	ORIGINATE DATE 14 August, 200114 <small>August 20017 May</small>	EDIT DATE 4 September, 201524 <small>August 20016 August</small>	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	PAGE 1 of 20
--	--	--	---------------------------------------	-----------------

Author: Laurent Lefebvre

Issue To: Copy No:

## R400 Sequencer Specification

### SEQ

Version 0.42

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

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

#### APPROVALS

Name/Dept	Signature/Date

Remarks:

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


"Copyright 2001, ATI Technologies Inc. All rights reserved. The material in this document constitutes an unpublished work created in 2001. The use of this copyright notice is intended to provide notice that ATI owns a copyright in this unpublished work. The copyright notice is not an admission that publication has occurred. This work contains [REDACTED] proprietary information and trade secrets of ATI. No part of this document may be used, reproduced, or transmitted in any form or by any means without the prior written permission of ATI Technologies Inc."

ATI 2010  
 LG v. ATI  
 IPR2015-00325

AMD1044 0016660






	ORIGINATE DATE 14 August, 200114 August, 20017 May	EDIT DATE 4 September, 201524 August, 20016 August	R400 Sequencer Specification	PAGE 2 of 20
--	--	--	------------------------------	-----------------

**Table Of Contents**

<b>1. OVERVIEW..... 43</b>	<b>2. INTERPOLATED DATA BUS..... 10</b>
1.1 Top Level Block Diagram ..... 54	3. INSTRUCTION STORE ..... 10
1.2 Data Flow graph..... 97	4. CONSTANT STORE..... 11
1.3 Control Graph..... 1240	5. LOOPING AND BRANCHES..... 11
<b>2. INTERPOLATED DATA BUS..... 1240</b>	6. REGISTER FILE ALLOCATION..... 11
<b>3. INSTRUCTION STORE..... 1240</b>	7. TEXTURE ARBITRATION..... 12
<b>4. CONSTANT STORE..... 1344</b>	8. ALU ARBITRATION..... 12
<b>5. LOOPING AND BRANCHES..... 1344</b>	9. HANDLING STALLS..... 13
<b>6. REGISTER FILE ALLOCATION... 1344</b>	10. CONTENT OF THE RESERVATION
<b>7. TEXTURE ARBITRATION..... 1442</b>	STATION FIFOS..... 13
<b>8. ALU ARBITRATION..... 1412</b>	11. THE OUTPUT FILE (RB FIFO AND
<b>9. HANDLING STALLS..... 1513</b>	PARAMETER CACHE)..... 13
<b>10. CONTENT OF THE RESERVATION</b>	<b>12. INTERFACES..... 13</b>
<b>STATION FIFOS..... 1543</b>	12.1 External Interfaces..... 13
<b>11. THE OUTPUT FILE (RB FIFO AND</b>	12.1.1 Sequencer to Shader
<b>PARAMETER CACHE)..... 1543</b>	Engine Bus..... 13
<b>12. INTERFACES..... 1543</b>	12.1.2 Shader Engine to Output
12.1 External Interfaces..... 1543	File..... 13
12.1.1 Sequencer to Shader	12.1.3 Shader Engine to Texture
Engine Bus..... 1513	Unit Bus (Fast Bus)..... 14
12.1.2 Shader Engine to Output	12.1.4 Sequencer to Texture Unit bus
File..... 1543	(Slow Bus) 14
12.1.3 Shader Engine to Texture	12.1.5 Shader Engine to RE/PA Bus 14
Unit Bus (Fast Bus)..... 1644	12.1.6 PA to sequencer..... 14
12.1.4 Sequencer to Texture Unit bus	<b>13. OPEN ISSUES..... 14</b>
(Slow Bus) 1644	<b>1. OVERVIEW..... 3</b>
12.1.5 Shader Engine to RE/PA Bus	1.1 Top Level Block Diagram ..... 4
..... 1644	<b>2. TEXTURE ARBITRATION..... 9</b>
12.1.6 PA? to sequencer..... 1644	<b>3. ALU ARBITRATION..... 9</b>
<b>13. EXAMPLES OF PROGRAM</b>	<b>4. HANDLING STALLS..... 10</b>
<b>EXECUTIONS..... 1744</b>	<b>5. CONTENT OF THE RESERVATION</b>
13.1.1 Sequencer Control of a Vector	STATION FIFOS..... 10
of Vertices 1744	<b>6. INTERFACES..... 10</b>
13.1.2 Sequencer Control of a Vector	6.1 External Interfaces..... 10
of Pixels 1846	6.1.1 Sequencer to Shader Engine
13.1.3 Notes..... 1947	Bus..... 10
<b>14. OPEN ISSUES..... 1947</b>	6.1.2 Shader Engine to Output File
<b>1. OVERVIEW..... 3</b>	..... 10
1.1 Top Level Block Diagram ..... 4	6.1.3 Shader Engine to Texture Unit
1.2 Data Flow graph..... 7	Bus (Fast Bus)..... 10
1.3 Control Graph..... 10	

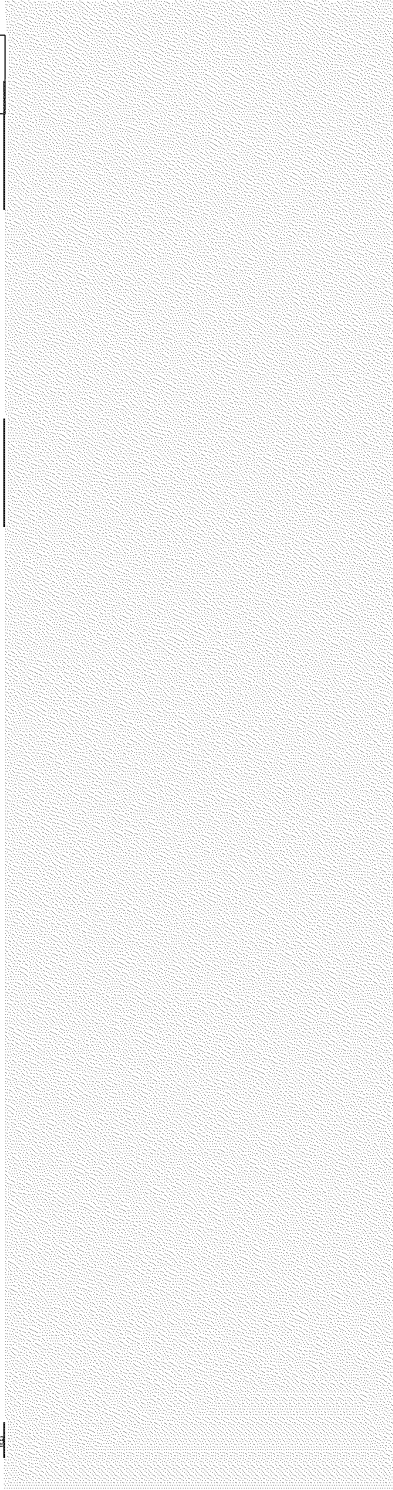




	ORIGINATE DATE 14 August, 2001 <del>August 2001</del> <del>May</del>	EDIT DATE 4 September, 2015 <del>August 2001</del> <del>August</del>	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	PAGE 3 of 20
6.1.4 Sequencer to Texture Unit bus (Slow Bus).....			7. OPEN ISSUES.....	11
6.1.5 Shader Engine to RE/PA Bus ..				11


Revision Changes:

Rev 0.1 (Laurent Lefebvre) Date: May 7, 2001	First draft.
Rev 0.2 (Laurent Lefebvre) Date : July 9, 2001	Changed the interfaces to reflect the changes in the SP. Added some details in the arbitration section.
Rev 0.3 (Laurent Lefebvre) Date : August 6, 2001	<u>Reviewed the Sequencer spec after the meeting on August 3, 2001.</u>
Rev 0.4 (Laurent Lefebvre) Date : August 24, 2001	<u>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.</u>







	<b>ORIGINATE DATE</b> <u>14 August, 2001</u> <small>14 August, 2001 14 August, 2001 17 May</small>	<b>EDIT DATE</b> <u>4 September, 2015</u> <small>4 September, 2015 24 August, 2001 16 August</small>	<b>R400 Sequencer Specification</b>	<b>PAGE</b> 4 of 20
--	--	--	-------------------------------------	------------------------

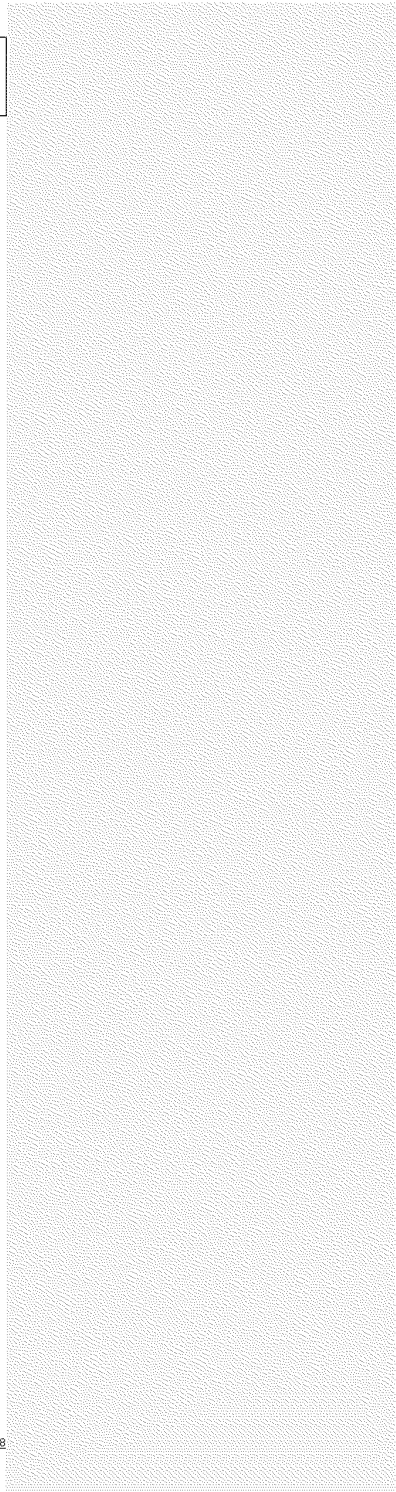
## 1. Overview

The sequencer first arbitrates between vectors of 16 ~~(maybe 32)~~ vertices that arrive directly from primitive assembly and vectors of 84 ~~quads (16 pixels) (32 pixels)~~ that are generated in the raster engine.


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

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

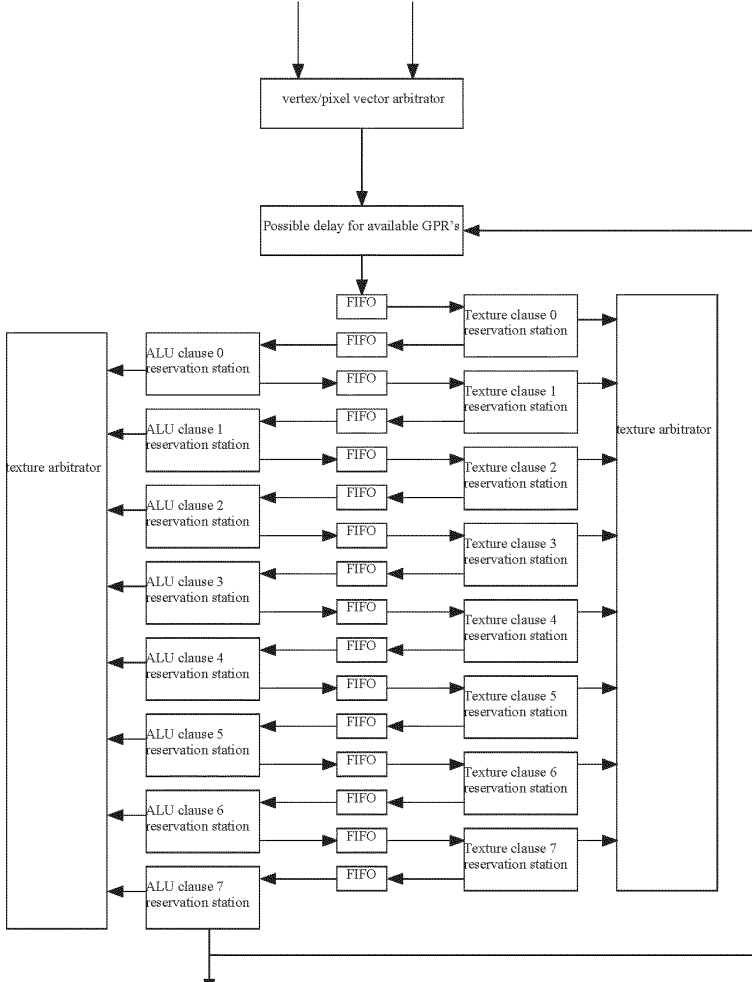
To support the shader pipe the raster engine also contains the shader instruction cache and constant store. There are only one constant store for the whole chip and one instruction store. These will be shared among the four shader pipes.





	ORIGINATE DATE 14 August, 200114 <small>August, 20017 May</small>	EDIT DATE 4 September, 201524 <small>August, 20016 August</small>	DOCUMENT-REV. NUM. GEN-CXXXXX-REVA	PAGE 5 of 20
--	---	---	---------------------------------------	-----------------

1.1 Top Level Block Diagram



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

The rasterizer always checks the vertices FIFO first and if allowed by the sequencer sends the data to the shader. If the vertex FIFO is empty then, the rasterizer takes the first entry of the pixel FIFO (a vector of 32\_16\_pixels) and sends it to the interpolators. Then the sequencer takes control of the packet. The packet consists of 3 bits of state, 6-7 bits for the base address of the Shader program and some information on the coverage to determine texture LOD. All other information (2x2 addresses) is put in a FIFO (one for the pixels and one for the vertices) and retrieved when the packet finishes its last clause.

# Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

## Real-Time Litigation Alerts



Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

## Advanced Docket Research



With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

## Analytics At Your Fingertips



Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

## API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

## LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

## FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

## E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.