserted  tion  send more texture request if asserted  e control and auto counter  ption  ddress  ddress  inable  nable for the GPRs  nase mux (arbitrates between inputs, annel mask  cel mask  cel mask		Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SP2p SQ_SP2p	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr pr_rd_addr pr_re_addr pr_ve_addr pr_we_addr	Direction TP→ SQ SP: Te Direction SQ→SP SP: GP Direction SQ→SP SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF	PR, Para	Bits 1 Bits 1 Bits 7 7 1 1 2 4 4 4 4	Descrip Do not Descrip Do not <i>r CaCh</i> Descri Write a Read a Read a Read a Read a The pin The pin The pin The pin	SU3     SU3     tion     send more texture request if asserted     e control and auto counter     ption     ddress     ddress     inable     inable for the GPRs     nase mux (arbitrates between inputs,     und writes)     annel mask     cel mask     cel mask     cel mask	ALU SRC	Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SP2_p SQ_SP1_p SQ_SP1_p SQ_SP2_p	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr pp_rd_addr pp_re_addr pp_re_addr pp_we_addr pp_we_addr pp_mask bixel_mask bixel_mask bixel_mask	Direction TP→ SQ SP: Te Direction SQ→SP SP: GI Direction SQ→SP SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF	PR, Para	Bits 1 Bits 1 mete Bits 7 7 1 1 2 4 4 4 4 4 4 4	Descrip Do not Descrip Do not r cach Write a Read a Read a Read a Read a The ch The pi The pi The pi The pi The pi	SU3     SU3     tion     send more texture request if asserted     e control and auto counter     ption     ddress     ddress     ddress     inable     inable for the GPRs     inase mux (arbitrates between inputs,     and writes)     annel mask     cel mask     cel mask     cel mask     cel mask     cel mask     cel mask	ALU SRC	Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_y SQ_SPx_y SQ_SPx_y SQ_SPx_y SQ_SPx_y SQ_SPx_y SQ_SPx_y SQ_SPx_c SQ_SPx_c SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr pr_rd_addr pr_re_addr pr_ve_addr pr_phase_mux thannel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$ SP: GI Direction $SQ \rightarrow SP$ $SQ \rightarrow SF$ $SQ \rightarrow SF$	PR, Para	Bits 1 Bits 1 Bits 1 Bits 1 Bits 7 7 7 1 1 2 4 4 4 4 4 4 1 2	Descrij Do not Descrij Do not r cach Write a Read a Read a Read a Read a Read a The ch The pi The pi The pi The pi The pi The pi	SU3 Stion Seend more texture request if asserted Seend more texture request if asserted Seend more texture request if asserted See Control and auto counter Settion Seend more texture request if asserted Settion Seend more texture request Settion	ALU SRC	Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_p SQ_SP2_p SQ_SP3_p SQ_SP3_p SQ_SP3_p SQ_SP3_p SQ_SP3_p SQ_SP3_p SQ_SP3_p	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr upr_rd_addr upr_rd_addr upr_re_addr upr_phase_mux hannel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask	Direction $TP \rightarrow SQ$ SP: Te Direction $SQ \rightarrow SP$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SF$ $SQ \rightarrow SF$	PR, Para	Bits 1 Bits 1 Bits 7 7 1 1 2 4 4 4 4 4 4 1 2	Descrip Do not Descrip Do not <i>r CaCh</i> Descri Write a Read a Read a Write B The pi The pi The pi The pi The pi Write B	SU3     SU3     ition     send more texture request if asserted     send more texture request if asserted     e control and auto counter     ption     ddress     ddress     inable or the GPRs     nase mux (arbitrates between inputs,     ind writes)     annel mask     tel m	ALU SRC	Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_p SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr pp_rd_addr pp_rd_addr pp_re_addr pp_re_addr pp_phase_mux hannel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask	Direction TP→ SQ SP: Te Direction SQ→SP SP: GP Direction SQ→SF SQ	PR, Para	Bits 1 Bits 1 Bits 7 7 1 2 4 4 4 4 4 1 2 1 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1	Descrip Do not Descrip Do not <i>r CaCh</i> Descri Write a Read a Read a Read a Read a The pi The pi The pi The pi The pi The pi Write B Write B	SU3     SU3     ition     send more texture request if asserted     e control and auto counter     ption     ddress     inable	ALU SRC	Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_c SQ_SP1_p SQ_SP1_p SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SPx_g SQ_SPx_g	tch_stall .1.11_SQ to etch_stall .1.12_SQ to wr_addr pp_rd_addr pp_re_addr pp_re_addr pp_we_addr pp_we_addr sixel_mask bixel_mask	Direction TP→ SQ SP: Te Direction SQ→SP SP: GI Direction SQ→SP SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF SQ→SF	PR, Para	Bits 1 Bits 1 Bits 7 7 1 1 1 2 4 4 4 4 4 4 4 4 1 2 2 12?	Descrip Do not Descrip Do not <i>r CaCh</i> Descri Write a Read a Read a Read a Read a The ch The pi The pi The pi The pi The pi The pi The pi The pi The pi A	SU3     SU3     tion     send more texture request if asserted     e control and auto counter     ption     ddress     ddress     ddress     ddress     inable     for the GPRs     nase mux (arbitrates between inputs,     und writes)     annel mask     tel mask	ALU SRC	Formatted: Bullets and Numbering
Name TP_SQ_fet .1.1326 Name SQ_SPx_fe .1.1426 Name SQ_SPx_y SQ_SPx_y SQ_SPx_y SQ_SPx_y SQ_SPx_y SQ_SPx_g SQ_SPx_g SQ_SPx_c SQ_SPx_c SQ_SPx_c SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p SQ_SP2_p	tch_stall .1.11_SQ to etch_stall .1.12_SQ to vr_addr pr_rd_addr pr_re_addr pr_ve_addr pr_ve_addr pr_phase_mux hannel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask bixel_mask	Direction TP $\rightarrow$ SQ SP: Te Direction SQ $\rightarrow$ SP: SQ $\rightarrow$ SP SQ $\rightarrow$ SF SQ $\rightarrow$ SF	n xture sta PR, Para n x x x x x x x x x x x x x	Bits 1 Bits 1 Bits 7 7 1 1 2 4 4 4 4 4 4 4 1 2 12?	Descrip Do not Descrip Do not r CaCh Descri Write a Read Read A Read A Read Read A Read A Rea	SU3     S	ALU SRC	Formatted: Bullets and Numbering

ATI Ex. 2107 IPR2023-00922 Page 43 of 260

	TE EI	DIT DATE	R400 Sequencer Specification	PAGE	
24 September, 20	001 <u>4 Sept</u>	tember, 20154		44 of 48	
5. <u>1.15</u> 26.1.13 SQ to S	Px: Instruc	ctions		▲ Formatter	1: Bullets and Numbering
Name SQ_SPx_instruct_start SQ_SP_instruct	Direction SQ→SPx SQ→SPx	Bits 1 21	Description Instruction start Transferred over 4 cycles 0: SRC A Select 2:0 SRC A Argument Modifier -3:3 SRC A swizzle 11:4 Unused 20:17 : SRC B Select 2:0 SRC B Argument Modifier -3:3 SRC B Select 2:0 SRC B Argument Modifier -3:3 SRC B swizzle 11:4 <u>Unused 20:18</u> - : SRC C Select 2:0 SRC C Select 11:4 Unused 20:12 - : : : : : : : : : : : : :	<u>VectorDst</u>	
SO SPx stall	SQ→SPx	1	Scalar Write Mask 20:17 Stall signal		
SQ_SPx_export_count	SQ→SPx	3	Each set of four pixels or vectors is of eight clocks. This field specifies where that sequence.	xported over the SP is in	
SQ_SPx_export_last	SQ→SPx	1	Asserted on the first shader count of t	e last export	
SQ_SP0_export_pvalid	SQ→SP0	4	Result of pixel kill in the shader pipe, v output for all pixel exports (depth a buffers). 4x4 because 16 pixels are o clock	hich must be nd all color omputed per	
SQ_SP0_export_wvalid	SQ→SP0	2	Specifies whether to write low and/or hig of the 64-bit export data from each of th vectors	h 32-bit word e 16 pixels or	
SQ_SP1_ export_pvalid	SQ→SP1	4	Result of pixel kill in the shader pipe, v output for all pixel exports (depth a buffers). 4x4 because 16 pixels are o clock	hich must be nd all color omputed per	
SQ_SP1_ export_wvalid	SQ→SP1	2	Specifies whether to write low and/or hig of the 64-bit export data from each of th vectors	h 32-bit word e 16 pixels or	
SQ_SP2_ export_pvalid	SQ→SP2	4	Result of pixel kill in the shader pipe, v output for all pixel exports (depth a buffers). 4x4 because 16 pixels are o clock	hich must be nd all color omputed per	
SQ_SP2_ export_wvalid	SQ→SP2	2	Specifies whether to write low and/or hig of the 64-bit export data from each of th vectors	h 32-bit word e 16 pixels or	

ATI Ex. 2107 IPR2023-00922 Page 44 of 260

A 1 1 1 1	ORIGINATE	DATE	EDIT [	DATE	DOCUMENT-REV. NUM.	PAGE	
	24 Septembe	r, 2001	4 Septemb	er, 2	0154 GEN-CXXXXX-REVA 4	5 of 48	
SQ_SP3_	export_pvalid	SQ→S	- <u>Marob 2002</u> P3	4	Result of pixel kill in the shader pipe, which r output for all pixel exports (depth and a buffers). 4x4 because 16 pixels are compu clock	nust be I color ed per	
SQ_SP3_	export_wvalid	SQ→S	P3	2	Specifies whether to write low and/or high 32- of the 64-bit export data from each of the 16 p vectors	it word ixels or	Provide de Dallahan ad Nambarian
5 <u>.1.1626</u>	.1.14_SP to	SQ: Co	nstant add	dres	ss load/ Predicate Set	۹-	Formatted: Bullets and Numbering
Name		Direction	E	Bits	Description	L	
SP0_SQ_c	const_addr	SP0→SQ	3	86	Constant address load / predicate vector load (4 b to the sequencer	its only)	
SP0_SQ_v	alid	SP0→SQ	1		Data valid		
5P1_SQ_0	const_addr	<sup>5</sup> 21→5Q	. 3	00	to the sequencer	its only)	
SP1_SQ_v	/alid	SP1→SQ	1		Data valid		
SP2_SQ_c	const_addr	SP2→SQ	3	86	Constant address load / predicate vector load (4 b to the sequencer	its only)	
SP2_SQ_v	valid	SP2→SQ	. 1		Data valid		
SP3_SQ_c	const_addr	SP3→SQ	. 3	56	Constant address load / predicate vector load (4 b to the sequencer	its only)	
SP3_SQ_v	valid	SP3→SQ	1		Data valid		
→ <u>.1.1726</u> Name SQ_SPx_c	o.1.15_SQ to	) SPX: C( Direction SQ→SPx	onstant bro	oad 3its 28	Cast Description Constant broadcast		
5 <u>.1.1826</u>	. <u>1.16</u> SP0 t	to SQ: K	ill vector l	oad		4-	Formatted: Bullets and Numbering
Name		Direction	. E	Bits	Description		
SP0_SQ_k	dill_vect	SP0→SQ	. 4	ł	Kill vector load		
SP1_SQ_k	dill_vect	SP1→SQ	4	ļ	Kill vector load		
				ł	Kill vector load		
SP3_SQ_K	dii_vect	SP3→SQ	4	1	KIII VEGOGI IGAO	18	
SP3_SQ_k SP3_SQ_k 5.1.1926	(ill_vect (ill_vect ().1.17_SQ to	SP3→SQ CP: RE	BM bus	•			
SP3_SQ_k SP3_SQ_k 5.1.1926 Name	6.1.17_SQ <i>tc</i>	OP: RE	3BM bus	l Bits	Description	• • • • • • • • • • • • • • • • • • •	
SP2_SQ_k SP3_SQ_k 5.1.1926 Name SQ_RBB_	n_vect ill_vect .1.17_SQ tc	$\begin{array}{c} \text{SP3} \rightarrow \text{SQ} \\ \text{SP3} \rightarrow \text{SQ} \\ \hline \text{Direction} \\ \text{SQ} \rightarrow \text{CP} \\ \hline \text{SQ} \rightarrow \text{CP} \\ \hline \end{array}$	3BM bus	Bits	Description Read Strobe		
SP2_SQ_k SP3_SQ_k .1.1926 Name SQ_RBB SQ_RBB SQ_RBB	rs	$OP_{2} \rightarrow SQ$ $OP_{2} \rightarrow SQ$ $OP_{2} \rightarrow SQ$ $OP_{2} \rightarrow SQ$ $SQ \rightarrow CP$ $SQ \rightarrow CP$ $SQ \rightarrow CP$	3BM bus	Bits B2	Description Read Strobe Read Data		
SP2_SQ_k SP3_SQ_k SP3_SQ_k SQ_RBB_ SQ_RBB_ SQ_RBB_ SQ_RBBM SQ_RBBM	rs <u>A_nrttr</u>	$OP_{2} = OQ_{2}$ $OP_{2} = OQ_{2}$ $OP_{2} = OQ_{2}$ $OP_{2} = OQ_{2}$ $OQ_{2} = O$	3BM bus	Bits B2	Description Read Strobe Read Data Optional Real-Time (Optional)		<b>Formatted:</b> Bullets and Numbering
SP2_30_K SP3_S0_K SP3_S0_K SP3_S0_K S0_RBB_ S0_RBB_ S0_RBBM S0_RBBM S0_RBBM	ill_vect dil_vect rs	SP3→SQ SP3→SQ Direction SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP	3BM bus	Bits 32	Description Read Strobe Read Data Optional Real-Time (Optional)		
SP2_SQ_K SP3_SQ_K .1.1926 Name SQ_RBB SQ_RBB SQ_RBB SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM	in_vect dil_vect i.1.17_SQ tc rs rd rd rtr .1.18_CP to	SP3→SQ SP3→SQ Direction SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP	3BM bus 1 3 1 3 3 3 3 3 8 3 8 5 8 5 5 5 5 5 5 5 5 5 5	Bits Bits Bits	Description Read Strobe Read Data Optional Real-Time (Optional) Description		
SP3_SQ_K SP3_SQ_K 5.1.1926 Name SQ_RBB SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM	ini_vect dil_vect i.1.17_SQ to rs SQ to SQ to SQ to SQ to SQ to	SI2→SQ SP3→SQ D CP: RE Direction SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP D SQ: RE Direction CP→SQ	3BM bus 3BM bus 1 3 1 3 3 BM bus 1 1 1 1 1 1 1 1 1 1 1 1 1	Bits Bits Bits	Description Read Strobe Read Data Optional Real-Time (Optional) Description Write Enable		<b>Formatted:</b> Bullets and Numbering
SP3_SQ_K SP3_SQ_K SP3_SQ_K SQ_RBB SQ_RBB SQ_RBBM SQ_RBM SQ_RM SQ_RBM SQ_RBM SQ_RM	III_vect III_17_SQ to rs rd A_nttr III8_CP to	SP3→SQ SP3→SQ DCP: RE Direction SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP	3BM bus 3BM bus 1 3 1 3 3 3 BM bus 1 1 1 1 1 1 1 1 1 1 1 1 1	3its 32 3its 5	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)		<b>Formatted:</b> Bullets and Numbering
SP3_SQ_K SP3_SQ_K SQ_RBB_ SQ_RBB_ SQ_RBBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBBM SQ_RBM SQ_	ill_vect dil_vect .1.17_SQ to rs SQ to SQ to SQ to SQ to	SP3→SQ SP3→SQ DCP: RE Direction SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP	3BM bus 3BM bus 1 3 1 3 3 3 3 3 3 8 5 8 5 8 1 1 1 1 1 1 1 1 1 1 1 1 1	Bits Bits Bits Bits Bits Bits	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data		Formatted: Bullets and Numbering
SP3_SQ_K SP3_SQ_K SQ_RBB_ SQ_RBB_ SQ_RBBM SQ_RBM	ill_vect dil_vect .1.17_SQ tc rs rd    .1.18_CP tc	SP3→SQ SP3→SQ DCP: RE Direction SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ: RE Direction CP→SQ CP→SQ CP→SQ	BBM bus	3its 32 3its 5 32	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data         Byte Enables		
SP2_302_k SP3_SQ_k D.1.1926 Name SQ_RBB_ SQ_RBB SQ_RBBM SQ_RBM SQ	ill_vect ill_vect <u>s.1.17_</u> SQ tc rs <u>rd</u> <u>A_nrtrtr</u> <u>0.1.18_</u> CP to	SP3→SQ SP3→SQ DCP: RE Direction SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ CP→SQ CP→SQ CP→SQ	BBM bus	Bits Bits 5 32	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data         Byte Enables         Read Enable		<b>Formatted:</b> Bullets and Numbering
SP2_SQ_K SP3_SQ_K SP3_SQ_K SQ_RBB SQ_RBB SQ_RBBM SQ_RBM	III_vect III_vect .1.17_SQ tc rs rd A_ntrtr 0.1.18_CP tc	SI2→SQ SP3→SQ DCP: RE Direction SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ CP→SQ CP→SQ CP→SQ CP→SQ	BBM bus	Bits 32 33its 5 32 4	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data         Byte Enables         Read Return Strobe 0         Pand Return Strobe 1		<b>Formatted:</b> Bullets and Numbering
SPSQ_K SP3_SQ_K SQ_RBB_ SQ_RBB_ SQ_RBB_ SQ_RBBM SQ_RBM S	ill_vect ill_vect <u>s.1.17_</u> SQ tc rs rd <u>A_ntrtr</u> <u>s.1.18_CP tc</u>	SI2→SQ SP3→SQ DCP: RE Direction SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ CP→SQ CP→SQ CP→SQ CP→SQ CP→SQ	BBM bus	Bits 32 5 5 32 4 	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data         Byte Enables         Read Return Strobe 0         Read Return Strobe 1         Read Return Optional		
SP3_SQ_K SP3_SQ_K SQ_RBB SQ_RBB SQ_RBB SQ_RBBM SQ_RBM SQ_RB	III_vect iII_vect .1.17_SQ tc rs rd .1.18_CP to .1.18_CP to	$SP3 \rightarrow SQ$ $P: RE$ Direction $SQ \rightarrow CP$ $SQ: RE$ Direction $CP \rightarrow SQ$ $CP \rightarrow S$	BBM bus	Bits Bits 5 5 32 4 32 32 32 32 32	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data         Byte Enables         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Pand Data 0		Formatted: Bullets and Numbering
SP3_SQ_k SP3_SQ_k SP3_SQ_k SQ_RBB SQ_RBB SQ_RBBM SQ_RBM SQ_RBM SM_SQ_RBM SM_SQ_SM SM_SM SM_	iiii_vect         iii_vect         iii_1_vect         iii_vect         rs         rd	$SP2 \rightarrow SQ$ $SP3 \rightarrow SQ$ $P : RE$ Direction $SQ \rightarrow CP$ $SQ : RE$ Direction $CP \rightarrow SQ$ $CP $	BBM bus	Bits 32 Bits 5 5 32 4 32 32 32	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data         Byte Enables         Read Enables         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Soft Besed		Formatted: Bullets and Numbering
SP3_SQ_k SP3_SQ_k SP3_SQ_k SQ_RBB SQ_RBB SQ_RBBM SQ_RBM SQ_RB	III_vect         III_vect         III_rect         III_IT_SQ to         rs         III_rect         III_IT_SQ to         III_IT_CD to         III_IT_CD to         III_IT_CD to         III_IT_CD to         III_IT_CD to         III_IT_CD to         IIII_IT_CD to         IIII_IT_CD to         IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	$SP3 \rightarrow SQ$ $SP3 \rightarrow SQ$ $P : RE$ Direction $SQ \rightarrow CP$ $SQ : RE$ Direction $CP \rightarrow SQ$	BBM bus BBM bus BBM bus BBM bus BBM bus E 1 3BM bus E 1 1 3 3 1 1 3 3 1 1	Bits Bits 55 52 4 32 32 32 32	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data         Byte Enables         Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Soft Reset		Formatted: Bullets and Numbering
SP3_SQ_K SP3_SQ_K SP3_SQ_K SQ_RBB SQ_RBB SQ_RBBM SQ_RBM	A_rtr 0.1.18_CP to 0.1.18_CP to 0.1.18_CP to 0.1.18_CP to	$SP3 \rightarrow SQ$ $SP3 \rightarrow SQ$ $P : RE$ Direction $SQ \rightarrow CP$ $SQ : RE$ Direction $CP \rightarrow SQ$	BBM bus	Bits Bits 5 5 32 4 32 32 32 32 32 32	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data         Byte Enables         Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Soft Reset		Formatted: Bullets and Numbering
SP3_SQ_K SP3_SQ_K SP3_SQ_K SQ_RBB_ SQ_RBB_ SQ_RBBM SQ_RBM SQ_RB	Anter a construction of the second se	$SP3 \rightarrow SQ$ $SP3 \rightarrow SQ$ $P : RE$ Direction $SQ \rightarrow CP$ $SQ \rightarrow SQ$ $CP $	BBM bus	Bits 32 55 32 4 32 32	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data         Byte Enables         Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Soft Reset		Formatted: Bullets and Numbering
SP3_SQ_K SP3_SQ_K SP3_SQ_K SQ_RBB_ SQ_RBB_ SQ_RBBM SQ_	III_vect       III_vect       III_vect       III_r       IIII_R       IIII_R       IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	SP3→SQ SP3→SQ DCP: RE Direction SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ→CP SQ CP→SQ CP→SQ CP→SQ CP→SQ CP→SQ CP→SQ CP→SQ	ATI Confide	Bits 32 55 32 4 32 32 32	Description         Read Strobe         Read Data         Optional         Real-Time (Optional)         Description         Write Enable         Address Upper Extent is TBD (16:2)         Data         Byte Enables         Read Enable         Read Return Strobe 0         Read Return Strobe 1         Read Data 0         Soft Reset		Formatted: Bullets and Numbering

ATI Ex. 2107 IPR2023-00922 Page 45 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
<u>AU</u>	24 September, 2001	4 September, 20154		46 of 48

#### 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 SEO then accesses the IS have for this chader using the load state pointer (
  - 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
- 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
- 9. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
- 10. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
- 11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
   position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
  - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
    - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

Exhibit 2024.docR400\_Sequences.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257180

ATI Ex. 2107 IPR2023-00922 Page 46 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 20154	GEN-CXXXXX-REVA	47 of 48

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 FIFO
  - the Pixel FIFO is wide enough to source four quad's worth of barycentrics per cycle
- 3. SEQ arbitrates between Pixel FIFO and Vertex FIFO when there are no vertices pending OR there is no space left in the register files for vertices, the Pixel FIFO is selected
- 4. SEQ allocates space in the SP register file for all the GPRs used by the program
  - the number of GPRs required by the program is stored in a local state register, which is accessed using the state pointer
  - SEQ will not allow interpolated data to be sent to the shader until space in the register file has been allocated
- 5. SEQ controls the transfer of interpolated data to the SP register file over the RE\_SP interface (which has a bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.
- 6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
  - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
  - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
  - all other 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 2024.docR400\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257181

ATI Ex. 2107 IPR2023-00922 Page 47 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 20154		48 of 48

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?

Exhibit 2024.doc R409\_Sequencer.doc 71269 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257182

ATI Ex. 2107 IPR2023-00922 Page 48 of 260

ALCONTRACTOR AND A DECEMBER OF A	24 September, 2001	<u>4 September, 2015<del>18</del></u>	GEN-CXXXXX-REVA	1 of 50
uthor:	Laurent Lefebvre	March 20024 March		I
sue To:		Copy No:		
		<u> </u>		
			alfiantian	
	K400 3	equencer spe	ecification	
		SQ		
		Version 1. <u>9</u> 8		
	, , , , , ,			
verview: Thi	is is an architectural specific quired capabilities and expe	cation for the R400 Sequen cted uses of the block. It a	cer block (SEQ). It provides an ov lso describes the block interfaces,	verview of the internal sub
blo	ocks, and provides internal st	ate diagrams.		
AUTOMATIC	CALLY UPDATED FIELDS:			
Document L Current Intr	Location: C:\pe anet Search Title: R400	erforce\r400\doc_lib\design\k ) Sequencer Specification	blocks\sq\R400_Sequencer.doc	
	· · · · ·	APPROVALS		
	Name/Dept		Signature/Date	
emarks:				
emarks:				
emarks:		S CONFIDENTIAL I	NFORMATION THAT CC	
Remarks: THIS DO SUBSTA	CUMENT CONTAIN	S CONFIDENTIAL I	NFORMATION THAT CC REST OF ATI TECHNOI	DULD BE LOGIES
emarks: THIS DO SUBSTA	CUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH L	S CONFIDENTIAL I NTAL TO THE INTE JNAUTHORIZED US	NFORMATION THAT CC EREST OF ATI TECHNOL SE OR DISCLOSURE.	OULD BE
Remarks: THIS DO SUBSTA	CUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH U	S CONFIDENTIAL I NTAL TO THE INTE	NFORMATION THAT CC REST OF ATI TECHNOL SE OR DISCLOSURE.	)ULD BE _OGIES
Remarks: THIS DO SUBSTA	CUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH L	S CONFIDENTIAL I NTAL TO THE INTE JNAUTHORIZED US	NFORMATION THAT CC REST OF ATI TECHNOL SE OR DISCLOSURE.	)ULD BE _OGIES
emarks: THIS DO SUBSTA	CUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH L	S CONFIDENTIAL I NTAL TO THE INTE JNAUTHORIZED US	NFORMATION THAT CC REST OF ATI TECHNOL SE OR DISCLOSURE.	)ULD BE _OGIES
emarks: THIS DO SUBSTA	CUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH U 01, ATI Technologies Inc. Al n 2001. The use of this cop vork. The copyright notice	S CONFIDENTIAL I NTAL TO THE INTE JNAUTHORIZED US	NFORMATION THAT CC REST OF ATI TECHNOL SE OR DISCLOSURE.	)ULD BE _OGIES
emarks: THIS DO SUBSTA	OCUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH L 01, ATI Technologies Inc. Al n 2001. The use of this cop vork. The copyright notice roprietary information and tra	S CONFIDENTIAL I NTAL TO THE INTE JNAUTHORIZED US	NFORMATION THAT CC REST OF ATI TECHNOL SE OR DISCLOSURE.	DULD BE OGIES
emarks: THIS DO SUBSTA SUBSTA	OCUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH U 1, ATI Technologies Inc. Al n 2001. The use of this cop vork. The copyright notice roprietary information and tra any form or by any means with	S CONFIDENTIAL I ENTAL TO THE INTE JNAUTHORIZED US	NFORMATION THAT CC REST OF ATI TECHNOL SE OR DISCLOSURE.	)ULD BE _OGIES
Copyright 200 ork created i npublished w onfidential, pr ansmitted in a	CUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH U 01, ATI Technologies Inc. Al n 2001. The use of this cop vork. The copyright notice roprietary information and tra any form or by any means wit	S CONFIDENTIAL I SNTAL TO THE INTE JNAUTHORIZED US	NFORMATION THAT CC REST OF ATI TECHNOL SE OR DISCLOSURE.	DULD BE OGIES
2.emarks: THIS DO SUBSTA	CUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH U	S CONFIDENTIAL I S CONFIDENTIAL I I rights reserved. The mate byright notice is intended to is not an admission that ade secrets of ATI. No part thout the prior written permis	NFORMATION THAT CC REST OF ATI TECHNOL SE OR DISCLOSURE.	DULD BE OGIES
emarks: THIS DO SUBSTA	CUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH U	S CONFIDENTIAL I SNTAL TO THE INTE JNAUTHORIZED US	NFORMATION THAT CC REST OF ATI TECHNOL SE OR DISCLOSURE. rial in this document constitutes an provide notice that ATI owns a co publication has occurred. This of this document may be used, re- ssion of ATI Technologies Inc."	DULD BE OGIES
Remarks: THIS DO SUBSTA Copyright 200 /ork created i npublished w onfidential, pr ansmitted in a	CUMENT CONTAIN ANTIALLY DETRIME INC. THROUGH U	S CONFIDENTIAL I S CONFIDENTIAL I I rights reserved. The mate byright notice is intended to is not an admission that ade secrets of ATI. No part thout the prior written permis	NFORMATION THAT CC REST OF ATI TECHNOL SE OR DISCLOSURE.	DULD BE OGIES

ATI Ex. 2107 IPR2023-00922 Page 49 of 260



# Table Of Contents

1.	OVERVIEW	<u> 86</u>
1.1	Top Level Block Diagram	108
1.2	Data Flow graph (SP)	1210
1.3	Control Graph	1311
2.	INTERPOLATED DATA BUS	1344
3.	INSTRUCTION STORE	1614
4.	SEQUENCER INSTRUCTIONS	1846
5.	CONSTANT STORES	1846
<u>5.1</u>	Memory organizations	<u>1846</u>
<u>5.2</u>	Management of the Control Flow Constants	<u>1846</u>
<u>5.3</u>	Management of the re-mapping tables	<u>18<del>16</del></u>
<u>5.3</u>	3.1 R400 Constant management	<u>1816</u>
5.2	3.2 Proposal for R400LE constant management	1917
<u>5.3</u>	3.3 Dirty bits	<u>2149</u>
5.3	3.4 Free List Block	<u>2149</u>
<u>5.3</u>	3.5 De-allocate Block	<u>22<del>2</del>0</u>
5.3	3.6 Operation of Incremental model	<u>2220</u>
<u>5.4</u>	Constant Store Indexing	<u>2220</u>
<u>5.5</u>	Real Time Commands	<u>2321</u>
5.6	Constant Waterfalling	<u>2321</u>
<u>6.</u>	LOOPING AND BRANCHES	<u>2422</u>
<u>6.1</u>	The controlling state	2422
6.2	The Control Flow Program	2422
6.3	Data dependant predicate instructions	2624
<u>6.4</u>	HW Detection of PV,PS	2725
<u>6.5</u>	Register file indexing	2725
6.6	Predicated Instruction support for Texture clauses	2725
<u>6.7</u>	Debugging the Shaders	28 <del>25</del>
<u>6.</u> 2	7.1 Method 1: Debugging registers	<u>28<del>25</del></u>
<u>6.</u> 2	7.2 Method 2: Exporting the values in the GPRs (12)	28 <del>26</del>
7.	PIXEL KILL MASK	<u>2826</u>
8.	MULTIPASS VERTEX SHADERS (HOS)	2926
9.	REGISTER FILE ALLOCATION	2926
10.	FETCH ARBITRATION	3028
11.	ALU ARBITRATION	30 <u>28</u>
12.	HANDLING STALLS	3129
13.	CONTENT OF THE RESERVATION STATION FIFOS	3129
14.	THE OUTPUT FILE	3129
15.	IJ FORMAT	3129
15.1	Interpolation of constant attributes	3230
16.	STAGING REGISTERS	3230
17.	THE PARAMETER CACHE	3432

Exhibit 2025. doc R400\_Sequencer.dec 71630 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

ORIGINATE DATE			EDITUATE	DOCUMENT-REV. NUM.	PAGE					
24 September, 2001			4 September, 201548	GEN-CXXXXX-REVA	3 of 50					
18.	. VERTEX POSITION EXPORTING									
19.	EXPORTING ARBITRATION									
<u>20.</u>	EXPO	RT TYPES			<u>3532</u>					
$\frac{20.1}{20.2}$	Verte	ex Shading			<u>3532</u>					
20.2	SPEC		MODES		<u> 3099</u>					
21.1	Real	time commands		***************************************	3633					
21.2	Sprit	tes/ XY screen coordina	ates/ FB information		3633					
21.3	Auto	generated counters			3634					
<u>21</u>	.3.1	Vertex shaders			3634					
<u>21</u>	.3.2	Pixel shaders			3634					
<u>22.</u>	STAT	E MANAGEMENT			3734					
<u>22.1</u>	Para	meter cache synchroni	ization		3734					
23.	XY AD	DRESS IMPORTS	****							
$\frac{23.1}{24}$	PECIS				3/ <del>33</del>					
24 1	Cont	trol	***************************************	******	3835					
24.2	Cont	text								
25.	DEBU	G REGISTERS	******		3936					
<u>25.1</u>	Cont	text		<u></u>	3936					
<u>25.2</u>	Coni				3936					
<u>26.</u>	INIE	RFACES								
<u>26.1</u>	Ext	ernal Interfaces	*****		3936					
<u>26</u>	.1.1	SC to SQ : IJ Contr	ol bus		3937					
<u>26</u>	.1.2	SQ to SP: Interpolato	r bus		40 <del>37</del>					
<u>26</u>	.1.3	SQ to SX: Interpolato	r bus		4037					
<u>26</u>	.1.4	SQ to SP: Staging Re	gister Data		4138					
<u>26</u>	.1.5	PA to SQ : Vertex inte	erface		4138					
<u>26</u>	.1.6	SQ to CP: State repo	rt		4441					
<u>26</u>	.1.7	SQ to SX: Control bus	S		4441					
<u>26</u>	.1.8	SX to SQ : Output file	control		4441					
<u>26</u>	.1.9	SQ to TP: Control bus	<u> </u>		4441					
<u>26</u>	.1.10	TP to SQ: Texture sta			4542					
<u>26</u>	.1.11	SQ to SP: Texture sta	all		4542					
<u>26</u>	.1.12	SQ to SP: GPR and a	uto counter		4542					
<u>26</u>	.1.13	SQ to SPx: Instruction	าร		4643					
<u>26</u>	.1.14	SP to SQ: Constant a	ddress load/ Predicate	Set	4744					
26.1.15 SQ to SPx: constant broadcast										
<u>26</u>	.1.16	SP0 to SQ: Kill vector	· load		4744					
<u>26</u>	.1.17	SQ to CP: RBBM bus			4744					
<u>26</u>	.1.18	CP to SQ: RBBM bus								
27.	EXAM	PLES OF PROGRAM	EXECUTIONS		4845					

Exhibit 2025 doc R400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

		EDITIDATE	R400 Sequencer Specification	PAGE					
	24 September, 2001	4 September, 201518		4 of 50					
27.1.1	Sequencer Control of	of a Vector of Vertices							
27.1.2	Sequencer Control of	of a Vector of Pixels							
27.1.3	Notes			504Z					
28. OPFN	ISSUES			5047					
1OVER	VIEW	*****		6					
	evel Block Diagram								
1.2 Data F	-low graph (SP)								
1.3 Contro	ol Graph	*****							
2. INTER	<b>POLATED DATA BL</b>	JS	******						
3INSTR	RUCTION STORE								
4.—SEQU	ENCER INSTRUCTION	əns	*** *** ****						
5. CONS	TANT STORES	*****							
5.1 Memo	ry organizations								
5.2 Manag	gement of the re-map	ping tables							
5.2.1	Dirty bits								
5 <u>.2.2</u>	=ree List Block	****							
523-1	De-allocate Block								
524	Operation of Incremen	tal model		10					
5.3 Const	ant Store Indexing			10					
5.4 Real T	ime Commands	********		20					
5.5 Const	ant Waterfalling			20					
6LOOP	ING AND BRANCHE	S		21					
6 1 The co	ontrolling state	• 1000° DON BROKE DOGEN		21					
62 The C	ontrol Flow Program			21					
6.3 Data c	lependant predicate i	nstructions		23					
6.4 HW-D	etection of PV.PS								
6.5 Regist	ter file indexing	*****							
6.6 Predic	ated Instruction supp	ort for Texture clauses	8						
6.7 Debug	ging the Shaders								
6.7.1-1	Method 1: Debugging	registers							
6.7.2-1	Vethod 2: Exporting th	ne values in the GPRs	; (12)						
7. PIXEL	. KILL MASK								
8. MULT	IPASS VERTEX SHA	DERS (HOS)							
9. REGI	STER FILE ALLOCA	FION							
10. FETC	H ARBITRATION	******							
11. ALU-A	RBITRATION								
12. HAND	LING STALLS		*****						
13CONT	ENT OF THE RESER	VATION STATION F	IFOS						
14. THE C	OUTPUT FILE	******							
15. IJ FOI	RMAT								
15.1 Inter	polation of constant a	ttributes							
16. STAG	ING REGISTERS	******	*****						
17. THE PARAMETER CACHE									
18. VERTEX POSITION EXPORTING									
19. EXPO	RTING ARBITRATIC	N	****						
20. EXPO	20. EXPORT TYPES								

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

V A	r I	ORIGINATE DATE	EDITUATE	DOCUMENT-REV. NUM.	PAGE
		24 September, 2001	4 September, 201548	GEN-CXXXXX-REVA	5 of 50
20.1	Verte	ex Shading		*****	
20.2	Pixel	Shading			
21. 8	SPECI	AL INTERPOLATION	MODES	*****	
21.1	<del>- Keal</del> Sprit	-ume-commands	ntos/ER information		<del>32</del> 22
21.2	Auto	aenerated counters	atoo/ I D IIIIOIIIIatioII		33
21.3	3_1	Vartav chadare			33
21.2	3-2-	Pixel shaders			33
22.	STATE	E MANAGEMENT		*****	
22.1	-Para	meter cache synchroni	ization		
<del>23.                                     </del>	XY AD	DRESS IMPORTS	******	****	
23.1	Verte	ex indexes imports			
24. F	REGIS	STERS	*************		
24.1		-101 ovt			
25.	DEBU	G-REGISTERS			
25.1	Cont	ext			
26	INTE	RFACES	******	*****	
26.1	Exte	ernal Interfaces			
26.1	1.1	SC to SQ : IJ Contr	ol bus		
<del>26.</del> 1	1.2—	SQ to SP: Interpolato	r bus		
<del>26.</del> 1	1.3	SQ to SP: Parameter	Cache Read control bu	IS	
26.1	1.4—	SQ to SX: Parameter	Cache Mux control Bus	8	
26.1	1.5	SQ to SP: Staging Re	gister Data	****	
26.1	1.6	PA to SQ : Vertex inte	erface		
26.1	1.7	SQ to CP: State report	r <del>t</del>		41
26.1	1.8	SQ to SX: Control but	s <del></del>		41
26.1	1.9	SX to SQ : Output file	-control		41
26.1	1.10-	SQ to TP: Control bus	S <del></del>		
26.1	1.11-	TP to SQ: Texture sta	HI	*****	
26.1	1.12-	SQ to SP: Texture sta	all		
26.1	1.13-	SQ to SP: GPR, Para	meter cache control an	d-auto-counter	
<del>26.</del> 1	1.14-	SQ to SPx: Instruction	۱۶ <del></del>		43
26.1	1.15_	SP to SQ: Constant a	ddress load		
26.1	1.16-	SQ to SPx: constant t	proadcast		
26.1	1.17_	SP0 to SQ: Kill vector	load		
26.1	1.18-	SQ to CP: RBBM bus			44
26.1	1.19-	CP to SQ: RBBM bus			
<del>27. E</del>	EXAM	PLES OF PROGRAM	EXECUTIONS	****	44
27.1	1.1	Sequencer Control of	a Vector of Vertices		44
27.1	1.2	Sequencer Control of	a Vector of Pixels		

Exhibit 2025 doc R400\_Sequences.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257187

ATI Ex. 2107 IPR2023-00922 Page 53 of 260

R400 Sequencer Specification PAGE
1518 6 of 50

Exhibit 2025 doc R400\_Sequencer.dec 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257188

ATI Ex. 2107 IPR2023-00922 Page 54 of 260

	March 20024 March		
24 September, 2001	4 September, 2015 <del>18</del>		
ORIGINATE DATE	EDITUATE		

### Revision Changes:

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

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

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

Rev 0.8 (Laurent Lefebvre) Date : October 8, 2001 Rev 0.9 (Laurent Lefebvre) Date : October 17, 2001

Rev 1.0 (Laurent Lefebvre) Date : October 19, 2001 Rev 1.1 (Laurent Lefebvre) Date : October 26, 2001

Rev 1.2 (Laurent Lefebvre) Date : November 16, 2001 Rev 1.3 (Laurent Lefebvre) Date : November 26, 2001 Rev 1.4 (Laurent Lefebvre) Date : December 6, 2001

Rev 1.5 (Laurent Lefebvre) Date : December 11, 2001

Rev 1.6 (Laurent Lefebvre) Date : January 7, 2002

Rev 1.7 (Laurent Lefebvre) Date : February 4, 2002 Rev 1.8 (Laurent Lefebvre) Date : March 4, 2002

Rev 1.9 (Laurent Lefebvre) Date :

#### First draft.

Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section. Reviewed the Sequencer spec after the meeting on August 3, 2001.

Added the dynamic allocation method for register file and an example (written in part by Vic) of the flow of pixels/vertices in the sequencer. Added timing diagrams (Vic)

Changed the spec to reflect the new R400 architecture. Added interfaces.

Added constant store management, instruction store management, control flow management and data dependant predication.

Changed the control flow method to be more flexible. Also updated the external interfaces.

Incorporated changes made in the 10/18/01 control flow meeting. Added a NOP instruction, removed the conditional\_execute\_or\_jump. Added debug registers.

Refined interfaces to RB. Added state registers.

Added SEQ $\rightarrow$ SP0 interfaces. Changed delta precision. Changed VGT $\rightarrow$ SP0 interface. Debug Methods added.

Interfaces greatly refined. Cleaned up the spec.

Added the different interpolation modes.

Added the auto incrementing counters. Changed the VGT→SQ interface. Added content on constant management. Updated GPRs.

Removed from the spec all interfaces that weren't directly tied to the SQ. Added explanations on constant management. Added  $PA \rightarrow SQ$  synchronization fields and explanation.

Added more details on the staging register. Added detail about the parameter caches. Changed the call instruction to a Conditionnal\_call instruction. Added details on constant management and updated the diagram.

Added Real Time parameter control in the SX interface. Updated the control flow section.

New interfaces to the SX block. Added the end of clause modifier, removed the end of clause instructions.

Rearangement of the CF instruction bits in order to ensure byte alignement

Exhibit 2025 docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*



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

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

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

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

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257190

ATI Ex. 2107 IPR2023-00922 Page 56 of 260





ATI Ex. 2107 IPR2023-00922 Page 57 of 260

ΛŇ	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
<u>400</u>	24 September, 2001	4 September, 201518		10 of 50

# 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.

Exhibit 2025 docR400\_Sequences.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

VAN	ORIGINATE DATE	EDITDATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 2015 <del>18</del>	GEN-CXXXXX-REVA	11 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 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 arbitres always favor the higher number state machines, preventing a bunch of half finished jobs from clogging up the register files.

Exhibit 2025.docR400\_Sequencer.dec 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257193

ATI Ex. 2107 IPR2023-00922 Page 59 of 260





Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257194

ATI Ex. 2107 IPR2023-00922 Page 60 of 260

URIGINATE DATE	EDITUATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201548	GEN-CXXXXX-REVA	13 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

Figure 4: Sequencer Control interfaces

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

# 2. Interpolated data bus

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

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257195

ATI Ex. 2107 IPR2023-00922 Page 61 of 260



Figure 5: Interpolation buffers

Exhibit 2025 doc R400\_Sequences.doc 71630 Bytes\*\*\* @ ATI Confidential. Reference Copyright Notice on Cover Page @ \*\*\*

AMD1044\_0257196

ATI Ex. 2107 IPR2023-00922 Page 62 of 260

			T23					5148 <	52- 55	59 -<	S ♀ <	
			T22					32- 35	33 3 <mark>9</mark> <	> 4₀ 4₀ <	44-(	X
			T21					76- 19	20- 23	<pre></pre> <pre></pre> <pre>24-</pre> <pre>27</pre>	28- <	5
			T20					V 0-3	> 4-7	⇒ ° £	15 - 15 -	
			T19							Ш	ш	
			T18								8	
	Щ	50	T17			≿ ଘ	×Ξ		8	5	3	
	PAG	15 of	T16			ЦО	ш				BO	$\sim$
Ł			T15			о Ш	Ш	5	D2			Ω
ERL	IUM.	٨A	T14	żБ	Za		Хß	C	Q 4	C5		
<b>AAT</b>	SEV. P	(X-RE	T13	5	D2		0	B1				
ERN	ENT-F	XXX	T12	Б	D2		8	AO	A1	A2		
)RD	DCUM	GEN-O	T11		≿ 2	C5 C5				С	Ē	
VE C	ă		T10		04 0	C5	L L				8	
CTI		201518 March	T9		0 4	C5			8	5	ß	
OTE	Ш		Т8	≿ଞ	≳ S	≿ວ	ζS				BO	- Constanting
PR	IT DA	mber,	17	IJ	8	ۍ	ß	51	D2			à
		Septe	TG	ß	8	5	C3	ü	5 2	C5		
		415	T5	ΣΈ			∑ß	B1				
	ATE	2001	T4	Ð			BO	AO	A A	42		
	VTE D	nber,	T3	Ξ			BO	51 48 X	55 <sup>2</sup> 2 ≺	20 20 X	≳ ¦ %	
	SIGIN <sup>A</sup>	Septer	T2	×δ	≿ ₽	× 2		X 32- 35	% % %	×4 &	× +4 +4 +7	$\succ$
	Ю	24 (	Ţ	AO	A1	A2		16- 19-	23 ° X	XY 24- 27	31 <sup>2</sup> 8 ×	$\times$
	00	5	TO	AO	A1	A2		0-3 X	¥ 4-7	≍ ∾ ≿	15 <del>1</del> 2 ×	
	K	2		g o	<del>с</del> –	⊳ SP	ო წ	O SP	- SP	2 SP	ი ი	

Exhibit 2025 too: Page Control of Control of

Figure 6: Interpolation timing diagram

AMD1044\_0257197

ATI Ex. 2107 IPR2023-00922 Page 63 of 260

	ORIGINATE DATE	EDITDATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 201548		16 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 2025 docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257198

ATI Ex. 2107 IPR2023-00922 Page 64 of 260



Exhibit 2025 doc Paramener-doc 71630 Bytes\*\*\* 

ATI Confidential. Reference Copyright Notice on Cover Page 

\*\*\*



# 4. <u>Sequencer Instructions</u>

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 <u>428x192\_320x96</u> bits (<u>128 texture states</u> for regular mode, <u>32 states for RT</u>). 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 <u>32</u> lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top <u>320</u> bits. It thus takes 6 clocks to write the texture state. Real time requires <u>32</u> lines in the physical memory (this is physically register mapped).

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

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

# 5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF\_WR\_BASE). On the read side, one level of indirection is used. A register (SQ\_CONTEXT\_MISC.CF\_RD\_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1 )% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

# 5.3 Management of the re-mapping tables

## 5.3.1 *R400* Constant management

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

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

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space

Exhibit 2025 doc R400\_Sequencer.dec 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

URIGINATE DATE	EDITUATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201548	GEN-CXXXXX-REVA	19 of 50

is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32\*2+32 = 96 entries and above.

#### 5.3.2 Proposal for R400LE constant management

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ\_IDLE and PA\_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in <u>Figure 9: De-allocation mechanismFigure 9: De-allocation mechanism</u>). Note that in the case a state is cleared a value of 0 is written to the corresponding de-allocation counter location so that when the SQ is going to report a state change, nothing will be de-allocated upon the first report.

The second path sets all context dirty bits that were used in the current state to 1 (thus allowing the new state to reuse these physical addresses if needed).

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257201

ATI Ex. 2107 IPR2023-00922 Page 67 of 260





Exhibit 2025 docR400\_Sequencer.doc 71630 Bytes\*\*\* @ ATI Confidential. Reference Copyright Notice on Cover Page @ \*\*\*

AMD1044\_0257202

ATI Ex. 2107 IPR2023-00922 Page 68 of 260



Figure 9: De-allocation mechanism for R400LE

## 5.3.3 Dirty bits

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

### 5.3.4 Free List Block

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

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

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

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

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

Exhibit 2025 doc R400\_Sequences.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

	ORIGINATE DATE	EDITUATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 201518 March, 20024 March		22 of 50

### 5.3.5 De-allocate Block

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

## 5.3.6 Operation of Incremental model

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

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

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

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

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

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

# 5.4 Constant Store Indexing

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

Exhibit 2025 doc R400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257204

ATI Ex. 2107 IPR2023-00922 Page 70 of 260

	EDITDATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201548	GEN-CXXXXX-REVA	23 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.

# 5.5 Real Time Commands

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

# 5.6 Constant Waterfalling

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



Figure 10: The instruction store

Exhibit 2025.docR400\_Sequences.doc 71630 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257205

ATI Ex. 2107 IPR2023-00922 Page 71 of 260



## 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 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.

The control program has nine basic instructions:

Execute Conditional\_execute Conditional\_Execute\_Predicates Conditional\_jump Conditionnal\_Call Return Loop\_start Loop\_end NOP

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

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

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

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

Conditionnal\_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return instruction, the IP is popped from the stack.

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* @ ATI Confidential. Reference Copyright Notice on Cover Page @ \*\*\*
ORIGINATE DATE	EDITUATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201518	GEN-CXXXXX-REVA	25 of 50

Conditional\_execute\_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition. Conditional\_jumps jumps to an address if the condition is met. NOP is a regular NOP

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

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

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

A value of 1 in the Addressing means that the address specified in the Exec Address field (or in the jump address field) is an ABSOLUTE address. If the addressing field is cleared (should be the default) then the address is relative to the base of the current shader program.

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

	Execute									
47	46 42	41	40 24	23 12	11 0					
Addressing	00001	Last	RESERVED	Instruction	Exec Address					
				count						

Execute up to 4k instructions at the specified address in the instruction memory. If Last is set, this is the last group of instructions of the clause.

	NOP						
47	46 42	41	40 0				
Addressing	00010	Last	RESERVED				

This is a regular NOP. If Last is set, this is the last instruction of the clause.

Conditional_Execute									
47	46 42	41	<u>40</u>	40- <u>39</u> 33 <u>32</u>	<u>3231</u>	<del>3</del> 1– <u>30</u> 24	23 12	11 0	
Addressing	00011	Last	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address	

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions). If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Conditional_Execute_Predicates										
47 46 42 41 40 40 34 33 3231 31 30 24 23 12 11 0								11 0		
			<u>3534</u>	<u>3332</u>						
Addressing	00100	Last	RESERVED	Predicate	Condition	RESERVED	Instruction	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. If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

		Loop_Start		
47	46 42	41 17	16 12	11 0

Exhibit 2025. doc R400\_Sequencer.doc 71630 Bytes\*\*\* 
California ATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\*

VAN		ORIGII	NATEDATE	EDITUATE	R400 Sequencer 8	Specification		PAGE
	24 Sept	ember, 2001	4 September, 201518 March, 20024 March				26 of 50	
	Addressing	00101		RESERVED		loop ID	J	ump address

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

		Loop_End		
47	46 42	41 17	16 12	11 0
	00110	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 35 <u>34</u>	<u>34-33</u>	312	<del>31</del> <u>30</u> 12	11 0					
			<u>3332</u>								
	00111	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							
	01000	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	41 401	40-39	3231	3130	30-29 12	11 0		
			3332						
	01001	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.

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

#### 6.3 Data dependant predicate instructions

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

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

For the scalar operations only we will also support the two following instructions: PRED\_SETE0\_# – SETE0

Exhibit 2025 doc R400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257208

ATI Ex. 2107 IPR2023-00922 Page 74 of 260

ORIGINATE DATE	EDITUATE	DOCOMENT-REV. NOW.	FAGE
24 September, 2001	4 September, 2015 <del>18</del> March, 20024 March	GEN-CXXXXX-REVA	27 of 50
DDED OFTEN # OFTE	4		

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.

We loop until loop\_iterator = loop\_count. Loop\_step is a signed value [-128...127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the "indexing logic" so that it knows when the provided index is out of range and thus can make the necessary arrangements.

## 6.6 Predicated Instruction support for Texture clauses

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

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257209

ATI Ex. 2107 IPR2023-00922 Page 75 of 260



March 20024 March

## 6.7 Debugging the Shaders

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

## 6.7.1 Method 1: Debugging registers

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

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

The sequencer will detect the following groups of errors:

- count overflow
- constant indexing overflow
- register indexing overflow

Compiler recognizable errors:

- jump errors

relative jump address > size of the control flow program

- call stack

call with stack full return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB\_PROB\_BREAK register is set.

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

{ISSUE : Interrupt to the driver or not?}

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

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

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

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

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

# 7. Pixel Kill Mask

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

MASK\_SETE MASK\_SETNE MASK\_SETGT

Exhibit 2025. doc R400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

	ORIGINATE DATE	EDITUATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201548	GEN-CXXXXX-REVA	29 of 50
MACH	OFTOTE			

MASK\_SETGTE

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

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

#### 9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX\_REG\_SIZE for vertices and PIXEL\_REG\_SIZE for pixels.

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257211

ATI Ex. 2107 IPR2023-00922 Page 77 of 260



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 2025 docR400\_Sequences.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

	EDITUATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201548	GEN-CXXXXX-REVA	31 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).

# $\Delta 01I = I(1) - I(0)$

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

P0	P1
P2	P3

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

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

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

Exhibit 2025 doc R400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

<b>V</b> A Ň	ORIGINATE DATE	EDII DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 201518 March, 20024 March		32 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) or (J = 0)) and
       ((J = 0) \text{ or } (1-I-J = 0)) and
       ((1-J-I=0) \text{ or } (I=0))) {
           if(| != 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 2025. doc R400\_Sequencer.doc 71630 Bytes\*\*\* @ ATI Confidential. Reference Copyright Notice on Cover Page @ \*\*\*

VAR	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201518	GEN-CXXXXX-REVA	33 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 <u>Figure 12Figure 12</u>. 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 <u>Figure 11Figure 11Figure 11</u>.

Basis for 8-deep Latch Memory (from	า R300)		
8x24-bit	11631 <sub>µ</sub>	u <sup>2</sup>	$60.57813\mu^2\text{per bit}$
Area of 96x8-deep Latch Memory Area of 24-bit Fix-to-float Converter	46524 <sub>µ</sub> 4712 <sub>µ</sub>	ι <sup>2</sup> ι <sup>2</sup> per conve	erter
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			$758520  \mu^2$

Figure 11: Area Estimate for VGT to Shader Interface

Exhibit 2025 doc R400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257215

ATI Ex. 2107 IPR2023-00922 Page 81 of 260



Figure 12:VGT to Shader Interface

#### 17. The parameter cache

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

MEMORY NUMBER	ADDRESS
4 bits	7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current\_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current\_Location by VS\_EXPORT\_COUNT\_7 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS\_EXPORT\_COUNT\_7 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 0010000000 and so on up to 11110000000. Then the next position received (the  $17^{th}$ ) is going to have the address 0000001000, the  $18^{th}$  00010001000, the  $19^{th}$  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  $2*VS_EXPORT_COUNT_7$  to Current\_Location and reset the memory count to 0 before the next vector begins).

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257216

ATI Ex. 2107 IPR2023-00922 Page 82 of 260

ORIGINATE DATE	EDITIDATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201518 March, 20024 March	GEN-CXXXXX-REVA	35 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?}

#### 20. Export Types

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

## 20.1 Vertex Shading

- 0:15 16 parameter cache
- 16:31 Empty (Reserved?)
- 32:43 12 vertex exports to the frame buffer and index
- 44:47 Empty
- 48:59 12 debug export (interpret as normal vertex export)
- 60 export addressing mode
- 61 Empty
- 62 position
- 63 sprite size export that goes with position export (point\_h,point\_w,edgeflag,misc)

#### 20.2 Pixel Shading

- 0 Color for buffer 0 (primary)
- 1 Color for buffer 1
- 2 Color for buffer 2
- 3 Color for buffer 3
- 4:7 Empty
- 8 Buffer 0 Color/Fog (primary)
- 9 Buffer 1 Color/Fog
- 10 Buffer 2 Color/Fog
- 11 Buffer 3 Color/Fog
- 12:15 Empty
- 16:31 Empty (Reserved?)
- 32:43 12 exports for multipass pixel shaders.
- 44:47 Empty
- 48:59 12 debug exports (interpret as normal pixel export)
- 60 export addressing mode
- 61:62 Empty
- 63 Z for primary buffer (Z exported to 'alpha' component)

Exhibit 2025.doc R400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

URIGINATE DATE	EDITUATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201548		36 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, no 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 2025. doc R400\_Sequences.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*



Figure 13: GPR input mux Control

#### 22. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

#### 22.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC\_SQ\_new\_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

#### 23. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to interpolate the IJ data or pass the XY data thru a Fix $\rightarrow$ 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.

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* 
ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*



## 24. Registers

24.1 Control	
REG_DYNAMIC REG_SIZE_PIX	Dynamic allocation (pixel/vertex) of the register file on or off. 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 INST_STORE_ALLOC	policy of the arbitration between vertexes and pixels interleaved, separate
INST_BASE_VTX	start point for the vertex instruction store (RT always ends at vertex_base and Begins at 0)
INST BASE PIX	start point for the pixel shader instruction store
ONE_THREAD	debug state register. Only allows one program at a time into the GPRs
ONE_ALU	debug state register. Only allows one ALU program at a time to be executed (instead of 2)
INSTRUCTION	This is where the CP puts the base address of the instruction writes and type (auto- incremented on reads/writes) Register mapped
CONSTANTS	512*4 ALU constants + 32*6 Texture state 32 bits registers (logically mapped)
CONSTANTS RT	256*4 ALU constants + 32*6 texture states? (physically mapped)
CONSTANT_EO_RT	This is the size of the space reserved for real time in the constant store (from 0 to CONSTANT EO RT). The re-mapping table operates on the rest of the memory
TSTATE_EO_RT	This is the size of the space reserved for real time in the fetch state store (from 0 to TSTATE EO RT). The re-mapping table operates on the rest of the memory
EXPORT_LATE	Controls whether or not we are exporting position from clause 3. If set, position exports occur at clause 7.

#### 24.2 Context

VS_FETCH_{07} VS_ALU_{07} PS_FETCH_{07} PS_ALU_{07} PS_BASE VS_BASE VS_CF_SIZE PS_CF_SIZE PS_SIZE VS_SIZE PS_NUM_REG VS_NUM_REG	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 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 base pointer for the pixel shader in the instruction store base pointer for the vertex shader in the instruction store size of the vertex shader (# of instructions in control program/2) size of the pixel shader (m of instructions) size of the pixel shader (cntl+instructions) size of the vertex shader (cntl+instructions) number of GPRs to allocate for pixel shader programs
PARAM_SHADE	One 16 bit register specifying which parameters are to be gouraud shaded (0 = flat, 1 = gouraud)
PROVO_VERT PARAM_WRAP	0 : vertex 0, 1: vertex 1, 2: vertex 2, 3: Last vertex of the primitive 64 bits: for which parameters (and channels (xyzw)) do we do the cyl wrapping (0=linear, 1=cylindrical)
PS_EXPORT_MODE	0xxxx : Normal mode 1xxxx : Multipass mode If normal, bbbz where bbb is how many colors (0-4) and z is export z or not If multipass 1-12 exports for color.
VS_EXPORT_MASK VS_EXPORT_MODE VS_EXPORT	which of the last 6 ALU clauses is exporting (multipass only) 0: position (1 vector), 1: position (2 vectors), 3:multipass
_COUNT_{06}	Six 4 bit counters representing the # of interpolated parameters exported in clause 7 (located in VS_EXPORT_COUNT_6) OR # of exported vectors to memory per clause in multipass mode (per clause)
PARAM_GEN_10	Do we overwrite or not the parameter 0 with XY data and generated T and S values
Exhibit 2025.docR400_Sequencer.doc	71630 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***

		IE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 20	001 <u>4</u>	September, 2015 <del>18</del> areb, 20024 March	GEN-CXXXXX-REVA	39 of 50	
GEN INDEX Auto generates an address from 0 to XX. Puts the results into R0-1 for pixel s						
	an	id R2 for ver	tex shaders			
CONST_E	BASE_VTX (9 bits)Lc	gical Base a	address for the constar	nts of the Vertex shader		
CONSTE	BASE PIX (9 bits) Lo	gical Base a	address for the constant	nts of the Pixel shader		
CONST	SIZE PIX (8 bits) Si	ze of the log	ical constant store for	pixel shaders		
CONST	SIZE VTX (8 bits) Si	ze of the log	ical constant store for	vertex shaders		
INST_PRED_OPTIMIZE Turns on the predicate bit optimization (if of, conditional_execute_predicates is always executed)						
CF BOOL	EANS 25	i6 boolean b	oits			
CF LOOP COUNT 32x8 bit counters (number of times we traverse the loop)						
CF_LOOP_START 32x8 bit counters (init value used in index computation)						
CF LOOP STEP 32x8 bit counters (step value used in index computation)						
			· ·	• • •		

#### 25. DEBUG Registers

#### 25.1 Context

DB_PROB_ADDR DB_PROB_COUNT	instruction address where the first problem occurred
DB_PROB_BREAK	break the clause if an error is found.
DB INST COUNT	instruction counter for debug method 2
DB_BREAK_ADDR	break address for method number 2
DB_CLAUSE	
_MODE_ALU_{07}	clause mode for debug method 2 (0: normal, 1: addr, 2: kill)
DB_CLAUSE	
_MODE_FETCH_{07	} clause mode for debug method 2 (0: normal, 1: addr, 2: kill)

#### 25.2 Control

DB_ALUCST_MEMSIZE	Size of the physical ALU constant memory
DB_TSTATE_MEMSIZE	Size of the physical texture state memory

#### 26. Interfaces

#### 26.1 External Interfaces

Whenever an x is used, it means that the bus is broadcast to all units of the same name. For example, if a bus is named  $SQ \rightarrow SPx$  it means that SQ is going to broadcast the same information to all SP instances.

#### 26.1.1 SC to SQ : IJ Control bus

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

Exhibit 2025 docR400\_Sequences.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257221

ATI Ex. 2107 IPR2023-00922 Page 87 of 260

	VAN	ORIGINATEL	JAIE	EDIT D	AIE		R400 Sequencer Specification	PAGE
		24 September	, 2001	4 Septembe	r, 2015	<u>518</u>		40 of 50
t	Name		Directio	on	Bits	Des	cription	
I	SC SQ q wr	mask	SC→S	Q	4	Qua	ad Write mask left to right	
	SC SQ lod co	orrect	SC→S	Q	24	LO	) correction per quad (6 bits per quad)	)
	SC_SQ_param	n_ptr0	SC→S	Q	11	PS	tore pointer for vertex 0	
	SC_SQ_param	n_ptr1	SC→S	Q	11	ΡS	tore pointer for vertex 1	
	SC_SQ_param	n_ptr2	SC→S	Q	11	ΡS	tore pointer for vertex 2	
	SC_SQ_end_c	of_vect	SC→S	Q	1	Enc	of the vector	
	SC_SQ_store_	dealloc	SC→S	Q	1	Dea	Illocation token for the P Store	
	SC_SQ_state		SC→S	Q	3	Sta	e/constant pointer	
	SC_SQ_valid_	pixel	SC→S	Q	16	Val	d bits for all pixels	
	SC_SQ_null_p	orim	SC→S	Q	1	Nul	Primitive (for PC deallocation purpose	es)
	SC_SQ_end_c	of_prim	SC→S	Q	1	Enc	Of the primitive	
	SC_SQ_send_	ху	SC→S	Q	1	Ser	ding XY information [XY information	is going to be
						sen	t on the next clock]	
	SC_SQ_prim_t	type	SC→S	Q	3	Rea	I time command need to load te	x cords from
						alte	rnate buffer. Line AA, Point AA and	i Sprite reads
						thei	r parameters from GEN_T and GEN_S	3 GPRs.
						000	: Normal	
						011	: Real Time	
						100	: Line AA	
						101	: Point AA	
						110	: Sprite	
	SC_SQ_new_	vector	SC→S	Q	1	This	s primitive comes from a new vecto	or of vertices.
						Mal	ke sure that the corresponding verte	x shader has
						finis	hed before starting the group of pixels	š
	SC_SQ_RTRn		SQ→S	С	1	Sta	Is the PA in n clocks	
	SC_SQ_RTS		SC→S	Q	1	SC	ready to send data	

#### 26.1.2 SQ to SP: Interpolator bus

Name	Direction	Bits	Description
SQ_SPx_interp_prim_type	SQ→SPx	3	Type of the primitive
			000 : Normal
			011 : Real Time
			100 : Line AA
			101 : Point AA
			110 : Sprite
SQ_SPx_interp_ijline	SQ→SPx	2	Line in the IJ/XY buffer to use to interpolate
SQ SPx interp mode	<u>SQ</u> →SPx	1	0: Use centroid buffer
			1: Use center buffer
SQ_SPx_interp_buff_swap	SQ→SPx	1	Swap the IJ/XY buffers at the end of the interpolation
SQ_SPx_interp_gen_I0	SQ→SPx	1	Generate I0 or not. This tells the interpolators not to
			use the parameter cache but rather overwrite the data
			with interpolated 1 and 0. Overwrite if gen_I0 is high.

# 26.1.3~ SQ to SX: Interpolator bus

Name	Direction	Bits	Description
SQ_SPx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SPx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SPx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_ptr1mux0	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_ptr2mux1	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_ptr3mux2	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_RT_switch	SQ→SXx	1	Selects between RT and Normal data
SQ SXx pc wr en	<u>SQ→SXx</u>	1	Write enable for the PC memories
SQ SXx pc wr addr	<u>SQ→SXx</u>	7	Write address for the PCs

Exhibit 2025 docR400\_Sequences.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*



#### 26.1.4 SQ to SP: Staging Register Data

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

Name	Direction	Bits	Description
SQ_SPx_vgt_vsisr_data	SQ→SPx	96	Pointers of indexes or HOS surface information
SQ_SPx_vgt_vsisr_double	SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ_SP0_data_valid	SQ→SP0	1	Data is valid
SQ_SP1_data_valid	SQ→SP1	1	Data is valid
SQ_SP2_data_valid	SQ→SP2	1	Data is valid
SQ_SP3_data_valid	SQ→SP3	1	Data is valid

26.1.5 PA to SQ : Vertex interface

#### 26.1.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. <u>Therefore, the POR for this interface is that the VGT will transmit the data to the</u> <u>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.

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

#### 26.1.5.2 Interface Diagrams

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257223

ATI Ex. 2107 IPR2023-00922 Page 89 of 260



Exhibit 2025. doc R400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page @ \*\*\*

AMD1044\_0257224

ATI Ex. 2107 IPR2023-00922 Page 90 of 260



Exhibit 2025 doc R400\_Sequencer doc 71630 Bytes\*\*\* 
 ATI Confidential. Reference Copyright Notice on Cover Page 
 \*\*\*

	24 September, 2001	4 September, 201518		44 of 50
	ORIGINATE DATE	EDITIDATE	R400 Sequencer Specification	PAGE

#### 26.1.6 SQ to CP: State report

Name	Direction	Bits	Description
SQ_CP_vrtx_ state	SEQ→CP	3	Oldest vertex state still in the pipe
SQ_CP_pix_state	SEQ→CP	3	Oldest pixel state still in the pipe

#### 26.1.7 SQ to SX: Control bus

Name	Direction	Bits	Description
SQ SXx exp Pixel	SQ→SXx	1	1: Pixel
			0: Vertex
SQ_SXx_exp_Clause	SQ→SXx	3	Clause number, which is needed for vertex clauses
SQ_SXx_exp_State	SQ→SXx	3	State ID
SQ_SXx_exp_exportID	SQ→SXx	1	ALU ID

These fields are sent synchronously with SP export data, described in SP0→SX0 interfaceevery time the sequencer picks an exporting clause for execution.

#### 26.1.8 SX to SQ : Output file control

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

#### 26.1.9 SQ to TP: Control bus

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

Name	Direction	Bits	Description
TPx_SQ_data_rdy	$TPx \rightarrow SQ$	1	Data ready
TPx_SQ_clause_num	TPx→ SQ	3	Clause number
TPx_SQ_Type	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instuct	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pmask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP2_pmask	SQ→TP2	4	Pixel mask 1 bit per pixel
SQ_TP3_lod_correct	SQ→TP3	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP3_pmask	SQ→TP3	4	Pixel mask 1 bit per pixel

Exhibit 2025 docR400\_Sequencer.dec 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

		ORIGINATE DATE		EDITUATE			DOCUMENT-REV. NUM.	PAGE
	400	24 September	, 2001	4 Septem	September, 2015 <del>18</del>		GEN-CXXXXX-REVA	45 of 50
	SQ_TPx_c	lause_num	SQ→TPx		3	Clause	number	
	SQ_TPx_w	/rite_gpr_index	SQ->TPx		7	Index i	nto Register file for write of returned Fet	ch Data

#### 26.1.10 TP to SQ: Texture stall

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



Name	Direction	Bits	Description
TP_SQ_fetch_stall	$TP \rightarrow SQ$	1	Do not send more texture request if asserted

#### 26.1.11 SQ to SP: Texture stall

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

#### 26.1.12 SQ to SP: GPR, Parameter cache control and auto counter

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_red_addren	SQ→SPx	1	Read Enable
SQ_SPx_gpr_wewr_addren	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC
SO SPy channel mask	SO SPV	1	The channel mask
SQ SP0 pixel mask	SQ→SP0	4	The pixel mask
SQ_SP1_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_pixel_mask	SQ→SP3	4	The pixel mask
SQ_SPx_gpr_input_mux	SQ→SPx	2	When the phase mux selects the inputs this tells from
			which source to read from: Interpolated data, VTX0,
			VTX1, autogen counter.
SQ SPx index count	SQ→SPx	12?	Index count, common for all shader pipes

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257227

ATI Ex. 2107 IPR2023-00922 Page 93 of 260



26.1.13 SQ to SPx: Instructions

Name	Direction	Bits	Description
SQ_SPx_instruct_start	SQ→SPx	1	Instruction start
SQ_SP_instruct	SQ→SPx	21	Transferred over 4 cycles0: SRC A Select2:0SRC A Argument Modifier3:3SRC A swizzle11:4VectorDst17:12Unused20:18
			- 1: SRC B Select 2:0 SRC B Argument Modifier 3:3 SRC B swizzle 11:4 ScalarDst 17:12 Unused 20:18
			- 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12
			- 3: Vector Opcode 4:0 Scalar Opcode 10:5 Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask 16:13 Scalar Write Mask 20:17
SQ SPx exp exportID	<u>SQ→SPx</u>	1	ALUID
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

Exhibit 2025 docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

6		24 September, 2	001	4 September, 201518 March 20024 March		5 <u>18</u>	GEN-CXXXXX-REVA	47 of 50
	00.000					outpi buffe clock	ut for all pixel exports (depth and rrs). 4x4 because 16 pixels are com	all color nputed per
	SQ_SP3_	export_wvalid	SQ→S	5P3	2	of the vector	orries whether to write low and/or high e 64-bit export data from each of the 1 ors	32-bit word 16 pixels or

#### 26.1.14 SP to SQ: Constant address load/ Predicate Set

Name	Direction	Bits	Description
SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load (4 bits only)
			to the sequencer
SP0_SQ_valid	SP0→SQ	1	Data valid
SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load (4 bits only)
			to the sequencer
SP1_SQ_valid	SP1→SQ	1	Data valid
SP2_SQ_const_addr	SP2→SQ	36	Constant address load / predicate vector load (4 bits only)
			to the sequencer
SP2_SQ_valid	SP2→SQ	1	Data valid
SP3_SQ_const_addr	SP3→SQ	36	Constant address load / predicate vector load (4 bits only)
			to the sequencer
SP3_SQ_valid	SP3→SQ	1	Data valid

#### 26.1.15 SQ to SPx: constant broadcast

Name	Direction	Bits	Description
SQ_SPx_constant	SQ→SPx	128	Constant broadcast

#### 26.1.16 SP0 to SQ: Kill vector load

Name	Direction	Bits	Description
SP0_SQ_kill_vect	SP0→SQ	4	Kill vector load
SP1_SQ_kill_vect	SP1→SQ	4	Kill vector load
SP2_SQ_kill_vect	SP2→SQ	4	Kill vector load
SP3_SQ_kill_vect	SP3→SQ	4	Kill vector load

#### 26.1.17 SQ to CP: RBBM bus

Name	Direction	Bits	Description
SQ_RBB_rs	SQ→CP	1	Read Strobe
SQ_RBB_rd	SQ→CP	32	Read Data
SQ_RBBM_nrtrtr	SQ→CP	1	Optional
SQ_RBBM_rtr	SQ→CP	1	Real-Time (Optional)

## 26.1.18 CP to SQ: RBBM bus

Name	Direction	Bits	Description
rbbm_we	CP→SQ	1	Write Enable
rbbm_a	CP→SQ	15	Address Upper Extent is TBD (16:2)
rbbm_wd	CP→SQ	32	Data
rbbm_be	CP→SQ	4	Byte Enables
rbbm_re	CP→SQ	1	Read Enable
rbb_rs0	CP→SQ	1	Read Return Strobe 0
rbb_rs1	CP→SQ	1	Read Return Strobe 1
rbb_rd0	CP→SQ	32	Read Data 0
rbb_rd1	CP→SQ	32	Read Data 0
RBBM_SQ_soft_reset	CP→SQ	1	Soft Reset

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257229

ATI Ex. 2107 IPR2023-00922 Page 95 of 260

V A M	ORIGINATE DATE	EDITDATE	R400 Sequencer Specification	PAGE	
	24 September, 2001	4 September, 201548		48 of 50	
27 Evon	aples of program	avagutiona			

#### 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)
- 11. the control packet continues to travel down the path of reservation stations until all clauses have been executed
  - position can be exported in ALU clause 3 (or 4?); the data (and the tag) is sent over a position bus (which is shared with all four shader pipes) back to the PA's position cache
  - A parameter cache pointer is also sent along with the position data. This tells to the PA where the data is going to be in the parameter cache.
    - there is a position export FIFO in the SP that buffers position data before it gets sent back to the PA

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

	ORIGINATE DATE	EDITDATE	DOCOMENT-REV. NOW.	PAGE
400	24 September, 2001	4 September, 201518	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.
- 6. SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)
  - note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels
  - the control packet contains the state pointer, the register file base pointer, and the LOD correction bits
  - all other information (such as quad address for example) travels in a separate FIFO
- 7. TSM0 accepts the control packet and fetches the instructions for fetch clause 0 from the global instruction store
  - TSM0 was first selected by the TSM arbiter before it could start
- 8. all instructions of fetch clause 0 are issued by TSM0
- 9. the control packet is passed to the next reservation station (the FIFO in front of ALU state machine 0, or ASM0 FIFO)
  - TSM0 does not wait for fetch requests made to the Fetch Unit to complete; it passes the register file write index for the fetch data to the TU, which will write the data to the RF as it is received
  - once the TU has written all the data for a particular clause to the register files, it increments a counter that is
    associated with the ASM0 FIFO; a count greater than zero indicates that the ALU state machine can go
    ahead and pop the FIFO and start to execute the ALU clause
- 10. ASM0 accepts the control packet (after being selected by the ASM arbiter) and gets the instructions for ALU clause 0 from the global instruction store
- 11. all instructions of ALU clause 0 are issued by ASM0, then the control packet is passed to the next reservation station (the FIFO in front of fetch state machine 1, or TSM1 FIFO)
- 12. the control packet continues to travel down the path of reservation stations until all clauses have been executed
  - pixel data is exported in the last ALU clause (clause 7)
    - it is sent to an output FIFO where it will be picked up by the render backend
    - the ASM arbiter will prevent a packet from starting on ASM7 if the output FIFO is full
- 13. after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*



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?

Exhibit 2025.docR400\_Sequencer.doc 71630 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257232

ATI Ex. 2107 IPR2023-00922 Page 98 of 260

		······································		
AR	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
<b>KUU</b>	24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	1 of 52
Author:	Laurent Lefebvre			
Issue To:		Copy No:		
	R400 S	equencer Spe	ecification	
		SQ		
		Version 1. <u>10</u> 8		
Overview: This req bloc	s is an architectural specific uired capabilities and expe cks, and provides internal st	cation for the R400 Sequenc cted uses of the block. It al ate diagrams.	cer block (SEQ). It provides an ov so describes the block interfaces,	erview of the internal sub-
AUTOMATIC/ Document Lo Current Intra	ALLY UPDATED FIELDS: ocation: C:\pe net Search Title: R400	erforce\r400\doc_lib\design\b D Sequencer Specification	locks\sq\R400_Sequencer.doc	
	Name/Dept	APPROVALS	Signature/Date	
Remarks: THIS DOC		S CONFIDENTIAL II		ULD BE
SUBSTA	INC. THROUGH	JNAUTHORIZED US	REST OF ATTTECHNOL	OGIES
"Copyright 2001 work created in unpublished we confidential, pro transmitted in ar	1, ATI Technologies Inc. Al 2001. The use of this cop ork. The copyright notice oprietary information and train ny form or by any means wi	I rights reserved. The mater yright notice is intended to p is not an admission that ade secrets of ATI. No part thout the prior written permiss	ial in this document constitutes ar provide notice that ATI owns a co publication has occurred. This v of this document may be used, re sion of ATI Technologies Inc."	n unpublished pyright in this vork contains eproduced, or
Exhibit 2026.doc	R400_Sequencer.doc 75288 Bytes*** @	ATI Confidential. Referen	ce Copyright Notice on Cover Pa	ige © ***

ATI 2026 LG v. ATI IPR2015-00325

AMD1044\_0257233

ATI Ex. 2107 IPR2023-00922 Page 99 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201525		2 of 52

#### Table Of Contents

1. OVERVIEW	
1.1 Top Level Block Diagram	108
1.2 Data Flow graph (SP)	1240
1.3 Control Graph	<u> 1311</u>
2. INTERPOLATED DATA BUS	<u> 1311</u>
3. INSTRUCTION STORE	<u> 1614</u>
4. SEQUENCER INSTRUCTIONS	<u> 1816</u>
5. CONSTANT STORES	<u> 1816</u>
5.1 Memory organizations	<u> 1846</u>
5.2 Management of the Control Flow Constants	<u> 1846</u>
5.3 Management of the re-mapping tables	1816
5.3.1 R400 Constant management	1816
5.3.2 Proposal for R400LE constant management	1917
5.3.3 Dirty bits	21 <del>19</del>
5.3.4 Free List Block	21 <del>19</del>
5.3.5 De-allocate Block	22 <del>20</del>
5.3.6 Operation of Incremental model	22 <del>20</del>
5.4 Constant Store Indexing	22 <del>20</del>
5.5 Real Time Commands.	2321
5.6 Constant Waterfalling	2321
6. LOOPING AND BRANCHES	2422
6.1 The controlling state	2422
6.2 The Control Flow Program	2422
6.3 Data dependant predicate instructions	2624
6.4 HW Detection of PV,PS	2725
6.5 Register file indexing	2725
6.6 Predicated Instruction support for Texture clauses	2725
6.7 Debugging the Shaders	28 <del>25</del>
6.7.1 Method 1: Debugging registers	28 <del>25</del>
6.7.2 Method 2: Exporting the values in the GPRs (12)	28 <del>26</del>
7. PIXEL KILL MASK	<u>28<del>26</del></u>
8. MULTIPASS VERTEX SHADERS (HOS)	2926
9. REGISTER FILE ALLOCATION	2926
10. FETCH ARBITRATION	3028
11. ALU ARBITRATION	30 <del>28</del>
12. HANDLING STALLS	3129
13. CONTENT OF THE RESERVATION STATION FIFOS	3129
14. THE OUTPUT FILE	3129
15. IJ FORMAT	31 <del>29</del>
15.1 Interpolation of constant attributes	3230
16. STAGING REGISTERS	3230
17. THE PARAMETER CACHE	3432
18. VERTEX POSITION EXPORTING	3532

Exhibit 2026.docR400\_Sequencer.doc 75288 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257234

ATI Ex. 2107 IPR2023-00922 Page 100 of 260

	2	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
<u> </u>	$\mathbf{M}$	24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	3 of 52
19. E	EXPO	RTING ARBITRATION			
<u>20. E</u>	EXPO	RT TYPES		******	3532
20.1	Vert	ex Shading	*****		<u>3532</u>
20.2 21 S	Pixe		MODES		
<u>21. c</u> 21.1	Real	time commands	MODES		3633
21.2	Sprit	es/ XY screen coordina	ates/ FB information		
21.3	Auto	generated counters			
<u>21.3</u>	3.1	Vertex shaders			
21.3	3.2	Pixel shaders			
<u>2. S</u>	STATI	E MANAGEMENT	********		3734
2.1	Para	meter cache synchron	ization		
<u>3. X</u>		DRESS IMPORTS			
<u>.3.1</u> 24 G		ex indexes imports			3835
4 1	Cont			***************************************	3835
24.2	Cont	text			
:5. C	DEBU	G REGISTERS			
5.1	Cont	iext			
5.2	Cont		******	******	
6.	NIE	RFACES	*********		
.6.1	Ext	ernal Interfaces			
<u>26.1</u>	1.1	SC to SQ : IJ Contr	ol bus		
<u>26.1</u>	1.2	SQ to SP: Interpolato	r bus		
<u>26.1</u>	1.3	SQ to SX: Interpolato	r bus	******	
<u>26.1</u>	1.4	SQ to SP: Staging Re	gister Data		
<u>26.1</u>	1.5	PA to SQ : Vertex inte	erface		
<u>26.1</u>	1.6	SQ to CP: State repo	<u>t</u>		
<u>26.1</u>	1.7	SQ to SX: Control bus	5		
<u>26.1</u>	1.8	SX to SQ : Output file	control		
<u>26.1</u>	1.9	SQ to TP: Control bus	5		
<u>26.1</u>	1.10	TP to SQ: Texture sta			
<u>26.1</u>	1.11	SQ to SP: Texture sta	<u>all</u>		
<u>26.1</u>	1.12	SQ to SP: GPR and a	uto counter		
<u>26.1</u>	1.13	SQ to SPx: Instruction	ns		
<u>26.1</u>	1.14	SP to SQ: Constant a	ddress load/ Predicate	Set	
26.1	1.15	SQ to SPx: constant I	proadcast		
26.1	1.16	SP0 to SQ: Kill vector	load		
26.1	1.17	SQ to CP: RBBM bus			
26.1	1.18	CP to SQ: RBBM bus			
27. E	EXAM	PLES OF PROGRAM	EXECUTIONS	*****	
27 1	1.1	Sequencer Control of	a Vector of Vertices		

Exhibit 2026.doc R400\_Sequences.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257235

ATI Ex. 2107 IPR2023-00922 Page 101 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 201525		4 of 52
07.1.0		March 20024 March		50.40
27.1.2	Sequencer Control	of a Vector of Pixels		
27.1.3	Notes			
28. OPEN	ISSUES	***********************************	*****	
I. OVER	VIEW	*****	*****	••••••••
	evel block blagram			Ö
1.2 Data F	- <del>IOW Graph (SP)</del>			<del></del> нч
	01 ATEN NATA DI	IC		
2 INISTE		/ <del>)</del>	***************************************	11
A SEOU	ENICED INICTRIJICTI	nic		16
5 CONS	TANT STORES	#19 <b>~</b> #	* * * * * * * * * * * * * * * * * * * *	16
5.1 Memo	rv organizations	**********************************	*****	16
5.2 Manac	rement of the re-man	ning tables		
521	Dirty hite	onig tabloo		10
522	The stat Dissi			
5.2.2	-ree List Block	*****		
<del>3.2.3</del>	Je-allocate Block	· · · · · ·		
3.2.4 (	Operation of Increment	ntal model		
5.3 Const	ant Store Indexing		***************************************	
5.4 Real I	Ime Commands		•••••••••••••••••••••••••••••••••••••••	
5.5 Const	ant Waterfalling			
6. LOOP	ING AND BRANCHE	.s		
6.1 The CO	ontrolling state			
6.2 Detec	<del>ontroi ⊨iow ⊭rogram .</del> Ionondont prodiosto i	aatriintiana		~~~~
	ependant predicate in ataction of DV/DS	15truguons		
6.5 Pogiet	er file indeving			24
6.6 Predic	oted Instruction supp	ort for Texture clauses	~	2/
6.7 Debuc	aina the Shaders	OILIOI I OXLUIO OIQUOO		
671	Method 1: Debugging	radictore		25
672	Method I: Debugging		(40)	
0.7.2	vietnod 2: Exporting ti	te values in the GPR	<del>S (12)</del>	
7. PIXEL	KILL MASK	DEDO (1100)	*****	
8. MULI	IPASS VERIEX SHA	WEKS (HUS)	* * * * * * * * * * * * * * * * * * * *	
9. KEGR	HEK FILE ALLUGA	HON		
	HARDHRAHUN	******		
12 UAND	INC STALLS	*****		At 20
	EVIT VE THE DECER		IFOS	60 ວວ
14THE C		CONTRACTION OF MILITON F		
15 LLEON	2MΔT	***************************************		28
151 Inter	polation of constant a	ittributes	* * * * * * * * * * * * * * * * * * * *	29
16 STAG	ING REGISTERS	n se son a present Theth San Theeth a and a <u>a a a a a a a a a a a a a a a a a a</u>		29
17. THE F	ARAMETER CACHE	• • • • • • • • • • • • • • • • • • •		
18 VERT	EX POSITION EXPO	RTING		31
19. EXPO	RTING ARBITRATIC	N		
20. EXPO	RT TYPES			
20.1 Vert	ex Shading			
	-			

Exhibit 2026.doc R409\_Sequences.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257236

ATI Ex. 2107 IPR2023-00922 Page 102 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	5 of 52
).2 Pixe	I Shading	Marah 20024 Marah		
SPEC	IAL INTERPOLATION	MODES	*****	
-l Keal	time commands	atao/ED information		
-2 - Ophi -3 - Auto	es/ AT screen coordin			
2131	Vertex shaders			33
21.3.1	Pivel chadere			33
STAT	E MANAGEMENT			
.1 Para	meter cache synchron	ization		
XY AD	DRESS IMPORTS	******		
1 Verte	ex indexes imports			34
- KEGIS	HEKS	******	******	34
.2 Cont	ext	*****		
DEBU	G REGISTERS	*****	****	
1-Cont	ext			
-INTE	RFACES	******		
I Exte	ernal Interfaces			
26.1.1-	SC to SQ : IJ Contr	ol bus		
26.1.2	SQ to SP: Interpolato	r bus		
26.1.3-	SQ to SP: Parameter	Cache Read control bu	IS	
26.1.4	SQ to SX: Parameter	Cache Mux control Bus	÷	
26.1.5	SQ to SP: Staging Re	gister Data		
26 <u>.1.6</u> —	PA to SQ : Vertex inte	erface		
26 <u>.1.7</u> —	SQ to CP: State repo	rt	****	
2 <del>6.1.8</del> —	SQ to SX: Control but	S		
26.1.9	SX to SQ : Output file	control		
26.1.10-	SQ to TP: Control but	S <del></del>	*****	
26.1.11-	TP to SQ: Texture sta	₩		
26.1.12-	SQ to SP: Texture sta	all		
26.1.13-	SQ to SP: GPR, Para	meter cache control an	d auto counter	
26.1.14_	SQ to SPx: Instruction	าธ	*****	
26.1.15-	SP to SQ: Constant a	ddress load		
26.1.16-	SQ to SPx: constant I	proadcast	*****	
26.1.17-	SP0 to SQ: Kill vector	load		
26.1.18-	-SQ to CP: RBBM hus			44
26.1.19	CP to SQ: RBBM bus			44
EXAM	PLES OF PROGRAM	EXECUTIONS		
27.1.1-	Sequencer Control of	a Vector of Vertices	****	
27.1.2	Sequencer Control of	a Vector of Pixels		
	Nintan			46

AMD1044\_0257237

ATI Ex. 2107 IPR2023-00922 Page 103 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE		
	24 September, 2001	4 September, 201525		6 of 52		
28. OPEN ISSUES						

Exhibit 2026 doc R409-Sequencer.doc 75288 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257238

ATI Ex. 2107 IPR2023-00922 Page 104 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE			
	24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	7 of 52			
Revision Changes:							
Rev 0.1 (Lauro Date: May 7, 2	ent Lefebvre) 1001	First dra	ft.				
Rev 0.2 (Laure	ent Lefebvre) 2001	Change SP Add	d the interfaces to reflect the changes and some details in the arbitration sec	s in the			
Rev 0.3 (Laure	ent Lefebvre)	Reviewe	ed the Sequencer spec after the mee	ting on			
Date : August	6, 2001	August	3, 2001.	•			
Rev 0.4 (Laure	ent Lefebvre)	Added file and	the dynamic allocation method for i	register			
Date : August :	24, 2001	flow of p	ixels/vertices in the sequencer.	or the			
Rev 0.5 (Laure	ent Lefebvre)	Added t	iming diagrams (Vic)				
Date : Septem	ber 7, 2001	Change	d the energy to reflect the new	B400			
Date : Septem	ber 24. 2001	architec	ture. Added interfaces.	R400			
Rev 0.7 (Laure	ent Lefebvre)	Added	constant store management, insi	truction			
Date : October	5, 2001	store m	anagement, control flow manageme	ent and			
Rev 0.8 (Laure	ent Lefebvre)	data dej Chande	d the control flow method to be	more			
Date : October	8, 2001	flexible.	Also updated the external interfaces.				
Rev 0.9 (Laure	ent Lefebvre)	Incorpor	rated changes made in the 10/18/01	control			
Date : October	17, 2001	tiow me	nditional execute or jump Added	debua			
		register	S.				
Rev 1.0 (Laure	ent Lefebvre)	Refined	interfaces to RB. Added state registe	rs.			
Rev 1.1 (Laure	ng, 2001 ent Lefebvre)	Added	SEQ→SP0 interfaces. Changed	delta			
Date : October	26, 2001	precisio	n. Changed VGT→SP0 interface.	Debug			
Devid O (Levin	while factories)	Methods	s added.				
Date : Novemb	per 16 2001	Interiace	as greatly relined. Cleaned up the spe	C.			
Rev 1.3 (Laure	ent Lefebvre)	Added t	he different interpolation modes.				
Date : Novemb	per 26, 2001	أحجاجا والمراجع					
Date : Decemb	ent Letebvre) ber 6. 2001	Added the VG1	the auto incrementing counters. Ci -→SQ interface Added content on c	nanged			
		manage	ment. Updated GPRs.				
Rev 1.5 (Laure	ent Lefebvre)	Remove	ed from the spec all interfaces that	weren't			
Date : Decemi	ber 11, 2001	constan	tied to the SQ. Added explanation to the SQ. Added explanation to the source of the source of the term of	A⊸SQ			
		synchro	nization fields and explanation.				
Rev 1.6 (Laure	ent Lefebvre)	Added I	nore details on the staging register.	Added			
Date . January	7, 2002	call ins	truction to a Conditionnal call inst	ruction.			
		Added	details on constant managemen	nt and			
Poy 17 (Lours	ant Lofohura)	updated	the diagram.	ha SV			
Date : Februar	v 4. 2002	interface	e. Updated the control flow section.	ne ox			
Rev 1.8 (Laure	ent Lefebvre)	New int	erfaces to the SX block. Added the	end of			
Date : March 4	, 2002	clause	modifier, removed the end of	clause			
Rev 1.9 (Laure	ent Lefebvre)	Rearan	gement of the CF instruction bits in c	order to			
Date : March 1	8, 2002	ensure	oyte alignement.				
Rev 1.10 (Lau	rent Lefebvre)	Updated	the interfaces and added a sect	ion on			
Date . March 2		exportin	g rules.				
Exhibit 2026.docR-	400_Sequencer.doc 75288 Bytes*** ©	ATI Confidential. Refere	nce Copyright Notice on Cover Pag	e © ***			

AMD1044\_0257239

ATI Ex. 2107 IPR2023-00922 Page 105 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201525		8 of 52

#### 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 2026.docR499\_Sequences.doc 75288 Bytes\*\*\* <sup>©</sup> ATI Confidential. Reference Copyright Notice on Cover Page <sup>©</sup> \*\*\*

AMD1044\_0257240

ATI Ex. 2107 IPR2023-00922 Page 106 of 260

# PROTECTIVE ORDER MATERIAL



Exhibit 2026 doe R400\_Sequencer abo 72288 Bytes \*\*\* 
 ATI Confidential. Reference Copyright Notice on Cover Page 
 \*\*\*

AMD1044\_0257241

ATI Ex. 2107 IPR2023-00922 Page 107 of 260



#### 1.1 Top Level Block Diagram



#### Figure 2: Reservation stations and arbiters

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

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

Exhibit 2026.docR499\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257242

ATI Ex. 2107 IPR2023-00922 Page 108 of 260
ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	11 of 52

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator, which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

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

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

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

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

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

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

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

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

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

Exhibit 2026.doc R400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257243

ATI Ex. 2107 IPR2023-00922 Page 109 of 260



ATI Ex. 2107 IPR2023-00922 Page 110 of 260



Exhibit 2026.docR400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257245

ATI Ex. 2107 IPR2023-00922 Page 111 of 260



ATI Ex. 2107 IPR2023-00922 Page 112 of 260 PROTECTIVE ORDER MATERIAL

		53	I	1			1 ~ ~ ~	2 4 5	\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \		
		22 T3					2 5 5 5 7 4 8 7 7 8 7 7	> 400 202	> 0 th > 0 th > 0 th	>47 >00%	$\rightarrow$
		21 T					- 00 - 60	7 0 0 7 0 0	740	7 % -	
		20 T.					<u>√=</u> +	~~~ ~ ~	~~~~	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	>
		19 T					<u> </u>	>4	<u>&gt;∞</u> 0	- >255	
		- 80							ш	Ш	
<b></b>		7			<u> </u>					ă	
GE	of 52	11.			×В	ΧÜ		වි	5	G	
РA	15 0	T16			ЮШ	Щ				BO	N
		T15			В	Ш Т	5	D2			Ω
NUM.	NA.	Т14	₹₽	28		≿8	ប	5	C2		
REV. I	XX-RE	T13	5	D2		DO	B1				
AENT-	CXXX	T12	Б	D2		Do	AO	A1	A2		
ocun	GEN-	T11		≿2	Ç5	000	e f		ШО	Ш	
		T10	1	2	C5	0	y -			8	
	52	Т9		2	C5			8	ۍ ۲	C C	
TE	2015	Т8	≿ ຕ	≿ ୪	אַ ז	≳ റ്റ				BO	hamme
DIT D/	ember	17	ß	8	ъ	C2	5	D2			٩
Ξ	1 Septe	T6	ខ	8	ы С	8	ប	5	C5		
	~	T5	×₽			X B0 X	8				
DATE	2001	Τ4	B1			BO	AO	A1	A2		
ATE D	mber,	13	8			BO	X 48- 51	XY 52- 55	X≺ 56- 59	60 × X	
RIGIN	Septe	12	×۹	Υ¥	₹₹		XY 32- 35	အ အိ X	∑ 4 £	X 44 44 44 44 44 44 44 44 44 44 44 44 44	$\succ$
9 9	24	Ê	AO	A1	A2		, 16 19 19	3 \$ ≾	24- 27	3 % X	$\times$
PC		<u>р</u>	Ao	A1	A2		×°¦	¥ 7-4	≍ ∾ ≍	12 12 ≾	
	र्		d o	₽ C	⊳ SP	ი წ	С O	Ч <del>С</del>	N SP	ია	

Figure 6: Interpolation timing diagram

Exhibit 2028 doc/84.00\_Sequences 72288 Bytes\*\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257247

ATI Ex. 2107 IPR2023-00922 Page 113 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	7
24 September, 2001	4 September, 201525		16 of 52	

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 2026.docR400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257248

ATI Ex. 2107 IPR2023-00922 Page 114 of 260

# PROTECTIVE ORDER MATERIAL



# R400 CP's Views of Instruction Memory





Figure 7: The CP's view of the instruction memory

Exhibit 2028 doo Programmer doo 75288 Byres\*\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257249

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201525		18 of 52

# 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 <u>128x192\_320x96</u> bits (<u>128 texture states</u> for regular mode, <u>32 states for RT</u>). The memory thus holds 128 texture states (<u>192 bits per state</u>). The logical size exposes <u>32</u> 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 <u>32</u> lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or <u>192 bits</u>). The driver sends <u>512 bits</u> but the CP ignores the top <u>320 bits</u>. It thus takes <u>6 clocks</u> to write the texture state. Real time requires <u>32 lines</u> in the physical memory (this is physically register mapped).

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

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

# 5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF\_WR\_BASE). On the read side, one level of indirection is used. A register (SQ\_CONTEXT\_MISC.CF\_RD\_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

# 5.3 Management of the re-mapping tables

### 5.3.1 R400 Constant management

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

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

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space

Exhibit 2026.doc,R409\_Sequencer.dec 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257250

ATI Ex. 2107 IPR2023-00922 Page 116 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	19 of 52

is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32\*2+32 = 96 entries and above.

### 5.3.2 Proposal for R400LE constant management

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ\_IDLE and PA\_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 9: De-allocation mechanismFigure 9: De-allocation mechanism

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).

Exhibit 2026.docR400\_Sequencer.doc 75288 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257251

ATI Ex. 2107 IPR2023-00922 Page 117 of 260



ATI Ex. 2107 IPR2023-00922 Page 118 of 260



### Figure 9: De-allocation mechanism for R400LE

### 5.3.3 Dirty bits

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

### 5.3.4 Free List Block

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

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

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

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

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

Exhibit 2026.doc R409\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

### AMD1044\_0257253

ATI Ex. 2107 IPR2023-00922 Page 119 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201525		22 of 52

5.3.5 De-allocate Block

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

### 5.3.6 Operation of Incremental model

The basic operation of the model would start with the write\_ptr, stop\_ptr, read\_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter is 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 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.
- Reset dirty set and Context dirty not set. A new physical address is allocated, the physical address in the renaming table is put on the free list at write\_ptr and it is incremented along with the de-allocate counter for the last context.
- 3.) Context dirty is set then the data will be written into the physical address specified by the logical address.

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

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

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

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

# 5.4 Constant Store Indexing

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

Exhibit 2026.docR400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257254

ATI Ex. 2107 IPR2023-00922 Page 120 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	23 of 52

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

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

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

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

### 5.5 Real Time Commands

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

# 5.6 Constant Waterfalling

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



Figure 10: The instruction store

Exhibit 2026 docR409\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257255

ATI Ex. 2107 IPR2023-00922 Page 121 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201525		24 of 52

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 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.

The control program has nine basic instructions:

Execute Conditional\_execute Conditional\_Execute\_Predicates Conditional\_jump Conditionnal\_Call Return Loop\_start Loop\_end NOP

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

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

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

Exhibit 2026.doc R400\_Sequencer.doc 75288 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257256

ATI Ex. 2107 IPR2023-00922 Page 122 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	100000
24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	25 of 52	

Conditional\_execute\_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition. Conditional\_jumps jumps to an address if the condition is met. NOP is a regular NOP

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

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

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

A value of 1 in the Addressing means that the address specified in the Exec Address field (or in the jump address field) is an ABSOLUTE address. If the addressing field is cleared (should be the default) then the address is relative to the base of the current shader program.

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

	Execute					
47	46 42	41	40 24	23 12	11 0	
Addressing	00001	Last	RESERVED	Instruction	Exec Address	
				count		

Execute up to 4k instructions at the specified address in the instruction memory. If Last is set, this is the last group of instructions of the clause.

			NOP
47	46 42	41	40 0
Addressing	00010	Last	RESERVED

This is a regular NOP. If Last is set, this is the last instruction of the clause.

	Conditional_Execute							
47         46 42         41         40         40 <u>39</u> <u>3332</u> 32 <u>31</u> 34 <u>30</u> 24         23 12         11 0							11 0	
Addressing	00011	Last	RESERVED	Boolean address	Condition	RESERVED	Instruction count	Exec Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions). If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Conditional_Execute_Predicates								
47	47         46 42         41         40.40         34.33         3231         31-30 24         23 12         11 0							11 0
Addressing	00100	Last	RESERVED	Predicate vector	Condition	RESERVED	Instruction count	Exec Address

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

		Loop_Start		
47	46 42	41 17	16 12	11 0

Exhibit 2026.doc R400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257257

ATI Ex. 2107 IPR2023-00922 Page 123 of 260

		ORIGINATE DATE		EDIT DATE	R400 Sequencer S	PAGE	_	
		24 Sept	tember, 2001	4 September, 201525			26 of 52	
		00101		RESERVED		loop ID	Jump address	T
1	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
	00110	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 35 <u>34</u>	34- <u>33</u> 33 <u>32</u>	3 <u>1</u> 2	<del>31_<u>30</u> 12</del>	11 0	
Addressing	00111	RESERVED	Predicate vector	Condition	RESERVED	Jump address	

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

	Return							
47	46 42	41 0						
	01000	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	47         46 42         4 <u>1 40</u> <sup>1</sup> 40 <u>.39</u> 32 <u>31</u> 34 <u>30</u> 30 <u>.29</u> 12         11 0						
	01001 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.

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

# 6.3 Data dependant predicate instructions

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

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

For the scalar operations only we will also support the two following instructions: PRED\_SETE0\_# – SETE0

Exhibit 2026.doc/R409\_Sequencer.doc 75288 Bytes\*\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257258

ATI Ex. 2107 IPR2023-00922 Page 124 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	]
24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	27 of 52	

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.

We loop until loop\_iterator = loop\_count. Loop\_step is a signed value [-128...127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the "indexing logic" so that it knows when the provided index is out of range and thus can make the necessary arrangements.

### 6.6 Predicated Instruction support for Texture clauses

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

Exhibit 2026.docR400\_Sequencer.dec 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257259

ATI Ex. 2107 IPR2023-00922 Page 125 of 260

 		-			
	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 2001	4 September, 201525 March, 20024 March		28 of 52	
6.7 Debu	gging the Shade	ers			
In order to be a	ble to debug the pixel/ve	rtex shaders efficiently, we	e provide 2 methods.		
671 Moth	and 1: Dobugging	radiatara			
Current plans a	ing in expose 2 debugging	a or error potification red	ietore ·		
1. address regis 2. count of the	ster where the first error on the first error of the first error of errors the first error errors the first error erro	occurred	191619.		
The sequencer	will detect the following g	groups of errors:			
<ul> <li>count overflow</li> <li>constant index</li> </ul>	v king overflow				
- jump errors	nizable errors:				
relative - call stack	jump address > size of t	he control flow program			
call with return v	h stack full with stack empty				
A jump error wi	Il always cause the prog	ram to break. In this case	a break means that a clause will halt	execution but	
allowing further	clauses to be executed.	,		,	
With all the oth the DB_PROB_	er errors, program can c _BREAK register is set.	continue to run, potentially	to worst-case limits. The program wi	ll only break if	
If indexing outs the value with register (or con	ide of the constant or the an index of 0. This coul stant) for errors.	e register range, causing a d be exploited to generat	an overflow error, the hardware is spece e error tokens, by reserving and initia	cified to return alizing the Oth	
{ISSUE : Interru	upt to the driver or not?}				
6.7.2 Meth	nod 2: Exporting th	he values in the GF	PRs (12)		
The sequencer execution mode	will have a count registe e for each clause. The me 1) Normal 2) Debug Kill	er and an address registe odes can be :	r for this mode and 3 bits per clause	specifying the	
Under the norm executed but a clause 7 will be an address spe After we have h	3) Debug Addr + Cournal mode execution folic II normal shader instruct executed under the deb ecified by the address re hit the instruction n times	t wws the normal course. L ions of the clause are rej ug kill setting. Under the o gister and instruction cou (n=count) we switch the c	Inder the kill mode, all control flow in blaced by NOPs. Only debug_export other mode, normal execution is done nt (useful for loops) specified by the o lause to the kill mode.	structions are instructions of until we reach count register.	
Under the debu vectors and tha the sequencer /	ig mode (debug kill OR d at all other exports to the (even if they occur before	lebug Addr + count), it is a SX block (position, color, e the address stated by the	assumed that clause 7 is always expor z, ect) will been turned off (changed i e ADDR debug register).	ting 12 debug into NOPs) by	
7. Pixel K					
A vector of 64 requests and a					
MASK_ MASK_ MASK_	_SETE _SETNE _SETGT				
Exhibit 2026.docR4	00_Sequencer.doc 75288 Bytes***	© ATI Confidential. Ref	erence Copyright Notice on Cover P	'age © ***	

ATI Ex. 2107 IPR2023-00922 Page 126 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	29 of 52	
MASK	_SETGTE				

8. Multipass vertex shaders (HOS)

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

# 9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX\_REG\_SIZE for vertices and PIXEL\_REG\_SIZE for pixels.

Exhibit 2026.docR400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257261

ATI Ex. 2107 IPR2023-00922 Page 127 of 260



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 2026.docR400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257262

ATI Ex. 2107 IPR2023-00922 Page 128 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1
24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	31 of 52	

# 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).

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

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

Exhibit 2026.doc R400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257263

ATI Ex. 2107 IPR2023-00922 Page 129 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201525		32 of 52

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(| != 0) {
              P0 = A
           } else if(J != 0) {
              P0 = B;
           } else {
              PO = 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 2026.docR409\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257264

ATI Ex. 2107 IPR2023-00922 Page 130 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	33 of 52

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 12. The area of the fixed-to-float converters and the VSISRs for this method is roughly estimated as 0.759sqmm using the R300 process. The gate count estimate is shown in Figure 11Figure 11Figure 11.

Basis for 8-deep Latch Memory (fror	n R300)		
8x24-bit	11631	$\mu^2$	$60.57813\mu^2\text{per}$ bit
Area of 96x8-deep Latch Memory	46524	$\mu^2$	
Area of 24-bit Fix-to-float Converter	4712	$\mu^2$ per conv	erter
Method 1	Block	Quantity	Area
	F2F	3	14136
	8x96 Latch	16	744384
			758520 µ <sup>2</sup>

Figure 11:Area Estimate for VGT to Shader Interface

Exhibit 2026.doc R400\_Sequencer.doc 75288 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257265

ATI Ex. 2107 IPR2023-00922 Page 131 of 260



### 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.

r	
MEMORY NUMBER	ADDRESS
4 bits	7 bits
	1 6165

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current\_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current\_Location by VS\_EXPORT\_COUNT\_7 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS\_EXPORT\_COUNT\_7 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 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> 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 2\*VS\_EXPORT\_COUNT\_7to Current\_Location and reset the memory count to 0 before the next vector begins).

Exhibit 2026.docR400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257266

ATI Ex. 2107 IPR2023-00922 Page 132 of 260

	,				
AP	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	35 of 52	
18. Verte	ex position export	ing			
On clause 3 th clause 7 if no the position and The clause with the export is g	he vertex shader can expor t done at clause 3. The sto nd 64x32 memories for the here the position export oc joing to occur at ALU clause	t to the PA both the vertex p rage needed to perform the sprite size. It is going to be curs is specified by the EXP e 7 if unset position export of	position and the point sprite. It can a position export is at least 64x128 m taken in the pixel output fifo from the 'ORT_LATE register. If turned on, it cours at clause 3.	lso do so at emories for SX blocks. means that	
19. <b>Expo</b>	orting Arbitration				
Here are the r 1) P	ules for co-issuing exporting osition exports and position	g ALU clauses. exports cannot be co-issuec	ł.		
All other types	s of exports can be co-issue	d as long as there is place ir	the receiving buffer.		
{ISSUE: Do w	e move the parameter cach	es to the SX?}			Formatted: Bullets and Numbering
20. Expo	orting Rules			*1	······
20.1 Par	ameter caches e	orts			
We support m	asking and out of order exp	ports to the parameter cache	s. So one can export multiple times	to the same	
PC line using	different masks.			<b>4</b>	
20.2 Wer	nory exports		of order to memory locations		
	ition oxports	iowever, you can export out	of order to memory locations.	4-	Formatted: Bullets and Numbering
20.3 FUS	rts have to be done IN ORD	IFR and don't support maski	na		
				* 1	Formatted: Bullets and Numbering
20- <u>21. E</u> )	kport Types		····		
ALU instructio	n. Here is a list of all possib	le data should be put) is spe le export modes:	ecified using the destination address	field in the	
00 101 1	Vertex Chading			*	Formatted: Bullets and Numbering
20.121.1	vertex Shading				
	0:15 - 16 parameter c 16:31 - Empty (Reserve	ache ed?)			
	32:43 - 12 vertex export 44:47 - Empty	s to the frame buffer and ind	ex		
	48:59 - 12 debug expor	t (interpret as normal vertex	export)		
	61 - Empty	ng mode			
	62 - position 63 - sprite size expo (point_h,point_v	rt that goes with position exp v,edgeflag,misc)	port		
<del>20.2</del> 21.2	Pixel Shading			*-	
	0 - Color for buffer 1 - Color for buffer	0 (primary) 1			
Exhibit 2026.doc	1400_Sequencer.dec 75288 Bytes*** 🤅	> ATI Confidential. Referer	nce Copyright Notice on Cover Pag	je © ***	

ATI Ex. 2107 IPR2023-00922 Page 133 of 260

	T	r	r			
	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE		
	24 September, 2001	4 September, 201525 March 20024 March		36 of 52		
	2         - Color for buffe           3         - Color for buffe           4:7         - Empty           8         - Buffer 0 Color           9         - Buffer 1 Color           10         - Buffer 2 Color           11         - Buffer 2 Color           12:15         - Empty           16:31         - Empty           16:32         - 12 exports for           44:47         - Empty           48:59         - 12 debug exp           60         - export address           61:62         - Empty           63         - Z for primary b	<pre>/F 2 /Fog (primary) /Fog /Fog /Fog ved?) multipass pixel shaders. orts (interpret as normal p sing mode buffer (Z exported to 'alph;</pre>	ixel export) a' component)			Formatted: Bullets and Numbering
21-22. S	pecial Interpolati	<u>on modes</u>		*		
<del>21.1</del> 22.1	Real time comm	ands				
We are unable need to add tf register bus a be able to add should be abl other is raster memory to 16 view support f stream), then parameters in the SX blocks mapped mem	e to use the parameter montree 16x128 memories (or nd written by type 0 packed dress the reatime paramet e able to view them as tw rized with. Most overlay sl x64 or 32x64 allowing on for 16 vector-4 interpolants the PA/sequencer need to stead of 16. This mode is the parameter data mer ory.	emory since there is no way ne for each of three vertice ets, and output to the the p er memory as well as the vo banks of 16 and do do haders will need 2 or 4 sc ly two interpolated scalars is important (true only if way to support a realtime-spec triggered by the primitive i mories are hooked on the	ay for a command stream to write into es x 16 interpolants). These will be ma parameter busses (the sequencer and/ regular parameter store. For higher pe- vable buffering allowing one to be loa ealar coordinates, one option might be s per cycle, the only problem I see with e map Microsoft's high priority stream cific mode where we need to address type: REAL TIME. The actual memoric RBBM bus and are loaded by the CP	• it. Instead we apped onto the /or PA need to erformance we ded, while the to restrict the th this is, if we to the realtime of 32 vectors of es are in the in using register		
21.222.2	Sprites/ XY scre	en coordinates/	FB information	*	1 N N	Formatted: Bullets and Numbering
When working coordinates m conjunction w special operat together:	g with sprites, one may v nay be needed in the sha ith the SND_XY register (i tions) to the shader using	vant to overwrite the para der program. This function n SC). Also it is possible t the same control register.	ameter 0 with SC generated data. Als nality is controlled by the gen_l0 regis o send the faceness information (for C Here is a list of all the modes and how	so, XY screen ster (in SQ) in DGL front/back w they interact		
Gen_st is a bi set, it means between 0 and	it taken from the interface we are dealing with Point / d 1.	between the SC and the S AA, Line AA or sprite and i	SQ. This is the MSB of the primitive ty in this case the vertex values are going	pe. If the bit is g to generated		
Param_Gen_ Param_Gen_ Param_Gen_ Param_Gen_ Param_Gen_ Param_Gen_ Param_Gen_	0 disable, snd_xy disable, 10 disable, snd_xy disable, 10 disable, snd_xy disable, 10 disable, snd_xy enable, 10 enable, snd_xy disable, 10 enable, snd_xy disable, 10 enable, snd_xy enable, 10 enable, snd_xy enable,	no gen_st - I0 = No mod , gen_st - I0 = No modifica no gen_st - I0 = No modifica no gen_st - I0 = No modifica no gen_st - I0 = garbage gen_st - I0 = garbage, ga no gen_st - I0 = screen x gen_st - I0 = screen x, sc	ification ation fication tition , garbage, garbage, faceness arbage, s, t , screen y, garbage, faceness reen y, s, t			
21.322.3	Auto generated	counters		4-	A.	Formatted: Bullets and Numbering
In the cases we both use this The count is a	we are dealing with multip count to write the 1 <sup>st</sup> pass always generated in the sa	ass shaders, the sequences data to memory and there are way but it is passed to	er is going to generate a vector coun n use the count to retrieve the data or o the shader in a slightly different way	t to be able to the 2 <sup>nd</sup> pass. depending on		

Exhibit 2026.doc R400\_Sequences.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257268

ATI Ex. 2107 IPR2023-00922 Page 134 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1
24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	37 of 52	
 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		and and a state of the second state of the sec		and the second second second second

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.

### 21.3.122.3.1 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).

### 21.3.222.3.2 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).



### 22.23. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

# 22.123.1 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.

Exhibit 2026.doc R409\_Sequencer.dec 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0257269

ATI Ex. 2107 IPR2023-00922 Page 135 of 260

		OPICINAT				0.15.11	DACE		
		24 Sentemi	e DATE	4 Sentember 201525	R400 Sequen	cer Specification	38 of 52		
2	3 24 XV	/ Addree	e impor	March 20024 March			*	Formatted: Bullets and Numbering	)
	$= \frac{24}{100}$	able to send	the XX ad	LO	toos so hy interlea	ving the writes of the	alle (to the II		
bu in th	uffer) with X terpolate the e GPRs. The ection <u>22.2</u> 24	Y writes (to IJ data or pase e Xys are cu 2 for details of	the XY buf ss the XY da rrently SCR	fer). Then when writing ata thru a Fix→float con EEN SPACE COORDII ontrol the interpolation ir	the data to the C verter and expander NATES. The values this mode.	GPRs, the sequence r and write the conve s in the XY buffers v	er is going to erted values to will wrap. See		
2	<u>3.124.1</u>	Vertex in	idexes i	mports			4-		
່ In bl	order to imp ock (96 bits).	ort vertex ind They are loa	exes, we ha ded in floati	ve 16 8x96 staging regi ng point format and can	sters. These are loa be transferred in 4 d	aded one line at a tim or 8 clocks to the GP	ne by the VGT 'Rs.	<b>Formatted:</b> Bullets and Numbering	
2	4. <u>25. Re</u>	egisters					*	~ (	
2	4.125.1	Control							
	REG_DYN REG_SIZE	IAMIC E_PIX	Dynam Size of	ic allocation (pixel/verte) the register file's pixel p	) of the register file ortion (minimal size	on or off. when dynamic alloc	ation turned		
	REG_SIZE	E_VTX	Size of on)	the register file's vertex	portion (minimal siz	e when dynamic allo	ecation turned		
	ARBITRAT	TION_POLICY DRE_ALLOC DE_VTX	/ policy of interleat start policy of Begins	of the arbitration between ved, separate int for the vertex instruc at 0)	vertexes and pixel ion store (RT alway	s /s ends at vertex_ba	se and		
	INST_BAS ONE_THR ONE_ALU	SE_PIX READ	start po debug debug	int for the pixel shader i state register. Only allow state register. Only allow	nstruction store s one program at a rs one ALU program	time into the GPRs n at a time to be exe	cuted (instead		
	INSTRUC	TION	This is	where the CP puts the	base address of the	e instruction writes a	nd type (auto-		
	CONSTAN CONSTAN CONSTAN	NTS_RT NTS_RT NT_EO_RT	512*4 / 256*4 / This is CONS <sup>-</sup>	ALU constants + 32*6 Te ALU constants + 32*6 te the size of the space r ANT EO RT). The re-r	egister mapped xture state 32 bits r (ture states? (physic eserved for real tim napping table opera	registers (logically ma cally mapped) ne in the constant sto ates on the rest of the	apped) ore (from 0 to e memory		
	TSTATE_E	EO_RT	This is TSTAT	the size of the space re	served for real time	in the fetch state st on the rest of the me	tore (from 0 to		
	EXPORT_	LATE	Control exports	s whether or not we a occur at clause 7.	re exporting position	on from clause 3. If	f set, position		
2	4 <u>.2</u> 25.2	Context					*-	Formatted: Bullets and Numbering	
I	VS_FETC VS_ALU_{ PS_FETC_{ PS_BASE VS_BASE VS_CF_SI PS_CF_SI PS_SIZE PS_NUM_ VS_NUM_ PARAM_S PROVO_V	H_{07} {07} H_{07} {07} IZE IZE REG REG SHADE /ERT	eight 8 eight 8 eight 8 base pu base po size of size of size of numbe One 16 = goura 0 : verti	bit pointers to the location bit pointers to the location bit pointers to the location bit pointers to the location pointer for the pixel shade binter for the vertex shader the vertex shader (# of inst the pixel shader (# of inst the vertex shader (cnt1+in the vertex shader (cnt1+in the vertex shader (cnt1+in of GPRs to allocate for to it register specifying with und) ex 0, 1: vertex 1, 2: vertex	on where each claus on where each claus on where each claus on where each claus on where each claus or in the instruction s ier in the instruction s ier in the instructions structions in control structions) nstructions) nstructions) pixel shader progra vertex shader progra hich parameters are x 2, 3: Last vertex of	ses control program i ses control program i ses control program i store of program/2) program/2) mms rams e to be gouraud shace of the primitive	is located is located is located is located ded (0 = flat, 1		
	Exhibit 2026.docR-	400_Sequencer.doc	75288 Bytes***	© ATI Confidential. R	eference Copyrigh	t Notice on Cover P	age © ***		

ATI Ex. 2107 IPR2023-00922 Page 136 of 260

					1
Zan	ORIGINATE	DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 Septembe	er, 2001	4 September, 201525	GEN-CXXXXX-REVA	39 of 52
PARAM	_WRAP	64 bits: f	or which parameters (and ch	annels (xyzw)) do we do the cyl wra	pping
PS_EXF	PORT_MODE		lormal mode		
		1xxxx : N	lultipass mode bbbz where bbb is bow mai	ny colors (0-4) and z is export z or n	ot
		lf multipa	iss 1-12 exports for color.		
VS_EXF	PORT_MASK	Which of 0: positic	the last 6 ALU clauses is exp in (1 vector), 1: position (2 ve	porting (multipass only) actors), 3:multipass	
VS_EXF		0		// · · ·	in
_COUN	1_{06}	(located	in VS_EXPORT_COUNT_6)	or interpolated parameters exported OR	In clause 7
	GEN IO	# of expo	orted vectors to memory per	clause in multipass mode (per claus	e) d S values
GEN_IN	_GEN_IO	Auto gen	erates an address from 0 to	XX. Puts the results into R0-1 for pi	el shaders
CONST	BASE VTX (9 hit	and R2 f	or vertex shaders	ts of the Vertex shader	
CONST	BASE_PIX (9 bits	s) Logical E	ase address for the constan	ts of the Pixel shader	
CONST	_SIZE_PIX (8 bits) SIZE_VTX (8 bits	Size of the Size o	ne logical constant store for p ne logical constant store for v	oixel shaders rertex shaders	
INST_P	RED_OPTIMIZE	Turns on	the predicate bit optimizatio	n (if of, conditional_execute_predica	tes is
CF_BO	OLEANS	always e 256 bool	ean bits		
CF_LOC	DP_COUNT	32x8 bit	counters (number of times w	e traverse the loop)	
CF_LOC	OP_STEP	32x8 bit	counters (step value used in	index computation)	
25.26 E		vietore			4-
<u> </u>		131613			
25.126.1	Context				
DB	 PROB_ADDR	instructio	n address where the first pro	oblem occurred	
DB_	PROB_COUNT	number o	of problems encountered dur	ing the execution of the program	
DB_ DB_	INST_COUNT	instructio	n counter for debug method	2	
DB_ DB	BREAK_ADDR	break ad	dress for method number 2		
_MC	DDE_ALU_{07}	clause m	ode for debug method 2 (0:	normal, 1: addr, 2: kill)	
MC	_CLAUSE DDE_FETCH_{0	7} c	lause mode for debug metho	od 2 (0: normal, 1: addr, 2: kill)	
_		-	-		
<del>25.2</del> 26.2	2_Control				4-
DB_	ALUCST_MEMSI	ZE S	Size of the physical ALU con	stant memory	
DB_	TSTATE_MEMSIZ	ZE S	Size of the physical texture s	tate memory	
26. <u>27. l</u> i	nterfaces				
26.127 1	External In	iterface	S		
Whenever a		ane that th	o hue is broadcast to all up	ite of the same name. For example	ifabueie
named SQ-	→SPx it means that	SQ is goir	ig to broadcast the same info	prmation to all SP instances.	, ii a bus is
27.2 SC	to SP Inter	faces			4-
*****	******	**********************			
Exhibit 2026.d	ocR400_Sequencer.doc 75	5288 Bytes*** 🧿	ATI Confidential. Referer	ice Copyright Notice on Cover Pa	ge © ***

ATI Ex. 2107 IPR2023-00922 Page 137 of 260

0				PAGE								
	24 September 2001	4 Sentember 201525	R400 Sequencer Specification	40 of 52								
07.01.00		March 20024 March		40 01 02								
There is one of these interfaces at front of each of the SP (buffer to stage pixel interpolators). This interface transmits the I,J data for pixel interpolation. For the entire system, two quads per clock are transferred to the 4 SPs, so each of these 4 interfaces transmits one half of a quad per clock. The interface below describes a half of a quad worth of data. The actual data which is transferred per quad is Ref Pix I => S4.20 Floating Point I value Delta Pix I (x3) => S4.8 Floating Point Delta I value This equates to a total of 128 bits which transferred over 2 clocks and therefor needs an interface 64 bits wide Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an												
Additionally, X additional cloc Transfers acro The data trans SC has sent a sending the ne the SC will sta Note: We cou it is planned fo Centroids are o In at least the increment buff EndOfVector s packet and m performance h	A data (12-bit unsigned C. The X,Y data is sent of ss these interfaces are s fer across each of these pixel vector's worth of d initial vector's data, he Il until the SQ returns a Id/may optimize for the or the SP to hold 2 double on, then the SC can send initial version, the SC service address pointers ignal on all interfaces to ultiple new vector signal it.) Bits	I fixed) is conditionally se on the lower 24 bits of the c ynchronized with the SC S busses is controlled by a ata to the SPs, he will incr will check to make sure the pipelined pulse to decrem ase of only sending only I, abuffers of I,J data and two two Buffers. shall send 16 quads per j o correctly all the time. (W quit early, We opted for the should cause a partial Description	nt across this data bus over the sam lata bus with faceness in the msb. SQ IJ Control Bus transfers. IJ BUF INUSE COUNT in the SC. rement the IJ BUF INUSE COUNT c the count is less than MAX_BUFER_Milent the count when he has scheduled J to use all the buffers to pre-load mor o buffers of X,Y data, so if either X,Y o bixel vector even if the vector is not fe may revisit this for both the SX,SP,S fe simple mode first with a belief that o vector and that this would not really	Each time the ount. Prior to NUS 2, if not a buffer free. e. Currently r Centers and full. This will SQ and add a nly the end of be significant								
<u>SC</u> SP#_data	<u>64</u>	Description           I jinformation sent over 2 c           Type 0 or 1, First clock I, s           Field         ULC           Bits         [63:39]           Format         SE4M20           SE4M20         SE4M           Type 2         Field           Field         Face           Bits         [63]           Format         Bit	Stocks (or X,Y in 24 LSBs with faceness i           econd clk J           C         LLC           C6         [25:13]           [12:0]           A8         SE4M8           X         Y           [3:12]         [11:0]           Isigned         Unsigned	n upper bit)								
SC_SP# valid       1       Valid         SC_SP# last guad       1       This bit will be set on the last transfer of data per guad.         SC_SP# type       2       0> Indicates centroids 1> Indicates centers 2> Indicates Centers         2> Indicates Centers       1> Indicates Centers         1> Indicates Centers       1> Indicates         1> Indicates Centers       1> Indicates         1> Indicates Centers       1> Indicates         1> Indicates       1> Indicates         1> Indicates       1> Indicates         1> Indicates       1> Indicates         1> Indicates       1> Indicates         1> Indicates </td												
The # is include the SC and the 27.2.2 SC This is the co loading data in clocks per trans	The # is included for clarity in the spec and will be replaced with a prefix of u#_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix. 27.2.2 SC_SQ This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 92 bits) could be folded in half to approx 46 bits.											
Exhibit 2026.docR-	100_Sequencer.doc 75288 Bytes***	© ATI Confidential. Ref	erence Copyright Notice on Cover P	age © ***								

ATI Ex. 2107 IPR2023-00922 Page 138 of 260

			1									
	ORIGINATE DATE			EDIT DATE	DOCUMENT-REV. NUM.	DOCUMENT-REV. NUM. PAGE						
	24 September	r, 2001	<u>4 Se</u>	ptember, 201525	GEN-CXXXXX-REVA	41 of 52						
ł.			0.0ar	ab 20024 Blacob								
Name	Name Bits Description											
SC_SQ_data		46										
		-	1 CIK tran	<u>sters</u> ntvalid data	consist of event id and							
	state id. Instruct SQ to post an											
	event vector to send state id and											
		-		event_id through request fifo								
		-		making su	re state id and/or event id							
		-		gets back t	to the CP. Events only							
				follow end	of packets so no pixel							
				vectors wil	l be in progress.							
			Fm	oty Quad Mask – Tra	nsfer Control data							
				consisting	of pc_dealloc							
				or new ve	ctor. Receipt of this is to							
		-		transfer pc	dealloc or new vector							
		-		vector will	always be posted to							
				request fife	and pc_dealloc will be							
		-		attached to	any pixel vector							
		-		outstandin	g or posted in request fifo							
			2 clk tran	sfers	diau outstanding.							
			Qua	d Data Valid – Send	ing quad data with or							
				without nev	w vector or pc_dealloc.							
		-		New vecto	r will be posted to request							
		-										
				vector unless none is in progress. In								
		-		this case the pc_dealloc will be								
		-		posted in t	he request queue.							
		-		The Quad	mask set but the nixel							
		-		correspond	ding pixel mask set to							
	Zero.											
SC SO valid		1	SC send	ing valid data 2 <sup>nd</sup> clk	could be all zeroes							
00 00 Valid		<u> </u>										
SC_SQ_data -	- first clock and s	econd cl	ock trans	fers are shown in the	e table below.							
88		maniate	1 P3:4-	Nr. 1 Ph. 1 (*								
wame		BITLIEIC	I BITS	Description								
1 <sup>st</sup> Clock Trans	sfer											
SC SQ event		0	1	This transfer is a 1	clock event vector							
				Force quad_mask =	= new_vector=pc_dealloc=0							
SC_SQ_event	event_id [2:1] 2 This field identifies the event											
				$\frac{U \Rightarrow \text{denotes an Er}}{1 \Rightarrow \text{TRD}}$	to Of State Event							
SC SQ pc de	alloc	3	I => IBD     Deallocation token for the Parameter Cache									
SC SQ new V	/ector	4	1	The SQ must wai	t for Vertex shader done count >	0 and after						
			dispatching the Pixel Vector the SQ will decrement the count.									
SC_SQ_quad	mask [8:5] 4 Quad Write mask left to right SP0 => SP3											
SC SQ end c	of prim	9	1 End Of the primitive									
SC SQ state	<u>lu</u> ask	[12.10]	16	Valid hits for all nix	els SP0=>SP3 (ULURUE)							
SC_SQ prim	type	[31:29]	3	Stippled line and R	Real time command need to load tex	cords from						

Exhibit 2026.doc R400\_Sequences.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257273

ATI Ex. 2107 IPR2023-00922 Page 139 of 260

			-,			·		
	ORIGINATE DATE 24 September, 2001		EDIT DATE		R400 Sequencer Specification	PAGE		
			4 Sept	ember, 201525		42 of 52		
				alternate buffer 000: Normal 100: Realtime 101: Line AA 110: Point AA (	Sprite)			
SC SQ pc ptr0 [42:32]		1 11	Parameter Cache pointer for vertex 0					
2nd Clock Transfer			T		****			
SC SQ pc pl	SC_SQ_pc_ptr1 [10:0]		<u>11</u>	Parameter Cache pointer for vertex 1				
SC SQ pc pt	tr2	[21:11	] 11	Parameter Cache pointer for vertex 2				
SC_SQ_lod_correct [45:22]		24	LOD correction per quad (6 bits per quad)					
Name Bits		Description						
SQ_SC_free_buff 1			Pipeline	pelined bit that instructs SC to decrement count of buffers in use.				
SQ SC dec (	SQ SC dec cntr cnt 1			ined bit that instructs SC to decrement count of new vector and/or event				
			sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.					

The scan converter will submit a partial vector whenever:

He gets a primitive marked with an end of packet signal.

2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc dealloc (vertices maximum size). In this case two new vectors are submitted and processed, but then one valid quad with the pc dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc\_dealloc signal made it through and thus the hang.)

### 26.1.1 SC to SQ : IJ Control bus

This is the control information sent to the sequencer in order to control the IJ fifos and all other information needed to execute a shader program on the sent pixels. This information is sent over 2 clocks, if SENDXY is asserted the next control packet is going to be ignored and XY information is going to be sent on the IJ bus (for the quade that where just sent). All pixels from the group of quade are from the same primitive, all quade of a vector are from the same render state.

### 26.1.2 SQ to SP: Interpolator bus

Formatted: Bullets and Numbering

# 26.1.327.2.3 SQ to SX: Interpolator bus

Name	Direction	Bits	Description	18					
SQ_SPxSXx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading	1990					
SQ_SPXx_interp_flat_gourau	SQ→SPx	1	Flat or gouraud shading	1000					
d									
SQ_S₽Xx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped	188					
SQ_SXXx_ptr1mux0	SQ→SXx	11	Parameter Cache Pointer						
SQ_SXx_ptr2mux1	SQ→SXx	11	Parameter Cache Pointer						
SQ_SXx_ptr3mux2	SQ→SXx	11	Parameter Cache Pointer						
SQ_SXx_RT_switchrt_sel	SQ→SXx	1	Selects between RT and Normal data						
SQ SXx pc wr en	<u>SQ→SXx</u>	1	Write enable for the PC memories						
SQ_SXx pc_wr_addr	<u>SQ→SXx</u>	7	Write address for the PCs						
SQ_SXx_pc_cmask	<u>SQ→SXx</u>	4	Channel mask						
26.1.427.2.4 SQ to SP: Staging Register Data									

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

 Name
 Direction
 Bits
 Description

Exhibit 2026.doc R400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257274

ATI Ex. 2107 IPR2023-00922 Page 140 of 260

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

	ORIGINATE	DATE	EDI	T DATE		DOCUMENT-REV. NUM.	PAGE
	24 September	r, 2001	4 Septem	ber, 20	01525	GEN-CXXXXX-REVA	43 of 52
SQ_SPx_vgt_v	sisrvsr_data	SQ→SPx		96	Pointer	s of indexes or HOS surface information	on
SQ_SPx_vgt_v	sisrvsr_double	SQ→SPx		1	0: Norn	nal 96 bits per vert 1: double 192 bits p	er 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	SQ_SP3_data_valid			1	Data is	valid	

26.1.527.2.5 PA-VGT to SQ : Vertex interface

# 26.1.5.127.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. <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.

Name	Bits	Description	
PAVGT_SQvgt_vsisr_data	96	Pointers of indexes or HOS surface information	
VGTPA_SQvgt_vsisr_doubl	1	0: Normal 96 bits per vert 1: double 192 bits per vert	
e			
VGTPASQvgt_end_of_v	1	Indicates the last VSISR data set for the current process vector (for double vector	
ector		data, "end_of_vector" is set on the second vector)	
VGTPA_SQvgt_vsisrindx_v	1	Vsisr data is valid	
alid			
VGTPA_SQvgt_state	3	Render State (6*3+3 for constants). This signal is guaranteed to be correct when	I
		"PAVGT_SQ_vgt_end_of_vector" is high.	
VGTPA_SQvgt_send	1	Data on the VGT_SQ is valid receive (see write-up for standard R400 SEND/RTR	
		interface handshaking)	
SQ_VGT_PA_vgt_rtr	1	Ready to receive (see write-up for standard R400 SEND/RTR interface	l
		handshaking)	ľ

26.1.5.227.2.5.2 Interface Diagrams

Exhibit 2026.docR400\_Sequencer.dec 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

-

Formatted: Bullets and Numbering

AMD1044\_0257275

ATI Ex. 2107 IPR2023-00922 Page 141 of 260

# PROTECTIVE ORDER MATERIAL



Exhibit 2028 doc R400\_Sequencer doo 75288 Bytes\*\*\*\* (Confidential. Reference Copyright Notice on Cover Page (C \*\*\*

AMD1044\_0257276

ATI Ex. 2107 IPR2023-00922 Page 142 of 260 PROTECTIVE ORDER MATERIAL



AMD1044\_0257277

	ORIGINATE DA	TE	EDIT	DATE	R400 Sequencer Specification PAG			
24 September 2001		4 September 201525		525		46 of 52		
			March 200	)74 Mar	oh			E
26.1.627.2	2.6 SQ to CP.	: Sta	te report				*	- ( <b>Formatted:</b> Bullets and Numbering
Name		Directi	on	Bits	Des	cription		
SQ_CP_vrtx_s	state	SEQ→	CP	3	Olde	st vertex state still in the pipe		
SQ_CP_pix_sta	ate	SEQ→	CP	3	Olde	st pixel state still in the pipe		
<del>26.1.7</del> <u>27.2</u>	<u>2.7_</u> SQ to SX.	: Cor	ntrol bus				*	Formatted: Bullets and Numbering
Name		Directi	on	Bits	Des	cription		
SQ_SXx_exp_	Pixelpix	SQ→S	Xx	1	1: P 0: V	xel ertex		
SQ_SXx_exp_	_cGlause 3	SQ→S	Хх	3	Clau	ise number, which is needed for verte	x clauses	
SQ_SXx_exp_	sState	SQ→S	Xx	3	Stat	e ID		
SQ_SXx_exp_	_exportIDalu_id	SQ→S	Xx	1	ALU	ID	]	
26.1.827.2	2.8_SX to SQ	: Ou	tput file c	ontrol	Bits	Description	····	(
SXx_SQ_Expe	ortexp_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_Expo	ort <u>exp_Pposition_s</u>	pac	SXx→SQ		1 Specifies whether there is room for another position.			
SXx_SQ_expExport_bBuffer_space SXx→			SXx→SQ		<ul> <li>7 Specifies the space available in the output buffers.</li> <li>0: buffers are full</li> <li>1: 2K-bits available (32-bits for each of the 64 pixels in a clause)</li> <li></li> </ul>			
						64: 128K-bits available (16 128-bit e each of 64 pixels) 65-127: RESERVED	ntries for	
<del>26.1.9</del> 27.2	2.9_SQ to TP:	: Con	ntrol bus				4	Formatted: Bullets and Numbering
Once every clo is ready or not also provides t	ock, the fetch unit s t. This way the sequ the instruction and	ends to	o the sequen can update t	cer on v	which n coun	clause it is now working and if the data ters for the reservation station fifos. Th	a in the GPRs he sequencer	

Name	Direction	Bits	Description
TPx_SQ_data_rdy	TPx→ SQ	1	Data ready
TPx_SQ_clause_num	$TPx \rightarrow SQ$	3	Clause number
TPx_SQ_t∓ype	TPx→ SQ	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_send	<u>SQ→TPx</u>	1	Sending valid data
SQ_TPx_const	SQ→TPx	48	Fetch state sent over 4 clocks (192 bits total)
SQ_TPx_instuctinstr	SQ→TPx	24	Fetch instruction sent over 4 clocks
SQ_TPx_end_of_clause	SQ→TPx	1	Last instruction of the clause
SQ_TPx_Type	SQ→TPx	1	Type of data sent (0:PIXEL, 1:VERTEX)
SQ_TPx_phase	SQ→TPx	2	Write phase signal
SQ_TP0_lod_correct	SQ→TP0	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP0_pmask	SQ→TP0	4	Pixel mask 1 bit per pixel
SQ_TP1_lod_correct	SQ→TP1	6	LOD correct 3 bits per comp 2 components per quad
SQ_TP1_pmask	SQ→TP1	4	Pixel mask 1 bit per pixel
SQ_TP2_lod_correct	SQ→TP2	6	LOD correct 3 bits per comp 2 components per quad

Exhibit 2026.docR400\_Sequencer.doc 75288 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257278

ATI Ex. 2107 IPR2023-00922 Page 144 of 260
	ORIGINATE	DATE	EDI	T DATE	Ξ	DOCUMENT-REV. NUM.	PAGE
	24 Septembe	r, 2001	4 Septen	nber, 20	01525	GEN-CXXXXX-REVA	47 of 52
SQ_TP2_pmas	ik	SQ→TP2		4	Pixel m	nask 1 bit per pixel	
SQ_TP3_lod_c	orrect	SQ→TP3		6	LOD co	orrect 3 bits per comp 2 components pe	er quad
SQ_TP3_pmas	sk 🛛	SQ→TP3		4	Pixel m	nask 1 bit per pixel	
SQ_TPx_claus	e_num	SQ→TPx		3	Clause	number	
SQ_TPx_write	_gpr_index	SQ->TPx		7	Index i	nto Register file for write of returned Fe	tch Data

## 26.1.1027.2.10 TP to SQ: Texture stall

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



Name	Direction	Bits	Description	]
TP_SQ_fetch_stall	$TP \rightarrow SQ$	1	Do not send more texture request if asserted	]

# 26.1.1127.2.11 SQ to SP: Texture stall

Name	Direction	Bits	Description
SQ SPx fetch stall	SQ→SPx	1	Do not send more texture request if asserted

## 26.1.1227.2.12 SQ to SP: GPR, Parameter cache control and auto counter

Name	Direction	Bits	Description
SQ_SPx_gpr_wr_addr	SQ→SPx	7	Write address
SQ_SPx_gpr_rd_addr	SQ→SPx	7	Read address
SQ_SPx_gpr_red_addren	SQ→SPx	1	Read Enable
SQ_SPx_gpr_wewr_addren	SQ→SPx	1	Write Enable for the GPRs
SQ_SPx_gpr_phase_mux	SQ→SPx	2	The phase mux (arbitrates between inputs, ALU SRC
			reads and writes)
SQ_SPx_channel_mask	SQ→SPx	4	The channel mask
SQ_SP0_pixel_mask	SQ→SP0	4	The pixel mask
SQ_SP1_pixel_mask	SQ→SP1	4	The pixel mask
SQ_SP2_pixel_mask	SQ→SP2	4	The pixel mask
SQ_SP3_pixel_mask	SQ→SP3	4	The pixel mask
SQ_SPx_gpr_input_mux	SQ→SPx	2	When the phase mux selects the inputs this tells from
			which source to read from: Interpolated data, VTX0,
			VTX1, autogen counter.
SQ_SPx_indexauto_count	SQ→SPx	12?	Index_Auto_count_generated by the SQ, common for all
			shader pipes

Exhibit 2026.doc.R400\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

AMD1044\_0257279

ATI Ex. 2107 IPR2023-00922 Page 145 of 260

ORIGINATE	DATE	EDIT [	DATE	R400 Sequencer Specification	PAGE	
6 1 1207 0 12 00 M	- CD	March 200	24 Marok	2	40 01 32	Formatted: Bullets and Numbering
0 <del>.1.13</del> <u>27.2.13</u> SQ 70	D SPX:	Instruction	75			
Name	Dire	ection	Bits	Description		
SQ_SPX_Instruct_start	<u>SQ</u> -	→SPX	21	Instruction start		
			21	O: SRC A Select 2:0 SRC A Argument Modifier 3:3 SRC A swizzle 11:4 VectorDst 17:12 Unused 20:18		
				1: SRC B Select     2:0       SRC B Argument Modifier     3:3       SRC B swizzle     11:4       ScalarDst     17:12       Unused     20:18		
				- 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12		
				- 3: Vector Opcode Scalar Opcode Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask Scalar Write Mask 20:17		
SQ SPx exp alu id	<u>SQ</u>	<u>→SPx</u>	1	<u>ALU ID</u>		
SQ SPx exporting	<u>SQ-</u>	→SPx	2	0: Not Exporting 1: Vector Exporting 2: Scalar Exporting		
SQ SPx stall	SQ	→SPx	1	Stall signal		
SQ_SP0_export_pvalid	SQ-	→SP0	4	Result of pixel kill in the shader pipe, whi output for all pixel exports (depth an buffers). 4x4 because 16 pixels are con clock	ch must be d all color nputed per	
SQ_SP1_ expert_pvalid	SQ-	→SP1	4	Result of pixel kill in the shader pipe, whi output for all pixel exports (depth an buffers). 4x4 because 16 pixels are con clock	ch must be d all color mputed per	
SQ_SP2_ expert_pvalid	SQ-	→SP2	4	Result of pixel kill in the shader pipe, whi output for all pixel exports (depth an buffers). 4x4 because 16 pixels are con clock	ch must be d all color mputed per	
SQ_SP3_ expert_pvalid	SQ-	→SP3	4	Result of pixel kill in the shader pipe, whi output for all pixel exports (depth an buffers). 4x4 because 16 pixels are con clock	ch must be d all color mputed per	
6.1.14 <u>27.2.14</u> SP to	SQ: C	Constant a	ddress	s load/ Predicate Set		<b>Formatted:</b> Bullets and Numbering
Name	Directi	on	Bits	Description		
SP0_SQ_const_addr	SP0→	SQ	36	Constant address load / predicate vector load to the sequencer	(4 bits only)	
SP0_SQ_valid	SP0→	SQ	1	Data valid		
SP1_SQ_const_addr	SP1→	SQ	36	Constant address load / predicate vector load	(4 bits only)	

Exhibit 2026.docR409\_Sequencer.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

to the sequencer

AMD1044\_0257280

ATI Ex. 2107 IPR2023-00922 Page 146 of 260

		ORIGINATE	DATE	EDIT	DATE	.	DOCUMENT-REV. NUM.	PAGE			
	UU-	24 Septemb	er, 2001	4 Septemi	ber, 20	01525	GEN-CXXXXX-REVA	49 of 52			
	D4 00		001 00	Marah 20	024 M	arah	····				
S	P1_SQ_0 P2_SQ_0	const_addr	SP1→SQ SP2→SQ		36	Constan	nd ht address load / predicate vector lo equencer	ad (4 bits only)			
S	P2 S0 1	valid	SP2-SO		1	Data valid					
s	P3 SQ d	const addr	SP3→SQ		36	Constan	nt address load / predicate vector lo	ad (4 bits only)			
		_				to the se	equencer	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,			
S	P3_SQ_V	/alid	SP3→SQ		1	Data val	lid				
26.1.1527.2.15 SQ to SPx: constant broadcast								Formatted: Bullets and Numbering			
N	lame		Direction		Bits	Descrip	tion				
S	Q_SPx_c	constant	SQ→SPx		128	Constan	it broadcast			~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	
26.1	1 <u>.1627</u>	<u>.2.16</u> SP0	to SQ: K	ill vector	load			-	~~~	Formatted: Bullets and Numbering	
N	lame		Direction		Bits	Descrip	tion				
S	P0_SQ_I	<pre>dil_vect</pre>	SP0→SQ		4	Kill vecto	or load				
S	P1_SQ_I	kill_vect	SP1→SQ		4	Kill vecto	or load				
S	P2_SQ_I	dill_vect	SP2→SQ		4	Kill vecto	or load				
S	P3_SQ_I	dil_vect	SP3→SQ		4	Kill vecto	or load				
26.1	1 <u>.1727</u>	<u>.2.17</u> SQ t	o CP: RE	BM bus						Formatted: Bullets and Numbering	
N	lame		Direction		Bits	Descrip	tion				
S	Q_RBB_	rs	SQ→CP		1	Read St	trobe				
S	Q_RBB_	_rd	SQ→CP		32	Read D	ata				
S	Q_RBBM	∕I_nrtrtr	SQ→CP		1	Optiona					
S	Q_RBB	۸_rtr	SQ→CP		1	Real-Tir	me (Optional)				
26.1	1 <u>.1827</u>	<u>.2.18</u> CP to	o SQ: RE	BM bus				<b>4</b>		Formatted: Bullets and Numbering	
N	lame		Direction		Bits	Descrip	tion	\			
rt	obm_we		CP→SQ		1	Write Ei	nable				
rt	obm_a		CP→SQ		15	Address	s Upper Extent is TBD (16:2)				
rt	obm_wd		CP→SQ		32	Data					
rt	obm_be		CP→SQ		4	Byte En	ables				
rk	obm_re		CP→SQ		1	Read E	nable				
rk	ob_rs0		CP→SQ		1	Read R	eturn Strobe 0				
rt	ob_rs1		CP→SQ		1	Read R	eturn Strobe 1				
rk	ob_rd0		CP→SQ		32	Read D	ata 0				
rk	ob_rd1		CP→SQ		32	Read D	ata 0				
R	BBM_SC	Q_soft_reset	CP→SQ		1	Soft Re	set				
27.2	28. Ex	camples of	<sup>f</sup> progra	m execı	utior	<u>15</u>		• • • • • • • • • • • • • • • • • • •		Formatted: Bullets and Numbering	$\square$
27.1	1 <u>.128.</u>	1.1_Sequer	ncer Cont	trol of a V	/ecto	or of Ve	ertices				
1. P	A sends state	a vector of 64 ve pointer as well a	ertices (actu s tag into po	ally vertex in sition cache	dices is sen	– 32 bits/ t along w	index for 2048 bit total) to the RE's	s Vertex FIFO			
e e	<ul> <li>space was 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 MH?)</li> <li>The vertex program is assumed to be loaded when we receive the vertex vector.</li> </ul>							ith the vertex			
	<ul> <li>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)</li> </ul>							all			
2. S	EQ arbit at this	rates between th point the vector	e Pixel FIFC	) and the Ve from the Ver	rtex F rtex Fl	IFO – bas FO	sically the Vertex FIFO always has	s priority			
Exh	Exhibit 2026.doc R400_Sequences.doc 75288 Bytes*** © ATI Confidential. Reference Copyright Notice on Cover Page © ***							Page © ***			

ATI Ex. 2107 IPR2023-00922 Page 147 of 260

	AA	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	<u>AU</u>	24 September, 2001	4 September, 201525		50 of 52
	<ul> <li>the a noth</li> </ul>	arbiter is not going to select ing else to do (ie no pixels a	a vector to be transformed are in the pixel fifo).	d if the parameter cache is full unless t	the pipe as
3.	SEQ allo • the r state • SEQ	ocates space in the SP regis number of GPRs required by pointer that came down wi will not send vertex data u	ster file for index data plus y the program is stored in th the vertices ntil space in the register fil	GPRs used by the program a local state register, which is accesse e has been allocated	ed using the
4.	SEQ ser • the 6 • 1 • 1 • 1 • 1 • the i of th	nds the vector to the SP reg 54 vertex indices are sent to RF0 of SU0, SU1, SU2, and RF1 of SU0, SU1, SU2, and RF2 of SU0, SU1, SU2, and RF3 of SU0, SU1, SU2, and ndex is written to the leasts e 128-bit location within the	ister file over the RE_SP i the 64 register files over I SU3 is written the first cy I SU3 is written the secon I SU3 is written the third cy I SU3 is written the fourth significant 32 bits (floating register file (w); the rema	nterface (which has a bandwidth of 20 4 cycles cle d cycle ycle cycle I point format?) (what about compo ining data bits are set to zero (x, y, z)	u48 bits/cycle) und indices)
5.	SEQ cor fetch sta • the c	nstructs a control packet for te machine 0, or TSM0 FIF0 control packet contains the s	the vector and sends it to O) state pointer, the tag to the	the first reservation station (the FIFO e position cache and a register file bas	in front of e pointer.
6.	TSM0 ao ● TSM	ccepts the control packet an 0 was first selected by the	d fetches the instructions TSM arbiter before it could	for fetch clause 0 from the global instr I start	uction store
7.	all instru	ctions of fetch clause 0 are	issued by TSM0		
8.	the contr FIFO) • TSM the f • once FIFC clause	ol packet is passed to the n 0 does not wait for requests etch data to the TU, which v e the TU has written all the o ); a count greater than zero se	ext reservation station (th s made to the Fetch Unit to will write the data to the Ri data to the register files, it indicates that the ALU sta	e FIFO in front of ALU state machine ( o complete; it passes the register file v F as it is received increments a counter that is associate the machine can go ahead start to exec	D, or ASM0 write index for ed with ASM0 cute the ALU
9.	ASM0 ao clause 0	ccepts the control packet (a from the global instruction	fter being selected by the store	ASM arbiter) and gets the instructions	for ALU
10	). all instru station (t	ctions of ALU clause 0 are i he FIFO in front of fetch sta	ssued by ASM0, then the te machine 1, or TSM1 FI	control packet is passed to the next re FO)	eservation
1-	I. the contr • positi shar • A pa goin • 1 • 1 • 1 • 1 • 1 • 1 • 1 • 1	ol packet continues to trave- tion can be exported in ALU ed with all four shader piper rameter cache pointer is als g to be in the parameter cac- there is a position export FIF the ASM arbiter will prevent meter data is exported in cl- parameter data is sent to the the SEQ allocates storage in onger a need for the param the ASM arbiter will prevent f position is being exported);	el down the path of reserva l clause 3 (or 4?); the data s) back to the PA's positio so sent along with the pos she. FO in the SP that buffers p a packet from starting an ause 7 (as well as positior e Parameter Cache over a n the Parameter Cache, ai eters (it is told by the PA v a packet from starting on ) is full	ation stations until all clauses have bee (and the tag) is sent over a position b n cache ition data. This tells to the PA where th position data before it gets sent back to exporting clause if the position export n data if it was not exported earlier) a dedicated bus nd the SEQ deallocates that space wh when using a token). ASM7 if the parameter cache (or the p	en executed ous (which is ne data is o the PA FIFO is full nen there is no position buffer
12	2. after the shader p	shader program has compl program	eted, the SEQ will free up	the GPRs so that they can be used by	y another
2	7.1.2 <u>28</u>	. <u>1.2_</u> Sequencer Co	ntrol of a Vector of	<sup>F</sup> Pixels	-4
1.	As with	vertex shader programs,	pixel shaders are loadec	I into the global instruction store by	/ the CP
	<ul> <li>At th</li> </ul>	is point it is assumed that th	ne pixel program is loaded	into the instruction store and thus rea	dy to be read.
	Exhibit 2026.dc	cR400_Sequencer.doc 75288 Bytes***	ATI Confidential. Reference     ATI Confidential. Reference     ATI Confidential. Reference     ATI Confidential. ATI Confidential. Reference     ATI Confidential. ATI Confidential. Reference     ATI Confidential. Reference     ATI Confidential. ATI Confidential. Reference     ATI Confidential. Reference     ATI Confidential. ATI Confidential. A	erence Copyright Notice on Cover F	Page © ***

ATI Ex. 2107 IPR2023-00922 Page 148 of 260

9	29	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
$\boldsymbol{\mathcal{G}}$	900	24 September, 2001	4 September, 201525	GEN-CXXXXX-REVA	51 of 52
2.	the RE's the s the F	Pixel FIFO is loaded with the tate pointer and the LOD cor Pixel FIFO is wide enough to	<ul> <li>barycentric coordinates for rection bits are also placed source four quad's worth of</li> </ul>	r pixel quads by the detailed walker in the Pixel FIF0 barycentrics per cycle	
3.	SEQ arbi left in the	trates between Pixel FIFO an register files for vertices, the	nd Vertex FIFO – when ther Pixel FIFO is selected	re are no vertices pending OR there is	no space
4.	SEQ allo the n state SEQ	cates space in the SP registe umber of GPRs required by t pointer will not allow interpolated da	er file for all the GPRs used the program is stored in a lo ita to be sent to the shader	by the program ocal state register, which is accessed u until space in the register file has beer	using the
5.	SEQ con bandwidt	trols the transfer of interpolat h of 2048 bits/cycle). See int	ted data to the SP register f erpolated data bus diagram	ile over the RE_SP interface (which hat is for details.	as a
6.	SEQ con fetch stat • note • the c • all ot	structs a control packet for the machine 0, or TSMO FIFO; that there is a separate set o ontrol packet contains the sta her information (such as qua	ne vector and sends it to the ) of reservation stations/arbite ate pointer, the register file l d address for example) trav	e first reservation station (the FIFO in fi rs/state machines for vertices and for base pointer, and the LOD correction b rels in a separate FIFO	ront of pixels pits
7.	TSM0 ac • TSM	cepts the control packet and 0 was first selected by the TS	fetches the instructions for SM arbiter before it could st	fetch clause 0 from the global instructi art	ion store
8.	all instruc	ctions of fetch clause 0 are is	sued by TSM0		
9.	<ul> <li>FIFO)</li> <li>TSM</li> <li>index</li> <li>once asso ahea</li> </ul>	0 does not wait for fetch requ to the fetch data to the TU, the TU has written all the da ciated with the ASM0 FIFO; a d and pop the FIFO and star	uests made to the Fetch Uni which will write the data to ta for a particular clause to a count greater than zero in t to execute the ALU clause	it to complete; it passes the register file the RF as it is received the register files, it increments a count dicates that the ALU state machine ca	e write ter that is n go
10.	ASM0 ac clause 0	cepts the control packet (afte from the global instruction st	er being selected by the AS ore	M arbiter) and gets the instructions for	ALU
11.	all instruction (the station (the station state) and the station (the state) and the state of th	ctions of ALU clause 0 are iss he FIFO in front of fetch state	sued by ASM0, then the cor e machine 1, or TSM1 FIFO	ntrol packet is passed to the next reser )	vation
12.	<ul> <li>the contr</li> <li>pixel</li> <li>i</li> <li>the contr</li> <li>the contr</li> <li>the contr</li> </ul>	ol packet continues to travel data is exported in the last A it is sent to an output FIFO w he ASM arbiter will prevent a	down the path of reservatio LU clause (clause 7) there it will be picked up by packet from starting on AS	n stations until all clauses have been e the render backend M7 if the output FIFO is full	executed
15.	shader p	rogram		GERS SO that they can be used by a	lother
27	<del>.1.3</del> 28	.1.3_Notes			4-
14.	The state threads o	e machines and arbiters will c or stall.	operate ahead of time so the	at they will be able to immediately start	the real
15.	The regis instructio pointer is	ster file base pointer for a vec n store base pointer does no only different for each state	ctor needs to travel with the t – this is because the RF p and thus can be accessed	vector through the reservation stations pointer is different for all threads, but th via the state pointer.	s, but the le IS
28	<del>.</del> 29. O	pen issues			*
Nee stat	ed to do s ic).	ome testing on the size of th	ne register file as well as or	n the register file allocation method (d	ynamic VS
ļ	Exhibit 2026.dod	R400_Sequencer.dec 75288 Bytes*** ©	) ATI Confidential. Refere	nce Copyright Notice on Cover Pag	e © ***

ATI Ex. 2107 IPR2023-00922 Page 149 of 260



	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	7
	24 September, 2001	4 September, 201525		52 of 52	
Saving power?	)				—

Parameter caches in SX?

Using both IJ buffers for center + centroid interpolation?

Exhibit 2026.doc R400\_Sequences.doc 75288 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257284

ATI Ex. 2107 IPR2023-00922 Page 150 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	1 of 52
Author:	Laurent Lefebvre			•
Issue To:		Copy No:		
		I		
	R400 S	equencer Spe	ecification	
		SQ		
		Version 1.1 <u>1</u> 9		
Overview: This req blo	s is an architectural specific luired capabilities and expe cks, and provides internal st	cation for the R400 Sequenc cted uses of the block. It a ate diagrams.	cer block (SEQ). It provides an ov lso describes the block interfaces,	erview of the internal sub-
AUTOMATIC Document L Current Intra	ALLY UPDATED FIELDS: ocation: C:\pa anet Search Title: R400	erforce\r400\doc_lib\design\b D Sequencer Specification APPROVALS	locks\sq\R400_Sequencer.doc	
	Name/Dept		Signature/Date	
Remarks:				
THIS DOO SUBSTA	CUMENT CONTAIN NTIALLY DETRIME INC. THROUGH U	S CONFIDENTIAL II NTAL TO THE INTE JNAUTHORIZED US	NFORMATION THAT CO REST OF ATI TECHNOL E OR DISCLOSURE.	ULD BE .OGIES
"Copyright 200 work created ir unpublished w confidential, pro transmitted in a	1, ATI Technologies Inc. Al 2001. The use of this cop ork. The copyright notice oprietary information and tr. ny form or by any means wi	I rights reserved. The mater yright notice is intended to is not an admission that ade secrets of ATI. No part thout the prior written permis	rial in this document constitutes ar provide notice that ATI owns a cop publication has occurred. This w of this document may be used, re sion of ATI Technologies Inc."	unpublished oyright in this ork contains oproduced, or
Exhibit 2027.doc	:R400_Sequencer.doc 68205 Bytes*** 🤅	ATI Confidential. Referen	ice Copyright Notice on Cover Pa	ge © ***

# ATI 2027 LG v. ATI IPR2015-00325

AMD1044\_0257285

ATI Ex. 2107 IPR2023-00922 Page 151 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519 April 200225 March		2 of 52

# Table Of Contents

1.	OVERVIEW	86
1.1	Top Level Block Diagram	108
1.2	Data Flow graph (SP)	<u> 1240</u>
1.3	Control Graph	<u> 1311</u>
<u>2.</u>	INTERPOLATED DATA BUS	<u> 1311</u>
<u>3.</u>	INSTRUCTION STORE	1614
<u>4.</u>	SEQUENCER INSTRUCTIONS	1814
<u>5.</u>	CONSTANT STORES	1814
5.1	Memory organizations	1814
5.2	Management of the Control Flow Constants	1815
5.3	Management of the re-mapping tables	1845
<u>5.</u>	3.1 R400 Constant management	1815
<u>5.</u>	3.2 Proposal for R400LE constant management	<u> 1915</u>
<u>5.</u>	3.3 Dirty bits	21 <u>17</u>
<u>5.</u>	3.4 Free List Block	2117
5.	3.5 De-allocate Block	22 <del>18</del>
5.1	3.6 Operation of Incremental model	22 <del>18</del>
5.4	Constant Store Indexing	22 <del>18</del>
5.5	Real Time Commands	2319
5.6	Constant Waterfalling	2319
6.	LOOPING AND BRANCHES	2420
<u>6.1</u>	The controlling state	2420
<u>6.2</u>	The Control Flow Program	2420
<u>6.3</u>	Data dependant predicate instructions	2622
<u>6.4</u>	HW Detection of PV,PS	<u>2723</u>
<u>6.5</u>	Register file indexing	2723
<u>6.6</u>	Predicated Instruction support for Texture clauses	<u> 27<del>23</del></u>
<u>6.7</u>	Debugging the Shaders	2723
<u>6.</u> ′	7.1 Method 1: Debugging registers	28 <del>23</del>
<u>6.</u> ′	7.2 Method 2: Exporting the values in the GPRs (12)	2824
7.	PIXEL KILL MASK	2824
<u>8.</u>	MULTIPASS VERTEX SHADERS (HOS)	<u> 2924</u>
9	REGISTER FILE ALLOCATION	2924
<u>10.</u>	FETCH ARBITRATION	3026
<u>11.</u>	ALU ARBITRATION	<u>3026</u>
<u>12.</u>	HANDLING STALLS	3127
<u>13.</u>	CONTENT OF THE RESERVATION STATION FIFOS	<u> 3127</u>
<u>14.</u>	THE OUTPUT FILE	3127
<u>15.</u>	IJ FORMAT	3127
<u>15.1</u>	Interpolation of constant attributes	3228
<u>16.</u>	STAGING REGISTERS	3228
<u>17.</u>	THE PARAMETER CACHE	3430
<u>18.</u>	VERTEX POSITION EXPORTING	3530

Exhibit 2027. doc R400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257286

ATI Ex. 2107 IPR2023-00922 Page 152 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	3 of 52
9. EXPO	RTING ARBITRATION	April 200225 March		
0. EXPO	RTING RULES			3530
0.1 Para	ameter caches exports			3530
0.2 Men	nory exports			
).3 Pos	ition exports			
<u>1. EXPO</u>	RT TYPES	***************************************	********	3530
1.1 Vert	ex Shading			<u></u>
1.2 Pixe	I Shading			
<u>2. SPEC</u>	IAL INTERPOLATION	MODES		
2.1 Rea	I time commands			
2.2 Spri	tes/ XY screen coordin	ates/ FB information		
$\frac{1.5}{200} = \frac{1}{2}$	generaled counters			
22.3.1	Vertex shaders			
<u>22.3.2</u>	Pixel shaders			
<u>. STAT</u>	E MANAGEMENT	****************		<u>3733</u>
3.1 Para	ameter cache synchron	ization		<u>37<del>33</del></u>
<u>I. XY AI</u>	DDRESS IMPORTS	************************************	*******	
1.1 Vert	ex indexes imports			
5. REGI	STERS	*****************	****	
5.1 Con	trol			
2 Con				
DEBU	IG REGISTERS	* * * * * * * * * * * * * * * * * * * *	**********	
5.1 Con	text			
	<u>KFAUES</u>		**********	
7.1 <b>Ext</b>	ernal Interfaces			
7.2 SC 1	to SP Interfaces			
<u>27.2.1</u>	<u>SC SP#</u>			
27.2.2	SC SQ			
27.2.3	SQ to SX: Interpolato	r bus		
27.2.4	SQ to SP: Staging Re	gister Data		4237
27.2.5	VGT to SQ : Vertex ir	terface		42 <del>38</del>
27.2.6	SQ to SX: Control bu	S		
27.2.7	SX to SQ : Output file	control		4641
27.2.8	SQ to TP: Control bus	\$		
27.2.9	TP to SQ: Texture sta	a 11		
27.2.10	SQ to SP: Texture sta	all		
27.2.11	SQ to SP: GPR and a	auto counter		4742
27.2.12	SQ to SPx: Instruction	ns		
27.2.13	SP to SQ: Constant a	ddress load/ Predicate	Set	4843
27.2.14	SQ to SPx: constant l	proadcast		4944
27 2 15	SP0 to SQ: Kill vector	load		4944
27.2.16	SO to CP: RBBM bue			<u>лолл</u>
<u>/////////////////////////////////////</u>	SQ IU CF. RBBM Dus		O D D	

Exhibit 2027.doc R400\_Sequences.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257287

ATI Ex. 2107 IPR2023-00922 Page 153 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 201519 April 200225 March		4 of 52
27.2.17	CP to SQ: RBBM bu	IS		
27.2.18	SQ to CP: State rep	ort		
8. OPEN	ISSUES	* * * * * * * * * * * * * * * * * * * *		
OVER	VIEW			6
1 Top Le	vel Block Diagram			8
.2 Data F	low graph (SP)			
.3 Contro	I Graph			
. INTER	POLATED DATA BI	JS		
INSTR	UCTION STORE	*********		
. SEQU	ENCER INSTRUCTION	ONS	* * * * * * * * * * * * * * * * * * * *	
CONS	TANT STORES	*******		
.1 Memor	y organizations	*********	****	
.2 Manag	ement of the Control	Flow Constants		
3 Manag	ement of the re-map	ping tables		
5.3.1-F	400 Constant manad	aement		
532 0	repeal for D4001 E	anatant managaman	+	47
5.3.2	TOPOSal IOF RADULE	эөныана шанауешен	L	
5.3.3-E	)irty bits			
5.3.4 F	ree List Block			
5.3.5	e-allocate Block	****		
5.3.6	Operation of Incremer	ntal-model		
4-Consta	nt Store Indexing			
5 Real T	ime Commands			
6 Consta	nt Waterfalling			
LOOP	ING AND BRĂNCHE	s		
1 The co	ntrolling state			
2 The Co	ontrol Flow Program.			
.3 Data d	ependant predicate i	nstructions		
4 HW De	etection of PV.PS	****		
5 Registe	er file indexing			
6 Predici	ated Instruction supp	ort for Texture clause	3	
7 Debug	ging the Shaders			
671 N	Aethod 1: Debugging	radictore		25
670	Asthead Or Evenentian t		(40)	
0 <del>././</del>	Aethod 2: Exporting ti	he values in the GPR	s (12)	
PIXEL	KILL MASK		******	
MULH	PASS VERIEX SHA	DERS (HOS)	*****	
REGIS	HER FILE ALLOCA	HON	******	
U. FETCH	AKBITKATION	*****	* * * * * * * * * * * * * * * * * * * *	
I. ALU A	KRIIKAHON	*****		
2. HAND	LING STALLS			
3. CONTI	ENT OF THE RESER	VATION STATION F	IFOS	
4. THE O	UIPUT FILE	****	****	
5. J FOF	RMAT	****		
5.1 Inter	polation of constant a	ittributes		
6.STAGI	ING REGISTERS	*****	*****	
7 TUED	ARAMETER CACHE	» a		32

Exhibit 2027.doc/R409\_Sequences.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257288

ATI Ex. 2107 IPR2023-00922 Page 154 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	5 of 52
18. VERT	EX POSITION EXPOR	TING	*****	
19. EXPO	RTING ARBITRATION	L		
20. EXPO	RT TYPES	****	*****	
20.1 Vert	ex Shading			
21SPEC		MODES	******	
21.1 Rea	I time commands	110 km ( + + + + + + + + + + + + + + + + + +		
21.2 Sprid	tes/ XY screen coordina	ates/ FB information		
21.3 Auto	generated counters			
21.3.1	Vertex shaders			
21.3.2	Pixel shaders			
22. STATI	E MANAGEMENT	·····		
22.1 Para 23 XV AF	AMELET CACHE SYNCHION	IZation		34 35
23.1 Vert	ex indexes imports	****		35
24. REGIS	STERS	*****	*****	
24.1 Con	trol			
24.2 Con	text			35
25.1 Con	tevt	*****	***************************************	
25.2 Gon	trol			
26INTE	RFACES	*****		
26.1 Ext	ernal Interfaces			
26.1.1	SC to SQ : IJ Contr	ol bus		
26.1.2		r bus	*****	
26.1.3-	SQ to SX: Interpolato	r bus		
26.1.4-	SQ to SP: Staging Re	gister Data		
26.1.5	PA to SQ : Vertex inte	erface		
26.1.6_	-SQ to CP: State repo	r <del>t</del>		
26.1.7	-SQ to SX: Control but	S		
26.1.8_	SX to SQ : Output file	control		
26.1.9_	-SQ to TP: Control bus	S		41
26.1.10	TP to SO: Texture etc	۱۱		42
26.1.10-	<u>SO to SP:</u> Texture etc	sli		/2
26.1.11	SO to SP: CPP and a	uta ocuntor		42 10
20.1.12	OQ to ODin Instruction	1010-00011101	******	
20.1.13	SQ to SPX: Instruction	18		43
<del>20.1.14</del> -	SP to SQ: Constant a	aaress load/ Predicate	-96f	
26.1.15-	SQ to SPx: constant l	proadcast	*****	
26.1.16-	-SP0 to SQ: Kill vector	load		
26.1.17-	-SQ to CP: RBBM bus			
06 1 10	CP to SQ: RBBM bus			
<del>20.1.18</del> -				

AMD1044\_0257289

ATI Ex. 2107 IPR2023-00922 Page 155 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE			
	24 September, 2001	4 September, 201519		6 of 52			
27.1.1 Sequencer Control of a Vector of Vertices							
27.1.2	27.1.2 Sequencer Control of a Vector of Pixels						
27.1.3 Notes							
28. OPEN	HSSUES	*****	*****				

Exhibit 2027.doc R400\_Sequencer.doc 68205 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257290

ATI Ex. 2107 IPR2023-00922 Page 156 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	7 of 52
Revision	Changes:			
Day 0.4 (Laur	(ant Lafabura)	Eirot dro	A	
Date: May 7, 2	2001	First dra		
Day 0.0 (Law		Ohanna		
Date : July 9, 2	2001	SP. Add	led some details in the arbitration sect	ion.
Rev 0.3 (Laure	ent Lefebvre)	Reviewe	ed the Sequencer spec after the mee	ting on
Date : August	6, 2001	August	3, 2001. the dynamic allocation method for r	aniatar
Date : August	24. 2001	file and	an example (written in part by Vic)	of the
Ū	,	flow of p	pixels/vertices in the sequencer.	
Rev 0.5 (Laure	ent Lefebvre)	Added t	iming diagrams (Vic)	
Rev 0.6 (Laure	ent Lefehvre)	Change	d the spec to reflect the new	R400
Date : Septem	iber 24, 2001	architec	ture. Added interfaces.	1000
Rev 0.7 (Laure	ent Lefebvre)	Added	constant store management, inst	ruction
Date : Octobe	r 5, 2001	store m data der	anagement, control flow manageme	nt and
Rev 0.8 (Laure	ent Lefebvre)	Change	d the control flow method to be	more
Date : Octobe	r 8, 2001	flexible.	Also updated the external interfaces.	
Rev 0.9 (Laure	ent Lefebvre)	Incorpoi	rated changes made in the 10/18/01	control
Date . October	117,2001	the co	nditional execute or jump. Added	debua
		register	S.	
Rev 1.0 (Laure	ent Lefebvre)	Refined	interfaces to RB. Added state register	S.
Rev 11 (Laure	ent Lefebvre)	Added	SEQ→SP0 interfaces Changed	delta
Date : Octobe	r 26, 2001	precisio	n. Changed VGT→SP0 interface.	Debug
-		Method	s added.	
Rev 1.2 (Laure Date : Novemi	ent Lefebvre) ber 16, 2001	Interface	es greatly refined. Cleaned up the spe	с.
Rev 1.3 (Laure	ent Lefebvre)	Added t	he different interpolation modes.	
Date : Noveml	ber 26, 2001			
Rev 1.4 (Laure	ent Lefebvre)	Added	the auto incrementing counters. Ch	nanged
Date . Decemi	ber 6, 2001	manade	ment. Updated GPRs.	nstant
Rev 1.5 (Laure	ent Lefebvre)	Remove	ed from the spec all interfaces that v	weren't
Date : Decemi	ber 11, 2001	directly	tied to the SQ. Added explanation	ns on
		constan synchro	nization fields and explanation	A→SQ
Rev 1.6 (Laure	ent Lefebvre)	Added i	more details on the staging register.	Added
Date : January	y 7, 2002	detail a	bout the parameter caches. Chang	ed the
		call ins Added	truction to a Conditionnal_call instr details on constant managemen	uction. t and
		updated	the diagram.	
Rev 1.7 (Laure	ent Lefebvre)	Added	Real Time parameter control in the	he SX
Date : Februar	ry 4, 2002	interface New jet	e. Updated the control flow section.	and of
Date : March 4	4. 2002	clause	modifier, removed the end of	clause
	·, <del></del>	instructi	ons.	
Rev 1.9 (Laure	ent Lefebvre)	Rearang	gement of the CF instruction bits in o	rder to
Date : March 1	18, 2002 (rent Lefebyre)	ensure l	byte alignement.	ion on
Date : March 2	25, 2002	exportin	g rules.	
Rev 1.11 (Lau	rent Lefebvre)	Added (	CP state report interface. Last version	of the
Date :		spec wit	th the old control flow scheme	
Exhibit 2027.doc	1400_Sequencer.dec 68205 Bytes*** ©	ATI Confidential. Refere	nce Copyright Notice on Cover Pag	e © ***

ATI Ex. 2107 IPR2023-00922 Page 157 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		8 of 52

# 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 2027.doc R409\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257292

ATI Ex. 2107 IPR2023-00922 Page 158 of 260

# PROTECTIVE ORDER MATERIAL



Exhibit 2027 docR400\_Sequencer.doc 68205 Bytes\*\*\* @ ATI Confidential. Reference Copyright Notice on Cover Page @ \*\*\*

AMD1044\_0257293

ATI Ex. 2107 IPR2023-00922 Page 159 of 260



# 1.1 Top Level Block Diagram



#### Figure 2: Reservation stations and arbiters

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

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

Exhibit 2027.docR409\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257294

ATI Ex. 2107 IPR2023-00922 Page 160 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	11 of 52

On receipt of a packet, the input state machine (not pictured but just before the first FIFO) allocated enough space in the GPRs to store the interpolated values and temporaries. Following this, the barycentric coordinates (and XY screen position if needed) are sent to the interpolator, which will use them to interpolate the parameters and place the results into the GPRs. Then, the input state machine stacks the packet in the first FIFO.

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

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

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

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

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

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

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

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

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

Exhibit 2027.doc R400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257295

ATI Ex. 2107 IPR2023-00922 Page 161 of 260



ATI Ex. 2107 IPR2023-00922 Page 162 of 260



# 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 2027.docR400\_Sequencer.doc 68205 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257297

ATI Ex. 2107 IPR2023-00922 Page 163 of 260



ATI Ex. 2107 IPR2023-00922 Page 164 of 260 PROTECTIVE ORDER MATERIAL

		r23					5 48 <	55 <	20 - C	2 2 2 2	
		122					32-7	သ မို ဗို	> 4 <sup>4</sup> €	> 44 47 6	X
		T21 -					2 <sup>1</sup> 6 − 2	53 c <	24- 27	31 <sup>2</sup> 8 <	5
		T20					> <sup>0</sup> -3	> <sup>4</sup> -7	, 2 ° ∠	15 <sup>-</sup> 7 <	
		T19							ů	ш	
		T18								å	
щ	52	T17			≿ ଘ	×Ξ		ວິ	5	S	
PA(	15 of	T16			ы Ш	ш				BO	2
		T15			В	Ш Ц	5	D2			Ω
NUM.	NA	T14	≳Б	≿8		≿8	ប	5	C5		
REV. I	XX-RE	T13	D	D2		8	B1				
IENT-	XXX	T12	Б	<b>D2</b>		ß	AO	A1	A2		
ocuN	GEN-(	T11		≿2	Χü	000			В	Ξ	
ā		T10	i.	5	C5	ŭ	2			8	
	9	19 1		5	C5			ວິ	5	S	
TE	2015:	Т8	≿ ເິ	≿ ୪	צַט	≳ છ				BO	Carriera
DIT D/	ember	1	S	8	5	8	D	D2			ف
	Septe	T6	ទ	ပိ	5	8	ប	5	C5		
	41	T5	×₽			∑ 8	B1				
ATE	2001	Т4	<u>B</u>			BO	AO	A1	A2		
ATE D	mber,	13	ñ			BO	X 48- 51	X 52- 55	56- 56-	60 × 33 60 ×	
SIGIN	Septe	Т2	×δ	۶¥	≿		X 32- 35	39 % ×	∑ \$ \$	XY 44- 74	$\succ$
ō	24	Ţ	AO	A1	A2		16 <sup>†</sup> X	% % %	24- 27	33 28 ×	X
PC		10 T	Ao	A1	A2		×°¦	× × -4	≍ ∾ ≍	13 <sup>1</sup> 7 ×	
	र		d o	₽ C	g ∽	ი წ	С o	С –	S N	ია	

Figure 6: Interpolation timing diagram

AMD1044\_0257299

ATI Ex. 2107 IPR2023-00922 Page 165 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	1
24 September, 2001	4 September, 201519		16 of 52	

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

## 3. Instruction Store

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

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

The instruction store is loaded by the CP thru the register mapped registers.

The next picture shows the various modes the CP can load the memory. The Sequencer has to keep track of the loading modes in order to wrap around the correct boundaries. The wrap-around points are arbitrary and they are specified in the VS\_BASE and PIX\_BASE control registers. The VS\_BASE and PS\_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

Exhibit 2027.docR400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257300

ATI Ex. 2107 IPR2023-00922 Page 166 of 260





Exhibit 2027 doc R400\_Sequences 62205 Bytes 444 © ATI Confidential. Reference Copyright Notice on Cover Page © +++

AMD1044\_0257301

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		18 of 52

# 5.4. Sequencer Instructions

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

# 6.5. Constant Stores

# 6.15.1 Memory organizations

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

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

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

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

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

# 6.25.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF\_WR\_BASE). On the read side, one level of indirection is used. A register (SQ\_CONTEXT\_MISC.CF\_RD\_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

6.35.3 Management of the re-mapping tables

# 6.3.15.3.1 R400 Constant management

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

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

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space

Exhibit 2027.docR409\_Sequencer.dec 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257302

ATI Ex. 2107 IPR2023-00922 Page 168 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	19 of 52

is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32\*2+32 = 96 entries and above.

## 6.3.25.3.2 Proposal for R400LE constant management

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ\_IDLE and PA\_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation mechanismFigure 9: De-allocation mechanism

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).

Exhibit 2027.docR400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257303

ATI Ex. 2107 IPR2023-00922 Page 169 of 260



ATI Ex. 2107 IPR2023-00922 Page 170 of 260



Figure 89: De-allocation mechanism for R400LE

## 6.3.35.3.3 Dirty bits

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

#### 6.3.45.3.4 Free List Block

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

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

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

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

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

Exhibit 2027.doc R409\_Sequencer.dec 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

#### AMD1044\_0257305

ATI Ex. 2107 IPR2023-00922 Page 171 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		22 of 52

6.3.55.3.5 De-allocate Block

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

## 6.3.65.3.6 Operation of Incremental model

The basic operation of the model would start with the write\_ptr, stop\_ptr, read\_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter because its not at the max value. The data will be written into physical address zero. Both the additional copy of the renaming table and the context zeros of the big renaming table will be updated for the logical address that was written by set start with physical address of 0. This process will be repeated for any logical address that are not dirty until the context changes. If a logical address is hit that has its dirty bits set while in the same context, both dirty bits would be set, so the new data will be over-written to the last physical address assigned for this logical address. When the first draw command of the context is detected, the previous context 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.

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

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

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

# 6.45.4 Constant Store Indexing

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

Exhibit 2027.docR400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257306

ATI Ex. 2107 IPR2023-00922 Page 172 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	23 of 52

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

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

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

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

## 6.55.5 Real Time Commands

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

# 6.65.6 Constant Waterfalling

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



Figure 240: The instruction store

Exhibit 2027.doc/R400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257307

ATI Ex. 2107 IPR2023-00922 Page 173 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	7
24 September, 2001	4 September, 201519		24 of 52	

7.6. Looping and Branches

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

# 7.16.1 The controlling state.

The R400 controling state consists of:

Boolean[256:0] Loop\_count[7:0][31:0] Loop\_Start[7:0][31:0] Loop\_Step[7:0][31:0]

That is 256 Booleans and 32 loops.

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

This state is available on a per shader program basis.

## 7.26.2 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 alu[7:0][7:0] Pixel\_shader\_fetch[7:0][7:0] Pixel\_shader\_alu[7:0][7:0]

Vertex\_shader\_fetch[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 // 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 nine basic instructions:

Execute Conditional\_execute Conditional\_Execute\_Predicates Conditional jump Conditionnal\_Call Return Loop\_start Loop\_end NOP

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

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

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

Exhibit 2027.doc R400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257308

ATI Ex. 2107 IPR2023-00922 Page 174 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201519 April, 200225 March	GEN-CXXXXX-REVA	25 of 52

Conditional\_execute\_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition. Conditional\_jumps jumps to an address if the condition is met. NOP is a regular NOP

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

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

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

A value of 1 in the Addressing means that the address specified in the Exec Address field (or in the jump address field) is an ABSOLUTE address. If the addressing field is cleared (should be the default) then the address is relative to the base of the current shader program.

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

Execute							
47	46 42	41	40 24	23 12	11 0		
Addressing	00001	Last	RESERVED	Instruction	Exec Address		
				count			

Execute up to 4k instructions at the specified address in the instruction memory. If Last is set, this is the last group of instructions of the clause.

NOP						
47	46 42	41	40 0			
Addressing	00010	Last	RESERVED			

This is a regular NOP. If Last is set, this is the last instruction of the clause.

Conditional_Execute								
47	46 42	41	40	39 32	31	30 24	23 12	11 0
Addressing	00011	Last	RESERVED	Boolean	Condition	RESERVED	Instruction	Exec
				address			count	Address

If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions). If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

Conditional_Execute_Predicates								
47	46 42	41	40 34	33 32	31	30 24	23 12	11 0
Addressing	00100	Last	RESERVED	Predicate	Condition	RESERVED	Instruction	Exec Address
				vector			count	

Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.

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

Exhibit 2027.doc R400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257309

ATI Ex. 2107 IPR2023-00922 Page 175 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		26 of 52

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
	00110	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 34	33 32	31	30 12	11 0			
	00111	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           46 42         41 0           01000         RESERVED		
47	46 42	41 0	1
	01000	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	39 32	31	30	29 12	11 0
	01001	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.

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

## 7.36.3 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

Exhibit 2027.docR400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257310

ATI Ex. 2107 IPR2023-00922 Page 176 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	27 of 52

exposed) and use it to control the write masking. This predicate is not maintained across clause boundaries. The # sign is used to specify which predicate set you want to use 0 thru 3.

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

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

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

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

## 7.46.4 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.

## 7.56.5 Register file indexing

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

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

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

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

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

We loop until loop\_iterator = loop\_count. Loop\_step is a signed value [-128...127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the "indexing logic" so that it knows when the provided index is out of range and thus can make the necessary arrangements.

## 7.66.6 Predicated Instruction support for Texture clauses

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

# 7.76.7 Debugging the Shaders

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

Exhibit 2027.doc R400\_Sequencer.dec 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257311

ATI Ex. 2107 IPR2023-00922 Page 177 of 260

_							
	AA	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE		
	<u>vy</u>	24 September, 2001	4 September, 201519 April 200225 March		28 of 52		
	7.7.1 <u>6.7.1</u>	_Method 1: Debug	iging registers				
•	Current plans are to expose 2 debugging, or error notification, registers: 1. address register where the first error occurred 2. count of the number of errors						
	The sequencer will detect the following groups of errors: - count overflow - constant indexing overflow - register indexing overflow						
	Compiler reco - jump errors relativ - call stack call w return	gnizable errors: e jump address > size of t ith stack full with stack empty	he control flow program				
	A jump error v allowing furthe	vill always cause the progr er clauses to be executed.	ram to break. In this case	, a break means that a clause will halt	execution, but		
	With all the ot the DB_PROE	her errors, program can c 3_BREAK register is set.	ontinue to run, potentially	y to worst-case limits. The program wi	ll only break if		
	If indexing out the value with register (or co	tside of the constant or the an index of 0. This coul nstant) for errors.	e register range, causing a d be exploited to genera	an overflow error, the hardware is spe te error tokens, by reserving and initia	cified to return alizing the 0th		
	{ISSUE : Inter	rupt to the driver or not?}					
	7.7.26.7.2	2_Method 2: Export	ting the values in t	he 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 :						
	Under the deb vectors and the the sequences	bug mode (debug kill OR d nat all other exports to the r (even if they occur before	lebug Addr + count), it is SX block (position, color a the address stated by th	assumed that clause 7 is always expor , z, ect) will been turned off (changed e ADDR debug register).	rting 12 debug into NOPs) by		
	8 <u>-7. Pixe</u>	l Kill Mask					
	A vector of 64 requests and	t bits is kept by the seque allow the shader pipe to ki	encer per group of pixels/ Il pixels using the followin	/vertices. Its purpose is to optimize the g instructions:	e texture fetch		
	MASH MASH MASH MASH	(_SETE (_SETNE (_SETGT (_SETGTE					
	Exhibit 2027.docF	2400_Sequencer.doc 68205 Bytes***	© ATI Confidential. Ref	ference Copyright Notice on Cover F	age © ***		

ATI Ex. 2107 IPR2023-00922 Page 178 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	]		
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	29 of 52			
9.8. Multipass vertex shaders (HOS)							

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

## 10.9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX\_REG\_SIZE for vertices and PIXEL\_REG\_SIZE for pixels.

Exhibit 2027. doc R400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257313

ATI Ex. 2107 IPR2023-00922 Page 179 of 260



Above is an example of how the algorithm works. Vertices come in from top to bottom; pixels come in from bottom to top. Vertices are in orange and pixels in green. The blue line is the tail of the vertices and the green line is the tail of the pixels. Thus anything between the two lines is shared. When pixels meets vertices the line turns white and the boundary is static until both vertices and pixels share the same "unallocated bubble". Then the boundary is allowed to move again. The numbering of the GPRs starts from the bottom of the picture at index 0 and goes up to the top at index 127.

# 11.10. Fetch Arbitration

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

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

# 12.11. ALU Arbitration

ALU arbitration proceeds in almost the same way than fetch arbitration. The ALU arbitration logic chooses one of the 8 potentially pending ALU clauses to be executed. The choice is made by looking at the fifos from 7 to 0 and picking the first one ready to execute. There are two ALU 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 2027.docR400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257314

ATI Ex. 2107 IPR2023-00922 Page 180 of 260
ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	31 of 52	

# 13.12. Handling Stalls

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

# 14-13. Content of the reservation station FIFOs

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

## 15.14. The Output File

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

### 16.15. IJ Format

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

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

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

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

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

Exhibit 2027.docR409\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257315

ATI Ex. 2107 IPR2023-00922 Page 181 of 260

Añ	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September, 2001	4 September, 201519		32 of 52

Adds: 8

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

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

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

All numbers are kept using the un-normalized floating point convention: if exponent is different than 0 the number is normalized if not, then the number is un-normalized. The maximum range for the IJs (Full precision) is +/- 63 and the range for the Deltas is +/- 127.

16.115.1 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:

normal interpolation

}

### 17.16. Staging Registers

In order for the reuse of the vertices to be 14, the sequencer will have to re-order the data sent IN ORDER by the VGT for it to be aligned with the parameter cache memory arrangement. Given the following group of vertices sent by the VGT:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 || 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 || 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 || 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

The sequencer will re-arrange them in this fashion:

0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 || 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 || 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 || 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63

The || markers show the SP divisions. In the event a shader pipe is broken, the VGT will send padding to account for the missing pipe. For example, if SP1 is broken, vertices 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 will still be sent by the VGT to the SQ **BUT** will not be processed by the SP and thus should be considered invalid (by the SU and VGT).

Exhibit 2027. doc R409\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257316

ATI Ex. 2107 IPR2023-00922 Page 182 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	33 of 52

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 <u>11Figure 12Figure 12</u>. 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 <u>10Figure 14.</u>Figure <u>11</u>.

Basis for 8-deep Latch Memory (fror	n R300)		
8x24-bit	11631	$\mu^2$	$60.57813\mu^2\text{per}$ bit
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 1011:Area Estimate for VGT to Shader Interface

Exhibit 2027. doc R400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257317

ATI Ex. 2107 IPR2023-00922 Page 183 of 260



number field wraps around, the PA increments the Current\_Location by VS\_EXPORT\_COUNT\_7 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS\_EXPORT\_COUNT\_7 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 0010000000 and so on up to 1111000000. Then the next position received (the 17<sup>th</sup>) is going to have the address 00000001000, 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 2\*VS\_EXPORT\_COUNT\_7to Current\_Location and reset the memory count to 0 before the next vector begins).

Exhibit 2027.doc.R400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257318

ATI Ex. 2107 IPR2023-00922 Page 184 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	35 of 52	

# 19.18. Vertex position exporting

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

### 20.19. Exporting Arbitration

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

Position exports and position exports cannot be co-issued.

All other types of exports can be co-issued as long as there is place in the receiving buffer.

# 21.20. Exporting Rules

### 21.120.1 Parameter caches exports

We support masking and out of order exports to the parameter caches. So one can export multiple times to the same PC line using different masks.

### 21.220.2 Memory exports

Memory exports don't support masking. However, you can export out of order to memory locations.

### 21.320.3 Position exports

Position exports have to be done IN ORDER and don't support masking.

### 22.21. Export Types

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

## 22.121.1 Vertex Shading

- 16 parameter cache 0:15 16:31<u>31</u> - Empty (Reserved?) Export Address 32 3233:43 40 - 12-8 vertex exports to the frame buffer and index 4441:47 - Empty 48:5955 - 12 8 debug export (interpret as normal vertex export) 60 - export addressing mode 61 - Empty 62 - position

63 - sprite size export that goes with position export (point\_h,point\_w,edgeflag,misc)

# 22.221.2 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

Exhibit 2027.docR400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257319

ATI Ex. 2107 IPR2023-00922 Page 185 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE			
24 September, 2001	4 September, 201519		36 of 52			
8 - Buffer 0 Color.	/Fog (primary)		1			
9 - Buffer 1 Color.	/Fog					
10 - Buffer 2 Color.	/Fog					
11 - Buffer 3 Color	/Fog					
12:15 - Empty						
16:3131 - Empty	(Reserved?)					
32 - Export Addres	SS					
 3233:4340 - 12-8 e	xports for multipass pixel	shaders.				
4441:47 - Empty	,					
48:5955 - 12-8 d	lebug exports (interpret as	normal pixel export)				
60 - export address	sing mode	· · · · · · · · · · · · · · · · · · ·				
61:62 - Empty	ang mode					
62 7 for primory k	uffor (7 overated to 'alpha	a component)				
05 - Z lor primary i	uner (z exported to alpha	a component)				

# 23.22. Special Interpolation modes

# 23.122.1 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 to view them as two banks of 16 and do double buffering allowing one to be loaded, while the other is rasterized with. Most overlay shaders will need 2 or 4 scalar coordinates, one option might be to restrict the memory to 16x64 or 32x64 allowing only two interpolated scalars per cycle, the only problem I see with this is, if we view support for 16 vector-4 interpolants important (true only if we map Microsoft's high priority stream to the realtime stream), then the PA/sequencer need to support a realtime-specific mode where we need to address 32 vectors of parameters instead of 16. This mode is triggered by the primitive type: REAL TIME. The actual memories are in the in the SX blocks. The parameter data memories are hooked on the RBBM bus and are loaded by the CP using register mapped memory.

# 23.222.2 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 Param\_Gen\_I0 enable, snd\_xy enable, gen\_st - I0 = screen x, screen y, s, t

### 23.322.3 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

Exhibit 2027.docR409\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257320

ATI Ex. 2107 IPR2023-00922 Page 186 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201519 April 200225 March	GEN-CXXXXX-REVA	37 of 52

GPRs the counter is incremented. Every time a state change is detected, the corresponding counter is reset. While there is only one count broadcast to the GPRs, the LSB are hardwired to specific values making the index different for all elements in the vector.

### 23.3.122.3.1 Vertex shaders

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

### 23.3.222.3.2 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).



Figure 1213: GPR input mux Control

### 24.23. State management

Every clock, the sequencer will report to the CP the oldest states still in the pipe. These are the states of the programs as they enter the last ALU clause.

## 24.123.1 Parameter cache synchronization

In order for the sequencer not to begin a group of pixels before the associated group of vertices has finished, the sequencer will keep a 6 bit count per state (for a total of 8 counters). These counters are initialized to 0 and every time a vertex shader exports its data TO THE PARAMETER CACHE, the corresponding pointer is incremented. When the SC sends a new vector of pixels with the SC\_SQ\_new\_vector bit asserted, the sequencer will first check if the count is greater than 0 before accepting the transmission (it will in fact accept the transmission but then lower its ready to receive). Then the sequencer waits for the count to go to one and decrements it. The sequencer can then issue the group of pixels to the interpolators. Every time the state changes, the new state counter is initialized to 0.

## 25.24. XY Address imports

The SC will be able to send the XY addresses to the GPRs. It does so by interleaving the writes of the IJs (to the IJ buffer) with XY writes (to the XY buffer). Then when writing the data to the GPRs, the sequencer is going to

Exhibit 2027.doc R409\_Sequencer.dec 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257321

ATI Ex. 2107 IPR2023-00922 Page 187 of 260

					ę	
	AR	ORIGINATE	DATE	EDIT DATE	R400 Sequencer Specification	PAGE
		24 Septembe	er, 2001	4 September, 201519		38 of 52
	interpolate the the GPRs. Th section 22.2 fc 25.124.1	IJ data or pass e Xys are curre or details on how Vertex inc	the XY da ently SCR w to contro	ta thru a Fix→float conve EEN SPACE COORDINA the interpolation in this n mports	erter and expander and write the conve ATES. The values in the XY buffers v node.	erted values to will wrap. See
I	In order to imp	ort vertex index	xes, we ha	ve 16 8x96 staging regist	ers. These are loaded one line at a tin	ne by the VGT
	block (96 bits)	. They are loade	ed in floatir	ng point format and can be	e transferred in 4 or 8 clocks to the GP	PRs.
	<del>26.1</del> 25.1	Control				
	REG_DYN REG_SIZI	IAMIC E_PIX	Dynami Size of on)	c allocation (pixel/vertex) the register file's pixel por	of the register file on or off. tion (minimal size when dynamic alloc	ation turned
	REG_SIZI	E_VTX	Size of on)	the register file's vertex po	ortion (minimal size when dynamic allo	ocation turned
	ARBITRA INST_STO	TION_POLICY	policy o interlea start no	f the arbitration between v ved, separate int for the vertex instruction	vertexes and pixels	se and
	INST PAG		Begins	at 0) int for the pixel shader inc	struction store	
	ONE_THE	READ	debug s debug s	state register. Only allows state register. Only allows	one program at a time into the GPRs one ALU program at a time to be exe	cuted (instead
	INSTRUC	TION	This is	where the CP puts the ba	ase address of the instruction writes a	nd type (auto-
	CONSTAN CONSTAN CONSTAN TSTATE_ EXPORT_	ITS ITS_RT IT_EO_RT EO_RT LATE	TSTATI Control control control	Inted on reads/writes) Re; LU constants + 32*6 Text LU constants + 32*6 text the size of the space ress ANT_EO_RT). The re-map the size of the space ress E_EO_RT). The re-map is whether or not we are occur at clause 7.	gister mapped ture state 32 bits registers (logically m ure states? (physically mapped) served for real time in the constant st apping table operates on the rest of the erved for real time in the fetch state si ing table operates on the rest of the me e exporting position from clause 3. I	apped) ore (from 0 to e memory tore (from 0 to emory f set, position
	26.225.2 VS_FETC VS_ALU_ PS_FETC PS_ALU_ PS_BASE VS_CF_S PS_CF_S PS_CF_S PS_SIZE PS_NUM VS_NUM PARAM_S PROVO_V PARAM_V PS_EXPC	Context H_{07} (07} H_{07} (07} (07} IZE IZE REG REG HADE /ERT VRAP	eight 8 eight 8 eight 8 base po base po base po size of 1 size of 1 size of 1 size of 1 size of 1 number One 16 = goura 0 : verte 64 bits: (0=linea 0xxxx : 1xxxx :	bit pointers to the location bit pointers to the location bit pointers to the location bit pointers to the location inter for the pixel shader inter for the vertex shade he vertex shader (# of inst he pixel shader (# of inst he pixel shader (cntl+inst he vertex shader (cntl+inst of GPRs to allocate for v bit register specifying wh ud) xo 0, 1: vertex 1, 2: vertex for which parameters (an ir, 1=cylindrical). Normal mode Multipass mode	n where each clauses control program in the instruction store structions in control program/2) ructions in control program/2) ructions) structions) ixel shader programs ertex shader programs ich parameters are to be gouraud shad 2, 3: Last vertex of the primitive d channels (xyzw)) do we do the cyl w	is located is located is located is located ded (0 = flat, 1 rrapping
I	Exhibit 2027.docR	400_Sequencer.doc 6	68205 Bytes***	© ATI Confidential. Ref	erence Copyright Notice on Cover F	age © ***

ATI Ex. 2107 IPR2023-00922 Page 188 of 260

n b	ORIGINATE		EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September	r, 2001	4 September, 201519	GEN-CXXXXX-REVA	39 of 52
VS_EXPO VS_EXPO _COUNT	DRT_MODE DRT _{06}	If multipa 0: positio	ss 1-12 exports for color. n (1 vector), 1: position (2 vector), 2 p	of interpolated parameters exported in	n clause 7
PARAM_ GEN_INE	GEN_IO DEX	Hocated I # of expo Do we ov Auto gen and R2 fo	n vs_export_count_6, rted vectors to memory per rerwrite or not the parameter erates an address from 0 to or vertex shaders	Clause in multipass mode (per clause r 0 with XY data and generated T and XX. Puts the results into R0-1 for pix	) S values el shaders
CONST_ CONST_ CONST_S CONST_S INST_PR	BASE_VIX (9 bits) BASE_PIX (9 bits) SIZE_PIX (8 bits) SIZE_VTX (8 bits) ED_OPTIMIZE	Logical B Logical E Size of th Size of th Turns on always e	ase address for the constar lase address for the constar le logical constant store for le logical constant store for y the predicate bit optimizatic xecuted).	its of the Vertex shader its of the Pixel shader pixel shaders vertex shaders on (if of, conditional_execute_predicat	es is
CF_BOO CF_LOOI CF_LOOI CF_LOOI	LEANS P_COUNT P_START P_STEP	256 book 32x8 bit o 32x8 bit o 32x8 bit o	ean bits counters (number of times w counters (init value used in i counters (step value used in	re traverse the loop) ndex computation) index computation)	
<u>27-26. D</u>	EBUG Regi	isters			
<del>27.1</del> 26.1	Context				
DB_F DB_F DB_II DB_E DB_C _MOI DB_C _MOI	PROB_ADDR PROB_COUNT PROB_BREAK NST_COUNT BREAK_ADDR DE_ALU_{07} DE_ALU_{07} DE_FETCH_{07	instructio number of break the instructio break ad clause m } of	n address where the first pro of problems encountered due e clause if an error is found. n counter for debug method dress for method number 2 ode for debug method 2 (0: lause mode for debug meth	oblem occurred ring the execution of the program 2 normal, 1: addr, 2: kill) od 2 (0: normal, 1: addr, 2: kill)	
<del>27.2</del> 26.2	Control				
DB_A DB_T	LUCST_MEMSIZ	E S	Size of the physical ALU con Size of the physical texture s	stant memory tate memory	
<del>28.</del> 27. In	terfaces				
<del>28.1</del> 27.1	External Inf	terface	S		
Whenever an named SQ→S	x is used, it mea SPx it means that	ins that th SQ is goin	e bus is broadcast to all ur g to broadcast the same inf	nits of the same name. For example, ormation to all SP instances.	if a bus is
<del>28.2</del> 27.2	SC to SP I	nterfac	ces		
2 <del>8.2.1</del> 27.	<u>2.1_</u> SC_SP#	g je N			
There is one of the I,J data fo these 4 interf data.	of these interfaces r pixel interpolatio aces transmits on	at front o n. For the le half of a	f each of the SP (buffer to st e entire system, two quads p a quad per clock. The inter	age pixel interpolators). This interfac er clock are transferred to the 4 SPs, face below describes a half of a qua	e transmits so each of ad worth of
Exhibit 2027.doc	R400_Sequencer.doc 682	05 Bytes*** C	ATI Confidential. Referen	nce Copyright Notice on Cover Pag	e © ***

ATI Ex. 2107 IPR2023-00922 Page 189 of 260

		ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE		
		24 September, 2001	4 September, 201519		40 of 52		
The actual data which is transferred per guad is							
Ref Pix I => S4.20 Floating Point I value							
Ref Pix J => S4.20 Floating Point J value							
Delta Pix I (x3) => S4.8 Floating Point Delta I value							
Delta Pix J (x3) => S4.8 Floating Point Delta J value							
This equates to a total of 128 bits which transferred over 2 clocks							
and therefor needs an interface 64 bits wide							

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb. Transfers across these interfaces are synchronized with the SC\_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ\_BUF\_INUSE\_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ\_BUF\_INUSE\_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX\_BUFER\_MINUS\_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	64	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit) <b>Type 0 or 1</b> , First clock I, second clk J Field ULC URC LLC LRC Bits [63:39] [38:26] [25:13] [12:0]
		Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	<ul> <li>0 -&gt; Indicates centroids</li> <li>1 -&gt; Indicates centers</li> <li>2 -&gt; Indicates X,Y Data and faceness on data bus</li> <li>The SC shall look at state data to determine how many types to send for the interpolation process.</li> </ul>

The # is included for clarity in the spec and will be replaced with a prefix of u#\_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

### 28.2.227.2.2 SC\_SQ

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 92 bits) could be folded in half to approx 46-47 bits.

Name	Bits	Description
SC_SQ_data	46	Control Data sent to the SQ
		1 clk transfers
		Event – valid data consist of event_id and

Exhibit 2027.docR400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257324

ATI Ex. 2107 IPR2023-00922 Page 190 of 260

	ORIGINATE DAT	TE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 20	001	4 September, 201519	GEN-CXXXXX-REVA	41 of 52
		2	state_id. In event vecto event_id th and onto th making suu gets back 1 follow end vectors will Empty Quad Mask – Tra consisting or new_vector transfer po without any vector will request fifo attached to outstandim, if no valid of 2 clk transfers Quad Data Valid – Send without new New vecto fifo with or pc_dealloc vector unle this case th posted in th Filler quad The Quad correspond zero.	Instruct SQ to post an protosend state id and prough request fifo- ne reservation stations re state id and/or event_id to the CP. Events only of packets so no pixel l be in progress. Insfer Control data of pc_dealloc ctor. Receipt of this is to dealloc or new_vector y valid quad data. New always be posted to to and pc_dealloc will be to any pixel vector g or posted in request fifo- quad data with or w_vector or pc_dealloc. r will be posted to request without a pixel vector and will be posted with a pixel tess none is in progress. In the pc_dealloc will be he request queue. s will be transferred with mask set but the pixel ding pixel mask set to test the set to the set to test to the set to the set to test to the pixel test to	
SC_SQ_valid	1	S	SC sending valid data, 2 <sup>nd</sup> clk	could be all zeroes	
SC_SQ_data	<ul> <li>first clock and second</li> </ul>	nd clo	ock transfers are shown in the	e table below.	

Ditrieiu	DIIS	Description
0	1	This transfer is a 1 clock event vector
		Force quad_mask = new_vector=pc_dealloc=0
[2:1]	2	This field identifies the event
		0 => denotes an End Of State Event
		1 => TBD
[5:3]3	31	Deallocation token for the Parameter Cache
64	1	The SQ must wait for Vertex shader done count > 0
		and after dispatching the Pixel Vector the SQ will
		decrement the count.
[108:75]	4	Quad Write mask left to right SP0 => SP3
119	1	End Of the primitive
[142:120]	3	State/constant pointer (6*3+3)
[3028:153]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
[ <u>33</u> 4: <u>31</u> 29]	3	Stippled line and Real time command need to load tex
		cords from alternate buffer
		000: Normal
		100 <u>010</u> : Realtime
		101: Line AA
	0 [2:1] [5:3]3 <u>6</u> 4 [108:75] 119 [142:120] [3028:153] [3 <u>3</u> 4: <u>31</u> 29]	Dialeta         Dits           0         1           [2:1]         2           [5:3]3         34           64         1           [108:75]         4           119         1           [142:120]         3           [3028:153]         16           [331:3129]         3

Exhibit 2027.doc R400\_Sequences.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257325

ATI Ex. 2107 IPR2023-00922 Page 191 of 260

		ORIGINATE DATE	EDITE	DATE		R400 Sequencer Specification	PAGE
		24 September, 2001	4 Septembe	er, 2015	5 <u>19</u>		42 of 52
Τ					110	: Point AA (Sprite)	
	SC_SQ_provo	<u>k_vtx</u> SC_SQ_pc_ptr	[35:34][42:32]	211	Pro poir	voking vertex for flat shadingPara hter for vertex 0	imeter Cache
	SC SQ pc pt	rO	[46:36]	11	Par	ameter Cache pointer for vertex 0	
	2nd Clock Tra	nsfer					
	SC_SQ_pc_pt	r1	[10:0]	11	Par	ameter Cache pointer for vertex 1	
	SC_SQ_pc_pt	r2	[21:11]	11	Par	ameter Cache pointer for vertex 2	
	SC_SQ_lod_c	orrect	[45:22]	24	LO	D correction per quad (6 bits per quad	)

Name	Bits	Description
SQ_SC_free_buff	1	Pipelined bit that instructs SC to decrement count of buffers in use.
SQ_SC_dec_cntr_cnt	1	Pipelined bit that instructs SC to decrement count of new vector and/or event sent to prevent SC from overflowing SQ interpolator/Reservation request fifo.

The scan converter will submit a partial vector whenever:

1.) He gets a primitive marked with an end of packet signal.

2.) A current pixel vector is being assembled with at least one or more valid quads and the vector has been marked for deallocate when a primitive marked new\_vector arrives. The Scan Converter will submit a partial vector (up to 16quads with zero pixel mask to fill out the vector) prior to submitting the new\_vector marker\primitive.

(This will prevent a hang which can be demonstrated when all primitives in a packet three vectors are culled except for a one quad primitive that gets marked pc\_dealloc (vertices maximum size). In this case two new\_vectors are submitted and processed, but then one valid quad with the pc\_dealloc creates a vector and then the new would wait for another vertex vector to be processed, but the one being waited for could never export until the pc\_dealloc signal made it through and thus the hang.)

### 28.2.327.2.3 SQ to SX: Interpolator bus

Name	Direction	Bits	Description
SQ_SXx_interp_flat_vtx	SQ→SPx	2	Provoking vertex for flat shading
SQ_SXx_interp_flat_gouraud	SQ→SPx	1	Flat or gouraud shading
SQ_SXx_interp_cyl_wrap	SQ→SPx	4	Wich channel needs to be cylindrical wrapped
SQ_SXx_pc_ptr04	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr12	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_pc_ptr23	SQ→SXx	11	Parameter Cache Pointer
SQ_SXx_rt_sel	SQ→SXx	1	Selects between RT and Normal data
SQ_SXx_pc_wr_en	SQ→SXx	1	Write enable for the PC memories
SQ_SXx_pc_wr_addr	SQ→SXx	7	Write address for the PCs
SQ_SXx_pc_channel_mask	SQ→SXx	4	Channel mask

### 28.2.427.2.4 SQ to SP: Staging Register Data

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

Direction	Bits	Description
SQ→SPx	96	Pointers of indexes or HOS surface information
SQ→SPx	1	0: Normal 96 bits per vert 1: double 192 bits per vert
SQ→SP0	1	Data is valid
SQ→SP1	1	Data is valid
SQ→SP2	1	Data is valid
SQ→SP3	1	Data is valid
<u>SQ→SPx</u>	1	Increment the read pointers
	Direction           SQ→SPx           SQ→SPx           SQ→SP1           SQ→SP1           SQ→SP2           SQ→SP3           SQ→SPx	$\begin{array}{c c c c c c c c c c c c c c c c c c c $

28.2.527.2.5 VGT to SQ : Vertex interface

28.2.5.127.2.5.1 Interface Signal Table

The area difference between the two methods is not sufficient to warrant complicating the interface or the state requirements of the VSISRs. <u>Therefore, the POR for this interface is that the VGT will transmit the data to the VSISRs</u> (via the Shader Sequencer) in full, 32-bit floating-point format. The VGT can transmit up to six 32-bit

Exhibit 2027.doc.R400\_Sequences.dec 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257326

ATI Ex. 2107 IPR2023-00922 Page 192 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201519 April, 200225 March	GEN-CXXXXX-REVA	43 of 52
floating points	aluge to each VOICD when	a fair ar maara valuaa raav	vive two transmission cleaks. The det	hun in OC

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

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

28.2.5.227.2.5.2 Interface Diagrams

Exhibit 2027. doc R400\_Sequences.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257327

ATI Ex. 2107 IPR2023-00922 Page 193 of 260

# PROTECTIVE ORDER MATERIAL



Exhibit 2027 dooR400\_Sequencer.doc 62205 Byres\*\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257328

ATI Ex. 2107 IPR2023-00922 Page 194 of 260 PROTECTIVE ORDER MATERIAL



AMD1044\_0257329

				F	1	_
	ORIGINATE DATE	E EDI	T DATE	R400 Sequencer Specification	PAGE	
24	4 September, 200	11 <u>4 Septer</u>	nber, 2015	519	46 of 52	
2726 SO to	CD: State re	- Anni - Do	(YTTE MAR		•	Formatted: Bullets and Numbering
27.2.0-04.0		<del>sport</del>				
<del>27.2.7</del> <u>27.2.6</u>	_SQ to SX: C	Control bus	)			
Name	Dir	ection	Bits	Description		]
SQ_SXx_exp_pix	SQ	≀→SXx	1	1: Pixel		1
				0: Vertex		
SQ_SXx_exp_clau	use SQ	l→SXX	3	Clause number, which is needed for verte	ex clauses	
SQ_SAX_exp_stat		Z→SAX	1			
SO SXy exp vali	_iu idSQ	SXY	1	Valid hit		
			<u>.</u>	valid bit		J
These fields are se	ent every time the	sequencer pic	ks an exp	orting clause for execution.		
27 2 827 2 7	SX to SQ ·	Output file	control		-4	Formatted: Bullets and Numbering
Name		Direction	Bi	ts Description		1
SXx_SQ_exp_cou	int_rdy	SXx→SQ	1	Raised by SX0 to indicate that the follo	owing two	
				fields reflect the result of the most reco	ent export	
SXx_SQ_exp_pos	sition_availepac	SXx→SQ	1	Specifies whether there is room	for another	
SXx SQ exp buf	fer availspace	SXx→SQ	7	Specifies the space available in the ou	utput buffers.	
				0: buffers are full		
				1: 2K-bits available (32-bits for each o	f the 64	
				pixels in a clause)		
				pixels in a clause)  64: 128K-bits available (16 128-bit ent	ries for each	
				pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels)	ries for each	
				pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED	ries for each	
				pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED	ries for each	Esmatted Dillet and Numbering
<del>27.2.9</del> 27.2.8	SQ to TP: 0	Control bus		pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED	ries for each	
27.2.927.2.8	_SQ to TP: C	Control bus	encer on v	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat	ries for each	<b>Formatted:</b> Bullets and Numbering
27.2.927.2.8 Once every clock, is ready or not. Thi	_SQ to TP: C the fetch unit sen is way the sequer	Control bus ds to the seque cer can updat	encer on v	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat o counters for the reservation station fifos. T	ries for each	<b>Formatted:</b> Bullets and Numbering
27.2.9 <u>27.2.8</u> Once every clock, is ready or not. Thi also provides the i	_SQ to TP: C the fetch unit sen is way the sequer instruction and co	Control bus ds to the sequencer can updat nstants for the	encer on v e the fetch fetch to e	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat ocounters for the reservation station fifos. To execute and the address in the register file	ries for each a in the GPRs The sequencer where to write	<b>Formatted:</b> Bullets and Numbering
27.2.9 <u>27.2.8</u> Once every clock, is ready or not. Thi also provides the i the fetch return dat	_SQ to TP: C the fetch unit sen- is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat nstants for the	encer on v e the fetch fetch to e	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat a counters for the reservation station fifos. The execute and the address in the register file	tries for each ta in the GPRs The sequencer where to write	Formatted: Bullets and Numbering
27.2.9 <u>27.2.8</u> Once every clock, is ready or not. Thi also provides the i the fetch return dat Name	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta. Dir	Control bus ds to the sequencer can updat instants for the rection	encer on v e the fetch fetch to e Bits	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat o counters for the reservation station fifos. T execute and the address in the register file Description	ries for each a in the GPRs The sequencer where to write	<b>Formatted:</b> Bullets and Numbering
27.2.9 <u>27.2.8</u> Dnce every clock, s ready or not. Thi also provides the i the fetch return dat Name TPx_SQ_data_rdy	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta. Dir TP	Control bus ds to the sequencer can updat instants for the rection $x \rightarrow SQ$	encer on v e the fetch fetch to e	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat a counters for the reservation station fifos. The execute and the address in the register file Description Data ready	ries for each a in the GPRs The sequencer where to write	<b>Formatted:</b> Bullets and Numbering
27.2.927.2.8 Once every clock, is ready or not. Thi also provides the i the fetch return dat Name TPX_SQ_data_rdy TPX_SQ_clause_nu	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat instants for the rection $x \rightarrow SQ$ $x \rightarrow SQ$	encer on v e the fetch fetch to e Bits 1 3	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat ocunters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number	ries for each a in the GPRs The sequencer where to write	
27.2.927.2.8 Once every clock, is ready or not. Thi also provides the i the fetch return dat <b>Name</b> TPX_SQ_data_rdy TPX_SQ_clause_nu TPX_SQ_type	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat instants for the rection $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$	encer on v e the fetch fetch to e Bits 1 3 1	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat a counters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX)	ries for each a in the GPRs The sequencer where to write	
27.2.927.2.8 Once every clock, is ready or not. Thi also provides the i the fetch return dat <b>Name</b> TPx_SQ_data_rdy TPx_SQ_clause_nu TPx_SQ_type SQ_TPx_send	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat instants for the rection $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$	encer on v e the fetch fetch to e Bits 1 3 1 1	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat o counters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data	ries for each a in the GPRs The sequencer where to write	
27.2.927.2.8 Once every clock, s ready or not. Th also provides the i the fetch return dat TPX_SQ_data_rdy TPX_SQ_clause_nu TPX_SQ_clause_nu TPX_SQ_type SQ_TPX_send SQ_TPX_const	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat instants for the rection x→ SQ x→ SQ x→ SQ t→ TPx t→ TPx	encer on v e the fetch fetch to e Bits 1 3 1 1 1 48	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat a counters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch state sent over 4 clocks (192 bits tota	a in the GPRs The sequencer where to write	<b>Formatted:</b> Bullets and Numbering
27.2.927.2.8 Once every clock, s ready or not. Th also provides the i the fetch return dat Name TPX_SQ_data_rdy TPX_SQ_clause_nu TPX_SQ_type SQ_TPX_send SQ_TPX_send SQ_TPX_instr	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat netants for the rection $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow TPx$ $\mapsto TPx$ $\mapsto TPx$	encer on v e the fetch fetch to e Bits 1 3 1 1 48 24	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat a counters for the reservation station fifos. The execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch state sent over 4 clocks (192 bits tota Fetch instruction sent over 4 clocks	ta in the GPRs The sequencer where to write	Formatted: Bullets and Numbering
27.2.927.2.8 Dnce every clock, s ready or not. Th also provides the i the fetch return dat Name TPX_SQ_data_rdy TPX_SQ_clause_nt TPX_SQ_type SQ_TPX_send SQ_TPX_const SQ_TPX_end_of_cl SQ_TPX_tend	_SQ to TP: C the fetch unit sen- is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat incer can updat nstants for the rection $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow TPx$ $x \rightarrow TPx$ $x \rightarrow TPx$	encer on v ethe fetch fetch to e 1 3 1 1 48 24 1	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat a counters for the reservation station fifos. The execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch state sent over 4 clocks (192 bits total Fetch instruction sent over 4 clocks Last instruction of the clause Three 6 data sent (0:PIXEL, 1:VERTEX)	ta in the GPRs The sequencer where to write	Formatted: Bullets and Numbering
27.2.927.2.8 Drice every clock, s ready or not. Thi also provides the i he fetch return dati TPx_SQ_data_rdy TPx_SQ_clause_nu TPx_SQ_type SQ_TPx_send SQ_TPx_send SQ_TPx_instr SQ_TPx_instr SQ_TPx_instr SQ_TPx_rope	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat nstants for the ection $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$	encer on v e the fetch fetch to e 1 3 1 1 1 48 24 24 1 1 1 2	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat o counters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch instruction sent over 4 clocks Last instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Write, base simples	a in the GPRs The sequencer where to write	Formatted: Bullets and Numbering
27.2.927.2.8 Dnce every clock, s ready or not. Tha also provides the i the fetch return dar TPx_SQ_data_rdy TPx_SQ_clause_nu TPx_SQ_type SQ_TPx_send SQ_TPx_const SQ_TPx_end_of_cl SQ_TPx_gpr_phas. SQ_TPx_gpr_phas. SQ_TPx_ded_core	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta.	Control buss ds to the sequencer can updat nstants for the $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$	encer on v e the fetch fetch to e Bits 1 3 1 1 4 8 24 1 1 1 2 6	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat o counters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch instruction sent over 4 clocks Last instruction sent over 4 clocks Last instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per come 2 comments	a in the GPRs the sequencer where to write	
27.2.927.2.8 Once every clock, s ready or not. Th also provides the i the fetch return dar TPX_SQ_data_rdy TPX_SQ_clause_nu TPX_SQ_clause_nu TPX_SQ_type SQ_TPX_send SQ_TPX_const SQ_TPX_end_of_cl SQ_TPX_end_of_cl SQ_TPX_end_of_cl SQ_TPX_end_of_cl SQ_TPX_gpr_phase SQ_TPA_gpr_phase SQ_TPO_lod_corres SQ_TPO_lod_corres	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat nstants for the $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$	encer on v e the fetch fetch to e Bits 1 3 1 1 48 24 1 1 1 2 6 6	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat ocunters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch state sent over 4 clocks (192 bits tota Fetch instruction sent over 4 clocks Last instruction sent over 4 clocks Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components Pixel mask 1 bit per rivel	a in the GPRs The sequencer where to write	Formatted: Bullets and Numbering
27.2.927.2.8 Once every clock, s ready or not. Th also provides the i the fetch return dar <b>Name</b> TPX_SQ_data_rdy TPX_SQ_clause_nu TPX_SQ_type SQ_TPX_send SQ_TPX_send SQ_TPX_send SQ_TPX_end_of_cl SQ_TPA_gpr_phas SQ_TP0_lod_corre SQ_TP0_lod_corre SQ_TP0_lod_corre	_SQ to TP: C the fetch unit sen- is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat nstants for the rection x→ SQ x→ SQ t→ TPx t→ TPx t→ TPx t→ TPx t→ TPx t→ TPx t→ TP0 t→ TP1	encer on v e the fetch fetch to e Bits 1 3 1 1 48 24 1 1 1 2 6 4 6 4 6	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat o counters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch state sent over 4 clocks (192 bits tota Fetch instruction sent over 4 clocks Last instruction sent over 4 clocks Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components Pixel mask 1 bits per comp 2 components	ries for each	Formatted: Bullets and Numbering
27.2.927.2.8 Once every clock, s ready or not. Th also provides the i the fetch return dar TPx_SQ_data_rdy TPx_SQ_clause_nu TPx_SQ_type SQ_TPx_send SQ_TPx_send SQ_TPx_const SQ_TPx_end_of_cl SQ_TPx_upr_bas SQ_TP0_lod_corre SQ_TP1_lod_corre SQ_TP1_lod_corre SQ_TP1_lod_corre SQ_TP1_lod_corre	SQ to TP: C the fetch unit sen- is way the sequer instruction and co ta. TP um TP: TP um TP: SQ S	Control bus ds to the sequencer can updat incer can updat nstants for the rection $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow TPx$ $a \rightarrow TP1$	encer on v ethe fetch fetch to e Bits 1 1 3 3 1 1 48 24 1 1 1 2 2 6 6 4 6 4	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat a counters for the reservation station fifos. The execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch state sent over 4 clocks (192 bits tota Fetch instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel LOD cornects bits per comp 2 components Pixel mask 1 bit per pixel	ries for each a in the GPRs for esequencer where to write	Formatted: Bullets and Numbering
27.2.927.2.8 Once every clock, s ready or not. Th also provides the i the fetch return dat TPx_SQ_data_rdy TPx_SQ_clause_nu TPx_SQ_clause_nu TPx_SQ_type SQ_TPx_send SQ_TPx_const SQ_TPx_end_of_cl SQ_TPx_upe SQ_TPx_opts SQ_TPA_opts_mask SQ_TP1_lod_corre SQ_TP1_pix_mask SQ_TP1_lod_corre SQ_TP1_pix_mask SQ_TP1_lod_corre	_SQ to TP: C the fetch unit sen- is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat nstants for the ection $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow TPx$ $\rightarrow TP1$ $\rightarrow TP1$ $\rightarrow TP1$	encer on v e the fetch fetch to e 1 3 3 1 1 48 24 1 1 1 2 6 6 4 6 6 4 6	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat accounters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch state sent over 4 clocks (192 bits tota Fetch instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel	ries for each	Formatted: Bullets and Numbering
27.2.927.2.8 Once every clock, is ready or not. Th also provides the i the fetch return dat TPx_SQ_data_rdy TPx_SQ_clause_nu TPx_SQ_clause_nu TPx_SQ_type SQ_TPx_send SQ_TPx_const SQ_TPx_const SQ_TPx_instr SQ_TPx_end_of_cl SQ_TP2_inters SQ_TP0_pix_mask SQ_TP1_lod_corre SQ_TP1_pix_mask SQ_TP2_lod_corre SQ_TP2_pix_mask	_SQ to TP: C the fetch unit sen- is way the sequer instruction and co ta.	Control bus ds to the sequencer can updat nstants for the $x \rightarrow SQ$ $x \rightarrow SQ$ $\rightarrow TPx$ $\rightarrow TPy$ $\rightarrow TP1$ $\rightarrow TP1$ $\rightarrow TP2$	encer on v e the fetch fetch to e	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat o counters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch instruction sent over 4 clocks (192 bits tota Fetch instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch instruction of the clause Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel	a in the GPRs The sequencer where to write	Formatted: Bullets and Numbering
27.2.927.2.8 Once every clock, is ready or not. Th also provides the i the fetch return dar TPx_SQ_data_rdy TPx_SQ_clause_nu TPx_SQ_clause_nu TPx_SQ_type SQ_TPx_send SQ_TPx_const SQ_TPx_const SQ_TPx_end_of_cl SQ_TP2_instr SQ_TP2_instr SQ_TP0_lod_corre SQ_TP1_lod_corre SQ_TP1_pix_mask SQ_TP2_lod_corre SQ_TP2_pix_mask SQ_TP2_lod_corre SQ_TP2_pix_mask SQ_TP2_lod_corre SQ_TP2_pix_mask SQ_TP2_lod_corre	_SQ to TP: C the fetch unit sen is way the sequer instruction and co ta. Dir TP. um TP. um TP. SQ sQ sQ sQ sQ sQ sQ sQ sQ sQ sQ sQ sQ sQ	Control buss ds to the sequencer can updat nstants for the $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow TPx$ $\rightarrow TP3$	encer on v e the fetch fetch to e Bits 1 3 1 1 4 8 24 1 1 1 2 6 6 4 6 4 6 6 4 6 6	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat the counters for the reservation station fifos. The execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch state sent over 4 clocks (192 bits tota Fetch instruction sent over 4 clocks Last instruction fithe clause Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components	initian in the GPRs The sequencer where to write	Formatted: Bullets and Numbering
27.2.927.2.8 Once every clock, is ready or not. Th also provides the i the fetch return dar TPX_SQ_data_rdy TPX_SQ_clause_nu TPX_SQ_clause_nu TPX_SQ_type SQ_TPX_send SQ_TPX_send SQ_TPX_send SQ_TPX_end_of_cl SQ_TPX_end_of_cl SQ_TPX_end_of_cl SQ_TPX_end_of_cl SQ_TPA_end_of_cl SQ_TPA_end_of_cl SQ_TPA_end_of_cl SQ_TPA_end_of_cl SQ_TPA_end_of_cl SQ_TPA_end_of_cl SQ_TPA_end_of_cl SQ_TP1_lod_corre SQ_TP1_lod_corre SQ_TP2_lod_corre SQ_TP2_lod_corre SQ_TP3_pl_mask SQ_TP3_lod_corre	_SQ to TP: C the fetch unit sem is way the sequer instruction and co ta. um TP: um TP: SQ SQ SQ Iause SQ sQ ist SQ sct SQ sct SQ sct SQ sct SQ sct SQ sct SQ sct SQ	Control bus ds to the sequencer can updat nstants for the $x \rightarrow SQ$ $x \rightarrow SQ$ $x \rightarrow SQ$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TPx$ $\rightarrow TP1$ $\rightarrow TP1$ $\rightarrow TP2$ $\rightarrow TP3$ $\rightarrow TP3$	encer on v e the fetch fetch to e Bits 1 3 1 1 1 4 8 24 1 1 1 2 6 6 4 6 6 4 6 6 4 6 4	pixels in a clause)  64: 128K-bits available (16 128-bit ent of 64 pixels) 65-127: RESERVED which clause it is now working and if the dat ocunters for the reservation station fifos. T execute and the address in the register file Description Data ready Clause number Type of data sent (0:PIXEL, 1:VERTEX) Sending valid data Fetch state sent over 4 clocks (192 bits tota Fetch instruction sent over 4 clocks Last instruction sent over 4 clocks Type of data sent (0:PIXEL, 1:VERTEX) Write phase signal LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel LOD correct 3 bits per comp 2 components Pixel mask 1 bit per pixel	ries for each	Formatted: Bullets and Numbering

Exhibit 2027.docR400\_Sequencer.doc 68205 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257330

ATI Ex. 2107 IPR2023-00922 Page 196 of 260

	ORIGINATE	DATE	EDIT	DATE		DOCUMEN	T-REV. NUM.	PAGE		
	24 Sentember	2001	4 Sentemb	er 201	1519	GEN-CXX	XXX-REVA	47 of 52		
SO TO with	ant index	, 2001	Anril 2002	05. Mai		Deminter file for	urite of returns -17	Totah Data		
SQ_IPX_write_	_gpr_index	5Q->17X		(	index inte	o register file for	write of returned h	eich Data		Formatted: Bullets and Numbering
<del>27.2.10</del> 27	.2.9_TP to S	Q: Text	ture stall						•	
The TP sends	this signal to the	SQ and t	he SPs when	its inp	out buffer	is full. The SQ	is going to send	it to the SP X		
clocks after rec	ception (maximun	n of 3 clock	ks of pipeline	delay).						
TP_SP_fet	tch_Stall									
	Ļ									
SO SP wr add	ir									
	" <b>&gt;</b> SU0									
		-	SU1							
					¥	л				
					SU2					
					002					
						SU3				
Name		Direction	n	Bits	Descript	ion			] 📖	
TP_SQ_fet	ch_stall	$TP \rightarrow SQ$		1	Do not se	end more texture	request if asserted	k		
									Contraction in the second sec second second sec	
2721127	2 10 SQ to	SP. To	vture stall	1					*- N	Formatted: Bullets and Numbering
27.2.1127	<u>.2.10</u> SQ to	SP: Te	xture stall						•1	Formatted: Bullets and Numbering
27.2.1127 Name	.2.10_SQ to	SP: Te	xture stall	Bits	Descript	ion	request if asserted		•/~	Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe	.2.10_SQ to	SP: Te. Direction SQ→SPx	xture stall	Bits	Descript Do not se	ion and more texture	request if asserted	1	• [~]	Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227	<u>.2.10</u> SQ to etch_stall .2.11 SQ to	SP: Te. Direction SQ→SPx SP: GF	xture stall	Bits II	Descript Do not se Dunter	ion and more texture	request if asserted		* ~ ] * ~	Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227	.2.10_SQ to etch_stall .2.11_SQ to	SP: Te Direction SQ→SPx SP: GF	PR and au	Bits    1    1    1    1    1    1    1	Descript Do not se Dunter	ion and more texture	request if asserted		* **       	Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPX_fe 27.2.1227 Name SQ_SPX_g	<u>.2.10</u> SQ to etch_stall .2.11 SQ to pr wr addr	SP: Te. Direction SQ→SPx SP: GF Direction SQ→SP	PR and au	Bits 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Descript Do not se Dunter Descrip Write ad	ion end more texture tion dress	request if asserted		*  	Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr	SP: Te Direction SQ→SPx SP: GF Direction SQ→SP SQ→SP	xture stall	Bits 1 <i>ito co</i> Bits 7 7	Descript Do not se Dunter Descrip Write ad Read ac	ion end more texture tion dress dress	request if asserted	3	*	Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en	SP: Te Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$	PR and au	Bits 1 1 to co Bits 7 7 1	Descript Do not se Dunter Descrip Write ad Read ac Read Er	ion end more texture tion dress dress able	request if asserted	3	*	Formatted: Bullets and Numbering Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_nhase_mux	SP: Te. Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	PR and au	Bits 1 1 <i>to co</i> Bits 7 7 7 1 1 1 2	Descript Do not se Descrip Write ad Read ac Read ac Read Er Write Er	tion tion dress dress able table for the GPF see mux (arbitration	request if asserted			Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_wr_en	SP: Te Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	PR and au	Bits 1 <i>to co</i> Bits 7 7 1 1 2	Descript Do not se Dunter Descrip Write ad Read ac Read ac Read Er Write Er The pha reads ar	tion dress dress dress able nable for the GPR able for the GPR able for the drbtra d writes)	request if asserted	its, ALU SRC		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPX_fe 27.2.1227 Name SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_wr_en pr_phase_mux hannel_mask	SP: Te Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	xture stall	Bits 1 1 1 1 1 1 1 2 4	Descript Do not so Dunter Descrip Write ad Read ac Read ac Read ac Write ac Write ac The pha reads ar The cha	tion dress dress dress able nable for the GPF able for the GPF ase mux (arbitra d writes) nnel mask	request if asserted s tes between inpu	its, ALU SRC		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPX_fe 27.2.1227 Name SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_d SQ_SPX_d	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_wr_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel	SP: Te Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	xture stall	Bits         I           1         1           1         1           1         1           7         1           1         2           4         2	Descript Do not se Dunter Descrip Write ad Read ac Read Read Ac Read A	tion dress dress able table for the GPR able for the GPR able for the GPR able for the GPR able for the GPR table for th	request if asserted s tes between inputs room Interrolated	this tells from		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel	SP: Te Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	xture stall	Bits         I           1         1           7         7           7         1           1         2           4         2	Descript Do not se Dunter Descrip Write ad Read ac Read Er Write Er The pha reads ar The cha When t which s VTX1, a	tion dress dress able table for the GPF able for the GPF ase mux (arbitra d writes) nnel mask he phase mux s ource to read duore to read	request if asserted s tes between inpu elects the inputs from: Interpolated	this tells from I data, VTX0,		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPX_fe 27.2.1227 Name SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_d SQ_SPX_d SQ_SPX_d SQ_SPX_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_mux <u>sel</u> uto_count	SP: Te Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$	xture stall	Bits 1 1 to co Bits 7 7 1 1 1 2 4 2 12?	Descript Do not se Dunter Descrip Write ad Read ac Read Read Ac Read A	tion dress dress able able for the GPR able for the GPR d writes) nnel mask to phase mux s ource to read dubgen counter. ant generated by	request if asserted s tes between inputs from: Interpolated the SQ, common	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPX_fe 27.2.1227 Name SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	$\begin{array}{c} SP: \ Te, \\ \hline \textbf{Direction} \\ SQ \rightarrow SPx \\ \hline SP: \ GF \\ \hline \textbf{Direction} \\ SQ \rightarrow SP \\ \hline SQ \rightarrow SP \\ SQ \rightarrow SP \\ \hline SQ $	xture stall	Bits         I           1         1           1         1           1         1           2         1           4         2           12?         1	Descript Do not se Dunter Descrip Write ad Read ac Read Read Ac Read A	tion dress dress able able for the GPR able for the GPR a	request if asserted s tes between inputs from: Interpolated the SQ, common	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_c SQ_SPx_c SQ_SPx_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	SP: Te. Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$	PR and au	Bits 1 1 <i>to co</i> Bits 7 7 1 1 2 4 2 12?	Descript Do not se Dunter Descrip Write ad Read ac Read Er Write Er The pha reads ar The cha When t which s VTX1, a Auto coo pipes	tion dress dress dress able to the GPF able for the GPF a	request if asserted s tes between inputs from: Interpolated the SQ, common	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_c SQ_SPx_g SQ_SPx_c	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_ywr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	SP: Te. Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$	PR and au	Bits 1 1 1 1 1 1 1 1 2 4 2 12?	Descript Do not se Descrip Write ad Read ac Read Ac Re	tion dress dress able table for the GPF able for the GPF	request if asserted s tes between inputs rom: Interpolated the SQ, common	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_c SQ_SPx_g SQ_SPx_c	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	SP: Te. Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	PR and au	Bits 1 1 1 1 1 1 1 1 2 4 2 12?	Descript Do not se Dunter Descrip Write ad Read ac Read Read Ac Read A	tion dress dress able able for the GPR able for the GPR a	request if asserted s tes between inputs rom: Interpolated the SQ, common	this tells from I data, VTX0,		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_c SQ_SPx_g SQ_SPx_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	SP: Te. Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	PR and au	Bits 1 1 1 1 1 1 1 1 2 4 2 12?	Descript Do not se Descrip Write ad Read ac Read Ac Re	tion dress dress able able for the GPR able for the GPR ase mux (arbitra d writes) nnel mask te phase mux s ource to read dubgen counter. ant generated by	request if asserted s tes between inputs rom: Interpolated the SQ, common	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.11227 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_c SQ_SPx_g SQ_SPx_c SQ_SPx_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	SP: Te. Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	xture stall	Bits 1 1 1 1 1 1 1 1 2 4 2 12?	Descript Do not se Descrip Write ad Read ac Read Er Write Er The cha When th which s VTX1, a Auto coo pipes	tion tion dress dress able to the GPR able for the GPR able for the GPR able for the GPR to the other able for the GPR to the other to the other	request if asserted s tes between inpu elects the inputs from: Interpolated the SQ, common	this tells from I data, VTX0,		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.1127 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_c SQ_SPx_g SQ_SPx_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	SP: Te. Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	xture stall	Bits 1 1 1 1 1 1 1 1 2 4 2 12?	Descript Do not se Descrip Write ad Read ac Read Er Write Er The cha When th which s VTX1, a Auto coo pipes	tion tion dress dress able to the GPR able for the GPR able for the GPR able for the GPR to the other able for the GPR able for	request if asserted s tes between inpu elects the inputs from: Interpolated the SQ, common	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.11227 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_c SQ_SPx_g SQ_SPx_a SQ_SPx_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	SP: Te. Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	xture stall	Bits 1 1 <i>to co</i> Bits 7 7 1 1 2 4 2 12?	Descript Do not se Descrip Write ad Read ac Read Er Write Er The pha reads ar The cha When tt which s VTX1, a Auto cor pipes	tion tion dress dress able to the GPR able for the GPR able to the GPR abl	request if asserted s tes between inputs from: Interpolated the SQ, common	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering     Formatted: Bullets and Numbering
27.2.11227 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_a SQ_SPx_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	SP: Te. Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	xture stall	Bits         I           1         1           10         CO           Bits         7           7         1           1         2           4         2           12?         1	Descript Do not se Descrip Write ad Read ac Read Er The pha reads ar The cha When tt which s VTX1, a Auto coor pipes	tion tion dress idress able able for the GPR able for the GPR ase mux (arbitra d writes) nnel mask the phase mux s ource to read utogen counter. unt generated by	request if asserted s tes between inputs from: Interpolated the SQ, common	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering
27.2.11227 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_a SQ_SPx_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	SP: Te. Direction $SQ \rightarrow SPx$ SP: GF Direction $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$ $SQ \rightarrow SP$	xture stall	Bits 1 1 1 1 1 1 1 2 4 2 12?	Descript Do not se Descrip Write ad Read ac Read Er The pha reads ar The cha When tt which s VTX1, a Auto coor pipes	tion tion dress dress able able for the GPR ase mux (arbitra d writes) nnel mask the phase mux s ource to read utogen counter. unt generated by	request if asserted	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering
27.2.11227 Name SQ_SPx_fe 27.2.1227 Name SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_g SQ_SPx_c SQ_SPx_a SQ_SPx_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count	SP: Te. Direction SQ $\rightarrow$ SPx SQ $\rightarrow$ SP SQ $\rightarrow$ SP	ATI Confide	Bits 1 1 <i>to co</i> Bits 7 7 1 1 2 4 2 12?	Descript Do not se Descrip Write ad Read ac Read Er The pha reads ar The cha Write Er The pha reads ar The cha When th When th VTX1, a Auto com pipes	tion tion dress dress iable iable for the GPR ase mux (arbitra d writes) nnel mask te phase mux s ource to read utogen counter. unt generated by	request if asserted s tes between inputs from: Interpolated the SQ, common	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering
27.2.11227 Name SQ_SPX_fe 27.2.1227 Name SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_g SQ_SPX_c SQ_SPX_c SQ_SPX_a SQ_SPX_a	.2.10_SQ to etch_stall .2.11_SQ to pr_wr_addr pr_rd_addr pr_rd_en pr_wr_en pr_phase_mux hannel_mask pr_input_muxsel uto_count 400_Sequencer.doc 682	SP: Te. Direction SQ $\rightarrow$ SPx SQ $\rightarrow$ SP SQ $\rightarrow$ SP	ATI Confide	Bits 1 1 <i>to co</i> Bits 7 7 1 1 2 4 2 12?	Descript Do not se Descrip Write ad Read ac Read ac Read Er Write Er The pha reads ar The cha When tt which s VTX1, a Auto coo pipes	tion tion dress dress lable for the GPR ase mux (arbitra d writes) nnel mask the phase mux s ource to read utogen counter. ant generated by	request if asserted s tes between inputs from: Interpolated the SQ, common	tts, ALU SRC this tells from I data, VTX0, for all shader		Formatted: Bullets and Numbering

ATI Ex. 2107 IPR2023-00922 Page 197 of 260

	ORIGINATE DA	TE EDIT	DATE	R400 Sequencer Specification	PAGE	
	24 September, 2	001 <u>4 Septemb</u>	er, 20151	9	48 of 52	
t	27.2.1327.2.12 SQ to 5	SPx Instruction	ns	I I	*	Formatted: Bullets and Numbering
	<u>27.2.1527.2.12</u> 0Q100		110			
	Name SO SPy instr_start	Direction SO_SPy	Bits	Description		
	SQ_SP_instr	SP_instr SQ→SPx		Transferred over 4 cycles         0: SRC A Select       2:0         SRC A Argument Modifier       3:3         SRC A swizzle       11:4         VectorDst       17:12         Unused       20:18		
				1: SRC B Select     2:0       SRC B Argument Modifier     3:3       SRC B swizzle     11:4       ScalarDst     17:12       Unused     20:18		
				- 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12		
				- 3: Vector Opcode Scalar Opcode Vector Clamp 11:11 Scalar Clamp 12:12 Vector Write Mask Scalar Write Mask 20:17		
	SQ_SPx_exp_alu_id	SQ→SPx	1	ALU ID	11000	
	SQ_SPx_exporting	SQ→SPx	2	0: Not Exporting 1: Vector Exporting 2: Scalar Exporting		
	SQ_SPx_stall	SQ→SPx	1	Stall signal		
	SQ_SP0_exp_pvalid <u>write_ma</u> <u>k</u>	I <u>S</u> SQ→SP0	4	Result of pixel kill in the shader pipe, wh output for all pixel exports (depth ar buffers). 4x4 because 16 pixels are co clock	ich must be id all color mputed per	
I	SQ_SP1_ <u>write_maskexp_pvalid</u>	SQ→SP1	4	Result of pixel kill in the shader pipe, wh output for all pixel exports (depth ar buffers). 4x4 because 16 pixels are co clock	ich must be id all color mputed per	
I	SQ_SP2_ <u>write_mask</u> exp_pvalid	SQ→SP2	4	Result of pixel kill in the shader pipe, wh output for all pixel exports (depth ar buffers). 4x4 because 16 pixels are co clock	ich must be id all color mputed per	
	SQ_SP3_ <u>write_maskexp_pvalid</u>	SQ→SP3	4	Result of pixel kill in the shader pipe, wh output for all pixel exports (depth ar buffers). 4x4 because 16 pixels are co clock	ich must be id all color mputed per	
	27.2.14 <u>27.2.13</u> SP to S	SQ: Constant a	addres	s load/ Predicate Set	*-	
	Name	Direction	Bits	Description		
	SP0_SQ_const_addr	SP0→SQ	36	Constant address load / predicate vector load to the sequencer Data valid	(4 bits only)	
	SP1_SQ_const_addr	SP1→SQ	36	Constant address load / predicate vector load	(4 bits only)	
			1	to the sequencer		

Exhibit 2027.docR400\_Sequencer.doc 68205 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257332

ATI Ex. 2107 IPR2023-00922 Page 198 of 260

	ORIGINATE	DATE	EDIT	DATE		DOCUMENT-REV. NUM.	PAGE	
	24 Septembe	er, 2001	4 Septem	ber, 20	01519	GEN-CXXXXX-REVA	49 of 52	
SP1 SO	valid	1901 .50	<u>Anril_200</u>	1	Data va	lid	1	
SP2_SQ_	const_addr	SP2→SQ		36	Consta to the s	nt address load / predicate vector load equencer	I (4 bits only)	
SP2_SQ_ SP3_SQ_	valid const_addr	SP2→SQ SP3→SQ		1 36	Data va Constar	lid nt address load / predicate vector load	I (4 bits only)	
SP3_SQ_	valid	SP3→SQ		1	Data va	lid		
<u>27.2.15</u> 27	7.2.14_SQ te	o SPx: co	onstant b	road	lcast			
Name		Direction		Bits	Descrip	otion		
SQ SPx	const	SQ→SPx		128	Consta	nt broadcast		
27.2.1627	7.2.15_SP0	to SQ: K	ill vector	load	1			Formatted: Bullets and Numbering
Name		Direction		Bits	Descrip	otion		
SP0_SQ_	kill_vect	SP0→SQ		4	Kill vect	tor load		
SP1_SQ_	kill_vect	SP1→SQ		4	Kill vect	tor load		
SP2_SQ_	kill_vect	SP2→SQ		4	Kill vect	tor load		
SP3_SQ_	kill_vect	SP3→SQ		4	Kill vect	tor load		
<u>27.2.17</u> 27	7.2.16_SQ to	o CP: RB	BM bus				4-	
Name		Direction		Bits	Descrip	otion		
SQ RBB	rs	SQ→CP		1	Read S	Strobe		
SQ RBB	rd	SQ→CP		32	Read D	Data		
SQ RBBI	M nrtrtr	SQ→CP		1	Optiona	al		
SQ RBBI	M rtr	SQ→CP		1	Real-Ti	ime (Optional)		
<del>27.2.18</del> 27	7.2.17_CP to	SQ: RB	BM bus			······································		Formatted: Bullets and Numbering
Name		Direction		Bits	Descrip	otion		
rbbm_we		CP→SQ		1	Write E	nable		
rbbm_a		CP→SQ		15	Addres	s Upper Extent is TBD (16:2)		
rbbm_wd		CP→SQ		32	Data	······································		
rbbm be		CP→SQ		4	Byte Er	nables		
rbbm_re		CP→SQ		1	Read E	Inable		
rbb rs0		CP→SQ		1	Read R	Return Strobe 0		
rbb_rs1		CP→SQ		1	Read F	Return Strobe 1		
rbb rd0		CP→SQ		32	Read D	Data O		
rbb_rd1		CP→SQ		32	Read D	Data 0		
RBBM_S	Q_soft_reset	CP→SQ		1	Soft Re	eset		
27.2.18 5	SQ to CP: Si	tate repo	rt				4	- < (Formatted: Bullets and Numbering
Name		Direction		Bits	Descrit	otion	]	
SQ CP V	/s event	SQ→CP		1	Vertex	Shader Event		
SQ CP V	/s eventid	SQ→CP		2	Vertex	Shader Event ID		
SQ CP r	os event	SQ→CP		1	Pixel S	hader Event		
SQ CP r	os eventid	SQ→CP		2	Pixel S	hader Event ID		
event event So, the CP wi and the SQ_C	id = 0 => *sEndC id = 1 => *sDone II assume the Vs CP_vs_eventid =	DfState (i.e. (i.e is done with 0.	VsEndOfSt . VsDone) . a state whe	<u>ate)</u> enever	it gets a	pulse on the SQ_CP_vs_event		
Exhibit 2027.docl	R400_Sequencer.doc 6	8205 Bytes*** ©	ATI Confid	lential	. Referer	nce Copyright Notice on Cover Pa	ge © ***	

ATI Ex. 2107 IPR2023-00922 Page 199 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
	24 September, 2001	4 September, 201519		50 of 52	Formatted: Bullets and Numbering
<u>28.Exam</u>	ples of program	<u>executions</u>		*1	
28.1.1 Se	quencer Control o	f a Vector of Vertic	æ <del>s</del>		
1.PA sends a •state point •space was •also befr shade •The verte •the S by	vector of 64 vertices (actu hter as well as tag into po- as allocated in the positior ore the vector is sent to er program (using the M exprogram is assumed to EQ then accesses the IS / the RBBM when the CP				
2.SEQ arbitrat ●at this po ●the arbite nothin	tes between the Pixel FIF int the vector is removed or is not going to select a v g else to do (ie no pixels i	O and the Vertex FIFO — k from the Vertex FIFO vector to be transformed if are in the pixel fifo).	vasically the Vertex FIFO always has p the parameter cache is full unless the	riority pipe as	
3.SEQ allocat	es space in the SP registe per of GPRs required by the pointer that came down w not send vertex data until	er file for index data plus G ne program is stored in a k ith the vertices space in the register file f	iPRs-used by the program ocal state register, which is accessed u nas been allocated	lising the	
4.SEQ sends •the 64 vo •RF0 •RF1 •RF2 •RF3 •the index the 12	the vector to the SP regis of SUO, SU1, SU2, and S of SU0, SU1, SU2, and S is written to the least sign Solit location within the rec				
5.SEQ constru state mac •the contr	ucts a control packet for th hine 0, or TSM0 FIFO) ol packet contains the sta	ne vector and sends it to the pointer, the tag to the p	e first reservation station (the FIFO in osition cache and a register file base p	front of fetch ointer.	
6.TSM0 accep •TSM0 wa	ots the control packet and as first selected by the TS	fetches the instructions fo M arbiter before it could st	<del>r fetch clause 0 from the global instruc</del> art	tion store	
7.all-instructio	ns of fetch clause 0 are is	sued by TSM0			
8.the control p FIFO) •TSM0.do fetch 4 •once the FIFO; clause	eacket is passed to the ne: es not wait for requests n data to the TU, which will TU has written all the dat a count greater than zero e	xt reservation station (the nade to the Fetch Unit to c write the data to the RF as a to the register files, it inc indicates that the ALU sta	FIFO in front of ALU state machine 0, 4 omplete; it passes the register file write bit is received rements a counter that is associated w ate machine can go ahead start to exec	er ASM0 e index for the vith ASM0 cute the ALU	
9.ASM0 accer 0 from the	ots the control packet (afte global instruction store	er being selected by the A	SM arbiter) and gets the instructions fo	r ALU-clause	
10.all instructi station (th	ons of ALU clause 0 are it e FIFO in front of fetch sta	ssued by ASM0, then the ( ate machine 1, or TSM1 Fi	control packet is passed to the next res IFO)	servation	
11.the control	packet continues to trave can be exported in ALU cl d with all four shader pipe ster cache pointer is also- in the parameter cache. is a position export FLEO	n executed (which is data is going e PA			
Exhibit 2027.docF	1400_Sequencer.doc 68205 Bytes***	© ATI Confidential. Ref	erence Copyright Notice on Cover P	Page © ***	

ATI Ex. 2107 IPR2023-00922 Page 200 of 260

				1			
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE			
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	51 of 52			
• the A •paramete •para •the S lo •the A po	SM arbiter will prevent a pa or data is exported in clause meter data is sent to the Pa EQ allocates storage in the nger a need for the parame SM arbiter will prevent a pa osition is being exported) is	cket from starting an export 7 (as well as position data rameter Cache over a dedic Parameter Cache, and the ters (it is told by the PA whe teket from starting on ASM7 full	ing clause if the position export FIFO if it was not exported earlier) ated bus SEQ deallocates that space when the n using a token). if the parameter cache (or the position	is full ere is no on buffer if			
12.after the shader program has completed, the SEQ will free up the GPRs so that they can be used by another shader program							
28.1.2 Sequencer Control of a Vector of Pixels							
1.As with ver	tex shader programs, pixe	el shaders are loaded into	the global instruction store by the	CP			
At this po	pint it is assumed that the pi	xel-program is loaded into th	ne instruction store and thus ready to	be read.			
2.the RE's Pix ethe state ethe Pixel	el FIFO is loaded with the b pointer and the LOD correc FIFO is wide enough to sou	varycentric coordinates for p ition bits are also placed in t urce four quad's worth of ba	ixel quade by the detailed walker he Pixel FIF0 rycentrics per cycle				
3.SEQ arbitrat left in the	es between Pixel FIFO and register files for vertices, the	Vertex FIFO – when there a Pixel FIFO is selected	are no vertices pending OR there is n	io-space			
4.SEQ allocat	es space in the SP register oer of GPRs required by the pointer not allow interpolated data i	file for all the GPRs used by program is stored in a local to be sent to the shader unti	<ul> <li>the program</li> <li>state register, which is accessed usi</li> <li>space in the register file has been a</li> </ul>	ng the llocated			
5.SEQ control bandwidth	<ul> <li>SEQ controls the transfer of interpolated data to the SP register file over the RE_SP interface (which has a bandwidth of 2048 hits/cycle). See interpolated data bus diagrams for details.</li> </ul>						
6.SEQ constru state mac •note that •the contr •all other	<ul> <li>bandwidth of 2048 bits/cycle). See interpolated data bus diagrams for details.</li> <li>6.SEQ constructs a control packet for the vector and sends it to the first reservation station (the FIFO in front of fetch state machine 0, or TSM0 FIFO)</li> <li>•note that there is a separate set of reservation stations/arbiters/state machines for vertices and for pixels</li> <li>•the control packet contains the state pointer, the register file base pointer, and the LOD correction bits</li> </ul>						
7.TSM0 accep •TSM0 wa	ots the control packet and fe is first selected by the TSM	tches the instructions for fel arbiter before it could start	ch clause 0 from the global instructio	n store			
8.all instructio	ns of fetch clause 0 are issu	ied by TSM0					
9.the control p FIFO)	acket is passed to the next	reservation station (the FIF	O in front of ALU state machine 0, or	ASMO			
●TSM0 do for the ●once the assoc aheac	es not wait for fetch reques fetch data to the TU, which TU has written all the data - iated with the ASM0 FIFO; ( and pop the FIFO and star	ts made to the Fetch Unit to n will write the data to the RI for a particular clause to the a count greater than zero in t to execute the ALU clause	complete; it passes the register file v - as it is received register files, it increments a counter icates that the ALU state machine co i	vrite index - that is an go			
10.ASM0 acco clause 0 fi	opts the control packet (afte com the global instruction st	r being selected by the ASN ore	l arbiter) and gets the instructions for	ALU			
11.all instructi station (th	ons of ALU clause 0 are iss e FIFO in front of fetch state	ued by ASM0, then the cont machine 1, or TSM1 FIFO	rol packet is passed to the next reser )	vation			
12.the control ∗pixel data ∗it is⊣ ∗the A	packet continues to travel d a is exported in the last ALU sent to an output FIFO when SM arbiter will prevent a pa	lown the path of reservation <del>  clause (clause 7)</del> re it will be picked up by the acket from starting on ASM7	stations until all clauses have been c render-backend if the output FIFO is full	executed			
13.after the shader pro	ader program has complete ogram	ed, the SEQ will free up the	GPRs so that they can be used by ar	nother			
Exhibit 2027.docF	400_Sequencer.doc 68205 Bytes*** ⓒ	aTI Confidential. Refere	nce Copyright Notice on Cover Pag	je © ***			

ATI Ex. 2107 IPR2023-00922 Page 201 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	]
	24 September, 2001	4 September, 201519		52 of 52	
28.1.3	Notes	- ABRI 1007775 Warab		4	Formatted: Bullets and Numbering
14.The-sta threade	e machines and arbiters wi	Il operate ahead of time so	that they will be able to immediately st	art the real	
15.The reg instruc pointer	ister file base pointer for a v ion store base pointer does is only different for each st	vector needs to travel with the not – this is because the R ate and thus can be access	ne vector through the reservation static F pointer is different for all threads, bu ed via the state pointer.	ons, but the it the IS	
29.28	Onen issues			*	Formatted: Bullets and Numbering
Need to do static).	some testing on the size of	of the register file as well as	s on the register file allocation method	l (dynamic VS	
Saving pov	ver?				
Parameter	caches in SX?				
Using-both	IJ buffers for center + cent	roid interpolation?			
Exhibit 2027	docR400_Sequencer.dec 68205 Bytes**	🔹 © ATI Confidential. Ref	erence Copyright Notice on Cover P	age © ***	
				-	

ATI Ex. 2107 IPR2023-00922 Page 202 of 260

		·····		
	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	1 of 58
Author:	Laurent Lefebvre			· ·
Issue To:		Copy No:		
	R400 S	equencer Spe	ecification	
		SQ		
		Version <u>1.112.0</u>		
Overview: This req bloo	s is an architectural specifi uired capabilities and expe cks, and provides internal st	cation for the R400 Sequend cted uses of the block. It al ate diagrams.	cer block (SEQ). It provides an ov so describes the block interfaces,	erview of the internal sub-
AUTOMATIC Document Lo Current Intra	ALLY UPDATED FIELDS: ocation: C:\pa unet Search Title: R400	erforce\r400\doc_lib\design\b ) Sequencer Specification APPROVALS	locks\sq\R400_Sequencer.doc	
	Name/Dept		Signature/Date	
Remarks:				
THIS DOO SUBSTA	CUMENT CONTAIN NTIALLY DETRIME INC. THROUGH U	S CONFIDENTIAL II NTAL TO THE INTE JNAUTHORIZED US	NFORMATION THAT CO REST OF ATI TECHNOL E OR DISCLOSURE.	ULD BE .OGIES
"Copyright 200 work created ir unpublished w confidential, pro transmitted in a	1, ATI Technologies Inc. Al 2001. The use of this cop ork. The copyright notice oprietary information and tr ny form or by any means wi	I rights reserved. The mater yright notice is intended to is not an admission that ade secrets of ATI. No part thout the prior written permise	ial in this document constitutes ar provide notice that ATI owns a cop publication has occurred. This w of this document may be used, re sion of ATI Technologies Inc."	unpublished byright in this vork contains produced, or
Exhibit 2028.doc	R400_Sequencer.doc 73201 Bytes*** 🤇	atti Confidential. Referen	ce Copyright Notice on Cover Pa	ge © ***

ATI 2028 LG v. ATI IPR2015-00325

AMD1044\_0257337

ATI Ex. 2107 IPR2023-00922 Page 203 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		2 of 58

### Table Of Contents

1.	OVE	ERVIEW	6
1.1	1 op	) Level Block Diagram	1210
1.2	Con	a Flow graph (SF)	1411
2.	INTI	ERPOLATED DATA BUS	1411
3.	INS	TRUCTION STORE	1714
4.	SEC	QUENCER INSTRUCTIONS	1714
5.	CON	NSTANT STORES	<u>17</u> 14
5.1	Mer	mory organizations	<u>17</u> 14
5.2	Mar	nagement of the Control Flow Constants	1845
5.3	Mar	nagement of the re-mapping tables	1815
5.	3.1	R400 Constant management	<u>18</u> 15
5.	3.2	Proposal for R400LE constant management	<u>18</u> 15
5.	3.3	Dirty bits	<u>20</u> 17
5.	3.4	Free List Block	<u>20</u> 17
5.	3.5	De-allocate Block	<u>21</u> 48
5.	3.6	Operation of Incremental model	<u>21</u> 48
5.4	Con	nstant Store Indexing	<u>21</u> 48
5.5	Rea	al Time Commands	2219
5.6	Con	nstant Waterfalling	2219
<b>6</b> .	LOC	OPING AND BRANCHES	2320
6.2	The	Control Flow Program	2320
6.3	Data	a dependant predicate instructions.	2922
6.4	HW	/ Detection of PV,PS	2923
6.5	Reg	gister file indexing	<u>29</u> 23
6.6	Pred	dicated Instruction support for Texture clauses	3023
6.7	Deb	bugging the Shaders	<u>30</u> 23
6.	7.1	Method 1: Debugging registers	<u>30</u> 23
6.	7.2	Method 2: Exporting the values in the GPRs (12)	<u>30</u> 24
7.	PIX	EL KILL MASK	<u>31</u> 24
8.	MUI		3124
9.	REG	GISTER FILE ALLOCATION	3124
10.			3226
11.	HAN	NDLING STALLS	3327
13.	CON	NTENT OF THE RESERVATION STATION FIFOS	3327
14.	THE	E OUTPUT FILE	3327
15.	IJF	ORMAT	3327
15.1	In	terpolation of constant attributes	3428
16.	STA	AGING REGISTERS	3428
17.	THE		3630
18.	VEr		3130

Exhibit 2028.doc.R409\_Sequencer.doc 73201 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257338

ATI Ex. 2107 IPR2023-00922 Page 204 of 260

	2	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
	UU	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	3 of 58
19.	EXPO	RTING ARBITRATION	Anril 2002		<u>37</u> 30
20.	EXPO	RTING RULES	*****		<u>38</u> 30
20.1	Para	meter caches exports.			<u>38</u> 30
20.2	Nerr	tion exports		• • • • • • • • • • • • • • • • • • • •	<u>3830</u>
20.3	FYPO	RT TYPES			3830
21.1	Vert	ex Shading		***************************************	
21.2	Pixe	I Shading			
22.	SPEC	IAL INTERPOLATION	MODES		<u>39</u> 31
22.1	Rea	I time commands			<u>39</u> 31
22.2	Sprit	tes/ XY screen coordina	ates/ FB information		
22.3	Auto	generated counters			<u>39</u> 3£
	.3.1	Vertex shaders			<u>39</u> 32
22	.3.2	Pixel shaders			<u>39</u> 32
23.	STATI	E MANAGEMENT			
23.1	Para	Imeter cache synchroni	ization		
24. 24.1	Vert	ev indeves imports			4033
25.	REGIS	STERS			
25.1	Cont	trol			
25.2	Cont	text			<u>41</u> 33
26.	DEBU	G REGISTERS			<u>42</u> 34
26.1	Cont	text			
26.2	Con				
27.	INIC	KFAUES		*****	<u>42</u> 35
27.1	EXU				
27.2	21	SC SP#			1235
27	22.1	SC_SO			4336
27	23	SO to SX: Internolator	r hue		<u>45</u> 88
27	2.5	SO to SP: Staning Re	nister Data		
27	2.5	VGT to SO : Vertex in	iterface		<u>40</u> 07 1538
27	26	SO to SX: Control bus	2		
27	27	SX to SO : Output file	control		лол1
27	28	SO to TP: Control bus	s		5041
27	29	TP to SO: Tevture sta			5142
27	2 10	SO to SP. Texture sta			5142
27	2 11	SO to SP' GPR and a	uto counter		5142
27	2.12	SO to SPV: Instruction	nato obunitor		50192
21	212	SQ to SFX. Instruction	ddrace laad/ Dradiaata	9	<u>0240</u> 5040
21	.4.1J 211	SO to SDV: constant a	uuress iudu/ Freuicale	000	<u>32</u> 43 EDAA
27	.4.14	SQ to Smx. constant t	JI UAUGAST		
21	.4.13	SPU to SQ: KIII Vector	1090		<u>53</u> 44
· * * /	2.16	SQ to CP: RBBM bus			<u>53</u> 44

ATI Ex. 2107 IPR2023-00922 Page 205 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE				
	24 September, 2001	4 September, 201519		4 of 58				
27.2.17 CP to SQ: RBBM bus								
27.2.18	SQ to CP: State rep	ort		<u>53</u> 44				
28. OPEN	ISSUES			<u>58</u> 44				

Exhibit 2028.doc R400\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257340

ATI Ex. 2107 IPR2023-00922 Page 206 of 260

				PAGE
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	5 of 58
Revision	Changes:	April 2002		
		Einst due	a	
Date: May 7, 2	2001	First dra	π.	
Rev 0.2 (Lauro	ent Lefebvre)	Change	d the interfaces to reflect the changes	in the
Date : July 9, 2	2001	SP. Add	ed some details in the arbitration sect	ion.
Date : August	6, 2001	August	3, 2001.	ung on
Rev 0.4 (Laure	ent Lefebvre)	Added 1	the dynamic allocation method for r	egister
Date : August	24, 2001	file and flow of p	ixels/vertices in the sequencer.	or the
Rev 0.5 (Laure	ent Lefebvre)	Added t	ming diagrams (Vic)	
Date : Septem Rev 0.6 (Laure	iber 7, 2001 ent Lefebyre)	Change	d the spec to reflect the new	R400
Date : Septem	iber 24, 2001	architec	ture. Added interfaces.	
Rev 0.7 (Laure	ent Lefebvre) r 5-2001	Added store m	constant store management, inst anagement control flow manageme	ruction nt and
Dute . Cotobo	, 2001	data dej	pendant predication.	
Rev 0.8 (Laure	ent Lefebvre) r 8 2001	Change	d the control flow method to be Also updated the external interfaces	more
Rev 0.9 (Laure	ent Lefebvre)	Incorpor	rated changes made in the 10/18/01	control
Date : Octobe	r 17, 2001	flow me	eting. Added a NOP instruction, re	moved
		registers		uebug
Rev 1.0 (Laure	ent Lefebvre)	Refined	interfaces to RB. Added state register	ſS.
Rev 1.1 (Laure	ent Lefebvre)	Added	SEQ→SP0 interfaces. Changed	delta
Date : Octobe	r 26, 2001	precisio	n. Changed VGT→SP0 interface.	Debug
Rev 1.2 (Laure	ent Lefebvre)	Interface	s added. es greatly refined. Cleaned up the spe	с.
Date : Novemi	ber 16, 2001	فناسم اسلم ف		
Date : Novemi	ent Lefebvre) ber 26, 2001	Added t	he different interpolation modes.	
Rev 1.4 (Laure	ent Lefebvre)	Added	the auto incrementing counters. Ch	nanged
Date : Decemi	ber 6, 2001	the VGI manage	SQ Interface. Added content on coment. Updated GPRs.	onstant
Rev 1.5 (Laure	ent Lefebvre)	Remove	d from the spec all interfaces that	weren't
Date : Decemi	ber 11, 2001	directly	tied to the SQ. Added explanations to the SQ. Added P.	ns on A→SQ
		synchro	nization fields and explanation.	
Rev 1.6 (Laure	ent Lefebvre)	Added r detail a	nore details on the staging register.	Added
Date : barraary	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	call inst	ruction to a Conditionnal_call instr	uction.
		Added	details on constant management	t and
Rev 1.7 (Lauro	ent Lefebvre)	Added	Real Time parameter control in t	he SX
Date : Februar	ry 4, 2002	interface New jet	e. Updated the control flow section.	and of
Date : March 4	4, 2002	clause	modifier, removed the end of	clause
	·	instructi	ons.	
Date : March	ent Lefebvre) 18, 2002	ensure l	pement of the CF instruction bits in o byte alignement.	rder to
Rev 1.10 (Lau	rent Lefebvre)	Updated	I the interfaces and added a sect	ion on
Date : March 2 Rev 1.11 (Lau	25, 2002 rent Lefebvre)	exportin Added (	g rules. CP state report interface. Last versior	of the
Date : April 19	, 2002	spec wit	h the old control flow scheme	
Rev 2.0 (Lauro	ent Lefebvre)	New cor	ntrol flow scheme	
JAKO , MUTE 10	- 3 and not not here			
Exhibit 2028.doc	1400_Sequencer.doc 73201 Bytes*** 🤅	ATI Confidential. Refere	nce Copyright Notice on Cover Pag	e © ***

ATI Ex. 2107 IPR2023-00922 Page 207 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September, 2001	4 September, 201519		6 of 58	

# 1. Overview

The sequencer is based on the R300 design. It The sequencer chooses two ALU clauses threads and a fetch clause hread to execute, and executes all of the instructions in a clause-block before looking for a new clause of the same type. Two ALU clauses, <u>htreads</u> 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-ponge along the sequencer FIFO, bouncing from fetch reservation station to alu reservation station. A FIFO exists between each reservation stage, holding up vectors until the vector currently occupying a reservation stations has left. A vector at a reservation station can be chosen to execute. The sequencer looks at all eight alu reservation stations to choose an alu clause to execute and all eight fetch stations to choose a fetch clause to execute. The arbitrator will give priority to clauses/reservation stations closer to the bottom of the pipelineolder threads. It will not execute an alu clause until the fetch fetches initiated by the previous fetch clause have completed. There are two separate sets of reservation stations, one for pixel vectors and one for vertices vectors. This way a pixel can pass a vertex and a vertex can pass a pixel.

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

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

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

Exhibit 2028.docR490\_Sequences.doc 73201 Bytes\*\*\* <sup>©</sup> ATI Confidential. Reference Copyright Notice on Cover Page <sup>©</sup> \*\*\*

AMD1044\_0257342

ATI Ex. 2107 IPR2023-00922 Page 208 of 260

# PROTECTIVE ORDER MATERIAL



Exhibit 2028 dos RAUGE\_Sequenceardos 73201 Bytes\*\*\*\* 
ATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\*

AMD1044\_0257343

ATI Ex. 2107 IPR2023-00922 Page 209 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September, 2001	4 September, 201519		8 of 58	

Exhibit 2028 doc R400\_Sequencer.doc 73201 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257344

ATI Ex. 2107 IPR2023-00922 Page 210 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	7
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	9 of 58	
1 Top L	evel Block Diad	Iram		4	-1
		,			
Fuelda apaga 40 m	100 Carrier Jan 70003 D . date	© ATLConfidential Poteron	ce Convright Notice on Cover Pa	ne @ ***	
EXHIBIT 2028.docR4	400-aequencer.coc /3201 Bytes***		ce copyright notice on cover Pa	95 8	

ATI Ex. 2107 IPR2023-00922 Page 211 of 260



Exhibit 2028.doc R409\_Sequences.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257346

ATI Ex. 2107 IPR2023-00922 Page 212 of 260



Exhibit 2028.doc R400\_Sequences.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257347

ATI Ex. 2107 IPR2023-00922 Page 213 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
AU	24 September, 2001	4 September, 201519		12 of 58

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 (tomd) is passed to the fetch system identifying the current level number (0) as well as the GPR write address for the fetch return data. One fetch request is sent every 4 clocks causing the texturing of sixteen 2x2s worth of data (or 64 vertices). Once all the requests are sent the packet is put in FIFO 1.

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

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

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

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

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

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

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

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

Under this new scheme, the sequencer (SQ) will only use one global state management machine per vector type (pixel, vertex) that we call the reservation station (RS).

Exhibit 2028.doc R409\_Sequencer.doc 73201 Bytes\*\*\* 🕲 ATI Confidential. Reference Copyright Notice on Cover Page 🕲 \*\*\*

AMD1044\_0257348

ATI Ex. 2107 IPR2023-00922 Page 214 of 260



ATI Ex. 2107 IPR2023-00922 Page 215 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		14 of 58

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

# 1.3 Control Graph



Figure 4: Sequencer Control interfaces

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

# 2. Interpolated data bus

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

Exhibit 2028.docR499\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257350

ATI Ex. 2107 IPR2023-00922 Page 216 of 260


ATI Ex. 2107 IPR2023-00922 Page 217 of 260 PROTECTIVE ORDER MATERIAL

		r23					5 48 <	52- 55	20 - 20 20 - 20	> % ß	
		T22 <sup>-</sup>					35 2 <	39 go <	> 4 5	> 44 - 6 47 (6	×
		T21					2 <del>1</del> 0 − 2	33 c <	24-27	3 <sup>2</sup> 8 <	-5
		T20					> <sup>0</sup> -3	> -4	, ⇒ ⇔ =	→ <sup>1</sup> 7 <sup>2</sup> <	and the second se
		T19							Ш	E1	
		T18								D	
	80	T17			×₿	Ϋ́		8	5	C2	
PAG	6 of 5	T16			ы	Ш				BO	2
	~	T15			СШ	ш	5	D2			۵
tion		T14	≳Б	≿ä		≿8	с С	5	C5		
Decifica		T13	Б	D2		8	B1				
cer Sp	T12	Б	D2		8	AO	A1	A2			
equen	equenc	T11		≿ 2	ζ. Ω	000		***********	С	μ	
400 S		T10	Ē	С4	C5	ŭ				8	
<u>~</u>		T9		C4	C5			8	5	5 C	
	01549	Т8	≿ຬ	≿ଃ	≿ວ	≳8				BO	Mussimu
DATE	ber, 2(	11	S	8	δ	3	5	D2			À
EDIT	eptem	T6	ទ	ဗိ	5	C2	ប៊	<u>5</u>	C5		
	4 8	Т5	≿≌			≿ 8	B1				
ATE	2001	T4	<u>B</u>			BO	AO	A1	A2		
TE D	nber, 1	13	<u>छ</u>			BO	¥8- 51	55 55	56- 56-	3 ° X	
IGINA	Septer	12	۶X	۶¥	א אַ		X 32- 35	39 % X	∑ 4 £	X 44- 74	$\succ$
OR	24 (	F	Ao	A1	A2		16- 19	% % X	24-27-27	33 <sup>2</sup> 8 X	$\times$
9		٩	Ao	A1	A2		53 ₹	× × 7-4	≍ ∞ ≿	12 4 ₹	
K	3		P O	<del>с</del> –	S ∽	ი ი	с o	- SP	SP SP	ია	

Figure 6: Interpolation timing diagram

73201 Bytes #\*\*\* 
 ATI Confidential. Reference Copyright Notice on Cover Page 
 \*\*\*\*

Exhibit 2028.docR400\_Sequencer.doc

AMD1044\_0257352

ATI Ex. 2107 IPR2023-00922 Page 218 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
C QU	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	17 of 58	

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

# 3. Instruction Store

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

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

The instruction store is loaded by the CP thru the register mapped registers.

The VS\_BASE and PS\_BASE context registers are used to specify for each context where its shader is in the instruction memory.

For the Real time commands the story is quite the same but for some small differences. There are no wrap-around points for real time so the driver must be careful not to overwrite regular shader data. The shared code (shared subroutines) uses the same path as real time.

### 4. Sequencer Instructions

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

### 5. Constant Stores

# 5.1 Memory organizations

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

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

The texture state is also kept in a similar memory. The size of this memory is 320x96 bits (128 texture states for regular mode, 32 states for RT). The memory thus holds 128 texture states (192 bits per state). The logical size exposes 32 different states total, which are going to be shared between the pixel and the vertex shader. The size of the re-mapping table to for the texture state memory is 32 lines (each line addresses 1 texture state lines in the real memory). The CP write granularity is 1 texture state lines (or 192 bits). The driver sends 512 bits but the CP ignores the top 320 bits. It thus takes 6 clocks to write the texture state. Real time requires 32 lines in the physical memory (this is physically register mapped).

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

The constant re-mapping tables for texture state and ALU constants are logically register mapped for regular mode and physically register mapped for RT operation.

Exhibit 2028.docR409\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257353

ATI Ex. 2107 IPR2023-00922 Page 219 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		18 of 58

# 5.2 Management of the Control Flow Constants

The control flow constants are register mapped, thus the CP writes to the according register to set the constant, the SQ decodes the address and writes to the block pointed by its current base pointer (CF\_WR\_BASE). On the read side, one level of indirection is used. A register (SQ\_CONTEXT\_MISC.CF\_RD\_BASE) keeps the current base pointer to the control flow block. This register is copied whenever there is a state change. Should the CP write to CF after the state change, the base register is updated with the (current pointer number +1)% number of states. This way, if the CP doesn't write to CF the state is going to use the previous CF constants.

# 5.3 Management of the re-mapping tables

### 5.3.1 R400 Constant management

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

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

For this model to work in its simplest form, the requirement is that the physical memory MUST be at least twice as large as the logical address space + the space allocated for Real Time. In our case, since the logical address space is 512 and the reserved RT space can be up to 256 entries, the memory must be of sizes 1280 and above. Similarly the size of the texture store must be of 32\*2+32 = 96 entries and above.

### 5.3.2 Proposal for R400LE constant management

To make this scheme work with only 512+256 = 768 entries, upon reception of a CONTROL packet of state + 1, the sequencer would check for SQ\_IDLE and PA\_IDLE and if both are idle will erase the content of state to replace it with the new state (this is depicted in Figure 8: De-allocation 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.

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).

Exhibit 2028.docR400\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257354

ATI Ex. 2107 IPR2023-00922 Page 220 of 260



ATI Ex. 2107 IPR2023-00922 Page 221 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		20 of 58



### Figure 89: De-allocation mechanism for R400LE

### 5.3.3 Dirty bits

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

### 5.3.4 Free List Block

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

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

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

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

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

Exhibit 2028.doc R490\_Sequencer.doc 73201 Bytes\*\*\* 
C ATI Confidential. Reference Copyright Notice on Cover Page 
\*\*\*

AMD1044\_0257356

ATI Ex. 2107 IPR2023-00922 Page 222 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	21 of 58

5.3.5 De-allocate Block

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

### 5.3.6 Operation of Incremental model

The basic operation of the model would start with the write\_ptr, stop\_ptr, read\_ptr pointers in the free list set to zero and the free list counter is set to zero. Also all the dirty bits and the previous context will be initialized to zero. When the first set constants happen, the reset dirty bit will not be set, so we will allocate a physical location from the free list counter is 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:

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

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

Whenever a draw packet arrives, the content of the re-mapping table is written to the correct re-mapping table for the context number. Also if the next context uses less constants than the current one all exceeding lines are moved to the free list to be de-allocated later. This happens in parallel with the writing of the re-mapping table to the correct memory.

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

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

# 5.4 Constant Store Indexing

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

Exhibit 2023.doc/R400\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257357

ATI Ex. 2107 IPR2023-00922 Page 223 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		22 of 58

between the time the sequencer is loaded and the time one can index into the constant store. The assembly will look like this

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

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

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

## 5.5 Real Time Commands

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

# 5.6 Constant Waterfalling

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



Exhibit 2028.docR400\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257358

ATI Ex. 2107 IPR2023-00922 Page 224 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE							
C UU	24 September, 2001	4 September, 2015	GEN-CXXXXX-REVA	23 of 58							
6. <b>Loopi</b>	ng and Branches										
Loops and br supporting co	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 B	ooleans and 32 loops.										
We have a st	ack of 4 elements for nested	calls of subroutines and 4 lo	pop counters to allow for nested loops	3.							
This state is a	wailable on a per shader pro	gram basis.									
6.2 The	Control Flow Prog	Iram									
We'd like to b	e able to code up a program	of the form:									
2: Exec 3: 4: 5: 6: 7: End L 8: ALU But realize th these dependence clauses' and unit. This info	TexFetch TexFetch ALU ALU TexFetch .coop Export at 3: may be dependent on lencies need to be express 'ALU clauses' we need to kr rmation will be encapsulated	2: and 4: is almost certair ad in the Control Flow instr low which instructions to dis in the flow control instructic	nly dependent on 2: and 3:. Withou uctions. Additionally, without separ spatch to the Texture Unit and which ons.	<u>ut clausing,</u> ate 'texture to the ALU							
Each control	flow instruction will contain 2	bits of information for each	(non-control flow) instruction:								
*****	a) ALU or Texture b) Serialize Exect	<u>ation</u>									
(b) would forc have been fe would be limi same conditio	te the thread to stop execution tched. Given the allocation ted to about 8 (non-control- ons) would be issued.	on at this point (before the i on of reserved bits, this wo flow) instructions. If more th	nstruction is executed) and wait until uld mean that the count of an 'Exec' han this were needed, a second Exe	all textures instruction ec (with the							
Another funct vertices are e allocated in o	ion that relies upon 'clauses xported in the correct order rder. Additionally data can't	s' is allocation and order of (even if not all execution is o be exported until space is a	execution. We need to assure that ordered) and that space in the output llocated. A new control flow instructio	pixels and buffers are n:							
Alloc	<pre></pre>	parameter, pixel or vertex	memory. And the size required>.								
would be crea allocation for completed. Th Alloc will occu allow the exp	ated to mark where such all a given thread can not be p ne implementation would als ur in order at least until the ports to occur without any	ocation needs to be done. erformed unless the equiva o assure that execution of ir e next serialization or chang further synchronization. Or	To assure allocation is done in order lent allocation for all previous threads instruction(s) following the serialization ge from ALU to Texture. In most cash ly 'final' allocations or position allo	, the actual s is already n due to the ses this will cations are							
Exhibit 2028.doc	R400_Sequencer.doc 73201 Bytes*** ©	ATI Confidential. Referer	nce Copyright Notice on Cover Pag	e © ***							

ATI Ex. 2107 IPR2023-00922 Page 225 of 260

ORIGINATE	DATE EDIT D	ATE R400 Se	equencer Specification	n PAGE				
24 September	er, 2001 <u>4 September</u>	r, 201519		24 of 58				
guaranteed to be ordered. B a single alloc for these structu	lecause strict ordering is ires. Vertex exports to r	required for pixels, par nemory do not require o	ameters and position rdering during allocat	s, this implies only ion and so multiple	-			
allocs may be done.	- Formatted: Bullets and Numbering							
6.2.1 Control flow ins	structions table							
Here is the revised control flow	Here is the revised control flow instruction set.							
Note that whenever a field is	s marked as RESERVE	D, it is assumed that a	II the bits of the field	d are cleared (0).				
47 46 43	40 34	Execute 33 16	15 12	11 0				
Addressing 0001	RESERVED Inst	ructions type + serialize instructions)	(9 <u>Count</u>	Exec Address				
Execute up to 9 instructions sequencer the type of the inst (1 = Serialize, 0 = Non-Seriali:	at the specified addres ruction (LSB) (1 = Textu zed).	in the instruction mer re, 0 = ALU and whethe	nory. The Instruction r to serialize or not th	type field tells the e execution (MSB)				
47 40 40		NOP 10 0						
47 40 43 Addressing 0010		RESERVED						
This is a regular NOP.								
	Conc	itional Execute						
47         46 43         4           Addressing         0011         Cont	42         41 34           dition         Boolean           address         Inst	<u>3316</u> ructions type + serialize instructions)	<u>15 12</u> (9 <u>Count</u>	<u>11 0</u> Exec Address				
If the specified Boolean (8 bit instructions (up to 9 instruction	ts can address 256 Boo ns). If the condition is no	eans) meets the specif t met, we go on to the n	ed condition then exe ext control flow instru	ecute the specified ction.				
47 46 40	Conditiona	Execute Predicates	10 15 10	14 0	]			
<u>47</u> <u>4643</u> Addressing 0010 <u>Co</u>	<u>42</u> <u>41 36</u> ondition <u>RESERVED</u>	<u>35 34</u> 33 <u>Predicate</u> Instruct	ions Count	Exec Address	-			
		vector type + se (9 instrue	rialize ctions)					
Check the AND/OR of all cur instructions. We need to ANI condition is not met, we go on	rent predicate bits. If A D/OR this with the kill n to the next control flow	ND/OR matches the co nask in order not to co nstruction.	ndition execute the s nsider the pixels that	pecified number of aren't valid. If the				
		Loop Start			]			
47 46 43 Addressing 0101	42 RFS	17 ERVED	16 12 loop ID	<u> </u>	-			
Loop Start. Compares the loo jump only. Also computes the control flow constants should	op iterator with the end index value. The loop id be used with the loop.	value. If loop condition must match between t	not met jump to the ne start to end, and a	address. Forward lso indicates which	1			
Evbikit 2028 dor P400 Sommers des 7	73201 Butes**** @ ATI Confi	lential Reference Cor	vright Notice on Co	ver Page @ ***				
<u>cxiiidii 2020.000</u> 8444 <u>4</u> _309408666.doc /			yngnt wolice on oo	ven I age ₩				

ATI Ex. 2107 IPR2023-00922 Page 226 of 260

	ORIGI	NATE DATE		EDIT DAT	re	DOC	UMENT-REV. NU	JM.	PAGE
	24 Sept	tember, 2001	<u>4 S</u>	4 September, 201549 GEN-CXXXXX-REVA 25 of 58				25 of 58	
									L
47	16 13	12	2017	Loop	End	a 17	16 1	2	11 0
Addressing 0011 RESERVED Predicate break loop ID start address									
Loop end. In continue, else (specified by p The way this i	crements tl e, jump BA predicate br is described	he counter b \CK to the s reak number) I does not pre	y one, co tart of the . If all bits	ompares the loop. If p cleared the ed loops, ar	e loop co predicate t in break the nd the inclu	unt with t preak != 0 e loop. usion of th	he end value. If ), then compares e loop id make thi	loop con predicat s easy to	dition met, te vector n do.
Conditionnal_Call									
47 Addressing	<u>46 43</u> 0111	<u>42</u> Condition	41 RES	37 ERVED	<u>35</u> Predicate	. 34 e vector	<u>33 12</u> RESERVED	<u>1</u> Jump	1 0 o address
If the condition	n is met, jur	mps to the sp	ecified add	tress and p	ushes the	control flo	w program counte	er on the s	stack.
17	16 12			Ret	turn	2 0			
Addressing	1000				RES	SERVED			
Pops the topr continue to th	nost addres e next instru	ss from the st uction.	tack and ji	umps to tha	at address	. If nothing	is on the stack,	the progra	am will just
A.77	10 10	40	44 34	Condition	nal Jump	> 10		44 0	
<u>4/</u> Addressing	<u>40 43</u> 1001	<u>4∠</u> Condition	<u>41 34</u> Boolean	53 FW only	RES	SERVED	Ju	np addre	<u>SS</u>
			address						•
				Allo	cate	,			
47 Debug	46 43	4	241	D	40 4		<u>3</u>	.0	
Buffer Select 01 – position 10 – paramete 11 – pass thru If debug is set	takes a valu export (orde er cache or u (out of ord t this is a de	ue of the follow ered export) pixel export ( ler exports). ebug alloc (igr	wing: ordered e nore if deb	xport) ug DB_ON	l register is	set to off)	<u>-</u>		
				End Of I	Program				
47 RESERVED	46 43				pp	420			
	1 1911	1			i XL				
Marks the end	d of the prog	gram.							
6.3 Imple	ementa	tion							4
The envisione location in the enter. Actu would allow for 16 entries for 48 entries for	ed impleme buffer duri jally two bu pr: vertices pixels.	entation has a ng its entire li uffers are ma	a buffer th ife, but th intained -	nat maintair e buffer has - one for V	ns the stat s FIFO qua ′ertices an	te of each alities in th id one for	thread. A thr at threads leave i Pixels. The inter	ead lives n the orde ded impl	in a given er that they ementation
Exhibit 2028.doc	R400_Sequencer.d	lec 73201 Bytes**	•• © ATI (	Confidentia	al. Referer	асе Соруг	ight Notice on C	over Pag	e © ***

ATI Ex. 2107 IPR2023-00922 Page 227 of 260

	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE		
<u>Can</u>	24 September, 2001	4 September, 201519		26 of 58		
From each but ALU unit. Or execution unit instructions has the thread to be Each entry in device. Only be read port devit	uffer, arbitration logic attended is selected t. It is returned to the busine been executed. As a been executed. As a been executed to the buffer. the buffer will be stored at botto and the buffer will be stored at the buffer will be termed 'state'.	empts to select 1 thread fo it is read out of the buffe (ffer (at the same place) v witch from ALU to TEX or across two physical pieces pitration will be stored in a The bits kept in the multi-re	r the texture unit and 1 (interleaved) r, marked as invalid, and submitted 1 vith its status updated once all possil visa-versa or a Serialize Execution n of memory - most bits will be stored ir highly multi-ported structure. The bits ad ported device will be termed 'statu	thread for the to appropriate ble sequential modifier forces n a 1 read port s kept in the 1 is'.		
<u>'State Bits' ne</u>	eded include:					
1. Contr 2. Exect 3. Loop 4. Call rr 5. Predit 6. Expor 7. Parar 8. GPR 9. Conte 10. LOD 0	ol Flow Instruction Pointe ution Count Marker 4 bits) lterators (4x9 bits), eturn pointers (4x12 bits), cate Bits(4x64 bits), t ID (1 bit), neter Cache base Ptr (7 the Base Ptr (8 bits), ext Ptr (3 bits), corrections (6x16 bits)	r <u>(12 bits),</u> 		<.	Formatted:	Bullets and Numbering
Absent from t the GPRs. Th bits, PC base progress has throughout ex 'Status Bits' no	his list are 'Index' pointer the first seven fields above the ptr and export ID) are been mode on thread e ecution of the thread. eeded include:	s. These are costly enouge (Control Flow Ptr, Execu- updated every time the the xecution. GPR Base Ptr	that I'm presuming that they are ins tion Count, Loop Counts, call return p read is returned to the buffer based , Context Ptr and LOD corrections a	stead stored in otrs, Predicate on how much re unchanged		
<ul> <li>Valid Three</li> <li>Texture/A</li> <li>Texture R</li> <li>Waiting of</li> <li>Allocation</li> <li>O0 - No a</li> <li>O1 - Position</li> <li>10 - Para</li> <li>11 - pass</li> <li>Allocation</li> <li>Position A</li> <li>First threa</li> <li>Event three</li> <li>Last (1 bit)</li> </ul>	ead LU engine needed leads are outstanding n Texture Read to Compl Wait (2 bits) Illocation needed tion export allocation need meter or pixel export need thru (out of order export) Size (4 bits) Nocated ad of a new context ead (NULL thread that need)	ete ded (ordered export) ded (ordered export) eds to trickle down the pipe	<u>.)</u>	•	- <b>Formatted:</b>	Bullets and Numbering
All of the abo winner for bo pixels and on summary only	the fields from all of the th the Texture Engine ar le for vertices. A final s considers the 'first' level	y will select a tion one for nplementation				
Texture arbitration the Texture E	ation requires no allocation ngine.	on or ordering so it is purely	/ based on selecting the 'oldest' thread	d that requires		
ALU arbitration Waiting on T further filtered	on is a little more comp exture Read to Comple based on whether space	blicated. First, only thread te are '0' are considered. <sup>-</sup> e is available. If the alloc	Is where either of Texture Reads of Then if Allocation Wait is active, thes ation is position allocation, then the	outstanding or se threads are thread is only		
Exhibit 2028.doc	R400_Sequencer.doc 73201 Bytes**:	🕲 ATI Confidential. Ref	erence Copyright Notice on Cover F	Page © ***		

ATI Ex. 2107 IPR2023-00922 Page 228 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	27 of 58	

considered if all 'older' threads have already done their position allocation (position allocated bits set). If the allocation is parameter or pixel allocation, then the thread is only considered if it is the oldest thread. Also a thread is not considered if it is a parameter or pixel or position allocation, has its First thread of a new context bit set and would cause ALU interleaving with another thread performing the same parameter or pixel or position allocation. Finally the 'oldest' of the threads that pass through the above filters is selected. If the thread needed to allocate, then at this time the allocation is done, based on Allocation\_Size. If a thread has its "last" bit set, then it is also removed from the buffer, never to return.

If I now redefine 'clauses' to mean 'how many times the thread is removed from the thread buffer for the purpose of exection by either the ALU or Texture engine', then the minimum number of clauses needed is 2 -- one to perform the allocation for exports (execution automatically halts after an 'Alloc' instruction) (but doesn't performs the actual allocation) and one for the actual ALU/export instructions. As the 'Alloc' instruction could be part of a texture clause (presumably the final instruction in such a clause), a thread could still execute in this minimal number of 2 clauses, even if it involved texture fetching.

The Texture Reads Outstanding bit must be updated by the sequencer, based on keeping track of how many Texture Clauses have been executed by a given thread that have not yet had there data returned. Any number above 0 results in this bit being set. We could consider forcing synchronization such that two texture clauses for a given thread may not be outstanding at any time (that would be my preference for simplicity reasons and because it would require only very little change in the texture pipe interface). This would allow the sequencer to set the bit on execution of the texture clause, and allow the texture unit to return a pointer to the thread buffer on completion that clears the bit.

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

The basic model is as follows:

### The render state defined the clause boundaries:

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

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

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

The control program has nine basic instructions:

Execute Conditional\_execute Conditional\_Execute\_Predicates Conditional\_jump Conditional\_jump Conditional\_Call Return Loop\_start Loop\_end NOP

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

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

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

Loop\_end increments (decrements?) the loop counter and jumps back the specified number of instructions. Conditionnal\_Call jumps to an address and pushes the IP counter on the stack if the condition is met. On the return

instruction, the IP is popped from the stack.

Exhibit 2028.doc R400\_Sequencer.doc 73201 Bytes\*\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

### AMD1044\_0257363

ATI Ex. 2107 IPR2023-00922 Page 229 of 260

		1		1					
	AP	ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE				
*****	(AU)	24 September, 2001	4 September, 201519		28 of 58				
	Conditional_execute_Predicates executes a block of instructions if all bits in the predicate vectors meet the condition. Conditional_jumps jumps to an address if the condition is met. NOP is a regular NOP								
	NOTE THAT ALL JUMPS MUST JUMP TO EVEN CFP ADDRESSES since there are two control flow instructions per memory line. Thus the compiler must insert NOPs where needed to align the jumps on even CFP addresses.								
*****	Also if the jump is logically bigger than pshader_cntl_size (or vshader_cntl_size) we break the program (clause) and set the debug registers. If an execute or conditional_execute is lower than cntl_size or bigger than size we also break the program (clause) and set the debug registers.								
	We have to fi store.	t instructions into 48 bits	in order to be able to put t	two control flow instruction per line in t	the instruction				
	A value of 1 i field) is an AE to the base of	in the Addressing means SOLUTE address. If the the current shader progra	that the address specified addressing field is cleared am.	d in the Exec Address field (or in the I (should be the default) then the addr	jump-address ess is relative				
	Note that wh	enever a field is marked	l as RESERVED, it is assi	umed that all the bits of the field are	cleared (0).				
	Execute up to instructions of	4k instructions at the spo the clause.	ecified address in the instru	uction memory. If Last is set, this is the	e last group of				
	This is a regu	lar NOP. If Last is set, thi	s is the last instruction of th	<del>ne clause.</del>					
*****	If the specified Boolean (8 bits can address 256 Booleans) meets the specified condition then execute the specified instructions (up to 4k instructions). If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.								
	Check the AND/OR of all current predicate bits. If AND/OR matches the condition execute the specified number of instructions. We need to AND/OR this with the kill mask in order not to consider the pixels that aren't valid. If Last is set, then if the condition is met, this is the last group of instructions to be executed in the clause. If the condition is not met, we go on to the next control flow instruction.								
	Loop Start. C jump only. Ale control flow co	ompares the loop iterato to computes the index va onstants should be used v	r with the end value. If lo lue. The loop id must mate with the loop.	op-condition not met jump to the add h between the start to end, and also in	ress. Forward ndicates which				
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 i	s described does not prev	vent nested loops, and the	inclusion of the loop id make this easy	<del>' to do.</del>				
	If the condition	n is met, jumps to the spe	cified address and pushes	the control flow program counter on th	<del>re stack.</del>				
	Pops the topr continue to th	nost address from the sta e next instruction.	ack and jumps to that add	ress. If nothing is on the stack, the pro	og <del>ram will just</del>				
	If condition me compiler and	et, jumps to the address. should NOT be exposed 1	FORWARD jump only allow to the API.	wed if bit 31 set. Bit 31 is only an optim	ization for the				
	Exhibit 2028.doc	R400_Sequencer.doc 73201 Bytes**1	* © ATI Confidential. Ref	erence Copyright Notice on Cover P	age © ***				

ATI Ex. 2107 IPR2023-00922 Page 230 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	29 of 58	
To prevent inf counter goes debug GPRs.	i <mark>nite loops, we will keep 9</mark> higher than 255 then the l	bits loop iterators instead o oop_end or the loop_start ii	of 8 (we are only able to loop 256 ti struction is going to break the loop	mes). If the and set the	
6.3 <u>6.4</u> Da	ata dependant pr	Formatted: Bullets and Numbering			
Data dependa three vector/so	nt conditionals will be sup calar predicate operations o	ported in the R400. The onl of the form:	y way we plan to support those is by	supporting	
	PRED_SETE_# - simila PRED_SETNE_# - simil PRED_SETGT_# - simila PRED_SETGTE_# - simi	er			
For the scalar	operations only we will also PRED_SETE0_# – SETE PRED_SETE1_# – SETE	o support the two following ii E0 E1	nstructions:		
The export is a maintain 4 set exposed) and sign is used to	a single bit - 1 or 0 that is s s of 64 bit predicate vector use it to control the write m specify which predicate se	eent using the same data pairs (in fact 8 sets because we hasking. This predicate is no et you want to use 0 thru 3.	h as the MOVA instruction. The seq interleave two programs but only 4 w maintained across clause boundarie	uencer will vill be s. The #	
Then we have we execute or	two conditional execute bit 1 or 0. For example, the ir	ts. The first bit is a condition nstruction:	al execute "on" bit and the second bit	tells us if	
P0_AI	DD_# R0,R1,R2				
Is only going to only write the i sequencer wit	o write the result of the ADI results to the GPRs whose h a PRED instruction is uno	D into those GPRs whose pr predicate bit is set. The use lefined.	edicate bit is 0. Alternatively, P1_ADI of the P0 or P1 without precharging t	D_# would he	
{Issue: do we	have to have a NOP betwe	en PRED and the first instru	ction that uses a predicate?}		
6.46.5 HN	N Detection of P	V,PS		*-	
Because of the masked writes comparing the insert NOPs v	he control program, the co s and subsequent reads t read address and the writ wherever there is a depend	mpiler cannot detect statica he sequencer will insert us e address of consecutive ins ant read/write.	ally dependant instructions. In the ca les of PV,PS as needed. This will I structions. For masked writes, the sec	ase of non- be done by quencer will	
The sequence	r will also have to insert NC	OPs between PRED_SET ar	d MOVA instructions and their uses.		
6.5 <u>6.6</u> Re	egister file indexir	ng		4	Formatted: Bullets and Numbering
Because we c data created i register indexi	an have loops in fetch clain n a fetch clause loop and ng and the instruction will c	use, we need to be able to i use it into an ALU clause. contain these controls:	ndex into the register file in order to Fhe instruction will include the base	retrieve the address for	
	Bit7 Bit 6 0 0 0 1 1 0 1 1	'absolute register' 'relative register' 'previous vector' 'previous scalar'			
In the case of base address	an absolute register we ju and we add to it the loop_i	st take the address as is. Ir ndex and this becomes our r	the case of a relative register read new address that we give to the shade	we take the er pipe.	
Exhibit 2028.docR	1400_Sequencer.doc 73201 Bytes*** (	3 ATI Confidential. Refere	nce Copyright Notice on Cover Pag	je © ***	

ATI Ex. 2107 IPR2023-00922 Page 231 of 260

	<b>∕</b> Tî	ORIGINATE DATE 24 September, 2001	EDIT DATE <u>4 September, 2015</u> 49	R400 Sequencer Specification	PAGE 30 of 58		
_	The sequencer is going to keep a loop index computed as such:						

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

We loop until loop\_iterator = loop\_count. Loop\_step is a signed value [-128...127]. The computed index value is a 10 bit counter that is also signed. Its real range is [-256,256]. The tenth bit is only there so that we can provide an out of range value to the "indexing logic" so that it knows when the provided index is out of range and thus can make the necessary arrangements.

Predicated Instruction support for Texture clauses

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

# 6.66.7 Debugging the Shaders

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

# 6.6.16.7.1 Method 1: Debugging registers

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

The sequencer will detect the following groups of errors:

- count overflow
- constant indexing overflow - register indexing overflow

Compiler recognizable errors: - jump errors

relative jump address > size of the control flow program

- call stack

call with stack full return with stack empty

A jump error will always cause the program to break. In this case, a break means that a clause will halt execution, but allowing further clauses to be executed.

With all the other errors, program can continue to run, potentially to worst-case limits. The program will only break if the DB\_PROB\_BREAK register is set.

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

{ISSUE : Interrupt to the driver or not?}

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

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

> 2)Debug Kill 2)1)Debug Addr + Count

Exhibit 2028.doc R400\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257366

Formatted: Bullets and Numbering

ATI Ex. 2107 IPR2023-00922 Page 232 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	31 of 58

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 <u>(debug kill OR debug Addr + count)</u>, it is assumed that <u>the programelause 7</u> is always exporting <u>42-n</u> debug vectors and that all other exports to the SX block (position, color, z, ect) will been turned off (changed into NOPs) by the sequencer (even if they occur before the address stated by the ADDR debug register).

# 7. Pixel Kill Mask

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

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

### 8. Multipass vertex shaders (HOS)

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

### 9. Register file allocation

The register file allocation for vertices and pixels can either be static or dynamic. In both cases, the register file in managed using two round robins (one for pixels and one for vertices). In the dynamic case the boundary between pixels and vertices is allowed to move, in the static case it is fixed to 128-VERTEX\_REG\_SIZE for vertices and PIXEL\_REG\_SIZE for pixels.

Exhibit 2028.docR400\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257367

ATI Ex. 2107 IPR2023-00922 Page 233 of 260



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 2028.docR400\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257368

ATI Ex. 2107 IPR2023-00922 Page 234 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	]
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	33 of 58	

# 12. Handling Stalls

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

# 13. Content of the reservation station FIFOs

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

# 14. The Output File

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

# 15. IJ Format

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

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

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

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

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

Exhibit 2023.doc/R409\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257369

ATI Ex. 2107 IPR2023-00922 Page 235 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE
24 September, 2001	4 September, 201519		34 of 58

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(| != 0) {
              P0 = A
           } else if(J != 0) {
              P0 = B;
           } else {
              PO = 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 2028.docR409\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257370

ATI Ex. 2107 IPR2023-00922 Page 236 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	1000000 C
24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	35 of 58	12020102020

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 <u>Figure 11Figure 12</u>. 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 <u>Figure 10Figure 11</u>.

Basis for 8-deep Latch Memory (fror	n R300)			
8x24-bit	11631 $\mu^2$		$60.57813\mu^2\text{per}$ bit	
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	
		Ī	758520 u <sup>2</sup>	

Figure 1011:Area Estimate for VGT to Shader Interface

Exhibit 2028.doc R400\_Sequencer.doc 73201 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257371

ATI Ex. 2107 IPR2023-00922 Page 237 of 260



# 17. The parameter cache

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

MEMORY NUMBER	ADDRESS
4 bits	7 bits

The PA generates the parameter cache addresses as the positions come from the SQ. All it needs to do is keep a Current\_Location pointer (7 bits only) and as the positions comes increment the memory number. When the memory number field wraps around, the PA increments the Current\_Location by VS\_EXPORT\_COUNT\_7 (a snooped register from the SQ). As an example, say the memories are all empty to begin with and the vertex shader is exporting 8 parameters per vertex (VS\_EXPORT\_COUNT\_7 = 8). The first position received is going to have the PC address 00000000000 the second one 00010000000, third one 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> 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 2\*VS\_EXPORT\_COUNT \_-7to Current\_Location and reset the memory count to 0 before the next vector begins).

Exhibit 2028.doc.R400\_Sequences.dec 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257372

ATI Ex. 2107 IPR2023-00922 Page 238 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
CAUL	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	37 of 58	
17.1 Exp	ort restrictions			*-	Formatted: Bullets and Numbering
177 1 1 00:	in a superstant				
<u>17.1.1 Pl</u>	<u>xer exports:</u>				
Pixels can ex function has t be ordered to	port 1,2,3 or 4 color buffers o be turned of if the exports the SX.	are done using interleaved	s will be done in order. The PREL predicated instructions. The export	<u>s will always</u>	
<u>17.1.2</u> Ve	ertex exports:	Formatted: Bullets and Numbering			
Position or pa posistion as a placed betwee The PRED_O the Paramete	arameter caches can be ex soon as possible. Position f en the exports). Parameter of PTIMIZE function has to be r cache (see Arbitration rest	xported in any order in the nas to be exported in a sing sache exports can be done in turned of if the exports are of rictions for details). The expo	shader program. It is always bett le export block (no texture instruct n any order with texture instructions lone using interleaved predicated in orts will always be allocated in order	tions can be interleaved. interleaved of the structions to to the SX.	
17.1.3 Pa	ass thru exports:			*-	
Pass thru exp	orts have to be done in grou	ips of the form:			
Alloc 4 (8 Execute AL	<u>or 12)</u> U(ADDR) ALU(DATA) ALU	(DATA) ALU(DATA)			
They cannot ordered.	have texture instructions ir	nterleaved in the export blo	ck. These exports are not guara	nteed to be	
Also, when do used to synch	ping a pass thru export, Pos ironize the chip when doing	ition MUST be exported AFT a transition from pass thru sł	ER all pass thru exports. This posit nader to regular shader and vice ve	ion export is	
17.2 Arb	itration restriction	<u>S</u>		4-	Formatted: Bullets and Numbering
Here are the S	Sequencer arbitration restric	tions:			
1) Cann 2) Cann 3) If last thread 4) Cann 5) Cann	ot execute a serialized threa of allocate position if any old thread is marked as not va d also marked last then: . Both threads must be fror . Must turn off the predicate of execute a texture clause is of execute last if texture per	 and to oldest	( Formatted: Bullets and Numbering		
18 Vorte	x position exporti	ina		*-	Formatted: Bullets and Numbering
On clause 3 t clause 7 if no the position a The clause w the export is g	he vertex shader can export t done at clause 3. The sto nd 64x32 memories for the here the position export occ going to occur at ALU clause	t to the PA both the vertex-p rage needed to perform the sprite size. It is going to be t sure is specified by the EXP - 7 if unset position export oc	osition and the point sprite. It can a position export is at least 64x128 r aken in the pixel output fifo from th ORT_LATE register. If turned on, if curs at clause 3.	also do so at nemories for e SX blocks. t means that	
	orting Arbitration	Formatted: Bullets and Numbering			
Here are the	allee for on-issuing experting				
<del>1)Poe</del>	sition exports and position ex				
All other types	s of exports can be co-issue	d as long as there is place in	the receiving buffer.		
Exhibit 2028.doc	7400_Sequencer.dec 73201 Bytes*** ⓒ	ATI Confidential. Referen	ce Copyright Notice on Cover Pa	ge © ***	

ATI Ex. 2107 IPR2023-00922 Page 239 of 260

-						
	ORIGINATE DAT	E EDIT DATE	R400 Sequencer Specification	PAGE		
	24 September, 20	01 <u>4 September, 2015</u> 19		38 of 58	Formatted: Bullets and Numbering	)
20. Expo	rting Rules			*		
20.1 Para	ameter cache	s exports				
We-support-ma	asking and out of orc	ler exports to the parameter c	aches. So one can export multiple time	s to the same		
PC-line-using o	lifferent masks.				Formatted: Bullets and Numbering	
20.2 Men	nory exports	.,		•	<u> </u>	
Memory export	ts don't support mas	ang. However, you can export	out of order to memory locations.	**	Formatted: Bullets and Numbering	$\overline{)}$
20.3 POSI	tion exports					
Position expon	is have to be done in	I OKDER and don't support m	asking.		<b>Formatted:</b> Bullets and Numbering	
<u>21.18. Ex</u>	port Types			*		
The export typ ALU instructior	e (or the location wh n. Here is a list of all	ere the data should be put) is possible export modes:	s specified using the destination addre	ss field in the		
21-1181	Vertex Shadi	na		*-		$\Box$
	0:15 - 16 param	ieter cache				
	16:31 - Empty (R 32 - Export A	eserved?) ddress				
	33:40 - 8 vertex e	xports to the frame buffer and	index			
	48:55 - 8 debug	export (interpret as normal ver	tex export)			
	60 - export ad 61 - Empty	dressing mode				
	62 - position 63 - sprite size	e export that goes with positior	n export			
	(point_h,p	oint_w,edgeflag,misc)				
<u>21.2</u> 18.2	Pixel Shadin	g			Formatted: Bullets and Numbering	
	0 - Color for	buffer 0 (primary)				
	2 - Color for	buffer 2				
	3 - Color for 4:7 - Empty	buffer 3				
	8 - Buffer 0 0 9 - Buffer 1 0	Color/Fog (primary)				
	10 - Buffer 2 C	Color/Fog				
	12:15 - Empty	201017F0g				
	16:31 - Empty (R 32 - Export A	eserved?) ddress				
	33:40 - 8 exports 41:47 - Empty	for multipass pixel shaders.				
	48:55 - 8 debug e	exports (interpret as normal pivelarssing mode	(el export)			
	61:62 - Empty					
	o3 -∠for prim	ary puπer (∠ exported to 'alph	a component)			
Exhibit 2028.docR4	100_Sequencer.doc 73201 By	tes*** © ATI Confidential. Ref	erence Copyright Notice on Cover F	'age © ***		

ATI Ex. 2107 IPR2023-00922 Page 240 of 260

	ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	
	24 September, 2001	4 September, 201549	GEN-CXXXXX-REVA	39 of 58	Sec Formatted: Bullets and Numbering
<u>22.19. Sp</u>	pecial Interpolation	n modes		*1	
22 119 1	Real time comma	ands			
We are unable need to add th register bus at be able to add should be able other is raster memory to 16 view support stream), then parameters in: the SX blocks	e to use the parameter men nree 16x128 memories (one nd written by type 0 packets lress the reatime parameter e able to view them as two ized with. Most overlay sha x64 or 32x64 allowing only for 16 vector-4 interpolants i the PA/sequencer need to stead of 16. This mode is tri . The parameter data memo	for each of three vertices x a, and output to the the para memory as well as the reg banks of 16 and do doubl ders will need 2 or 4 scalar two interpolated scalars pe mportant (true only if we ma support a realtime-specific ggered by the primitive typo pries are hooked on the RB	or a command stream to write into it 16 interpolants). These will be map imeter busses (the sequencer and/or ular parameter store. For higher perf e buffering allowing one to be loade r coordinates, one option might be to r cycle, the only problem I see with ap Microsoft's high priority stream to mode where we need to address 3 2: REAL TIME. The actual memories BM bus and are loaded by the CP u	Instead we ped onto the r PA need to ormance we ed, while the prestrict the this is, if we the realtime 2 vectors of are in the in sing register	
mapped memo	ory. Sorites/XV scree	en coordinates/ Fl	3 information	<b>4</b>	Formatted: Bullets and Numbering
When working coordinates m conjunction wi special operat together:	g with sprites, one may wa hay be needed in the shade th the SND_XY register (in ions) to the shader using th	nt to overwrite the parame er program. This functionali SC). Also it is possible to se e same control register. He	ter 0 with SC generated data. Also ty is controlled by the gen_l0 registe end the faceness information (for OG re is a list of all the modes and how	, XY screen er (in SQ) in BL front/back they interact	
Gen_st is a bi set, it means v between 0 and	t taken from the interface be we are dealing with Point AA d 1.	etween the SC and the SQ. A, Line AA or sprite and in th	This is the MSB of the primitive type his case the vertex values are going t	e. If the bit is o generated	
Param_Gen_I Param_Gen_I Param_Gen_I Param_Gen_I Param_Gen_I Param_Gen_I Param_Gen_I	0 disable, snd_xy disable, n 0 disable, snd_xy disable, g 0 disable, snd_xy enable, n 0 disable, snd_xy enable, g 0 enable, snd_xy disable, n 0 enable, snd_xy disable, g 0 enable, snd_xy enable, g 0 enable, snd_xy enable, g	o gen_st – I0 = No modifica en_st – I0 = No modification o gen_st – I0 = No modification o gen_st – I0 = No modification o gen_st – I0 = garbage, ga en_st – I0 = garbage, garba o gen_st – I0 = screen x, screen an_st – I0 = screen x, screen	ition n tion rbage, garbage, faceness ige, s, t reen y, garbage, faceness n y, s, t		
<u>22.3</u> 19.3	Auto generated c	ounters		4-	Formatted: Bullets and Numbering
In the cases v both use this of The count is a the shader typ keep two cound GPRs the cound there is only of all elements in	ve are dealing with multipas count to write the 1 <sup>st</sup> pass of lways generated in the sam be (pixel or vertex). This is to nters, one for pixels and on unter is incremented. Every ne count broadcast to the G in the vector.	as shaders, the sequencer is lata to memory and then us way but it is passed to the oggled on and off using the le for vertices. Every time a time a state change is det iPRs, the LSB are hardwire	is going to generate a vector count t se the count to retrieve the data on t e shader in a slightly different way d GEN_INDEX register. The sequence a full vector of vertices or pixels is v ected, the corresponding counter is d to specific values making the index	o be able to he 2 <sup>nd</sup> pass. epending on er is going to vritten to the reset. While different for	
<del>22.3.1</del> 19.1	3.1_Vertex shaders			4-	Formatted: Bullets and Numbering
In the case of that the compi	vertex shaders, if GEN_IN ler must allocate 3 GPRs in	DEX is set, the data will be all multipass vertex shader	put into the x field of the third regist modes).	er (it means	
<u>22.3.2</u> 19.1	3.2 Pixel shaders			4-	
In the case of the 2 <sup>nd</sup> registe	pixel shaders, if GEN_INDI r (R1.x), else if GEN_INDE)	EX is set and Param_Gen_ ( is set the data will be put i	IO is enabled, the data will be put in the the time the the time the the time the time the the time the time the time the time time the time time time time time time time tim	the x field of	
Exhibit 2028.docF	1400_Sequencer.doc 73201 Bytes*** ©	ATI Confidential. Refere	nce Copyright Notice on Cover Pa	ge © ***	

ATI Ex. 2107 IPR2023-00922 Page 241 of 260



ATI Ex. 2107 IPR2023-00922 Page 242 of 260

	ORIGINAT	E DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE		
	24 Septemb	er, 2001	4 September, 201519	GEN-CXXXXX-REVA	41 of 58	Formatted: Bullets and Numbering	
2 <u>5.22. Re</u>	<u>egisters</u>				*1		
25.122.1	Control						
REG_DYN REG_SIZE	IAMIC E_PIX	Dynamic Size of th	allocation (pixel/vertex) of the register file's pixel portion	ne register file on or off. (minimal size when dynamic allocati	on turned		
REG_SIZE	E_VTX	Size of th	ne register file's vertex portio	n (minimal size when dynamic alloca	tion turned		
ARBITRAT	FION_POLICY SE_VTX	policy of start poin Begins at	the arbitration between verte it for the vertex instruction st t 0)	exes and pixels ore (RT always ends at vertex_base	and		
INST_BAS ONE_THR ONE_ALU	SE_PIX READ	start poin debug sta debug sta of 2)	t for the pixel shader instruc ate register. Only allows one ate register. Only allows one	tion store program at a time into the GPRs ALU program at a time to be execu	ted (instead		
INSTRUC	TION	This is w incremen	here the CP puts the base a ited on reads/writes) Registe	address of the instruction writes and r mapped	type (auto-		
CONSTAN CONSTAN CONSTAN	NTS_RT NTS_RT NT_EO_RT	512*4 AL 256*4 AL This is th CONSTA This is th	U constants + 32*6 Texture U constants + 32*6 texture s is size of the space reserve NT_EO_RT). The re-mapping the size of the space reserve	state 32 bits registers (logically map states? (physically mapped) ed for real time in the constant storr ng table operates on the rest of the r d for real time in the fatch state stor	ped) e (from 0 to nemory e (from 0 to		
	EControls whe	TSTATE	_EO_RT). The re-mapping to	able operates on the rest of the merr	iory		
lause 7.			the are experting position i		to occur at		
<u>25.222.2</u>	Context				*-	- Formatted: Bullets and Numbering	$\square$
VS_FETC VS_ALU_I PS_BASE VS_CF_SI PS_CF_SI PS_SIZE VS_SIZE PS_NUM_ VS_NUM_ PARAM_S PROVO_V PARAM_V PS_EXPO VS_EXPO	H_{07} [07] H_{07} [2E [2E [2E [2E [2E [2E [2E [2E	eight 8 bi eight 8 bi eight 8 bi eight 8 bi base poir base poir size of th size of th size of th size of th size of th number of One 16 b e gourau 0 : vertex 64 bits: fo (0=linear 0xxxx : N 1xxxx : N If normal If multipa 0: positio	t pointers to the location whit t pointers to the location whit there for the pixel shader in the evertex shader (# of instruction e pixel shader (# of instruction e pixel shader (cntl+instruction of GPRs to allocate for pixel of GPRs to allocate for pixel of GPRs to allocate for vertex it register specifying which p d) =0, 1: vertex 1, 2: vertex 2, 3 or which parameters (and ch , 1=cylindrical). ormal mode lultipass mode bbbz where bbb is how ma ss 1-12 exports for color. n (1 vector), 1: position (2 vector) (06) Six 4 bit course seported in clause 7 (locations)	ere each clauses control program is- ere each clauses control program is- ere each clauses control program is- ere each clauses control program is- e instruction store the instruction store tions in control program/2) ons in control program/2) ons in control program/2) shader programs shader programs canameters are to be gouraud shade i: Last vertex of the primitive annels (xyzw)) do we do the cyl wra ny colors (0-4) and z is export z or n- ectors), 3:multipass 1 by the VS (and thus number of inter hters representing the # of interpolation ated in VS_EXPORT_COUNT_6) OF	located locate		
		Do we ov	red vectors to memory per-	O with XY data and generated T and o with XY data and generated T and	e) d S values		
PARAM_G	SEN_IO	Do we ov	erwrite of not the parameter	-			

ATI Ex. 2107 IPR2023-00922 Page 243 of 260

	ATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE		
24 Septe	mber, 2001	4 September, 201519	R400 Sequencer Specification	42 of 58		
GEN_INDEX	Auto g	enerates an address from	0 to XX. Puts the results into R0-1 for	pixel shaders		
CONST_BASE_VTX CONST_BASE_PIX ( CONST_SIZE_PIX (8 CONST_SIZE_VTX (7 INST_PRED_OPTIMI CF_BOOLEANS CF_LOOP_COUNT CF_LOOP_START CF_LOOP_STEP	and R 9 bits) Logica bits) Logica bits) Size o bits) Size o ZE Turns always 256 bc 32x8 b 32x8 b 32x8 b	2 for vertex shaders I Base address for the con f the logical constant store f the logical constant store on the predicate bit optimiz s executed). solean bits bit counters (number of time it counters (step value used it counters (step value used	stants of the Vertex shader stants of the Pixel shader for pixel shaders for vertex shaders zation (if of, conditional_execute_predi es we traverse the loop) l in index computation) ed in index computation)	cates is		Proventient Dullaterand Municipa
26-23. DEBUG F	Registers	2		•^	~ ~	romated, builds and rumbering
26.123.1 Contex	t					
DB_PROB_ADDF DB_PROB_COUT DB_PROB_BREA DB_ON DB_INST_COUN DB_BREAK_ADE DB_CLAUSE _MODE_ALU_{0- DB_CLAUSE _MODE_FETCH_	tinstruc IT numbe K break turns c r instruc R break 7} clause {07}	tion address where the firs of problems encountered the clause if an error is fou on an off debug method 2 tion counter for debug met address for method numbe mode for debug method 2 —clause mode for debug n	t problem occurred d during the execution of the program ind. thod 2 er 2 <del>! (0: normal, 1: addr, 2: kill)</del> nethod <del>2 (0: normal, 1: addr, 2: kill)</del>			
26 222 2 Control				4-		Formatted: Bullets and Numbering
DB_ALUCST_ME DB_TSTATE_ME	MSIZE MSIZE	Size of the physical ALU Size of the physical textu	constant memory ire state memory			
					15	Formatted: Bullets and Numbering
27.24. Interfaces						
27.124.1 Externa	I Interfac	ces				
whenever an x is used, i named SQ→SPx it means	t means that that SQ is g	the bus is broadcast to a oing to broadcast the same	Il units of the same name. For examp e information to all SP instances.	ole, if a bus is		
27.224.2_SC to S	P Interfa	aces		+-	к. <sup>-</sup>	Formatted: Bullets and Numbering
27.2.124.2.1 SC	SP#			4-		Formatted: Bullets and Numbering
There is one of these inter the I,J data for pixel interp these 4 interfaces transm	faces at from olation. For t its one half o	t of each of the SP (buffer the entire system, two qua of a quad per clock. The	to stage pixel interpolators). This inter ds per clock are transferred to the 4 S interface below describes a half of a	face transmits Ps, so each of quad worth of		
The actual data which is t Ref Pix I => S4.2( Ref Pix J => S4.2 Delta Pix I (x3) => Delta Pix J (x3) => This equates to a total of and therefor needs an inte	ansferred pe 9 Floating Poi 0 Floating Po S4.8 Floatin 9 S4.8 Floatin 28 bits which rface 64 bits	r quad is nt I value int J value g Point Delta I value g Point Delta J value n transferred over 2 clocks wide				
Exhibit 2028.docR400_Sequencer.do	> 73201 Bytes**	* © ATI Confidential. Ref	erence Copyright Notice on Cover F	age © ***		

ATI Ex. 2107 IPR2023-00922 Page 244 of 260

ORIGINATE DATE	EDIT DATE	DOCUMENT-REV. NUM.	PAGE	01100
24 September, 2001	4 September, 201549	GEN-CXXXXX-REVA	43 of 58	

Additionally, X,Y data (12-bit unsigned fixed) is conditionally sent across this data bus over the same wires in an additional clock. The X,Y data is sent on the lower 24 bits of the data bus with faceness in the msb. Transfers across these interfaces are synchronized with the SC\_SQ IJ Control Bus transfers.

The data transfer across each of these busses is controlled by a IJ\_BUF\_INUSE\_COUNT in the SC. Each time the SC has sent a pixel vector's worth of data to the SPs, he will increment the IJ\_BUF\_INUSE\_COUNT count. Prior to sending the next pixel vectors data, he will check to make sure the count is less than MAX\_BUFER\_MINUS\_2, if not the SC will stall until the SQ returns a pipelined pulse to decrement the count when he has scheduled a buffer free. Note: We could/may optimize for the case of only sending only IJ to use all the buffers to pre-load more. Currently it is planned for the SP to hold 2 double buffers of I,J data and two buffers of X,Y data, so if either X,Y or Centers and Centroids are on, then the SC can send two Buffers.

In at least the initial version, the SC shall send 16 quads per pixel vector even if the vector is not full. This will increment buffer write address pointers correctly all the time. (We may revisit this for both the SX,SP,SQ and add a EndOfVector signal on all interfaces to quit early. We opted for the simple mode first with a belief that only the end of packet and multiple new vector signals should cause a partial vector and that this would not really be significant performance hit.)

Name	Bits	Description
SC_SP#_data	64	IJ information sent over 2 clocks (or X,Y in 24 LSBs with faceness in upper bit)
		Type 0 or 1, First clock I, second clk J
		Field ULC URC LLC LRC
		Bits [63:39] [38:26] [25:13] [12:0]
		Format SE4M20 SE4M8 SE4M8 SE4M8
		Type 2
		Field Face X Y
		Bits [63] [23:12] [11:0]
		Format Bit Unsigned Unsigned
SC_SP#_valid	1	Valid
SC_SP#_last_quad_data	1	This bit will be set on the last transfer of data per quad.
SC_SP#_type	2	0 -> Indicates centroids
		1 -> Indicates centers
		2 -> Indicates X,Y Data and faceness on data bus
		The SC shall look at state data to determine how many types to send for the
		interpolation process.

The *#* is included for clarity in the spec and will be replaced with a prefix of u#\_ in the verilog module statement for the SC and the SP block will have neither because the instantiation will insert the prefix.

# 27.2.224.2.2 SC\_SQ

This is the control information sent to the sequencer in order to synchronize and control the interpolation and/or loading data into the GPRs needed to execute a shader program on the sent pixels. This data will be sent over two clocks per transfer with 1 to 16 transfers. Therefore the bus (approx 92 bits) could be folded in half to approx 47 bits.

1010
1903
1-1-1-1

Exhibit 2028.docR409\_Sequencer.dec 73201 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

Formatted: Bullets and Numbering

AMD1044\_0257379

ATI Ex. 2107 IPR2023-00922 Page 245 of 260

	ORIGINATE I	DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 September	, 2001	4 September, 201519		44 of 58
			Empty Quad Mask – consist or new, transfe without vector reques attache outstar if no va 2 clk transfers Quad Data Valid – S without New ve fifo without pc_dea vector this cas posted Filler q The Qu corresp	Transfer Control data ing of pc_dealloc _vector. Receipt of this is to r pc_dealloc or new_vector : any valid quad data. New will always be posted to t fifo and pc_dealloc will be id to any pixel vector iding or posted in request fifo lid quad outstanding. ending quad data with or new_vector or pc_dealloc. ector will be posted to request n or without a pixel vector and ulloc will be posted with a pixel unless none is in progress. In se the pc_dealloc will be in the request queue. uads will be transferred with uad mask set but the pixel ponding pixel mask set to	
SC SC valid		4	20 conding valid data 0 <sup>n0</sup>	ally could be all payage	
		1	SC seriority valid data, 2	cik coulu be all zeroes	

SC\_SQ\_data - first clock and second clock transfers are shown in the table below.

Name	BitField	Bits	Description
1 <sup>st</sup> Clock Transfer			
SC_SQ_event	0	1	This transfer is a 1 clock event vector
			Force quad_mask = new_vector=pc_dealloc=0
SC_SQ_event_id	[2:1]	2	This field identifies the event
			0 => denotes an End Of State Event
			1 => TBD
SC_SQ_pc_dealloc	[5:3]	3	Deallocation token for the Parameter Cache
SC_SQ_new_vector	6	1	The SQ must wait for Vertex shader done count > 0 and after
			dispatching the Pixel Vector the SQ will decrement the count.
SC_SQ_quad_mask	[10:7]	4	Quad Write mask left to right SP0 => SP3
SC_SQ_end_of_prim	11	1	End Of the primitive
SC_SQ_state_id	[14:12]	3	State/constant pointer (6*3+3)
SC_SQ_pix_mask	[30:15]	16	Valid bits for all pixels SP0=>SP3 (UL,UR,LL,LR)
SC_SQ_prim_type	[33:31]	3	Stippled line and Real time command need to load tex cords from
			alternate buffer
			000: Normal
			010: Realtime
			101: Line AA
			110: Point AA (Sprite)
SC_SQ_provok_vtx	[35:34]	2	Provoking vertex for flat shading
SC_SQ_pc_ptr0	[46:36]	11	Parameter Cache pointer for vertex 0
2nd Clock Transfer			
SC_SQ_pc_ptr1	[10:0]	11	Parameter Cache pointer for vertex 1
SC_SQ_pc_ptr2	[21:11]	11	Parameter Cache pointer for vertex 2
SC_SQ_lod_correct	[45:22]	24	LOD correction per quad (6 bits per quad)
Name	Bits D	escript	ion

Exhibit 2028.doc R400\_Sequences.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257380

ATI Ex. 2107 IPR2023-00922 Page 246 of 260

Designation of the second second states								1	
	ORIGINATE	DATE	EDIT	DATE		DOCUMENT-REV. NUM.	PAGE		
	24 Septembe	r, 2001	4 Septem	ber, 20	<u>15</u> 19	GEN-CXXXXX-REVA	45 of 58		
SQ SC free	buff	1	Pipelined bit th	nat instr	ucts SC	to decrement count of buffers in use.			
SQ_SC_dec_		1	Pipelined bit th	nat instr	ucts S0	C to decrement count of new vector ar	nd/or event		
			sent to preven	t SC fro	m over	flowing SQ interpolator/Reservation re	quest fifo.		
The scan com 1.) He ge 2.) A cur marke vector marke (This will except for new_vect the new v until the p 27.2.324. Name SQ_SXX_inte SQ_SXX_inte SQ_SXX_inte SQ_SXX_pc_p SQ_SXX_pc_p SQ_SXX_rt_se	verter will submit tes a primitive mar- rent pixel vector ed for deallocate vector r (up to 16quade er/primitive. prevent a hang vector pors are submitted vould wait for and c_dealloc signal r 2.3_SQ to S. rp_flat_vtx rp_flat_gouraud rp_cyl_wrap tr0 tr1 tr2 el	a partial ked with is being when a partial which can rimitive and pro- other ver nade it t X: Inter Directi SQ $\rightarrow$ S SQ $\rightarrow$ S	vector whenev a an end of pac assembled wi primitive marke ero pixel mas that gets mar becessed, but the rtex vector to b hrough and thu erpolator but on Px Px Px Px Xx Xx Xx Xx	er: ket sign th at le d new_ k to fil trated v ked pc en one op process the has <i>JS</i> Bits 2 1 4 11 11 11 11	al. ast one vector i l out th vhen al _deallo valid qu essed, ang.) Descri Provok Flat or Wich c Param Selects	e or more valid quads and the vector arrives. The Scan Converter will subn he vector) prior to submitting the r I primitives in a packet three vectors c (vertices maximum size). In this ad with the pc_dealloc creates a vector but the one being waited for could ne ption ing vertex for flat shading gouraud shading hannel needs to be cylindrical wrappe ater Cache Pointer eter Cache Pointer ter Cache Pointer between RT and Normal data	has been hit a partial lew_vector are culled case two or and then ver export		Formatted: Bullets and Numbering
SQ SXx pc v	/r en	SQ→S	Xx	1	Write e	nable for the PC memories			
SQ SXx pc v	vr addr	SQ→S	Xx	7	Write a	ddress for the PCs			
SQ_SXx_pc_	channel_mask	SQ→S	SXx	4	Chann	el mask			
27.2.424. This is a broad Name SQ_SPX_vsr_( SQ_SP2_vsr_( SQ_SP1_vsr_( SQ_SP2_vsr_( SQ_SP2_vsr_( SQ_SP3_vsr_( SQ_SPX_vsr_()	2.4 SQ to S dcast bus that ser data double valid valid valid valid valid read	P: Stands the N Direction $SQ \rightarrow S$ $SQ \rightarrow S$ $SQ \rightarrow S$ $SQ \rightarrow S$ $SQ \rightarrow S$ $SQ \rightarrow S$ $SQ \rightarrow S$	rging Regis VSISR informat on Px Px P2 P1 P2 P3 Px Px	ter D ion to tl Bits 96 1 1 1 1 1 1 1	ata ne stagi Pointer O: Norr Data is Data is Data is Data is Increm	ng registers of the shader pipes. ption s of indexes or HOS surface information nal 96 bits per vert 1: double 192 bits per valid valid valid valid ent the read pointers	•		Formatted: Bullets and Numbering
27.2.524. 27.2.5.124 The area diff requirements <u>VSISRs (via</u> floating-point bits wide.	2.5_VGT to 3 2.5.1 Interfact erence between of the VSISRs. <u>T</u> the Shader Sequivalues to each V	SQ : V ce Sigr the two <u>herefor</u> <u>Jencer)</u> SISR wh	/ertex inter nal Table methods is no e, the POR fo in full, 32-bit here four or mo	face ot suffic <u>r this ir</u> floating ore valu	tient to <u>nterfac</u> g <u>-point</u> les requ	warrant complicating the interface o <u>a is that the VGT will transmit the d</u> <u>format.</u> The VGT can transmit up to lire two transmission clocks. The data	* <sup>-</sup> r the state <u>lata to the</u> six 32-bit i bus is 96		

Exhibit 2028.doc.R400\_Sequencer.dec 73201 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

AMD1044\_0257381

ATI Ex. 2107 IPR2023-00922 Page 247 of 260

	ORIGINATE	DATE	EDIT DATE	R400 Sequencer Specification	PAGE
	24 Septembe	r, 2001	4 September, 201519		46 of 58
Name		Bits	Description		
VGT_SQ_vsis	r_data	96	Pointers of indexes or HOS	surface information	
VGT_SQ_vsis	r_double	1	0: Normal 96 bits per vert 1	: double 192 bits per vert	
VGT_SQ_end	_of_vector	1	Indicates the last VSISR da data, "end_of_vector" is set	ata set for the current process vector (fo t on the first vector)	or double vector
VGT_SQ_indx	_valid	1	Vsisr data is valid		
VGT_SQ_state	e	3	Render State (6*3+3 for co "VGT_SQ_vgt_end_of_vec	onstants). This signal is guaranteed to b tor" is high.	e correct when
VGT_SQ_send	d	1	Data on the VGT_SQ is va interface handshaking)	lid receive (see write-up for standard R4	400 SEND/RTR
SQ_VGT_rtr		1	Ready to receive (see handshaking)	write-up for standard R400 SEND/	RTR interface

# 27.2.5.224.2.5.2 Interface Diagrams

Formatted: Bullets and Numbering

Exhibit 2028.doc R400\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257382

ATI Ex. 2107 IPR2023-00922 Page 248 of 260

# PROTECTIVE ORDER MATERIAL



Exhibit 2028 doo R4.00\_Sequences 73201 Bytes\*\*\*\* S ATI Confidential. Reference Copyright Notice on Cover Page S \*\*\*

AMD1044\_0257383

ATI Ex. 2107 IPR2023-00922 Page 249 of 260 PROTECTIVE ORDER MATERIAL



AMD1044\_0257384

	T					r			
	ORIGINATE	DATE	EDIT	DATE		DOCUMENT-REV. NUM.	PAGE		
	24 September	r, 2001	4 Septem	ber, 20	) <u>15</u> 19	GEN-CXXXXX-REVA	49 of 58		
27.2.624	<u>2.6_</u> SQ to S	X: Conti	rol bus	<u>1 - 11 w r 1</u>			*-		Formatted: Bullets and Numbering
Name	ι	Direction	1	Bits	Descri	ption			
<u>SQ SXX exp</u>	туре	$30 \rightarrow 30$			01: Pixe	el without z (1 to 4 buffers)		N. N.	Formatted
					10: Pos	sition (1 or 2 results)			
SQ SXx exp	number	SQ→SXX	r	2	11: Pas Numbe	s thru (4,8 or 12 results aligned) or of locations needed in the e	xnort buffer		Formatted
47					(encod	ing depends on the type see bellow			
SQ SXx exp	alu_id	SQ→SXX	(	1	ALU IC	) '1		1.2.2	Formatted
SQ SXX exp	state	SQ→SXX	[	3	State C	Context		122	Formatted
SQ SXx free	done	SQ→SXx		1	Pulse	to indicate that the previous expo	t is finished		Formatted
			<u></u>	_ =	(this ca	an be sent with or without the other	fields of the	-	Formatted
	g × 9	0.0 0.		4	interfac	20)			
SQ SXX free	alu id	SQ-SXX		1	ALUIL	}			Formatted
Depending on	the type the num	ber of exp	ort location c	hange	<u>s:</u>				
<ul> <li>Type</li> </ul>	00 : Pixels withou	t Z					4	1	Formatted: Bullets and Numbering
<u>_</u>	00 = 1 buffer 01 = 2 buffers								
0	10 = 3 buffers								
<u> </u>	11 = 4 buffer								
• Type	01: Pixels with Z	(color + 7)							
0	01 = 3 buffers (	2 color + Z	.)						
0	10 = 4 buffers (	3 color + Z	<u>)</u>						
<u>.</u> 	<u>11 = 5 buffers (</u>	4 color + Z	<u>.)</u>						
• Type	10 = 1  position expo	<u>irt</u>							
<u></u>	01 = 2 position	5							
<u> </u>	1X = Undefined								
• Type	$\frac{11: \text{Pass I hru}}{00 = 4 \text{ buffers}}$								
<u></u>	01 = 8 buffers								
<u> </u>	10 = 12 buffers								
<u>o</u>	11 = Undefined	_							
Below the thic	ck black line is th	e end of t	ransfer pack	cet that	t tells the	e SX that a given export is finished	. The report		
packet will al	ways arrive eithe	er before o	r at the san	ne time	<u>e than th</u>	ne next export to the same ALU id	These fields		
are sent every	/-time-tne-sequen	ser picks a	n exporting (	slause	tor exec	ution.			
~~ ~ ~ ~ .		-					<i>A</i>		Formatted: Bullets and Numbering
21.2.724.	2.7 SX to SC	2 : Outp	out file co	ntrol					
Name		Directior	1	Bits	Descri	ption			
SXx_SQ_exp	_count_rdy	SXx→SC	!	1	Raised	by SX0 to indicate that the following	two fields		
SXx SQ exp	pos avail	SXx→SC	!	1	Specifi	es whether there is room for another	position.		
SXx_SQ_exp	_buf_avail	SXx→SG	1	7	Specifi	es the space available in the output	buffers.		
					0: buffe	ers are full	64		
					pixels i	n a clause)	04		
						,			
					64: 128	3K-bits available (16 128-bit entries t	or each of		
					64 pixe				
		1			00-127				
		-	6 MU 6		-		<u>^</u>		
Exhibit 2028.doc	R400_Sequencer.doc 732	201 Bytes*** ©	A II Confid	iential.	. Ketere	nce Copyright Notice on Cover Pa	ge © ***		

ATI Ex. 2107 IPR2023-00922 Page 251 of 260

ORIGINATE DATE	EDIT DATE	R400 Sequencer Specification	PAGE	
24 September, 2001	<u>4 September, 2015</u> 19		50 of 58	

# 27.2.824.2.8 SQ to TP: Control bus

Once every clock, the fetch unit sends to the sequencer on which <u>clause\_RS line it</u> is now working and if the data in the GPRs is ready or not. This way the sequencer can update the fetch <u>valid bits counters\_flags</u> for the reservation station\_fifes. The sequencer also provides the instruction and constants for the fetch to execute and the address in the register file where to write the fetch return data.

Name Name	DirectionDirection	<b>Bits</b> Bits	DescriptionDescription	
TPx_SQ_data_rdyTPx_SQ_data_rdy	$\frac{\text{TPx} \rightarrow \text{SQ}}{\text{SQ}} \xrightarrow{\text{SQ}}$	<u>1</u> 1	Data readyData-ready	
<u>,TPx_SQ_rs_line_num</u> T <del>Px_SQ_clause_num</del>	<u>TPx→ SQ</u> TPx→ SQ	63	Line number in the Reservation stationClause number	Formatted
TPx_SQ_typeTPx_SQ_type	$\frac{\text{TPx} \rightarrow \text{SQ}}{\text{SQ}} \xrightarrow{\text{SQ}}$	11	Type of data sent (0:PIXEL, 1:VERTEX)Type of data sent (0:PIXEL, 1:VERTEX)	
SQ_TPx_send SQ_TPx_send	<u>SQ→TPx</u> SQ→TPx	11	Sending valid dataSending valid data	
<u>SQ_TPx_const</u> SQ_TPx_const	<u>SQ→TPx</u> SQ→TPx	<u>48</u> 48	Fetch state sent over 4 clocks (192 bits total)Fetch state sent over 4 clocks (192 bits total)	
SQ_TPx_instrSQ_TPx_instr	<u>SQ→TPx</u> SQ→TPx	2424	Fetch instruction sent over 4 clocksFetch instruction sent over 4 clocks	
SQ_TPx_end_of_groupSQ_TPx_end_of_clause	<u>SQ→TPx</u> SQ→TPx	17	Last instruction of the groupLast instruction of the clause	Formatted
SQ_TPx_TypeSQ_TPx_Type	<u>SQ→TPx</u> SQ→TPx	11	Type of data sent (0:PIXEL, 1:VERTEX)Type of data sent (0:PIXEL, 1:VERTEX)	
<u>SQ_TPx_gpr_phase</u> SQ_TPx_gpr_phase	<u>SQ→TPx</u> SQ→TPx	22	Write phase signalWrite phase signal	
SQ_TP0_lod_correctSQ_TP0_lod_correct	<u>SQ→TP0</u> SQ→TP0	<u>6</u> 6	LOD correct 3 bits per comp 2 components per quad LOD correct 3 bits per comp 2 components per quad	
SQ_TP0_pix_maskSQ_TP0_pix_mask	<u>SQ→TP0</u> SQ→TP0	<u>4</u> 4	Pixel mask 1 bit per pixel mask 1 bit per pixel	
SQ_TP1_lod_correctSQ_TP1_lod_correct	<u>SQ→TP1</u> SQ→TP1	<u>6</u> 6	LOD correct 3 bits per comp 2 components per quad LOD correct 3 bits per comp 2 components per quad	
<u>SQ_TP1_pix_mask</u> SQ_TP1_pix_mask	<u>SQ→TP1</u> SQ→TP1	<u>4</u> 4	Pixel mask 1 bit per pixel mask 1 bit per pixel	
<u>SQ_TP2_lod_correct</u> SQ_TP2_lod_correct	<u>SQ→TP2</u> SQ→TP2	<u>6</u> 6	LOD correct 3 bits per comp 2 components per quad LOD correct 3 bits per comp 2 components per quad	
SQ_TP2_pix_maskSQ_TP2_pix_mask	<u>SQ→TP2</u> SQ→TP2	44	Pixel mask 1 bit per pixel mask 1 bit per pixel	
SQ_TP3_lod_correctSQ_TP3_lod_correct	<u>SQ→TP3</u> SQ→TP3	<u>6</u> 6	LOD correct 3 bits per comp 2 components per quad LOD correct 3 bits per comp 2 components per quad	
<u>SQ_TP3_pix_mask</u> SQ_TP3_pix_mask	<u>SQ→TP3</u> SQ→TP3	44	Pixel mask 1 bit per pixel mask 1 bit per pixel	]

Exhibit 2028. doc R409\_Sequences. doc 73201 Bytes\*\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257386

Formatted: Bullets and Numbering

ATI Ex. 2107 IPR2023-00922 Page 252 of 260
A A	ORIGINATE	ORIGINATE DATE           24 September, 2001         4 Set		EDIT DATE		DOCUN	IENT-REV. NUM.	PAGE	
	24 Septembe			<u>mber, 201</u>	519	GEN-0	CXXXXX-REVA	51 of 58	
<u>TPx_rs_l</u>	ine_numSQ_TPx	_clause_n	JM	<u>SQ</u> →TP	<u>x</u> SQ→TPx	63	Line number Reservation s number	in the tationClause	Formatted
2_TPx_writ	e_gpr_indexSQ_1	FPx_write_	gpr_inde	<u>SQ-&gt;TP</u>	<u>x</u> SQ->TPx	77	Index into Register of returned Fetch into Register file returned Fetch Dat	r file for write n_DataIndex for_write_of a	
7 <del>.2.9</del> 24.	2.9 TP to S	Q: Textu	ıre stall						Formatted: Bullets and Numbering
e TP sends	this signal to the	SQ and th	e SPs whe	n its input	buffer is fu	11.			
TP_SP_fe	etch_Stall								
Q_SP_wr_ad	ldr SU0								
			SU1	]					
				•	SU2		V3		
Name		Direction	1	Bits	Descriptior	1			
[ ]F_3Q_6	ston_stan	1r→ 3Q			DO NOL SENC	i more text	ure request il asserteu		
7.2.1024	4.2.10_SQ to	SP: Te	xture st	all				4-	
Name		Direction	1	Bits	Descriptior	1			
SQ_SPx_	fetch_stall	SQ→SP	(	1	Do not senc	I more text	ture request if asserted		
7.2.1124	4.2.11_SQ to	SP: GI	PR and a	auto co	unter			*	(Formatted: Bullets and Numbering
Name		Directio	n	Bits	Descriptio	n		I	
SQ_SPx_	gpr_wr_addr	SQ→SF	<sup>y</sup> x	7	Write addre	ess			
SQ SPX	gpi_ra_addr apr rd en	SQ→SF	-x -x	1	Read Enab	ess			
SQ_SPx	gpr_wr_en	SQ→SF		1	Write Enab	le for the C	GPRs		
SQ_SPx_	gpr_phase	SQ→SF	Х	2	The phase reads and	mux (arl writes)	bitrates between input	s, ALU SRC	
SQ_SPx_	channel_mask	SQ→SF	<sup>2</sup> X	4	The channe	el mask		hin Anlin C	
SQ_SPX_	gpr_input_sel	SQ→SF	x	2	when the which sou VTX1, auto	phase mu rce to rea gen count	ix selects the inputs the inputs the inputs the inputs the input of th	data, VTX0,	
	auto_count	SQ→SF	x	12?	Auto count	generated	d by the SQ, common	for all shader	
SO SPV	uuto_count		^	12!	pipes	generated			

ATI Ex. 2107 IPR2023-00922 Page 253 of 260

	ORIGINATE DATE EDIT DATE 24 September, 2001 <u>4 September, 200</u>		ATE	R400 Sequencer Specification			
			<u>r, 2015</u> 1	9			
7 2 1 2 2 4	212 SO #	ODV.	Instruction	2002 20		4	<b>Formatted:</b> Bullets and Numbering
e 1 , har , 1, har <u>har 1</u>	<u></u>	101 X.	monucion	15	1		
Name		Dire	ction	Bits	Description		
SQ_SPX_I	nstr_start	<u>SQ</u> -	→SPX	21	Instruction start		
					0: SRC A Select     2:0       SRC A Argument Modifier     3:3       SRC A swizzle     11:4       VectorDst     17:12       Unused     20:18		
					1: SRC B Select     2:0       SRC B Argument Modifier     3:3       SRC B swizzle     11:4       ScalarDst     17:12       Unused     20:18		
					- 2: SRC C Select 2:0 SRC C Argument Modifier 3:3 SRC C swizzle 11:4 Unused 20:12		
					- 3: Vector Opcode Scalar Opcode Vector Clamp 11:11 Scalar Clamp Vector Write Mask Scalar Write Mask 20:17		
SQ_SPx_	exp_alu_id	SQ-	→SPx	1	ALU ID		
SQ_SPx_e	exporting	SQ-	→SPx	2	0: Not Exporting 1: Vector Exporting 2: Scalar Exporting		
SQ_SPx_s	tall	SQ-	→SPx	1	Stall signal		
SQ_SP0_\	vrite_mask	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	/hich must be and all color computed per	
SQ_SP1_	write_mask	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 c clock	vhich must be and all color computed per	
SQ_SP2_	write_mask	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	vhich must be and all color computed per	
SQ_SP3_	write_mask	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 c clock	vhich must be and all color computed per	
27.2.1324	.2.13_SP to	SQ: C	constant a	ddress	s load/ Predicate Set		Formatted: Bullets and Numbering
Name		Directi	on	Bits	Description	1/4 54	
SPU_SQ_0	onst_addr	SP0→S	SQ.	36	constant address load / predicate vector loa	ad (4 bits only)	

Exhibit 2028.docR400\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

1

36

Data valid

to the sequencer

Constant address load / predicate vector load (4 bits only)

SP0→SQ SP1→SQ

SP0\_SQ\_valid SP1\_SQ\_const\_addr

AMD1044\_0257388

ATI Ex. 2107 IPR2023-00922 Page 254 of 260

ORIGINATE DATE EDIT DA			E	DOCUMENT-REV. NUM.	PAGE			
	24 Septem	ber, 2001	4 September, 2	01519	GEN-CXXXXX-REVA	53 of 58		
SP1 SQ V	valid	SP1→SQ	<u>Anril 200</u>	Data va	lid			
SP2_SQ_d	const_addr	SP2→SQ	36	Constar to the s	nt address load / predicate vector load			
SP2 SQ V	valid	SP2→SQ	1	Data va	lid			
SP3_SQ_d	const_addr	SP3→SQ	36	Consta	nt address load / predicate vector load	d (4 bits only)		
				to the s	equencer			
SP3_SQ_V	valid	SP3→SQ	1	Data va	lid			
<del>27.2.14</del> 24	<u>1.2.14</u> SQ	to SPx: co	onstant broa	dcast		at	T.	Formatted: Bullets and Numbering
Name		Direction	Bits	Descrip	otion			
SQ_SPx_c	const	SQ→SPx	128	Consta	nt broadcast			
<del>27.2.15</del> 24	.2.15_SPC	) to SQ: Ki	ill vector load	4			E.	Formatted: Bullets and Numbering
Name		Direction	Bits	Descrip	otion			
SP0_SQ_I	kill_vect	SP0→SQ	4	Kill vect	or load			
SP2 SO 1	kiii_vect	<u>SP1→SQ</u>	4	Kill vect	or load			
SP3 SO 1	kill vect	SP3→SQ	4	Kill vect	or load			
27.2.1624	.2.16_SQ	to CP: RB	BM bus	1.00.000			Y	Formatted: Bullets and Numbering
Name		Direction	Bits	Descrip	otion			
SQ_RBB	rs	SQ→CP	1	Read S	strobe			
SQ_RBB_	rd	SQ→CP	32	Read D	Data			
SQ_RBBM	M_nrtrtr	SQ→CP	1	Optiona	al			
SQ_RBB	M_rtr	SQ→CP	1	Real-Ti	me (Optional)			
<del>27.2.17</del> <u>24</u>	<u>.2.17</u> CP	to SQ: RB	BM bus				A.	Formatted: Bullets and Numbering
Name		Direction	Bits	Descrip	otion			
rbbm_we		CP→SQ	1	Write E	nable			
rbbm_a		CP→SQ	15	Addres	s Upper Extent is TBD (16:2)			
rbbm_wd		CP→SQ	32	Data				
rbbm_be		<u>CP→SQ</u>	4	Byte Er	nables			
rbbm_re		CP→SQ	1	Read E	nable			
rbb_rs0			1	Read R	teturn Strobe U			
rbb_rd0		CP→SQ CP→SQ	32	Read P	ata 0			
rbb_rd1		CP→SQ	32	Read	Data 0			
RBBM SC	C soft reset	CP→SQ 1 Soft Reset						
		1		1		"		Formatted: Bullets and Numbering
27.2.1824	<u>.2.18</u> SQ	to CP: Sta	ite report					
Name		Direction	Bits	Descrip	otion			
SQ_CP_V	s_event	SQ→CP	1	Vertex	Snader Event			
SQ_CP_V	s_eventid	SQ->CP	2	Vertex Division	Snader Event ID			
SQ_CP_p	s_event	SQ→CP	2	Pixel S	hader Event ID			
5Q_CP_p	s_eventia	SQACP	2	Pixel 5	hader Event ID			
eventi eventi So, the CP wil and the SQ_C	id = 0 => *sEnd id = 1 => *sDor Il assume the \ CP_vs_eventid	IOfState (i.e. ne (i.e /s is done with = 0.	VsEndOfState) . VsDone) a state wheneve	r it gets a	pulse on the SQ_CP_vs_event			
Exhibit 2028.docF	1400_Sequencer.doc	73201 Bytes*** ©	ATI Confidentia	I. Referer	nce Copyright Notice on Cover Pa	ge © ***		

ATI Ex. 2107 IPR2023-00922 Page 255 of 260

	٦ <b>آ</b>	ORIGINATE DAT 24 September, 20	E 01 <u>4 S</u>	EDIT DATE	R	400 Sequer	ncer Specifica	ation	PAGE 54 of 58	]		
	24.3 Exa	mple of contr	ol flow	program e	xecut	tion		i	*		Formatted: Bullets and Numbering	
	We now provid	le some examples of										
	Given the proc	iram:										
	Alu 0											
	Alu 1											
	Tex 1											
	<u>Alu 3 Serial</u> Alu 4											
	Tex 2 Alu 5											
	Alu 6 Serial											
	Alu 7											
	Alloc Position Alu 8 Export	1 buffer										
	Tex 4 Alloc Paramet	er 3 buffers										
	Alu 9 Export 0											
	Alu 10 Serial E	xport 2										
	Alu II Export	<u>i Ena</u>										
	Would be conv	verted into the followi	ng CF inst	ructions:								
	Execute Alu Execute Alu	1 0 Alu 0 Tex 0 1 0	Tex 0 Al	u 1 Alu 0 Tex	: O Alı	u O Alu 1	Tex 0					
	Alloc Posit Execute Alu	<u>ion 1</u> 0 Tex 0										
	Alloc Parar Execute Alu	<u>1 3</u> 1 0 Tex 0 Alu 1	Alu O Er	ıd								
	And the execut	tion of this program.	would look	liko thia:								
	And the exect			ince unis.								
	Put thread in V	enex RS:										
	Control Flo Execution	ow Instruction Pointe Count Marker (3 or 4	<u>r (12 bits),</u> I bits), (EC	(CFP) CM)								
	Loop Itera	tors (4x9 bits), (LI)	(CRP)									
	Predicate	Bits(4x64 bits), (PB)	101(17									
	GPR Base	Ptr (8 bits), (GPR)										
	Export Ba Context P	se Ptr (7 bits), (EB) r (3 bits).(CPTR)										
	LOD corre	ction bits (16x6 bits)	(LOD)									
	State Bits											
	<u>CFP EC</u> 0 0		<u>CRP</u>	<u>PB EX</u> 0 0		<u>GPR</u> 0	<u>EB</u> 0	<u>CPTR</u>				
	Valid Thre	ad (VALID)										
	Texture A	U engine needed (T		3)								
	Waiting or	Texture Read to Co	mplete (SE	RIAL)								
1	Anocation	vvait (2 Dits) (ALLOU	<u>.</u>									
	Exhibit 2028.docR	400_Sequencer.doc 73201 By	tes*** © A1	1 Confidential. F	eferend	ce Copyrigi	nt Notice on	Cover Pag	ge © ***			

ATI Ex. 2107 IPR2023-00922 Page 256 of 260

			DOCUMENT-REV NUM	PAGE
	24 September, 2001	4 September, 201519	GEN-CXXXXX-REVA	55 of 58
00 - 01 - 11 - Allocatic Position First thr Last (1	No allocation needed     No allocation needed     Position export allocation ne     Parameter or pixel export ne     pass thru (out of order expo     n Size (4 bits) (SIZE)     Allocated (POS_ALLOC)     ead of a new context (FIRST)     bit), (LAST)	eded (ordered export) eded (ordered export) rt)		1
Status Bits				
VALID 1	ALU Q	SERIAL ALLOC S	BIZE     POS_ALLOC     FIRST       0     0     1	LAST 0
Then the Execut It execu state rel	e thread is picked up for the e the Alu O Alu O Tex O Te utes the first two ALU instruc- turned to the RS:	execution of the first control f ex 0 Alu 1 Alu 0 Tex tions and goes back to the	flow instruction: 0 Alu 0 Alu 1 Tex 0 RS for a resource request change.	<u>Here is the</u>
State Bits				
<u>CFP</u>	<u>ECM LI CRP</u> 2 0 0	PB         EXID           0         0	GPR         EB         CPTR           0         0         0	
Status Bits				
VALID	TYPE PENDING	SERIAL ALLOC S	SIZE POS ALLOC FIRST	LAST
Then wh back in this State Bits CFP 0	hen the texture pipe frees up, state: ECM LI CRP 4 0 0	the arbiter picks up the three	gene the texture reads. The th       GPR     EB     CPTR       0     0     0	LOD
Status Bits				
VALID	TYPE PENDING	SERIAL ALLOC S	BIZE POS ALLOC FIRST	LAST
1	ALU 1	1 0 0	0 1	0
Because pick the thre this state:	e of the serial bit the arbiter ead up. Lets say that the text	must wait for the texture to ure reads are complete, the	return and clear the PENDING bit b n the arbiter picks up the thread and	efore it can returns it in
State Bits				
CFP 0	<u>ECM LI CRP</u> 6 0 0	PB         EXID           0         0	GPR         EB         CPTR           0         0         0	LOD 0
Status Bits				
VALID 1	TYPE PENDING TEX 0	SERIAL ALLOC S	BIZE POS ALLOC FIRST	LAST 0
 Again th	ne TP frees up, the arbiter pic	ks up the thread and execut	es. It returns in this state:	

ATI Ex. 2107 IPR2023-00922 Page 257 of 260

State Bits           CFP         ECM         LI         CRP         PB         EXID         GPR         EB         CPTR         LOD           Q	ORIGINATE DATE 24 September, 2001	EDIT DATE <u>4 September, 2015</u> 19 April 2002	R400 Sequencer Specification	PAGE 56 of 58
Status Bits         VALID       TYPE       PENDING       SERIAL       ALLOC       SIZE       POS       ALLOC       FIRST       LAST         1       ALU       1       0       0       0       0       0       1       0         Now, even if the texture has not returned we can still pick up the thread for ALU execution because it has the serial bit is not set. The thread will however come back to the RS for the second ALU instruction because it has the serial bit set.         State Bits       CFP       ECM       Li       CRP       PB       EXID       GPR       EB       CPTR       LOD         0       8       0       0       0       0       0       0       0       0         Status Bits	ate Bits <u>-P ECM LI CF</u> 7 <u>0</u> 0	R <u>P PB EXID</u> 0 0	0 <u>GPR EB CP</u> 0 <u>0</u> 0	TR LOD 0
INTERMINENT OF THE INTERMITTION OF THE SECOND ALLO INSTRUCTION DECAUSE IT THIS THE SECTION ALLO INSTRUCTION TO DECAUSE IT THIS THE SECTION ALLO INSTRUCTION DECAUSE IT THIS THE SECTION ALLO IN THE SECTION ALLO INTERVIENTE.         Status Bits         VALID TYPE PENDING SERIAL ALLOC SIZE POS ALLOC FIRST LAST         State Bits         VALID TYPE PENDING SERIAL ALLOC SIZE POS ALLOC FIRST LAST         State Bits         State Bits         VALID TYPE PENDING SERIAL ALLOC SIZE POS ALLOC FIRST LAST         State Bits         State Bits         VALID TYPE PENDING SERIAL ALLOC SIZE POS ALLOC FIRST LAST         State Bits         VALID TYPE PENDING SERIAL ALLOC SIZE POS ALLOC FIRST LAST <td>ALID TYPE PENDING ALID ALU 1 Now, even if the texture has not re</td> <td>SERIAL ALLOC 0 0 turned we can still pick up</td> <td>SIZE POS_ALLOC FIRST</td> <td>LAST Q ause the serial bit bas the serial bit</td>	ALID TYPE PENDING ALID ALU 1 Now, even if the texture has not re	SERIAL ALLOC 0 0 turned we can still pick up	SIZE POS_ALLOC FIRST	LAST Q ause the serial bit bas the serial bit
Status Bits         VALID         TYPE         PENDING         SERIAL         ALLOC         SIZE         POS         ALLOC         FIRST         LAST           1         ALU         1         1         0         0         0         1         0           As soon as the TP clears the pending bit the thread is picked up and returns:         Status Bits         Status Bits         CFP         ECM         LI         CRP         PB         EXID         GPR         EB         CPTR         LOD         0	Increase         Increase		<u>) GPR EB CP</u>	
Image: August of the second	atus Bits ALID TYPE PENDING	SERIAL ALLOC	SIZE POS ALLOC FIRST	
OT         DOM         D         DAL         TD         DAL         D <thd< th=""> <thd< th="">         D         <thd< td=""><td>As soon as the TP clears the pendi ate Bits</td><td>ing bit the thread is picked</td><td>up and returns:</td><td></td></thd<></thd<></thd<>	As soon as the TP clears the pendi ate Bits	ing bit the thread is picked	up and returns:	
1       TEX       0       0       0       0       0       1       0         Picked up by the TP and returns: Execute Alu 0         State Bits         CFP       ECM       LI       CRP       PB       EXID       GPR       EB       CPTR       LOD         1       0       0       0       0       0       0       0       0       0         Status Bits         VALID       TYPE       PENDING       SERIAL       ALLOC       SIZE       POS ALLOC       FIRST       LAST         1       ALU       1       0       0       0       0       1       0	atus Bits ALID TYPE PENDING	SERIAL ALLOC	SIZE POS ALLOC FIRST	
CFP         ECM         LI         CRP         PB         EXID         GPR         EB         CPTR         LOD           1         0	TEX     Q       Picked up by the TP and returns:     Execute Alu 0	0 0	0 0 1	<u>0</u>
VALIDTYPEPENDINGSERIALALLOCSIZEPOS_ALLOCFIRSTLAST1ALU1000010	ECM     Ll     CF       0     0     0       atus Bits     0     0	<u>RP PB EXID</u> 0 0	<u>) GPR EB CP</u> 0 0 0	TR LOD
Picked up by the ALU and returns (lets say the TP has not returned yet): Alloc Position 1	TYPE         PENDING           ALU         1           Picked up by the ALU and returns ( Alloc Position 1	SERIAL     ALLOC       0     0       lets say the TP has not ret	SIZE POS_ALLOC FIRST	LAST Q
State Bits           CFP         ECM         LI         CRP         PB         EXID         GPR         EB         CPTR         LOD           2         0	ate Bits -P ECM LI CF 0 0 0 0	<u>RP PB EXIC</u> <u>0</u> 0	0 GPR EB CP 0 0 0	TR LOD Q

Exhibit 2028.docR400\_Sequencer.doc 73201 Bytes\*\*\* © ATI Confidential. Reference Copyright Notice on Cover Page © \*\*\*

AMD1044\_0257392

ATI Ex. 2107 IPR2023-00922 Page 258 of 260

Δň	ORIGINATE DATE	E ED	IT DATE	DOCU	MENT-REV	/. NUM.	PAGE	
Statue Bite	24 September, 200			GLIN	-0/////-1		57 01 58	
VALID 1	TYPE PENDIN ALU 1	G <u>SERIAL</u> O	ALLOC S		ALLOC	FIRST 1	LAST Q	
If the SX the RS in this	has the place for the ex state:	port, the SQ is g	joing to allocate	and pick up ti	ne thread fo	or execution	n. It returns to	
Execute Al	u 0 Tex 0							
State Bits								
<u>CFP</u> E 311	<u>CM LI</u> 0	CRP         PB           0         0	EXID 0	<u>GPR</u> 0	<u>EB</u> 0	<u>CPTR</u> 0	LOD 0	
Status Bits								
VALID 1	TYPE PENDIN TEX 1	G SERIAL 0	ALLOC S	IZE POS	ALLOC	FIRST 1	LAST 0	
Now ein	e the TP has not return	med vet we mu	ist wait for it to i	return hecous	e we cann	- Int issue m	ultiple texture	
requests. The	TP returns, clears the	PENDING bit ar	nd we proceed:	courr becaus	C WE Call	ior issue III	unpie texture	
<u>Alloc P</u>	aram 3							
State Bits								
CFP E	CM LI	CRP PB	EXID	GPR	EB	CPTR	LOD	
<u>+ U</u>	<u>v</u>	<u>v 10</u>		12	<u> </u>	0	12	
Status Bits								1
								1000000
VALID	TYPE PENDIN	G SERIAL	ALLOC S	IZE POS	ALLOC	FIRST	LAST	
VALID 1	TYPE PENDIN ALU 1	G <u>SERIAL</u> 0	ALLOC S	IZE POS	ALLOC	FIRST 1	LAST 0	
VALID 1 Once aga thread.	TYPE         PENDIN           ALU         1           ain the SQ makes sure	G SERIAL 0 e the SX has e	ALLOC S	IZE POS	ALLOC er cache be	FIRST 1 efore it car	LAST 0 pick up this	
VALID 1 Once aga thread. Execute	TYPE         PENDIN           ALU         1           ain the SQ makes sure           Alu 0 Tex 0 Alu	G SERIAL 0 e the SX has e 1 Alu 0 End	ALLOC S	IZE POS	ALLOC	FIRST 1 efore it car	LAST 0 pick up this	
VALID 1 Once aga thread. Execute State Bits	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu     0 Tex	G SERIAL 0 e the SX has e 1 Alu 0 End	ALLOC S	IZE POS ,	ALLOC	FIRST 1 efore it car	LAST 0 pick up this	
VALID 1 Once age thread. Execute State Bits CFP E	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu 0 Tex 0 Alu       CM     LI	G SERIAL O e the SX has e 1 Alu O End CRP PB	ALLOC S 10 3 nough room in EXID	IZE POS . 1 the Paramete	ALLOC er cache be	FIRST 1 efore it car	LAST 0 pick up this LOD	
VALID 1 Once aga thread. Execute State Bits CFP E 5 1	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu 0 Tex 0 Alu       CM     LI       Q	G SERIAL 0 e the SX has e 1 Alu 0 End CRP PB 0 0	ALLOC         S           10         3           nough room in 1 <u>EXID</u> 1	IZE POS . 1 the Paramete <u>GPR</u> 0	ALLOC er cache be	FIRST 1 efore it car <u>CPTR</u> 0	LAST 0 pick up this LOD 0	
VALID 1 Once age thread. Execute State Bits CFP 5 1 Status Bits	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu 0 Tex 0 Alu       CM     LI       Q	G SERIAL 0 e the SX has e 1 Alu 0 End <u>CRP PB</u> 0 0	ALLOC         S           10         3           nough room in         3           EXID         1	IZE POS . 1 the Paramete <u>GPR</u> 0	ALLOC er cache be	FIRST 1 efore it car CPTR 0	LAST 0 pick up this LOD 0	
VALID 1 Once aga thread. Execute State Bits CFP E 5 1 Status Bits VALID	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu 0 Tex 0 Alu       CM     LI       Q       TYPE       PENDIN	G SERIAL 0 e the SX has e 1 Alu 0 End CRP PB 0 0 G SERIAL	ALLOC         S           10         3           nough room in         1           EXID         1           ALLOC         S	IZE POS	ALLOC	FIRST 1 efore it car <u>CPTR</u> 0 FIRST	LAST 0 pick up this LOD 0 LAST	
VALID       1       Once aga       thread.       Execute       State Bits       CFP       5       1   Status Bits       VALID       1	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu     0 Tex       Alu     0 Tex       CM     LI       Q       TYPE     PENDIN       TEX     1	G SERIAL 0 e the SX has e 1 Alu 0 End CRP PB 0 0 G SERIAL 0	ALLOC         S           10         3           nough room in         3           EXID         1           1         1           ALLOC         S           Q         Q	IZE POS . 1 the Paramete GPR 0 SIZE POS . 1	ALLOC	FIRST 1 efore it car CPTR 0 <u>FIRST</u> 1	LAST 0 pick up this LOD 0	
VALID 1 Once aga thread. Execute State Bits CFP E 5 1 Status Bits VALID 1 This executor	TYPE         PENDIN           ALU         1           ain the SQ makes sure           Alu 0 Tex 0 Alu           CM         LI           0           TYPE         PENDIN           TEX         1           utes on the TP and the	G SERIAL 0 e the SX has e 1 Alu 0 End CRP PB 0 0 G SERIAL 0 n returns:	ALLOC         S           10         3           nough room in         3           EXID         1           1         1           ALLOC         S           0         0	IZE POS 1 the Paramete GPR 0 IZE POS 1	ALLOC er cache be	FIRST 1 efore it car CPTR 0 FIRST 1	LAST Q pick up this LOD Q LAST Q	
VALID       1       Once aga       thread.       Execute       State Bits       CFP       5       1       Status Bits       VALID       1       This exect       State Bits	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu     0 Tex       Alu     0 Tex       CM     LI       0         TYPE     PENDIN       TEX     1   utes on the TP and the	G SERIAL 0 e the SX has e 1 Alu 0 End CRP PB 0 0 G SERIAL 0 n returns:	ALLOC         S           10         3           nough room in         3           EXID         1           1         1           ALLOC         S           0         0	IZE POS . 1 the Paramete GPR 0 IZE POS .	ALLOC	FIRST 1 efore it car <u>CPTR</u> <u>0</u> <u>FIRST</u> 1	LAST 0 pick up this LOD 0 LAST 0	
VALID       1       Once aga       thread.       Execute       State Bits       CFP       5       1       Status Bits       VALID       1       This exec       State Bits       CFP       E	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu 0 Tex 0 Alu       CM     LI       0       TYPE       PENDIN       TEX     1       utes on the TP and the       CM     LI	G SERIAL 0 e the SX has e 1 Alu 0 End CRP PB 0 0 G SERIAL 0 n returns: CRP PB	ALLOC         S           10         3           nough room in         3           EXID         1           ALLOC         S           0         0           EXID         1	IZE POS 1 the Paramete GPR 0 IZE POS 1 GPR	ALLOC er cache be	FIRST 1 efore it car <u>CPTR</u> 0 FIRST 1 <u>CPTR</u>	LAST 0 pick up this LOD 0 LAST 0 LOD	
VALID         1         Once aga         thread.         Execute         State Bits         CFP       E         5       1         Status Bits       VALID         1	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu 0 Tex 0 Alu       CM     L       Q       TYPE     PENDIN       TEX     1       utes on the TP and the       CM     L       Q	G     SERIAL       0       e the SX has e       1 Alu 0 End       CRP     PB       0     0       G     SERIAL       0     0       G     SERIAL       0     0       CRP     PB       0     0	ALLOC         S           10         3           nough room in         3           EXID         1           ALLOC         S           Q         0           Q         0           EXID         1	IZE POS . 1 the Parameter <u>GPR</u> 0 IZE POS . 1 <u>GPR</u> 0	ALLOC er cache be	FIRST 1 efore it car <u>CPTR</u> 0 <u>FIRST</u> 1 <u>CPTR</u> 0 <u>CPTR</u> 0	LAST 0 pick up this 0 LOD 0 LAST 0 LOD 0 LOD 0 0	
VALID         1         Once aga         thread.         Execute         State Bits         CFP         5         1         Status Bits         VALID         1         This exec         State Bits         CFP       E         5       2         Status Bits	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu 0 Tex 0 Alu       CM     LI       0       TYPE     PENDIN       TEX     1       utes on the TP and the       CM     LI       0	G     SERIAL       0       e the SX has e       1 Alu 0 End       CRP     PB       0     0       G     SERIAL       0     0       n returns:       CRP     PB       0     0	ALLOC         S           10         3           nough room in         3           EXID         1           ALLOC         S           0         0           EXID         1	BIZE     POS       1       the     Paramete       GPR       0	ALLOC er cache be	FIRST 1 efore it car CPTR 0 FIRST 1 CPTR 0 CPTR 0 0	LAST 0 pick up this 0 0 0 LAST 0 LOD 0 0 0 0 0 0 0 0 0 0 0 0 0	
VALID         1         Once aga         thread.         Execute         State Bits         CFP       E         5       1         Status Bits         VALID       1         1       This exect         State Bits       CFP         5       2         Status Bits       Status Bits	TYPE     PENDIN       ALU     1       ain the SQ makes sure       Alu 0 Tex 0 Alu       CM     LI       Q       TYPE     PENDIN       TEX     1       utes on the TP and the       Q       Q	G SERIAL 0 e the SX has e 1 Alu 0 End CRP PB 0 0 G SERIAL 0 n returns: CRP PB 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	ALLOC         S           10         3           nough room in         3           EXID         1           ALLOC         S           Q         Q           EXID         1           EXID         1           EXID         1	POS       1       1       the Parameter       GPR       0	ALLOC er cache be	FIRST 1 efore it car <u>CPTR</u> 0 <u>FIRST</u> 1 <u>CPTR</u> 0 <u>CPTR</u> 0	LAST Q pick up this LOD Q LAST Q LOD Q	

ATI Ex. 2107 IPR2023-00922 Page 259 of 260

	<b>∕</b> Ĵĵ	ORIGINATE DATE 24 September, 2001		EDIT E	DATE er, 201519	R400	PAGE 58 of 58		
	VALID	TYPE	PENDING	SERIAL	ALLOC	SIZE	POS_ALLOC	FIRST	LAST
	1	ALU	1	1	0	0	1	1	1
Waits for the TP to return because of the textures reads are pending (and SERIAL in this case). Then execu and does not return to the RS because the LAST bit is set. This is the end of this thread and before dropping it on floor, the SQ notifies the SX of export completion.									

## 28-25. Open issues

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

Saving power?

Exhibit 2028.docR400\_Sequencer.doc 73201 Bytes\*\*\* C ATI Confidential. Reference Copyright Notice on Cover Page C \*\*\*

Formatted: Bullets and Numbering

AMD1044\_0257394

ATI Ex. 2107 IPR2023-00922 Page 260 of 260