CRIME 1.5 Specification

Linda Gardner

Zahid Hussain

Chris Johnson

Chris Kogelnik

Minghua Lin

Mike Nielsen

James Tornes

Mark Troeller

Revision 0.3

TABLE OF CONTENTS

CHAPTER 1 Introduction

CRIME (CPU, Rendering, Interconnect and Memory Engine) is the controller in an O2 workstation that provides the external agent function for a single MIPS R10000 or R12000 processor (Project names: T5 or TREX respectively). CRIME implements extensions to the SysAD protocol for use by VICE (Video Imaging and Compression Engine) to perform DMA between VICE internal RAM and Unified Workstation Memory. CRIME is also the memory controller for the SDRAM based Unified Workstation Memory architecture used in O2. CRIME also implements an on-board rendering engine that supports the OpenGL. CRIME provides I/O ports for the MACE (Multimedia, Audio and Communications Engine) and the GBE (Graphics Back End) chips that complete the functionality of an O2 workstation. These functions are outlined in this chapter.

This document is tied closely with the *MIPS R10000 Microprocessor User's Manual [5]*. It follows closely with the nomenclature and the terms used in these specs. In many places, the terms are used without definition. It is assumed that the same definitions in these specifications apply. If in doubt, these specifications can be referenced.

1.1 System Architecture Overview

CRIME connects the processors (Mips based and VICE), memory and IO. It channels data streams among them. A diagram showing CRIME and other main functional blocks in the O2 architecture is shown in Figure 1. Exploration of the architecture details can be found in the *G2 Architecture Issues [1]*. The*G2 Architecture* document discusses other possible implementations of a switched-bus memory controller using SDRAMs, so not all the information is directly applicable to the actual implementation of the CRIME ASIC.

A block diagram of the overall moosehead system can be found in *Moosehead System Architecture [2]*.

Referenced documents in italics are fully detailed in section 1.5 on page 10 in of this document. There you will find an address on the internal Silicon Graphics Mountain View Campus for ease of access to those documents that are on-line.

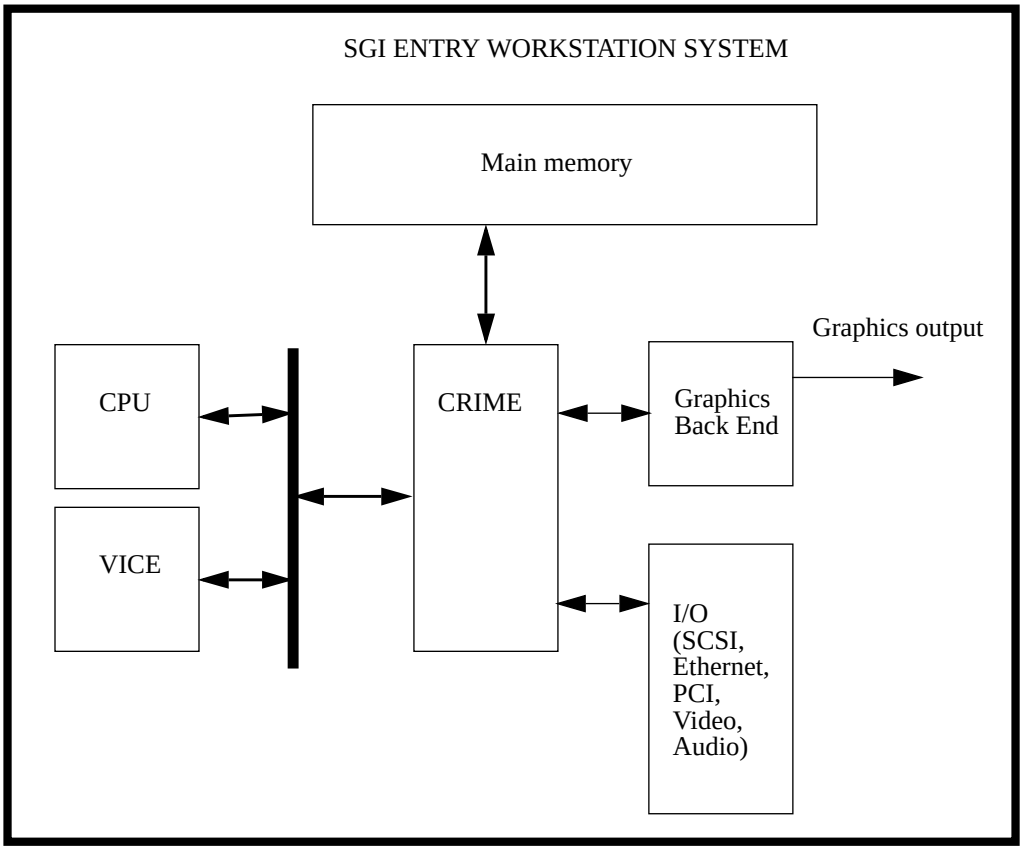


Figure 1-1 *O2* Block Diagram

1.2 Major Functions

1.2.1 External Agent Functions

1.2.1.1 For R10K and R12K processors

As the External Agent for the R10K/R12K processor, CRIME arbitrates the ownership of the SysAD bus.

CRIME supports a single R10K or R12K processor using HSTL logic levels. LVTTL is not supported for the SysAD interconnect.

CRIME supports a cache line size of 128 bytes and supports up to 4 outstanding reads for the R10K SysAD protocol.

CRIME resides on the System PC Board, while the R10K and VICE reside on a small daughter card called the CPU Board. The SysAD bus is limited to 100 MHz operation for this topology. Using this information, the correct divisor for a particular R10K or R12K can be chosen.

The clock divisor is programmed into the R10K/R12K at initialization time. The CRIME 1.5 ASIC reads a serial ROM on the CPU board and returns the information contained in the ROM to the processor in accordance with the SysAD initialization protocol for R10K/R12K. The ROM resides on the CPU board so that different clock speed processors will be able to communicate that to the CRIME ASIC which resides on the system board.

1.2.1.2 For VICE (Video Imaging and Compression Engine)

CRIME implements extensions to the SysAD bus protocol to allow block transfers of data between VICE and CRIME utilizing the SysAD bus and a few additional control lines.

This protocol extends the outstanding reads allowed by VICE to be a maximum of eight (8). Burst block writes of up to eight (8) are also supported.

The memory reference patterns of VICE when performing imaging and compression algorithms are supported by this protocol to allow efficient

"rectangle carving" of small blocks from larger image blocks stored in System Memory.

For more information about the VICE processor refer to the *VICE Design Specification 099-0123-003 [12]*

For more information about the changes to VICE to create VICE 1.5 that operates with CRIME 1.5 and an R10K/R12K processor on the SysAD bus refer to the *VICE 1.5 Design Specification 099-0TBA-x.x TBA [14]*

1.2.2 Memory Controller Functions

CRIME directly controls the Main Memory. The Main Memory is a 288-bit wide interleaved memory subsystem. The subsystem consists of up to 8 DIMM (Dual-in-line Memory Module) banks. Each DIMM bank consists of two identical 144-bit SDRAM-based DIMMs. The DIMMs have on-board control/address registers and a PLL clock driver.

The maximum memory configuration in a CRIME based workstation is 256 MBytes utilizing 16Meg SDRAM chips and 1 GByte using 64Meg SDRAM chips.

The memory controller acts as the central arbiter for the only memory in the workstation. It implements a An algorithm that gives up to 1/2 of all available system memory bandwidth to the GBE so that the graphics display refresh rate is maintained from system memory is further explained in the Memory Interface Unit Chapter. See section 6.2.3.2 on page 12 of Chapter 6.

1.2.3 Traffic Routing

CRIME has four data ports. They are: SysAD Bus, Memory Data Bus, CMI bus (CRIME MACE Interconnect) and CGI bus (CRIME GBE Interconnect). For the list below IO includes both the CMI and CGI buses.

The request streams flowing between these data ports are:

1. Processor to Memory Read
2. Processor to Memory Write
3. Processor to IO Read
4. Processor to IO Write
5. Processor to VICE Read
6. Processor to VICE Write

7. VICE to Memory Read
8. VICE to Memory Write
9. IO to Memory Read
10. IO to Memory Write
11. Processor to CRIME Internal Registers Read
12. Processor to CRIME Internal Registers Write
13. CRIME to Processors Interrupt Requests
14. IO to CRIME register writes for Interrupt Posting

Some of the above requests have responses to complete the transactions. The responses go the opposite direction.

CRIME has FIFOs and buffers to buffer these requests so that the Memory does not have to be locked out based on the ownership on the SysAD Bus or IO Buses. This maximizes the memory throughput.

Both SysAD Bus and the IO Buses are split-transaction buses.

1.3 Major Functional Units

The functions of the CRIME chip can be grouped into four major functional units, the Processor Interface Unit (PIU) comprised of the Unix Processor Interface and VICE Processor Interface sub-units, the IO Interface Unit (IOU) comprised of the CRIME MACE Interface and the CRIME GBE Interface sub-units, the Memory Interface Unit (MIU) and the Rendering Engine (RE).

The Processor Interface Unit can access all other Units. The Rendering Engine, VICE, GBE and MACE are restricted to communication with the MIU. As slaves, they respond to Unix processor PIO vi the PIU.

The exception to this would be a peer-to-peer transfer between a pair of clients on the PCI bus. Refer to Chapter 3 for the address map of the O2 workstation architecture.

This relationship is shown in Figure 1-2.

The PIU is operated at CPU_Clk rate, which has a maximum frequency of 100 MHz. The MIU, IOU and RE are operated at SysClk rate, which has a maximum frequency of 66.7 MHz. There is a small portion of the

MIU and IOU that communicate with their respective external interfaces that operate at SysClk2X so as to conserve pins on the ASIC.

Data is passed among these four functional units through two mechanisms. For transfers to/from memory, the PIU, RE and IOU each make requests of the MIU. The MIU acts as the arbiter and grants requests. The requestor is then switched into the memory path for the duration of the request which is limited to one 32 byte system memory word. Additional requests from the same client are honored based on the fixed bandwidth allocated to each requestor or lack of a competing request by any other requestor.

For I/O activity, the PIU always initiates the request. Read requests are split transaction similar to the SysAD so that the interconnect can be used for other data transfers while waiting for the read response.

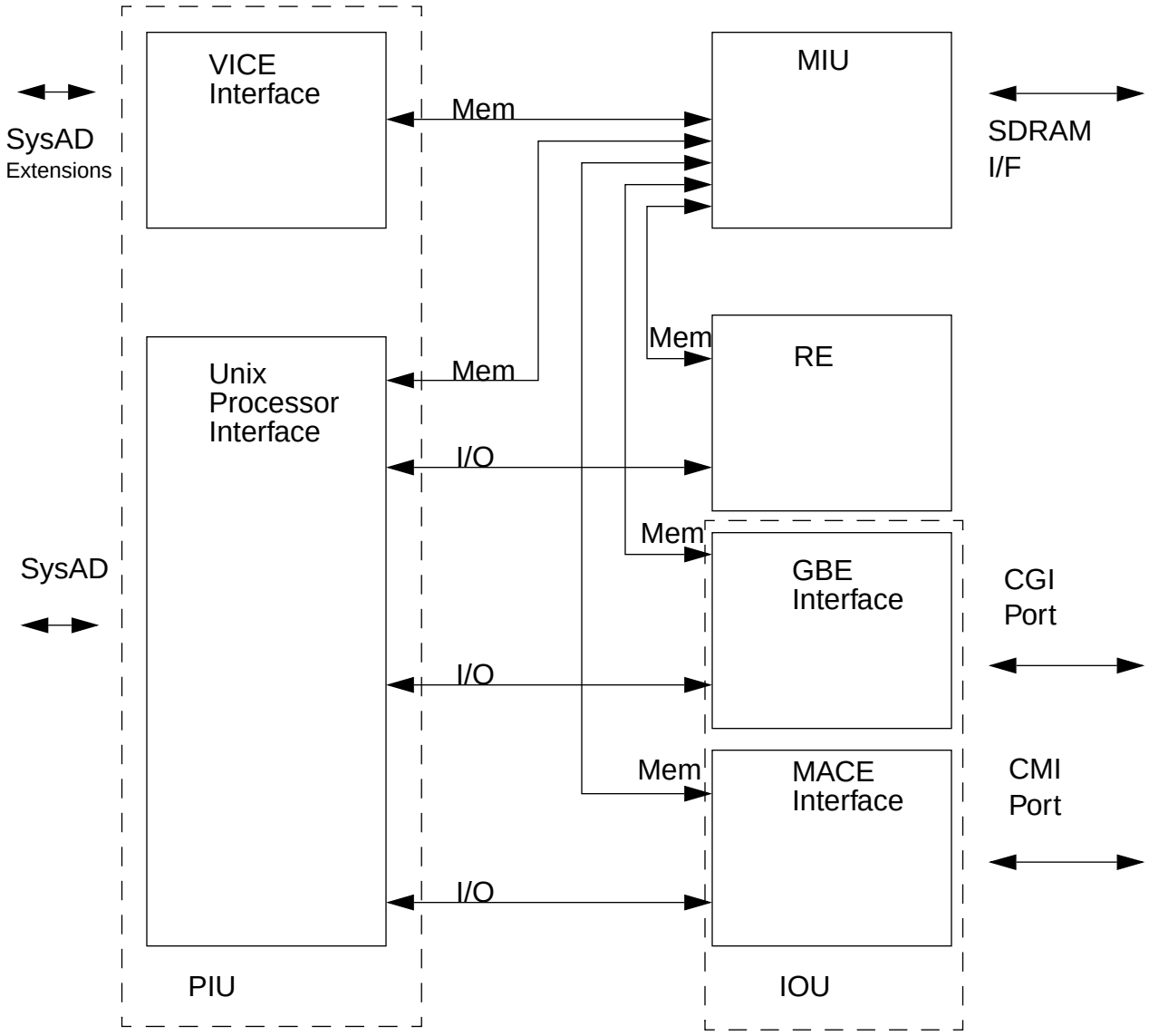


Figure 1-2 CRIME Internal Block Diagram

1.4 Key Features

CRIME's key features are as follows.

a. Uniprocessor R10K or R12K system support
b. VICE SysAD protocol extensions support
c. SDRAM memory subsystem control
d. Rendering Engine on-chip
e. CRIME MACE Interface
f. CRIME GBE Interface
g. 3.3 Volt core and IO buffers
h. IO buffer includes LVTTL, LVCMOS, PECL and HSTL logic levels
i. Package: **TBD** (was 584) TBGA
j. Technology: VLSI standard cell; 0.35 micron drawn gate length.
k. Power dissipation: **TBD** Watts

1.5 References

[1] *G2 Architecture Issues*, V0.3, by Mike Nielsen, 9/23/93.
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/doc/architecture.sc

[2] *Moosehead System Architecture*, by M. Nielsen, 7/20/94.
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/doc/archSlide.sc

[3] *Silicon Graphics Synchronous DRAM DIMM*, V0.7, by M. Nielsen, 12/15/95.
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/memory/sdsimm/doc/dimm.sc

[4] *16-bit to 32-bit Registered Bus Exchanger with Byte Masks*, SN74ALVC16280, by Texas Instrument / M. Nielsen, Rev ? 8/12/94.
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/memory/alvc16280/doc/alvc16280.sc

[5] *MIPS R10000 Microprocessor User's Manual, Version 2.0,1996.*
Network path:
http://www.t5.mti.sgi.com/t5home.html
Look for "R10K User Manual"

[6] *Moosehead CRIME/IO Interconnect*, V0.3, by M. Nielsen, 10/17/94.
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/perf/cii.sc

[7] *Moosehead Clock Distribution*, V0.1, by M. Nielsen, 10/28/93.
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/doc/clkDist.sc

[8] *Moosehead Interrupt System*, V0.0, by M. Nielsen, 7/6/94,
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/doc/interrupts.sc

[9] *Moosehead Address Space*, V0.4, by M. Nielsen, 6/19/95,
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/doc/addressSpace.sc

[10] *GBE ASIC specification for Rev 1.1*, by Rob Liston, Steve Ahlgrim, Mike Nielsen & Kamran Izadi, 7/16/96
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/gbe/doc/gbespec.fm (FRAME)
or
/hosts/wain/vault1/d1/moosehead/subsystem/gbe/doc/gbespec.ps (PostScript)

[11] *MACE I/O ASIC Specification*, Chip Revision 2.0, by Bill Dunham et. al., May 31, 1996
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/mace/doc/mace2.0/spec.book (FRAME)
or
/hosts/wain/vault1/d1/moosehead/subsystem/mace/doc/mace2.0/spec.ps (PostScript)

[12] *VICE Design Specification 099-0123-003*, Version 1.0, by Michael Fuccio et.al., 4/17/97
Network path:
/hosts/wain/vault1/d1/moosehead/subsystem/vice/doc/chip_spec/vice.book (FRAME 5)
or
/hosts/wain/vault1/d1/moosehead/subsystem/vice/doc/chip_spec/vice.ps (PostScript)

[13] *Moosehead R10000 Kernel Software*, Version 0.92, by William J. Earl, 5/12/97
Network path:
http://fir.esd.sgi.com/~wje/Miscellaneous/mh-r10000-speculation.html

[14] *VICE 1.5 Design Specification 099-0TBA-001*, Version 0.x, by Ephrem Wu et.al., 5/15/97(Mainly Chapter 3 changed from [12] above.
Network path:
/hosts/nanny/d2/doc/vice1_5/ch3_sysintfc (FRAME 5)

CHAPTER 2

Device Interface Description

The pins of the CRIME chip are divided into the following groups: Clock Pins, Host Interface Pins, Vice Interface Pins, Memory Interface Pins, CMI (CRIME-MACE Interface) Pins, CGI (CRIME-GBE Interface) Pins, Initialization Pins, Miscellaneous Functional Pins and Test Pins.

Pin names with trailing "_n" are active low signals. Otherwise, they are active high signals.

NOTE check NEC data book against original CRIME buffer types listed at: /hosts/wain/vault1/d1/moosehead/subsystem/crime/doc/crime628.pads to see if all output buffers are LVCMOS or LVTTL. Also double check SysClk1X and SysClk2X for input type LVCMOS or LVTTL. Assumed all non-SysAD inputs are LVTTL.

2.1 Summary of CRIME Pins

Table 2-1 CRIME Pin Descriptions

Pin Name	Type	Description	Output Drive	Logic Level
CPU_SysAD(63:0)	I/O	64-bit Multiplexed Address/Data bus. Bidirectional signals used to communicate with the Unix Processor and the VICE chip. Cycles on the SysAD which contain a valid address are called *address cycle*s. Cycles on the SysAD which contain valid data are called *data cycles*. For R10K/R12K Validity is identified by CPU_SysVal_n.	8 mA	HSTL 1A
CPU_SysADC(7:0) **NOT SUPPORTED**	I/O	NOT SUPPRTED ON CRIME. Tie Processor connections to Pullups.		
CPU_SysCmd(11:0)	I/O	12-bit command bus. The SysCmd bus identifies the contents of the SysAD bus during any cycle in which it is valid. SysCmd(11) is used to indicate whether the current cycle is an address cycle [SysCmd11)=0] or a data cycle [SysCmd11)=1]. For R10K/R12K connect CRIME CPU_SysCmd(11:0) to processor SysCmd(11:0).	8 mA	HSTL 1A
CPU_SysCmdP	I/O	1-bit SysCMD parity check for the SysCmd bus. For R10K/R12K odd parity?	8 mA	HSTL 1A
CPU_Clk, CPU_Clk_n	I	Differential Input System Clock for SysAD bus Synchronization. R10K/R12K differential clock input.		PECL
CPU_WrRdy_n	O	External agent ready to accept a new write.	8 mA	HSTL 1A
CPU_SysVal_n	I/O	Data from Unix processor is valid. The Unix processor and CRIME each drive this signal as bus ownership permits.	8 mA	HSTL 1A
SysResp(3,2,0)	O	System Response (R10K/R12K) SysResp(4,1) can be pulled up on the CPU Board and connected to the processor as only 4 outstanding read requests are used in a single processor R10K/R12K configuration.	8 mA	HSTL 1A
CPU_SysRespVal_n	O	System Response Valid(R10K/R12K)	8 mA	HSTL 1A

Table 2-1 CRIME Pin Descriptions

Pin Name	Type	Description	Output Drive	Logic Level
CPU_SysRespP	O	System Response Parity (R10K/R12K)	8 mA	HSTL 1A
CPU_SysRel_n	I/O	System Release (R10K/R12K) For R10K/R12K this pin is connected to processor SysRel*	8 mA	HSTL 1A
CPU_SysReq_n	I	System Request (R10K/R12K) For R10K/R12K this pin is connected to processor SysReq*		HSTL 1A
CPU_SysGnt_n	O	System Grant(R10K/R12K) For R10K/R12K this pin is connected to processor SysGnt*	8 mA	HSTL 1A
CPU_SysCorErr_n	I	CPU correctable error. R10K/R12K corrected an error on an internal datapath most likely a cache. Used by CRIME to set an interrupt to allow for quality checking of R10K/R12K processors while in factory test and during normal Unix system operation.	8 mA	HSTL 1A
ViceWrRdy_n	I	Vice Write Ready. Input to CRIME from Vice. CRIME uses this to assert CPU_SysWrRdy_n back to the processor to flow control writes on the SysAD bus to VICE. It is registered in CRIME and then logically “ored” with CRIME’s internal CPU_SysWrRdy_n generation circuitry.		LVTTL
ViceValidOut_n	I	Vice Valid Out. Indicates data from VICE is valid. VICE drives this signal which is monitored by Crime. All VICE address and data cycles to CRIME drive this signal active.		LVTTL
ViceValidIn_n	O	Vice Valid In. Indicates data from CRIME intended for VICE is valid. CRIME asserts this signal during read response data cycles for VICE.	8 mA	LVTTL
ViceSysRqst_n	I	VICE System Bus Request. VICE request to use SysAD bus. VICE drives this signal which is monitored by Crime. Any VICE initiated address or data cycle must be preceded by bus ownership. CRIME will request the SysAD bus from the Unix processor when VICE asserts ViceSysRqst_n. VICE will de-assert this signal in the cycle after it receives ViceSysGnt_n.		LVTTL

Table 2-1 CRIME Pin Descriptions

Pin Name	Type	Description	Output Drive	Logic Level
ViceSysGnt_n	O	VICE System Bus Grant. CRIME drives this signal when the Unix processor has Released the SysAD bus to CRIME as the result of a ViceSysRqst_n assertion. CRIME will assert this signal for one CPU_Sclk cycle when granting the SysAD bus to VICE.	8 mA	LVTTL
ViceRelease_n	I	VICE releases SysAD bus. VICE drives this signal which is monitored by CRIME. VICE will assert this signal for one CPU_Clk cycle when releasing the SysAD bus. VICE will perform SysAD transactions for up to 8 pipelined block transfers before releasing the SysAD bus.		LVTTL
VICE_Int_n	I	VICE drives this signal which is monitored by the CRIME chip. This is a level sensitive interrupt signal that is a collection of interrupts from devices internal to the VICE chip.		LVTTL
MEM_ADDR(13:0)	0	Memory address for SDRAM	8 mA	LVTTL
MEM_DATA(127:0)	I/0	Memory data for SDRAM	8 mA	LVTTL
MEM_DIR	0	Memory data bus direction control	8 mA	LVTTL
MEM_MASK(15:0)	0	Memory write mask (per byte)	8 mA	LVTTL
MEM_CS0_N(1:0)	0	Memory Chip Select Bank 0	8 mA	LVTTL
MEM_CS1_N(1:0)	0	Memory Chip Select Bank 1	8 mA	LVTTL
MEM_CS2_N(1:0)	0	Memory Chip Select Bank 2	8 mA	LVTTL
MEM_CAS_N	0	Memory Column Address Strobe	8 mA	LVTTL
MEM_RAS_N	0	Memory Row Address Strobe	8 mA	LVTTL
MEM_ECCMASK(1:0)	0	Memory ECC Mask	8 mA	LVTTL
MEM_ECC(15:0)	I/0	Memory Error Correction Code Field	8 mA	LVTTL
MEM_CLKE	0	Memory Clock Enable	8 mA	LVTTL
MEM_WE_N	0	Memory Write Enable	8 mA	LVTTL
GBE_DATA(63:0)	I/0	Muxed Address/Cmd & Data Bus CRIME <-> GBE	8 mA	LVTTL
GBE_TOKEN_IN	I	GBE send token to CRIME or ask for bus		LVTTL
GBE_TOKEN_OUT	0	CRIME send token to GBE or ask for bus	8 mA	LVTTL

Table 2-1 CRIME Pin Descriptions

Pin Name	Type	Description	Output Drive	Logic Level
MACE_DATA(31:0)	I/0	Muxed Address/Cmd & Data Bus CRIME <-> MACE	8 mA	LVTTL
MACE_TOKEN_IN	I	MACE send token to CRIME or ask for bus		LVTTL
MACE_TOKEN_OUT	0	CRIME send token to MACE or ask for bus	8 mA	LVTTL
MACE_WRITE_ACK	0	CRIME sends write acknowledge to MACE	8 mA	LVTTL
MACE_PIO_WR_ACK	I	MACE sends PIO write acknowledge to CRIME		LVTTL
MODE_CLK	O	CRIME drives clock to serial ROM for MODE bit initialization of R10K/R12K processor.	2 mA	LVTTL
MODE_DIN	I	CRIME receives data from serial ROM for MODE bit initialization of R10K/R12K processor.		LVTTL
CPU_SysRes_n	O	Reset to R10K/R12K processor	4 mA	LVTTL
DCOK	O	Power Stable indication to R10K/R12K processor	4 mA	LVTTL
Reset_Cond	I/O	Conditioned Reset	4 mA	LVTTL
Shorten_Reset	I/O	Shorten Reset?? What value	4 mA	LVTTL
Big_Endian	I	1 = CRIME / CPU protocol is big endian 0 = CRIME / CPU protocol is little endian **Note: Since CRIME now gets the Big Endian information as part of the R10K/R12K mode stream that CRIME reads from the Serial ROM, This pin could be eliminated.**		LVTTL
TCLK	I	Input for Boundary Scan Clock. Selected per JTAG 1149 controller mode.		LVTTL
TDI	I	Data Input Pin for Boundary Scan Test.		LVTTL
TMS	I	Test Mode Select - Boundary Scan Tap Controller		LVTTL
TRST	I	Test Mode Reset - Boundary Scan Tap Controller		LVTTL
TDO	O	Test Data Out - Boundary Scan Chain	8 mA	LVTTL
IO_OFF_N	I	1 - Ouputs Active 0 - Outputs 3-State This signal can be asserted asynchronously.		LVTTL
ROscIn	I	Ring Oscillator Input (NEC Process/Parametric Test?)		LVTTL

Table 2-1 CRIME Pin Descriptions

Pin Name	Type	Description	Output Drive	Logic Level
ROscOut1	O	Ring Oscillator Output 1 (NEC Process/Parametric Test?)	8 mA	LVTTL
ROscOut2	O	Ring Oscillator Output 2 (NEC Process/Parametric Test?)	8 mA	LVTTL
SYS_CLK1X	I	System 1X Input Clock - Nominally 66.67 MHz		LVC-MOS?
SYS_CLK2X	I	System 2X Input Clock - Nominally 133 MHz		LVC-MOS?
Sys_PLL_AGND	GND	Analog Phase Lock Loop Ground		
Sys_PLL_APWR	PWR	Analog Phase Lock Loop Power		
Sys_PLL_BYPASS	I	1- Bypass Internal Phase Lock Loop 0- Use Internal Phase Lock Loop - Normal System Config.		LVTTL
SysDIV2O	O	Phase Lock Loop Clock Signal / 2 Output for Test Purposes	2 mA	LVTTL
Sys_PLL_REF	O	Phase Lock Loop Reference Signal Output for Test Purposes	2 mA	LVTTL
Sys_PLL_FB	O	Phase Lock Loop Clock Signal Output for Test Purposes	2 mA	LVTTL
Sys_PLL_EN	I	Phase Lock Loop enabled.		LVTTL
CPU_PLL_AGND	GND	Analog Phase Lock Loop Ground		
CPU_PLL_APWR	PWR	Analog Phase Lock Loop Power		
CPU_PLL_BYPASS	I	1- Bypass Internal Phase Lock Loop 0- Use Internal Phase Lock Loop - Normal System Config.		LVTTL
CPUDIV2O	O	Phase Lock Loop Clock Signal / 2 Output for Test Purposes	2 mA	LVTTL
CPU_PLL_REF	O	Phase Lock Loop Reference Signal Output for Test Purposes	2 mA	LVTTL
CPU_PLL_FB	O	Phase Lock Loop Clock Signal Output for Test Purposes	2 mA	LVTTL
CPU_PLL_EN	I	Phase Lock Loop enabled.		LVTTL
VcQSys	I	VCC for the System Interface Output Drivers		
VrefSys	I	Voltage reference for the System interface input receivers		

Table 2-1 CRIME Pin Descriptions

Pin Name	Type	Description	Output Drive	Logic Level
HSTL_IEN	I	Input enable pin for HSTL inputs requested by NEC (foundry for the part). Tie to VDD in System for normal operation.		
VDDI	PWR	?? Pins Core +3.3V Power (24 more pad connections from die to package internal)		
VDDE	PWR	?? Pins I/O +3.3V Power		
VSSI	GND	?? Pins Core Ground (28 more pad connections from die to package internal)		
VSSE	GND	?? Pins I/O Ground (88 more pad connections from die to package internal)		
VSS_PKG	GND	?? Ground connections to TBGA internal Ground Plane.		

2.2 Clock Pins

2.2.1 CPU_Clk, CPU_CLK_n

Complementary system clock signals. Differential Low Voltage (3.3v) PECL Input.

When the CRIME chip is used in a R10K/R12K system, SysClk is the clock reference for the R10K/R12K processor, the CRIME chip and the VICE chip. A possible clock distribution scheme in a R10K/R12K system is shown in the following figure.

In the figure, it is shown that the SysClk is generated from a clock source, typically a crystal oscillator. The clock source is buffered through an output-matched clock driver. This clock driver is ***not*** PLL-based to reduce clock-jitter related uncertainty. The clock driver should have 3 pairs of the outputs. One output pair connects to CRIME's CPU_Clk input pins. One output pair to the R10K/R12K SysClk input pins and the final pair to the VICE Sclock input pins.

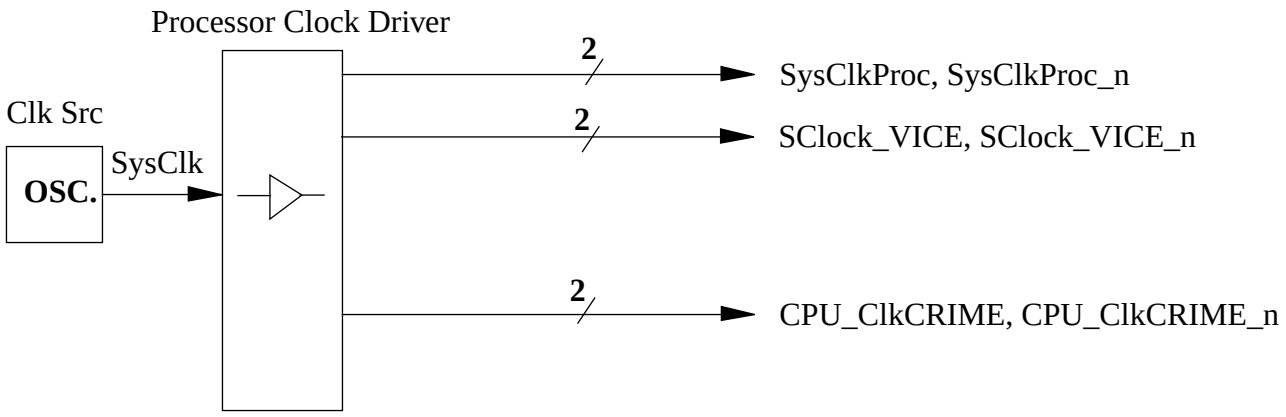


Figure 2-1 SysClk Distribution

CPU_Clk and CPU_Clk_n input pins to CRIME are used as the reference for the on-chip PLL-based clock driver, which drives the internal clock distribution network, which is used to control the circuitry in the PIU.

The differential inputs drive a differential input buffer, which shifts the logic level to the internal LVCMOS level. The buffered signal is used as the reference to the on-chip PLL clock driver.

2.2.2 SysClk2X, SysClk1X

These are the two System Clock inputs to CRIME. They are nominally 133 and 66.67 MHz respectively.

The 1X clock is for all internal CRIME processing that includes the Rendering Engine, the Memory Interface Unit and the System Clock Domain side of the Processor Interface Unit (PIU).

The 2X clock is for interconnects to the Memory at a 2X frequency data rate in order to save pins on CRIME. The same technique is used for the GBE and MACE interconnect.

The SysClk2X is reconstructed on-chip with a phase lock loop. An on-chip SysClk1X is also derived from this phase lock loop. The SysClk1X input pin is used to produce the correct phase relationship between the on-chip PLL derived SysClk1X with SysClk1X of other chips in the system.

Note that there is no relationship between the SysClk1X 66.67 MHz clock in CRIME and the Vice VCLOCK signal. The CRIME <-> VICE interconnect is all synchronized to the CPU_CLK, CPU_CLK_n domain.

2.3 Processor Interface Pins

Processor interface pins have HSTL interface logic level. HSTL mode will be used with R10K/R12K processors.

The processor interface pins are named so they match the pin names of the R10K/R12K processor. The exception to this is that "_n" is used in this document and in the RTL description of CRIME to indicate an active low signal pin. The R10K/R12K documentation uses the "*" character or sometimes "b" to indicate an active low signal.

The functions of the pins for use in a R10K/R12K system are described below.

2.3.1 CPU_SysCmd(11:0)

System command bus. HSTL Bidirectional.

See [5] *MIPS R10000 Microprocessor User's Manual, Version 2.0,1996.* for a detailed description of the encoding types.

According to the above document, the SysCmd encoding varies depending on the request and response types.

2.3.2 CPU_SysCmdPar

System command bus parity. HSTL Bidirectional. Odd parity for the Sys_cmd(11:0) bus.

The CRIME chip always checks odd parity, as specified for the R10K/R12K processor. It will produce odd parity when generating SysCmd codes.

2.3.3 CPU_SysAD(63:0)

System address/data bus. HSTL Bidirectional.

See [5] *MIPS R10000 Microprocessor User's Manual, Version 2.0,1996.* for a detailed description.

2.3.4 CPU_SysAD_Chk(7:0)

System address/data check bus. HSTL Bidirectional.

SysAD_Chk bus is used to transfer ECC codes in R10K/R12K systems.

CRIME will not perform ECC checking on transactions from the R10K/R12K or VICE.

CRIME will not provide ECC check bits on transactions to the R10K/R12K or VICE.

These pins should be pulled up on the R10K/R12K processor using large value resistors.

2.3.5 CPU_SysVal_n

System bus (SysAD and SysCmd) valid. HSTL Bidirectional.

See *System Design Using T5* for detailed description.

In a R4K system, this signal is an input. It is used to connect to ValidOut* pin on R4K.

2.3.6 CPU_SysRdRdy_n

System read ready. HSTL Output.

See [5] *MIPS R10000 Microprocessor User's Manual, Version 2.0,1996.* for a detailed description.

Not implemented in CRIME, as the input command queue in CRIME will always be able to accept four outstanding reads from the R10K/R12K processor, even when the CRIME input write buffer is full.

2.3.7 CPU_SysWrRdy_n

System write ready. HSTL Output.

See [5] *MIPS R10000 Microprocessor User's Manual, Version 2.0,1996.* for a detailed description.

This signal is negated when the internal resources in the CRIME chip are insufficient to accept more write requests from processor. There are a few such resources involved in determining when this signal should be negated.

2.3.8 CPU_SysReq_n

System bus requests. HSTL Input.

Asserted by the R10K/R12K processor to request ownership of the SysAD bus.

See [5] *MIPS R10000 Microprocessor User's Manual, Version 2.0,1996.* for a detailed description.

2.3.9 CPU_SysGnt_n

System bus grants. HSTL Output.

The grant signal is used with the corresponding request signal to form a pair of handshake signals between the CRIME and the R10K/R12K processor. See [5] *MIPS R10000 Microprocessor User's Manual, Version 2.0,1996.* for a detailed description.

This signal is also used to load the R10K/R12K processor's Mode Register during initialization. See the Chapter 8 for details.

2.3.10 CPU_SysRel_n

System bus release. HSTL Bidirectional.

See [5] *MIPS R10000 Microprocessor User's Manual, Version 2.0,1996.* for a detailed description.

2.3.11 CPU_SysState(2:0), CPU_SysStatePar, CPU_SysStateVal_n

These signals are not implemented in the single processor R10K/R12K CRIME design.

2.3.12 CPU_SysResp(4:0)

System response bus. HSTL Output.

CRIME drives response indications through this bus to the R10K/R12K processor.

See [5] *MIPS R10000 Microprocessor User's Manual, Version 2.0,1996.* for a detailed description.

CRIME only implements CPU_SysResp(3,2,0). CPU_SysResp(4,1) will be connected to the R10K/R12K as a pullup resistor.

2.3.13 CPU_SysRespPar

System response bus parity. HSTL Output.

Odd parity for the SysResp(4:0) bus. See [5] *MIPS R10000 Microprocessor User's Manual, Version 2.0,1996.* for a detailed description.

2.3.14 CPU_SysRespVal_n

System response bus valid. HSTL Output.

CRIME asserts this signal when there are valid information on the CPU_SysResp(4:0) bus.

2.3.15 CPU_SysCorErr_n

System Correctable Error. HSTL Input.

The assertion of the signals indicates the R10K/R12K corrected a single bit ECC error on it's internal data path most likely due to a cache bit error.

2.3.16 CPU_SysVref

Voltage reference for the SysAD interface. Analog Input.

This is the reference voltage for HSTL interface.

2.4 Memory Interface Pins

Memory interface pins have LVTTL interface logic levels. They are described in the table above. For more information on the Memory Interface Unit, refer to Chapter 6.

2.5 CRIME GBE Interface Pins

The CGI (CRIME GBE Interface) is described in detail in Chapter 4

2.6 CRIME MACE Interface Pins

The CMI (CRIME MACE Interface) is described in detail in Chapter 4

2.7 Initialization Pins

The CRIME chip supports the initialization sequences for the R10K/R12K processor family. The R10K/R12K processor has two reset pins: VCCOK and SysReset*.

CRIME supports the mode bit loading for the R10K/R12K processor.

The CRIME chip supports the Power-on reset and Cold Reset sequences for the R10K/R12K processor The details of the Reset Sequences are described in the Chapter 8, "*System Initialization and Reset*."

2.7.1 Pin Assignments

This table should be updated to reflect the mapping of signal names to physical Ball Grid Array alpha-numeric designations for the package chosen for CRIME 1.5

The following table provides the physical pin to signal name assignments sorted by the TGBA pin number.

Table 2-2 584 BGA Pin Assignments - Pin Order

Pin #	Signal	Pin #	Signal	Pin #	Signal
A01		A02		A03	
A04		A05		A06	
A07		A08		A09	
A10		A11		A12	
A13		A14		A15	
A16		A17		A18	
A19		A20		A21	
A22		A23		A24	
A25		A26		A27	
A28		A29			

The following table provides the physical pin to signal name assignments sorted by the Signal Name from the CRIME VHDL Data Base.

Example shown. Could be updated to show CRIME 1.1 for now.

Table 2-3 Signal Name - Pin Assignment

Signal Name	TBGA Pin #	Die Pad #
PLL_AG (Example)	L22	316

2.7.2 Test Modes

Describe JTAG mode here.

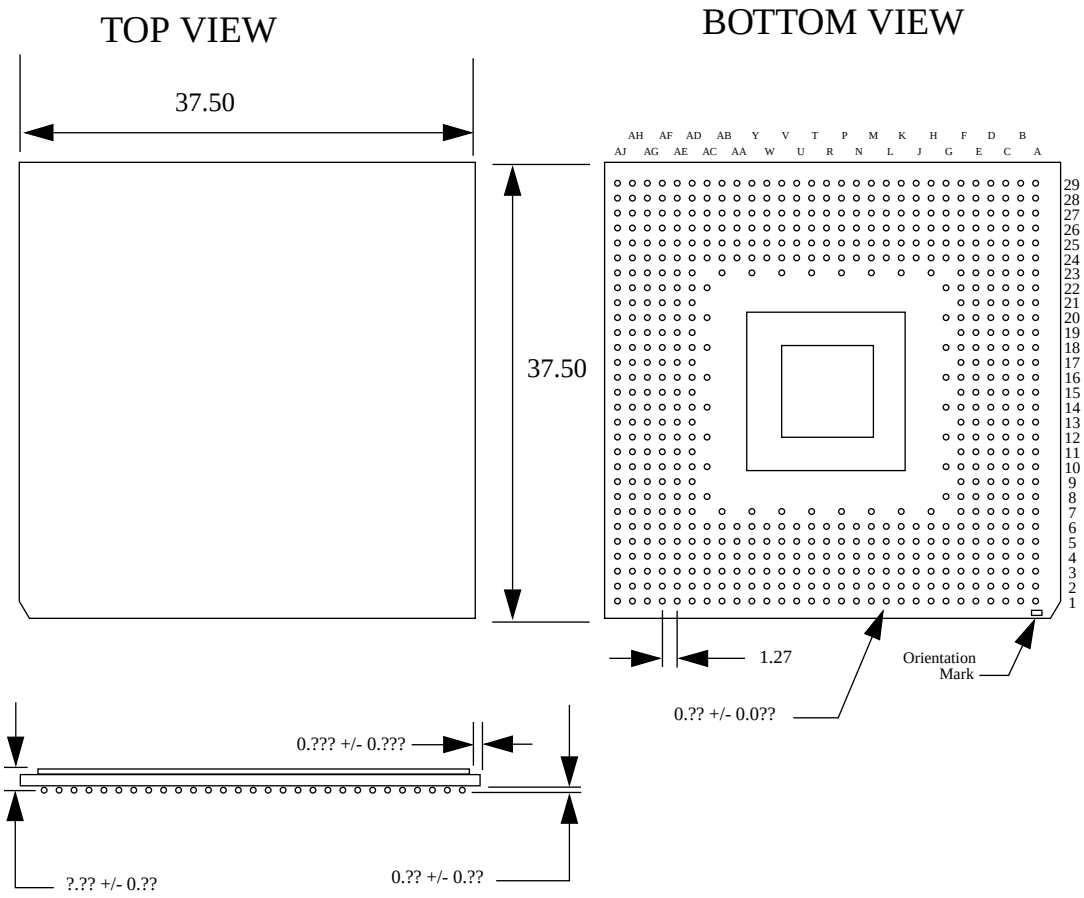
Table 2-4 JTAG Clock Test Modes

VICE Function				SysAD Flops	All Other Vice Flops

2.7.3 Schematic Icon

2.7.4 Physical Packaging Diagram

584 Lead Tab Ball Grid Array package from NEC. **Update this drawing to show correct package. Probably just fill in some of the empty ball locations on the inner perimeter.**



Refer to NEC Packaging Drawing for Co-planar specifications.

All Units in mm

Figure 2-2 584 Lead Tab Ball Grid Array

2.7.5 Physical Package Markings

Package Markings for the CRIME 1.5 chip

Figure 2-3 Package Markings

2.7.6 Bonding Diagram

Refer to Table 3, “Signal Name - Pin Assignment,” on page 15

CHAPTER 3 Internal Registers

3.1 System Address Map

This information is taken from the document *Moosehead Address Space [9]*. Refer to Chapter 1 for the location of this reference.

This section describes the system level partitioning of the Moosehead Address Space from the perspective of the various possible transaction initiators. The address space is only described to the resolution of an ASIC or option module. For further details of the lower level register and memory address assignments refer to the detailed ASIC or option module specifications. See *GBE ASIC specification for Rev 1.1 [10]*, *MACE I/O ASIC Specification [11]*, *VICE Design Specification 099-0123-003 [12]* referenced at the end of Chapter 1. For the CRIME registers, reference this document.

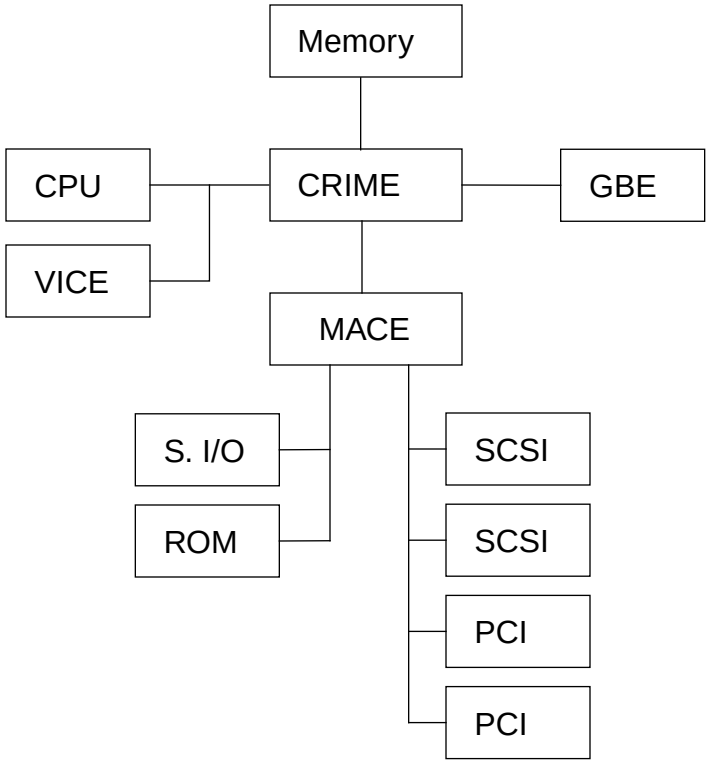
3.1.1 Overview

The possible initiators of transactions are the: CPU, VICE, CRIME rendering engine; GBE; MACE audio, video, and Ethernet DMA engines; Super I/O parallel and serial DMA engines; PCI SCSI interfaces, and PCI

option slots. The CPU can access any and all memory or I/O devices, whereas other initiators are restricted to accessing only system memory. The one exception is PCI options that can perform local transactions between option slots.

The following block diagram shows the key subsystems with respect to the address space partitioning described in this document.

Figure 3-1 Moosehead Address Clients



3.1.2 CPU Address Space

The CPU implements a 40-bit physical address space that allows direct access to all the base system resources as well as direct access to the PCI 32-bit memory space and PCI 32-bit I/O space. In order to allow the operating system to efficiently access the base system I/O resources, they are all mapped to the CPU's KSEG1 region. In addition, the first 32 MBytes of the PCI memory and I/O spaces are aliased into the KSEG1 region so that in typical configurations all PCI resources may also be accessed via KSEG1. The full 32-bit PCI I/O and memory spaces may be accessed but via translation buffer entries.

For system memory, the full 1GByte memory space is directly accessible to the CPU, but the first 256 MBytes are aliased to KSEG0 so that operating system structures may be mapped without using TLB entries. To access the full 1GByte memory configuration, translation buffer entries are required. In addition, the high 512MBytes of memory is aliased into the upper 512MBytes of the first 1GByte of address space. This allows the option of more compact page tables by accessing this region rather than the 1.5GByte to 2GByte address range. A given physical memory location must only be accessed at either the low memory alias, the high memory alias, or the full memory region. The three choices cannot be intermixed or uncoherent aliases will result in the caches. The frame buffer and depth buffer regions allow linear address space access to the current 2Kx2Kx32-bit frame buffer and depth buffer mapped by the CRIME rendering engine. In addition, there is a 1GByte no-ECC alias for memory that allows the CPU to access any location of physical memory without ECC checking or correction. This allows the CPU to read the raw contents of memory during ECC error fixup routines without disabling the ECC logic in the memory controller for other DMA transactions, or directly reading memory written by the CRIME rendering engine without generating ECC errors. Note that for references to the no-ECC alias, only checking and correction is suppressed; ECC is still generated for write transactions, and partial (sub-doubleword) writes still generate read-modify-write memory transactions.

After the CPU interface in CRIME performs a first level decode and forwards the transaction to the appropriate subsystem, each of the CRIME, GBE, MACE, and VICE ASICs is responsible for further decode within their assigned register/buffer regions.

Table 3-1: CPU Address Space

Start Address	Size	Function	ASIC Responding
0x10 0000 0000	960 GB	Reserved	-
0x03 0000 0000	52 GB	Reserved	-
0x02 0000 0000	4 GB	PCI Memory	MACE
0x01 0000 0000	4 GB	PCI I/O	MACE
0x00 C000 0000	1 GB	Reserved	-
0x00 8000 0000	1 GB	No-ECC Memory	CRIME
0x00 4000 0000	1 GB	Memory	CRIME
0x00 2000 0000	512 MB	Hi Memory	?
0x00 1FC0 0000	4 MB	System ROM	MACE
0x00 1F80 0000	4 MB	Super IO Regisers	MACE
0x00 1F40 0000	4 MB	A/V Registers	MACE
0x00 1F00 0000	4 MB	IO Registers	MACE
0x00 1C40 0000	44 MB	Reserved	MACE
0x00 1C00 0000	4 MB	PCI Configuration	MACE
0x00 1A00 0000	32 MB	PCI Low Memory	MACE
0x00 1800 0000	32 MB	PCI Low I/O	MACE
0x00 1700 0000	16 MB	VICE Registers/RAM	VICE
0x00 1600 0000	16 MB	GBE Registers	GBE
0x00 1400 0000	32 MB	CRIME Registers	CRIME
0x00 1200 0000	32 MB	Depth Buffer	CRIME
0x00 1000 0000	32 MB	Frame Buffer	CRIME
0x00 0000 0000	256 MB	Low Memory	CRIME

3.1.3 VICE Address Space

VICE implements a 40-bit physical address space as per the SYSAD protocol, however, VICE is restricted to only accessing system memory. Note that this address decode range has been extended from 36 bits over the CRIME 1.1 implementation for R5K processors.

VICE only needs to generate the low order 32 bits of the physical address and zero the most significant eight bits. The diagram below shows the address space for VICE DMA transactions. VICE must still decode the full 40-bit physical address space during CPU transactions and respond to references to its 16 MByte region of address space as defined for the CPU.

Table 3-2: VICE Address Space

Start Address	Size	Function	ASIC Responding
0x10 0000 0000	960 GB	Reserved	-
0x00 C000 0000	61 GB	Reserved	-
0x00 8000 0000	1 GB	No-ECC Memory	CRIME
0x00 4000 0000	1 GB	Memory	CRIME
0x00 0000 0000	1 GB	Reserved	-

3.1.4 GBE Address Space

GBE implements a 32-bit physical address space on its interconnect to CRIME, however it is restricted to only accessing system memory. The diagram below shows the address space for GBE DMA transactions. GBE must decode incoming transactions from the CPU (via CRIME) to its internal registers and buffers.

Table 3-3: GBE Address Space

Start Address	Size	Function	ASIC Responding
0x 8000 0000	2 GB	Reserved	-

Table 3-3: GBE Address Space

Start Address	Size	Function	ASIC Responding
0x 4000 0000	1 GB	Memory	CRIME
0x 0000 0000	1 GB	Reserved	-

3.1.5 MACE Address Space

MACE implements a 32-bit physical address space on its interconnect to CRIME, however it is restricted to only accessing system memory. The diagram below shows the address space for MACE DMA transactions from internal or Super I/O devices. MACE must decode incoming transactions from the CPU (via CRIME) to its internal registers and buffers, or forward them on to the PCI, Super I/O, or ROM devices as appropriate.

Table 3-4: MACE Address Space

Start Address	Size	Function	ASIC Responding
0x C000 0000	1 GB	Reserved	-
0x 8000 0000	1 GB	No-ECC Memory	CRIME
0x 4000 0000	1 GB	Memory	CRIME
0x 0000 0000	1 GB	Reserved	-

3.1.6 PCI I/O Address Space

PCI supports a 32-bit I/O address space. This is partitioned into a local region that corresponds to the PCI devices themselves and a memory region that allow access to system memory. PCI devices may also use the local region for peer-to-peer transactions as the IO ASIC does not respond to I/O transactions from PCI devices to the local region.

There are two aliases for system memory, one with native endian addressing semantics and one with reverse endian addressing semantics. For references to the native endian memory region, bytes are passed through in the native order of the memory system. For references to the reverse endian memory region, sub-64-bit references have the bytes swapped to the reverse endianess of the memory system. This allows device drivers to program DMA to the region that most closely matches the endianess of the device.

The diagram below shows the address space for PCI I/O transactions from the SCSI device or option slots. The IO ASIC should not cache references to system memory via I/O space.

The SCSI device and option slots must decode incoming I/O transactions as per their configuration register programming. Typically the PCI devices have their I/O space resources mapped to the first 32 MBytes of the local space so the CPU can access them via KSEG1.

Table 3-5: PCI I/O Address Space

Start Address	Size	Function	ASIC Responding
0x 8000 0000	2 GB	Local PCI I/O	PCI BUS
0x 4000 0000	1 GB	Native Endian Memory	CRIME
0x 0000 0000	1 GB	Reverse Endian Memory	CRIME

3.1.7 PCI Memory Address Space

PCI supports a 32-bit memory address space. This is partitioned into a local region that corresponds to the PCI devices themselves and a memory region that allow access to system memory. PCI devices may also use the local region for peer-to-peer transactions as the MACE ASIC does not respond to memory transactions from PCI devices to the local region.

There are two aliases for system memory, one with native endian addressing semantics and one with reverse endian addressing semantics. For references to the native endian memory region, bytes are passed through in the native order of the memory system. For references to the reverse endian memory region, sub-64-bit references have the bytes swapped to the reverse endianess of the memory system. This allows device drivers to program DMA to the region that most closely matches the endianess of the device.

The diagram below shows the address space for PCI memory transactions from the SCSI device or option slots. The MACE ASIC should cache references to system memory via memory space. There is no hardware coherency between CPU memory references and PCI memory space references to the cache in the MACE ASIC. Instead, device drivers must explicitly manage the coherency of MACE ASIC cached data via flushing, invalidation, and temporal mutual exclusivity of access to cached DMA buffers.

The SCSI device and option slots must decode incoming memory transactions as per their configuration register programming. Typically the PCI devices have their memory space resources mapped to the first 32 MBytes of the local space so the CPU can access them via KSEG1.

Table 3-6: PCI Memory Address Space

Start Address	Size	Function	ASIC Responding
0x 8000 0000	2 GB	Local PCI I/O	PCI BUS
0x 4000 0000	1 GB	Native Endian Memory	CRIME
0x 0000 0000	1 GB	Reverse Endian Memory	CRIME

3.1.8 PCI configuration Space

Refer to the *MACE I/O ASIC Specification [11]* referenced in Chapter 1 for PCI configuration Space details.

3.2 PIU Registers

Refer to Chapter 5 for the Processor Interface Register address offsets and functions.

3.3 MIU Registers

Refer to Chapter 6 for the Memory Interface Unit Register address offsets and functions.

3.4 Rendering Engine Registers

Refer to Chapter 7 for the Rendering Engine Register address offsets and functions.

CHAPTER 4 Input Output Unit

This block is comprised of the CGI (CRIME GBE Interconnect) and the CMI (CRIME MACE Interconnect).

For DMA requests generated by GBE or MACE chips, these blocks make requests to the internal CRIME MIU to perform DMA writes and reads.

For PIO requests from the Processor Interface Unit, these blocks perform writes and reads on the interconnect.

Interrupts from GBE or MACE are performed as transactions by GBE or MACE.

4.1 Overview

The Moosehead CRIME/MACE Interconnect (CMI) supports I/O devices integrated into the MACE ASIC and PCI I/O devices attached to the MACE ASIC. CMI has the following key characteristics:

Multiplexed command, address, and data signals

64-bit logical bus width

1- to 8-byte masked sub-block transfers
1- to 32-doubleword block transfers
Split read transactions
Routing tags in transfer headers
Interrupts posted via transactions

The fundamental 64-bit logical bus width is used to convey command/address fields as well as data. To support CPU access to registers and buffers in the I/O devices, 1- to 8-byte sub-block transfers allow load/store transfers as supported by the MIPS instruction set. The sub-block transfers also allow I/O devices to access memory at the byte level. However, since the memory system implements ECC protection on a 64-bit basis, sub-block DMA is very inefficient and strongly discouraged. CMI supports 1- to 32-doubleword block transfers to allow efficient use of the memory system with 32-doubleword block transfers being most efficient.

Since there is over an order-of-magnitude mismatch between the bandwidth of a PCI bus and the memory system, and there are multiple integrated I/O devices internal to the IO ASIC, CII implements split read transactions. This allows multiple outstanding memory read transactions to be transferred to the memory controller while other traffic such as memory writes from other devices and CPU accesses to I/O device registers continue to use the I/O interconnects.

Because of the split read transactions, as well as the distributed nature of the I/O adapters attached to the system ASIC and multiple processors (primary CPU and VICE), it is necessary to route read response data back to the appropriate source of the transaction. Thus the header of each transfer contains routing tag information.

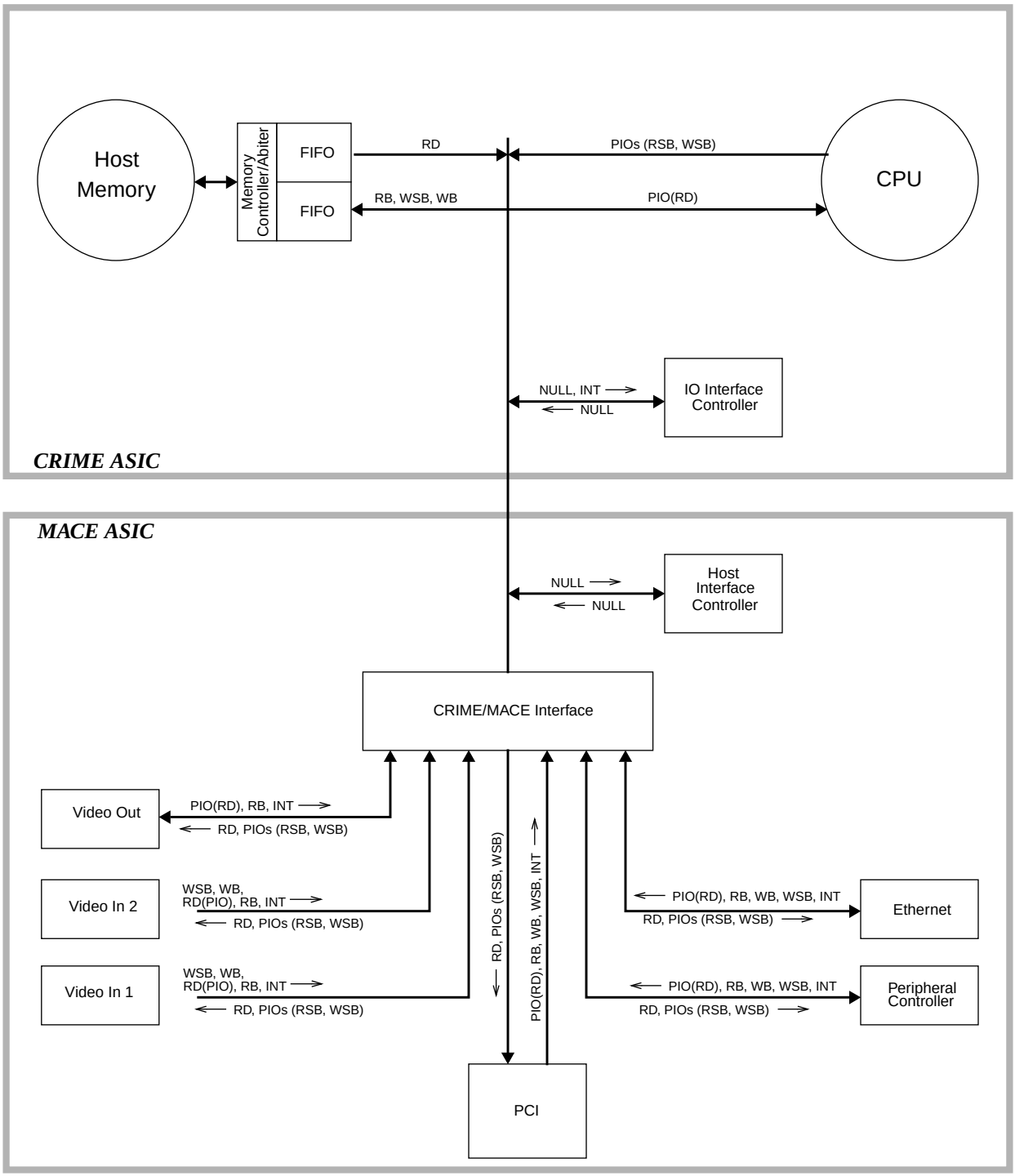
Because of the potentially large number of interrupt sources between the audio, video, and PCI I/O subsystems, interrupts are posted via transactions to central interrupt resources in the CRIME ASIC. This allows finer grain assignment of the interrupt sources to the hardware interrupt of the MIPS CPU.

4.2 CRIME MACE Interconnect Data Flow

The *Moosehead* system I/O asic contains a high speed link to the CRIME asic. The interface is a high speed queued message passing interface that uses 64-bit wide data and control words. The CRIME Link in-

terface controls all communication between MACE and CRIME. A high level message path picture is shown on the following page:

Figure 4-1 MACE - CRIME communication path



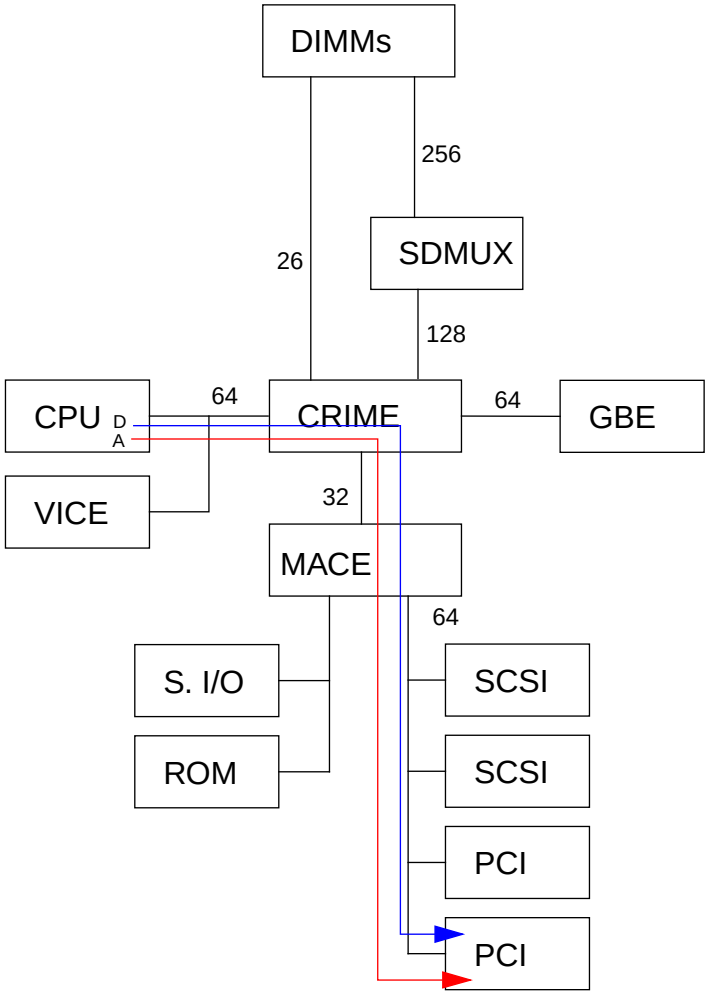


Figure 4-2 CRIME I/O Write Path

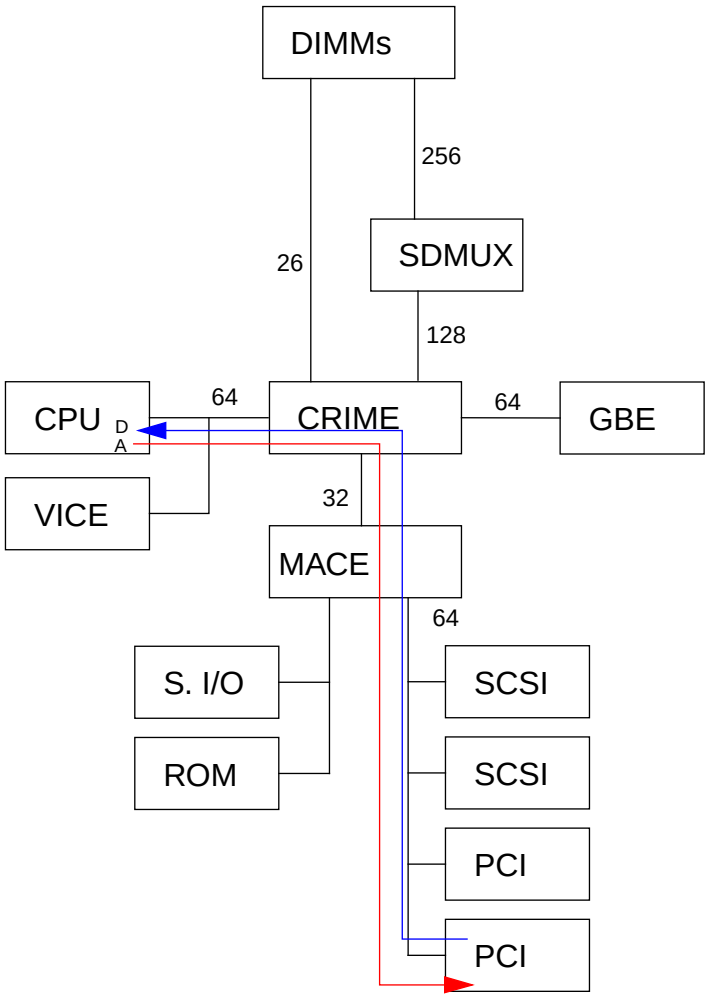


Figure 4-3 CRIME I/O Read Path

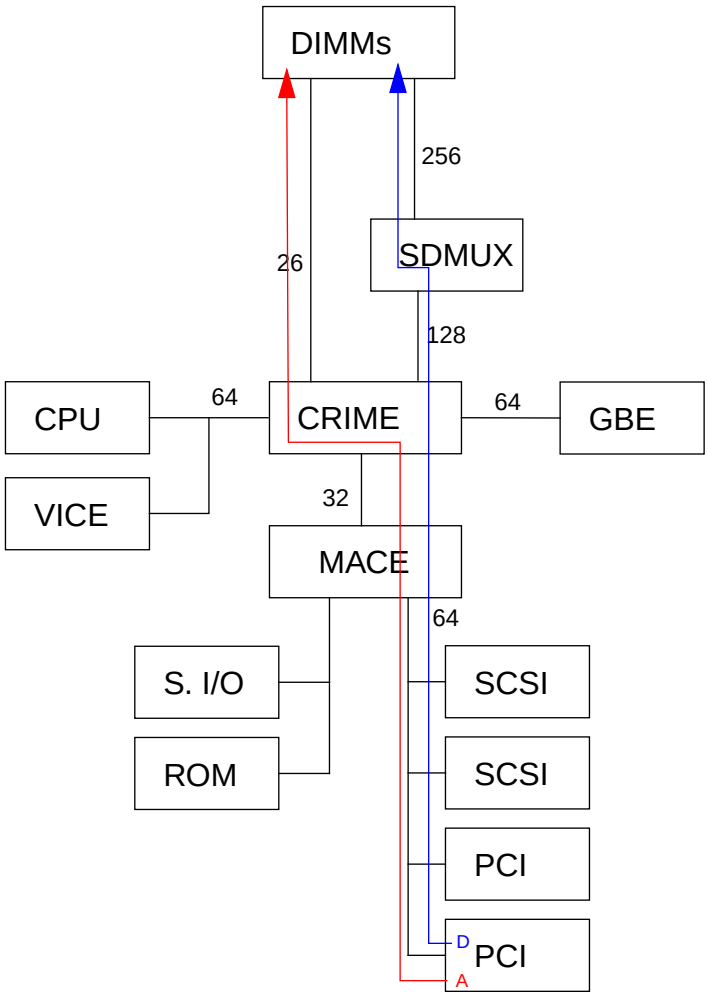


Figure 4-4 I/O Memory Block Write Path

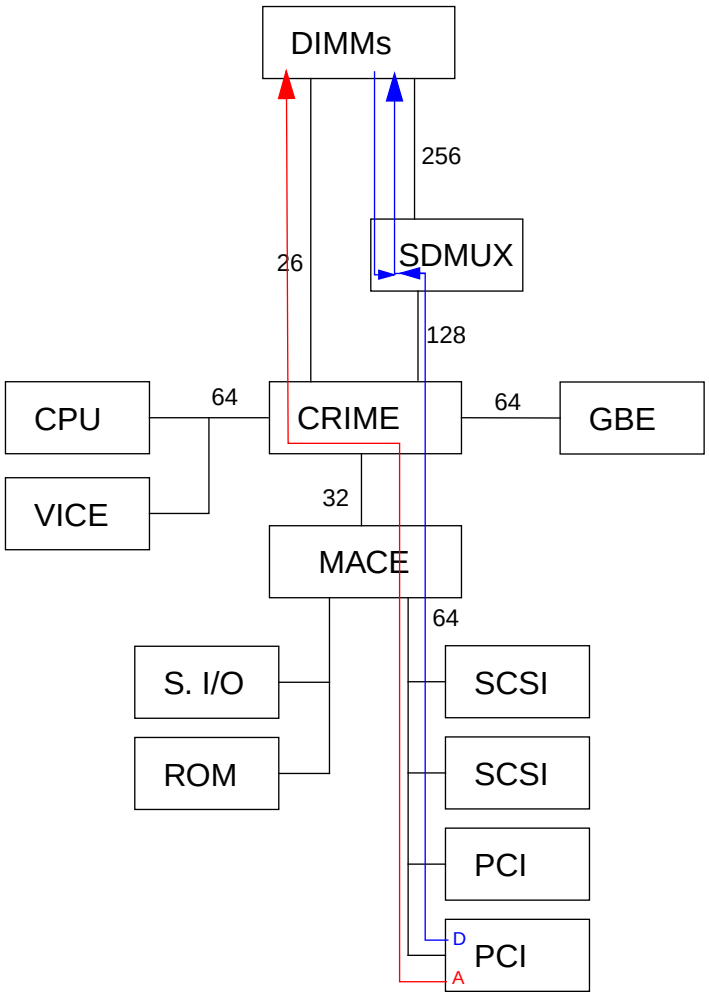


Figure 4-5 I/O Memory Sub-Block Write Path

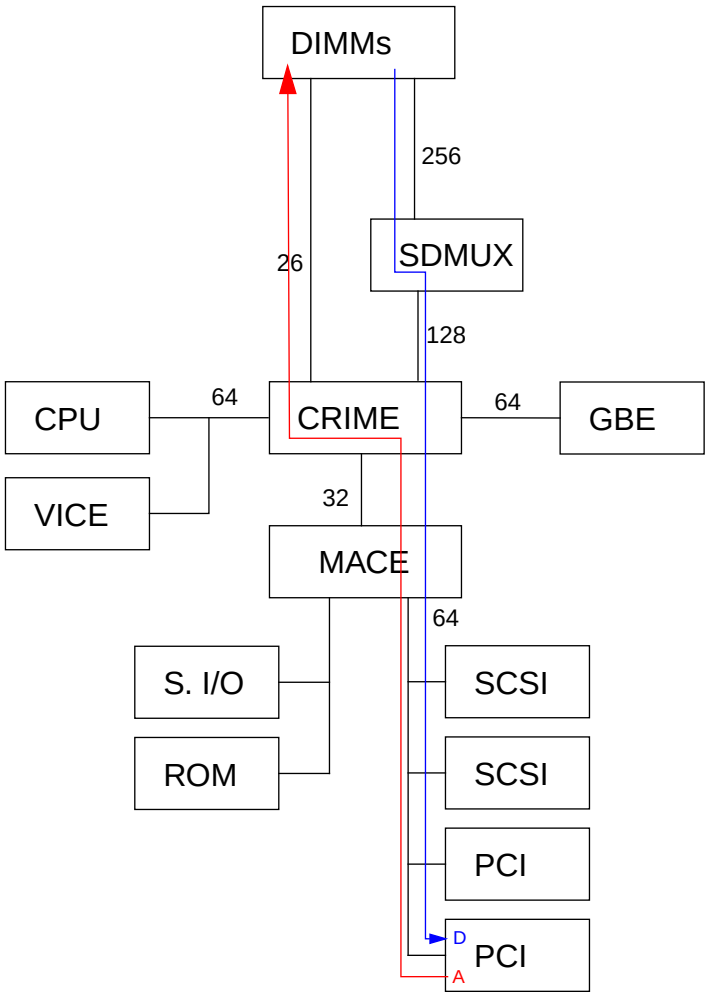


Figure 4-6 I/O Memory Read Path

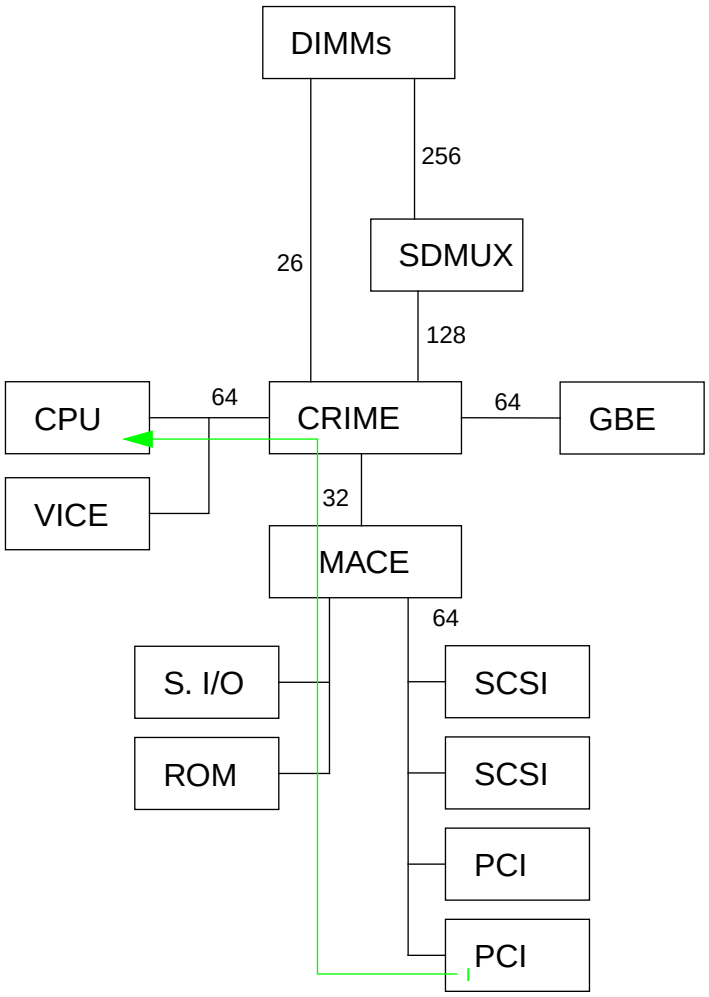


Figure 4-7 I/O Interrupt Path

Table 4-1 CMI Signal Definitions

Signal Name	Width	Direction	Description
MACE_DATA[31:0]	32	Bidirectional	Data bus
MACE_TOKEN_IN	1	Input	Command for CRIME. When asserted MACE waits for MACE_TOKEN_OUT to de-assert. This constitutes bus ownership by MACE
MACE_TOKEN_OUT	1	Output	Command for MACE. De-asserted, MACE can drive bus.
MACE_WRITE_ACK	1	Output	CRIME acknowledges a write by MACE
MACE_PIO_WR_ACK	1	Input	MACE acknowledges a PIO write by CRIME
	36		

4.3 General Description

The System Interface consists of the host bus interface, bus selector and FIFOs. Together they provide the communication path for DMA and PIO packets to be sent between CRIME and subsytemswithin the MACE ASIC. Packets received from the host are routed directly to the appropriate FIFOs. Packets sent from the subsystems are qued in their individual FIFOs and sent to the host based on a fixed arbitration scheme. The figure at the start of this chapter shows the basic block diagram of the System Interface and the data paths to/from the subsystems.

Key characteriests of the System Interface:

Half-duplex, Synchronous Host Interface
DMA Transaction Flow Control
FIFO and Host Bus Error Notification

4.3.1 CRIME Interface

The CRIME Interface implements a half-duplex, 32-bit, double-speed, synchronous bus. The data bus runs at twice the internal clock rate to achieve the effective 64-bit per cycle transfer rate. Bus mastership is negotiated via the TOKEN_IN and TOKEN_OUT signals. Upon reset and power up, bus mastership defaults to CRIME. MACE negotiates for bus ownership by asserting its TOKEN_OUT signal and waiting for TOKEN_IN to be released. To flow control write transactions there are write acknowledge signals back to each ASIC that indicate when an outstanding write transaction has completed.

4.3.1.1 Data Format and Endianness

All data passed through the CMI is expected to be byte ordered in big-endian format. No support is available for switching endianness within the CMI. The 64-bit doublewords passed between CRIME and MACE will be sent across the 32-bit bus as two 32-bit word tranfers.The following figure illustrates the byte/word ordering.

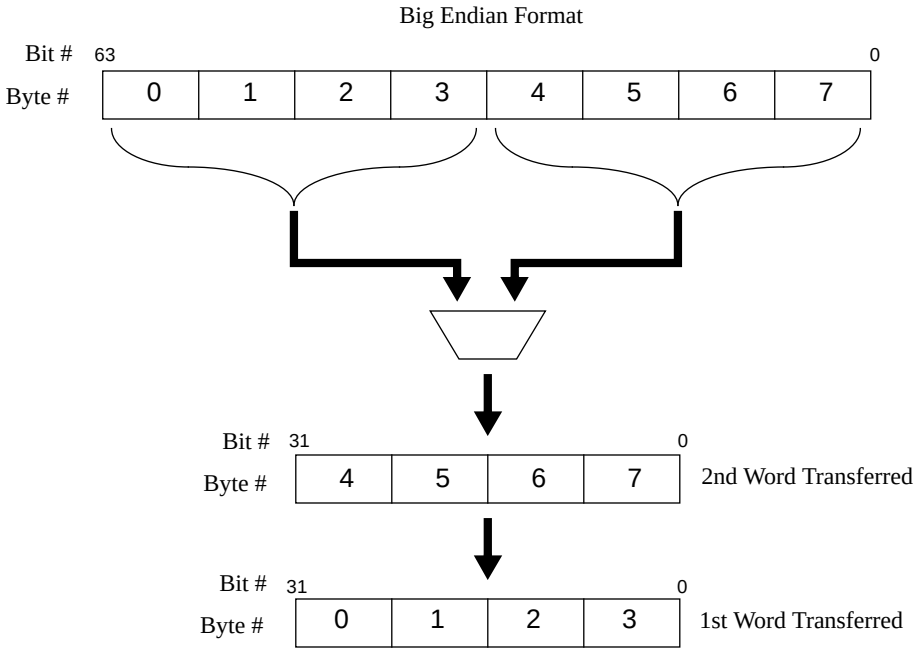


Figure 4-8 Byte Ordering of Bus Transfers

4.3.2 Transaction Ordering

Transactions from CRIME to MACE are loaded into the subsystem FIFOs in the order issued. Similarly, transactions issued from a subsystem are passed onto CRIME in the order issued. However, the ordering of transactions from different subsystems to CRIME is dependent upon the arbitration scheme. There is no ordering between transactions initiated by CRIME and transactions initiated by MACE's subsystems. These two types of transaction are flowing in opposite and independent directions.

4.3.3 Transaction Flow Control

Transaction flow control is maintained by the initiator assuring that the number of outstanding transactions it generates doesn't exceed the maximum allowed. This requires the host to keep track of the number of outstanding PIO transactions, and MACE to keep track of the number of outstanding Memory transactions.

The MACE bus arbiter keeps track of outstanding transactions to assure that the maximum number of memory transactions that CRIME's FIFO can handle is not exceeded. Within MACE, each subsystem's FIFO controller will be responsible for limiting the number of read request packets issued so that its return FIFO won't overflow.

DMA read transactions will be acknowledged after the Read Data Response packet has returned and loaded into the appropriate subsystem FIFO. This will cause the MACE bus arbiter to decrement the outstanding transaction counter, thus allowing another transaction to be sent to CRIME.

MACE expects CRIME to assert its Memory Write Acknowledge (MWA) signal for one clock period once a memory write transacion has completed.

4.3.3.1 PIO Transactions

PIO packets from CRIME to MACE will be routed to each subsystem within MACE based on the decoding of the MSBs of the header's ADR field. Each subsystem will be capable of storing 2 PIO write packets (total of four 64-bit words), or one write and one read. The following is a list of possible outstanding PIO combinations:

a) 1 or 2 PIO writes
b) 1 PIO write followed by a PIO read
c) 1 PIO read only

CRIME will assure that no more than 2 outstanding PIO writes will be issued and no more than 1 outstanding read. If there is an outstand PIO read then no other PIO transaction (read or write) will be issued until the current PIO read has completed. This will be determined by CRIME identifying the Read Data Response packet returned by MACE. The TAG and ADR fields of the Read Block packet will be returned in the corresponding Read Data Response packet's header.

Each completed PIO write will be acknowledged by PWA (PIO Write Acknowledge signal from MACE to CRIME) being asserted for 1 clock period of the 66Mhz system clock. MACE will NOT guarantee that 2

outstanding PIO writes to seperate subsystems will complete in the order issued. However 2 outstanding PIO transactions to the same subsystem will be completed in the order issued.

4.3.3.2 PIO Write Acknowledge

Each subsystem will supply a PIO write acknowledge signal to the CMI. Once a PIO write has completed, the subsystem will assert its PIO write acknowledge signal for one period of the system clock. The individual subsystems will be responsible for synchronizing their write acknowledge signal with the system clock. A simple state machine within the CMI will monitor the write acknowledge signals and assert the PWA signal to CRIME for one system clock period for each write acknowledge signal that is asserted. If two simultaneous write acknowledges from different subsystems occure, PWA will be asserted for two consecutive system clocks.

4.3.4 Interrupt Packet Flow Control

The CMI will assure that only one outstanding interrupt packet will be sent to CRIME at any one time. Interrupt packets will be treated as out of band messages, in that they bypass pending memory writes in the CRIME asic. This means that system software provide the synchronization to ensure that pending memory writes are flushed if needed to signal the completion of a DMA transaction. This can usually be done by doing a single PIO read.

4.3.5 Tag Code

Refer to the *MACE I/O ASIC Specification [11]* See Chapter 1 for individual Tag Codes used by the MACE ASIC.

4.3.6 Transaction Types

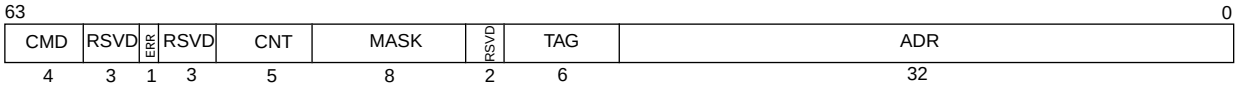
The following transaction types are supported:

Table 4-2 Supported Transaction Types

Type	CMD bits	# of Dwords	Description
NULL	0000	1	Null Transaction
INT	0001	1	Interrupt Update
WB	0010	2-33	Write Block
WSB	0011	2	Write Sub-Block
RB	0100	1	Read Block Request
RSB	0101	1	Read Sub-Block Request
RD	0110	2-33	Read Data Response

Null Packet

The Null (NULL) packet is one doubleword transfer of the following format:



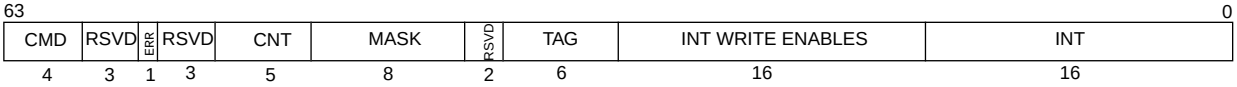
The fields are:

Bits	Width	Name	Description
63..60	4	CMD	Null command code (0000)
59..57	3	RSVD	Reserved
56	1	ERR	(Not Applicable) Error
55..53	3	RSVD	Reserved
52..48	5	CNT	(Not Applicable) Doubleword Count
47..40	8	MASK	(Not Applicable) Byte Mask
39..38	2	RSVD	Reserved
37..32	6	SRC	(Not Applicable) Tag
31..0	32	ADR	(Not Applicable) Byte Address

NOTE: 1. All bits of fields listed as *not applicable* or *reserved* must be set to zeroes. Therefore, all 64 bits must be set to zero (deasserted).

Interrupt Packet

The Interrupt (INT) packet is a doubleword transfer of the following format:



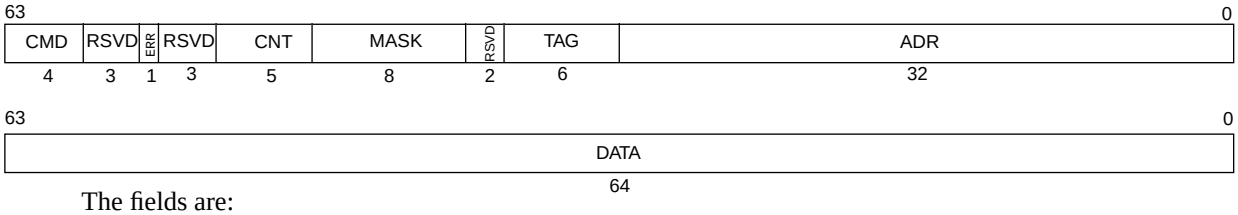
The fields are:

Bits	Width	Name	Description
63..60	4	CMD	Interrupt command code (0001)
59..57	3	RSVD	Reserved
56	1	ERR	(Not Applicable) Error
55..53	3	RSVD	Reserved
52..48	5	CNT	(Not Applicable) Doubleword Count
47..40	8	MASK	(Not Applicable) Byte Mask
39..38	2	RSVD	Reserved
37..32	6	SRC	Tag
31..16	16	INT MASK	Interrupt Bit Write Enables
15..0	16	INT	Interrupt Bits

NOTE: 1. All bits of fields listed as *not applicable* or *reserved* must be set to zeroes.

Write Block Packet

The Write Block (WB) packet is a doubleword header followed by 1 to 32 doubleword data transfers of the following format:



The fields are:

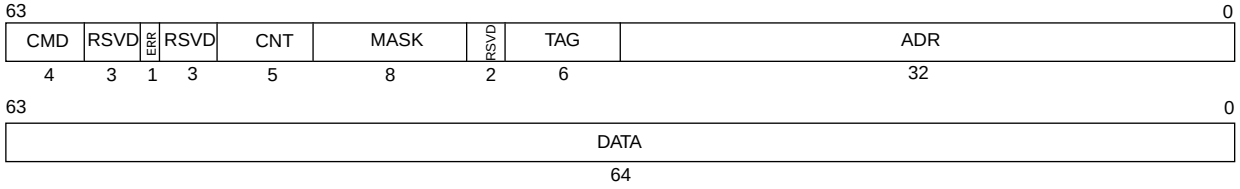
Bits	Width	Name	Description
63..60	4	CMD	Write Block command code (0010)
59..57	3	RSVD	Reserved
56	1	ERR	(Not Applicable) Error
55..53	3	RSVD	Reserved
52..48	5	CNT	Doubleword Count (00000 to 11111)
47..40	8	MASK	(Not Applicable) Byte Mask
39..38	2	RSVD	Reserved
37..32	6	SRC	Tag
31..0	32	ADR	Byte Address

NOTE: 1. All bits of fields listed as *not applicable* or *reserved* must be set to zeroes.

2. CNT field specifies the number *minus one* of data doublewords contained in the packet. (e.g. If CNT equals zero, then there is one data doubleword that follows the header.)

Write Sub-Block Packet

The Write Sub-Block (WSB) packet is a doubleword header followed by one doubleword data transfer of the following format:



The fields are:

Bits	Width	Name	Description
63..60	4	CMD	Write Sub-Block command code (0011)
59..57	3	RSVD	Reserved
56	1	ERR	(Not Applicable) Error
55..53	3	RSVD	Reserved
52..48	5	CNT	Doubleword Count (00000)
47..40	8	MASK	Byte Mask
39..38	2	RSVD	Reserved
37..32	6	SRC	Tag
31..0	32	ADR	Byte Address

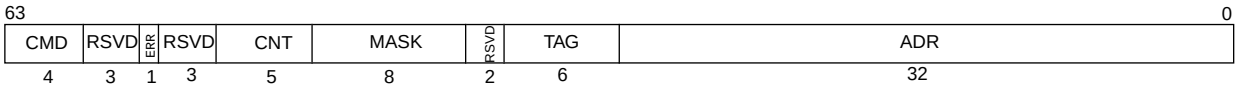
NOTE: 1. All bits of fields listed as *not applicable* or *reserved* must be set to zeroes.

2. CNT field specifies the number ***minus one*** of data doublewords contained in the packet. Only ***one*** data doubleword is allowed in a WSB packet, so the CNT field must be set to zero.

3. The MASK field specifies which bytes to be masked in the data doubleword.
   e.g. To mask byte #7 (big endian ordering - bits 7 down to 0) then the least significant bit of the MASK field should be asserted (bit #40 of the header).

Read Block Request Packet

The Read Block Request (RB) packet is a doubleword transfer of the following format:



The fields are:

Bits	Width	Name	Description
63..60	4	CMD	Read Block Request command code (0100)
59..57	3	RSVD	Reserved
56	1	ERR	(Not Applicable) Error
55..53	3	RSVD	Reserved
52..48	5	CNT	Doubleword Count (00000 to 11111)
47..40	8	MASK	(Not Applicable) Byte Mask
39..38	2	RSVD	Reserved
37..32	6	SRC	Tag
31..0	32	ADR	Byte Address

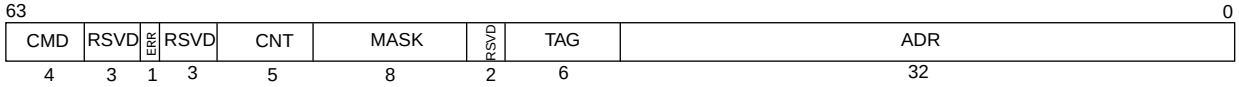
NOTE: 1. All bits of fields listed as *not applicable* or *reserved* must be set to zeroes.

2. CNT field specifies the number *minus one* of data doublewords requested.
(e.g. If CNT equals zero, then only one data doubleword is being requested.)

The RB packet is routed to the destination and at some later time a read data response packet (RD) will be sent back to the source that issued the RB packet.

Read Sub-Block Request Packet

The Read Sub-Block Request (RSB) packet is a doubleword transfer of the following format:



The fields are:

Bits	Width	Name	Description
63..60	4	CMD	Read Block Request command code (0101)
59..57	3	RSVD	Reserved
56	1	ERR	(Not Applicable) Error
55..53	3	RSVD	Reserved
52..48	5	CNT	Doubleword Count (00000)
47..40	8	MASK	Byte Mask
39..38	2	RSVD	Reserved
37..32	6	SRC	Tag
31..0	32	ADR	Byte Address

NOTE: 1. All bits of fields listed as *not applicable* or *reserved* must be set to zeroes.

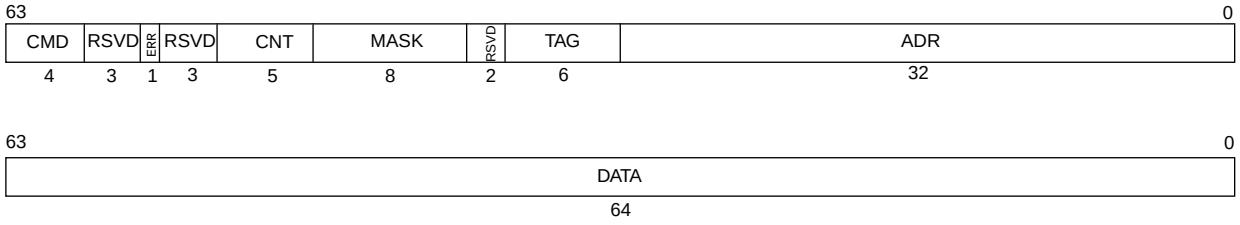
2. CNT field specifies the number ***minus one*** of data doublewords contained in the packet. Only ***one*** data doubleword is allowed to be requested in a RSB packet, so the CNT field must be set to zero.

3. The MASK field specifies which bytes to be masked.
   e.g. To mask byte #7 (big endian ordering - bits 7 down to 0) then the least significant bit of the MASK field should be asserted (bit #40 of the header).

The RSB packet is routed to the destination and at some later time a read data response packet (RD) will be sent back to the source that issued the RSB packet.

Read Data Response Packet

The Read Data Response packet is a doubleword header followed by 1 to 32 doubleword data transfers of the following format:



The fields are:

Bits	Width	Name	Description
63..60	4	CMD	Read Data Response command code (0110)
59..57	3	RSVD	Reserved
56	1	ERR	Error
55..53	3	RSVD	Reserved
52..48	5	CNT	Doubleword Count (00000 to 11111)
47..40	8	MASK	(Not Applicable) Byte Mask
39..38	2	RSVD	Reserved
37..32	6	SRC	Tag
31..0	32	ADR	Byte Address

NOTE: 1. All bits of fields listed as *not applicable* or *reserved* must be set to zeroes.

2. CNT field specifies the number *minus one* of data doublewords contained in the packet. (e.g. If CNT equals zero, then there is one data doubleword that follows the header.)

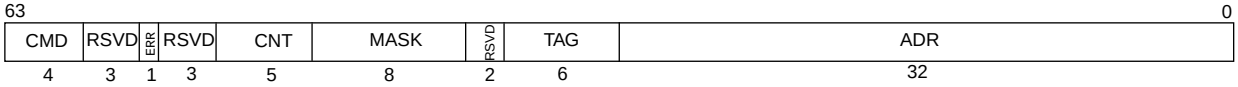
The RD packet is routed back to the source that issued the corresponding RB or RSB packet. The ERR field indicates the status of the read data as follows:

ERR	Encoding	Description
OK	0	Data valid
INVLD	1	Read request invalid

The INVLD status may occur because of a reference to an invalid address.

Read Block Request Packet

The Read Block Request (RB) packet is a doubleword transfer of the following format:



The fields are:

Bits	Width	Name	Description
63..60	4	CMD	Read Block Request command code (0100)
59..57	3	RSVD	Reserved
56	1	ERR	(Not Applicable) Error
55..53	3	RSVD	Reserved
52..48	5	CNT	Doubleword Count (00000 to 11111)
47..40	8	MASK	(Not Applicable) Byte Mask
39..38	2	RSVD	Reserved
37..32	6	SRC	Tag
31..0	32	ADR	Byte Address

NOTE: 1. All bits of fields listed as *not applicable* or *reserved* must be set to zeroes.

2. CNT field specifies the number *minus one* of data doublewords requested. (e.g. If CNT equals zero, then only one data doubleword is being requested.)

The RB packet is routed to the destination and at some later time a read data response packet (RD) will be sent back to the source that issued the RB packet. The source's identity is contained in the TAG field.

4.4 CRIME GBE Interconnect Data Flow

The CRIME GBE Interconnect is similar to the CRIME MACE Interconnet. Major differences are:

- Interconnect is 64 bits for CRIME/GBE vs 32 bits for CRIME/MACE
- CRIME/GBE connection does not provide pins for ACK flow control.

4.4.1 CRIME/GBE Use

The CRIME GBE Interconnect is used for writes and reads.

For Reads (GBE DMA engines read System Memory) GBE fetches descriptors for DMA transfers and pixels for display and to control display modes.

For Writes, GBE writes data into System Memory for Screen Capture.

4.4.2 CRIME/GBE Pins

A signal description for the CRIME GBE Interconnect is listed in the Table below.

Table 4-3 CGI Signal Definitions

Signal Name	Width	Direction	Description
GBE_DATA[31:0]	64	Bidirectional	Data bus
GBEE_TOKEN_IN	1	Input	Command for CRIME. When asserted GBE waits for GBE_TOKEN_OUT to de-assert. This constitutes bus ownership by GBE
GBE_TOKEN_OUT	1	Output	CRIME has command for GBE. De-asserted, GBE can drive bus.
	66		

4.5 CRIME interface

The CRIME to GBE interface is a point to point, burst oriented protocol with a peak bandwidth of 1 GB/s. The data path between chips consists of a 64 bit data bus toggling at 133 MHz. The interface is centrally clocked using a 66 MHz clock, with PLL cells in each chip to set the pin to core flip-flop delay to 0.0 ns. Care must be taken in the board level clock routing to ensure 0.0 ns clock skew at the pins of GBE and CRIME. The timing budget assumes minimum 1.0 ns and maximum 5.5 ns clock to out, plus chip and board level clock skew. Input path timing requires 0.0 ns hold time. The data bus input and output flip flops are clocked at 133 MHz.

The CRIME interface is based on the simple notion of a point to point link with one sender and one receiver. A full handshake is used to provide

an orderly transition when swapping receiver and sender. There are no explicit flow control signals between the chips. All flow control is performed implicitly through the data lines. The 64 data lines are clocked at 133 MHz and are internally demultiplexed to a 128 bit wide bus running at 66 MHz, synchronous to ref67. Therefore, the interface is logically 128 bits, and the smallest individual data transfer is 128 bits. Data transfers consist of a 128 bit header, optionally followed by some number of 128 bit data cycles. The header is formatted as follows.

cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

4.5.1 CRIME as sender commands

NOP					
0000	-	-	-	-	-
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

When either CRIME or GBE is the sender, it must drive NOP commands by default. CRIME is the sender on system reset, and must drive NOP commands during and after the reset period. This is to ensure that when GBE comes out of reset and begins receiving, it will be sampling known data.

PIO WRITE					
0001	-	-	-	**DATA**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

CRIME can write to GBE using the PIO write command. All writes are 32 data bits, and addresses are assumed to be 24 bits, and word-aligned (addr[1:0}= “00”). Address bits 31:24 are ignored. There is no flow control for PIO writes, so GBE must be able to accept writes at the full rate of 66 MHz. The only exception is color map writes, which go through a 64 deep fifo and must be throttled by software. The 24 bit address should be based at the start of the GBE address space.

PIO READ REQUEST					
0010	-	-	-	-	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

PIO reads from GBE are executed as a split transaction. To initiate the read, CRIME sends a PIO_READ_REQ command, which includes a 24 bit address field. GBE will respond with a PIO_READ_DATA command. There can only be one outstanding PIO read request. All reads are 32 bits, and the 24 bit read address is 32-bit aligned and based at the start of the GBE address space.

DMA READ DATA					
0011	**COUNT**	-	-	-	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

CRIME uses the DMA_READ_DATA command to send DMA data to GBE. The header contains a 5 bit COUNT field, which specifies the number of 256 bit data words which immediately follow the header. For example, if COUNT= “00002”, then the 128 bit header will be followed by 4 cycles of 128 bit data. DMA transfers are always aligned on 256 bit boundaries, and are always multiples of 256 bits. The address is returned with the data for debugging purposes; CRIME always returns DMA data in the same order that it was requested by GBE.

DMA WRITE DONE					
0100	-	-	-	-	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

CRIME sends a DMA_WR_DONE command to GBE to indicate that a DMA write has been flushed to main memory. CRIME has one DMA write buffer, and can therefore only handle one outstanding DMA write. GBE must wait for a DMA_WR_DONE message from CRIME before issuing a new DMA write.

4.5.2 GBE as sender commands

PIO READ DATA					
0101	**-**	-	-	**DATA**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

GBE sends a PIO_READ_DATA command to complete the PIO read transaction. GBE always returns 32 bits of data in the data field. The read address is returned for debugging purposes.

DMA READ REQUEST					
0110	**COUNT**	-	-	**-**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

GBE sends a DMA_READ_REQ command to request a block of main memory. DMA reads are executed as split transactions, and CRIME contains a memory transaction queue which can hold up to 16 requests. As a practical matter, it is essential for GBE to keep the request queue non-empty, so that the memory controller inside CRIME will operate at full bandwidth. The ADDR field is 256 bit aligned, so bits [4:0] must be “00000”. The COUNT field specifies the number of 256 bit words to transfer. CRIME guarantees that it will return DMA_READ_DATA blocks in the same order that they were received.

DMA WRITE DATA					
0111	**COUNT**	-	-	**-**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

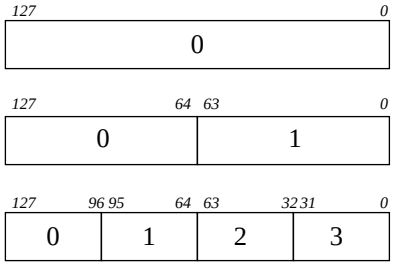
GBE uses the DMA_WRITE_DATA header, followed by COUNT*2 data cycles, to transfer DMA write data from GBE to one of the two DMA write buffers inside CRIME. The DMA will also be queued in the CRIME memory transaction queue, to be executed in turn. The COUNT refers to 256 bit words, and the address is 256 bit aligned. GBE must ensure that no more than two DMA writes are pending at any time.

INTERRUPT					
1000	**-**	-	-	**-**	**MASK**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

GBE sends an interrupt command to indicate the vertical retrace interrupt. Note that this is a one-shot event, so the interrupt must be latched inside CRIME. The lower 4 bits of the address field (MASK) identify the source of the interrupt within GBE. If a MASK bit is ‘1’, then the corresponding interrupt is set inside CRIME. If the MASK bit is ‘0’, there is no change to the interrupt register in CRIME. CRIME is only capable of clearing the interrupt.

To summarize the important assumptions:

- only 1 outstanding PIO read allowed
- only 1 outstanding DMA write allowed
- only 8 outstanding DMA reads/writes allowed
- no limits on PIO writes
- all PIO is 32 bits data, 24 bits address, 32 bit aligned, GBE start address => 0x000000
- all DMA is 512 byte aligned, multiples of 256 bits
- sender must drive NOP commands by default, also during reset
- data is always sent in big endian order



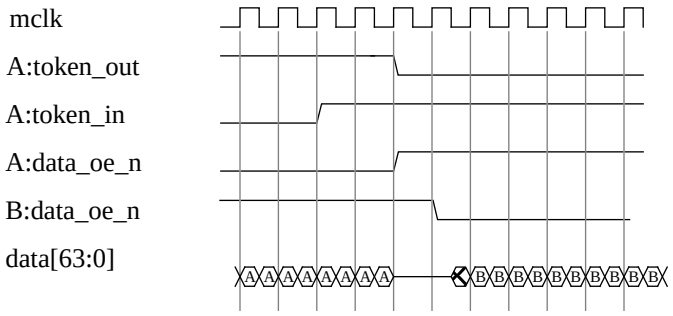
- no 8K DRAM page crossings

4.5.3 Transfer of mastership

The command/data structure describes the mechanism for transferring data. To complete the protocol we need to know how the two devices swap roles as sender and receiver. This is accomplished using the token_in and token_out signals. On the board, CRIME’s token_out should be wired to GBE’s token in, and GBE’s token_out should be wired to CRIME’s token_in. On system reset, CRIME is always the sender, which it indicates by driving token_out high and driving NOP commands. On reset, GBE is idle and will drive its token_out low. This state

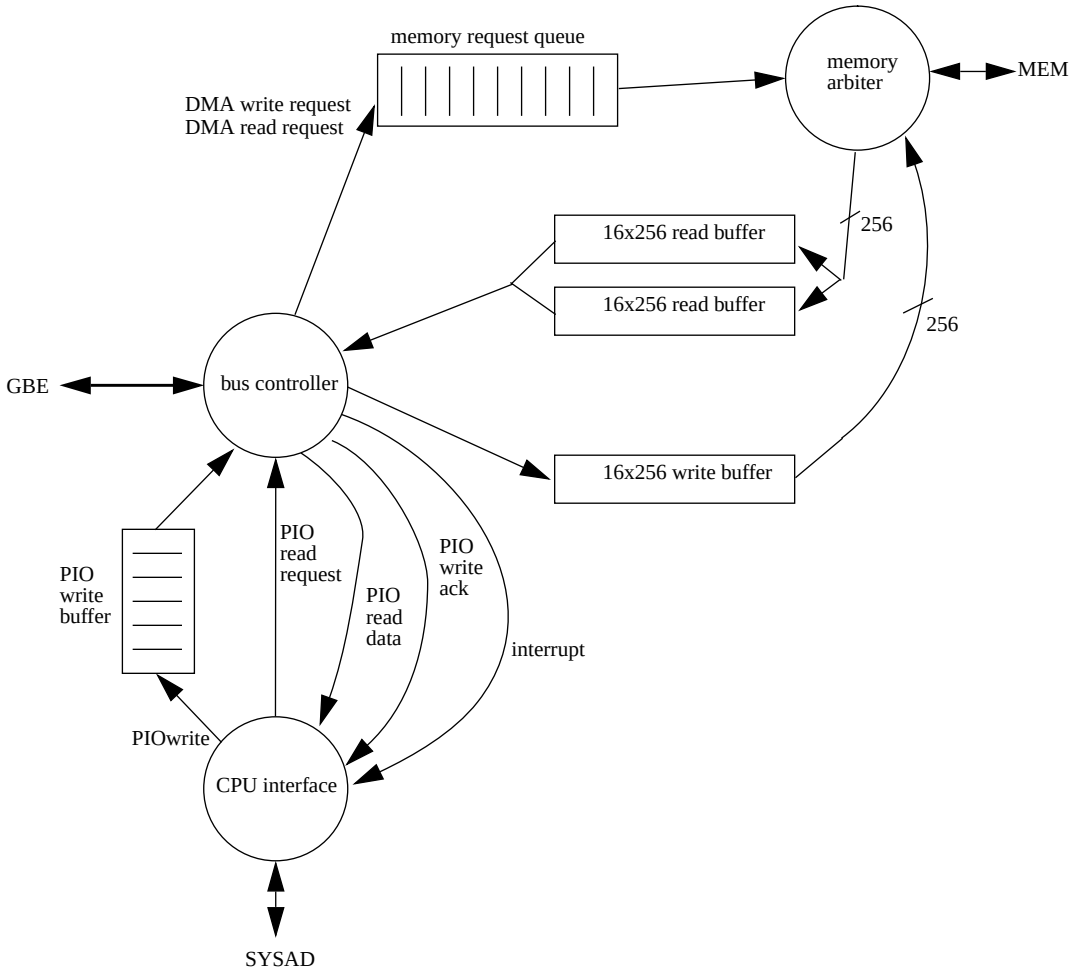
will continue until, through PIO programming, GBE needs to become the sender. GBE will then drive its token_out high, and wait for its token_in to go low. As soon as GBE detects that token_in is low, it should begin driving the data bus. At this point, GBE is the sender and CRIME is the receiver. The sequence is identical to switch mastership back to CRIME.

The following timing diagram shows the transition from Device A sending to Device B sending.



Note that the current sender should not release the bus until it has completed sending all of its pending commands. This will help to minimize the total number of bus turn arounds.

4.5.4 CRIME side of GBE interface, block diagram



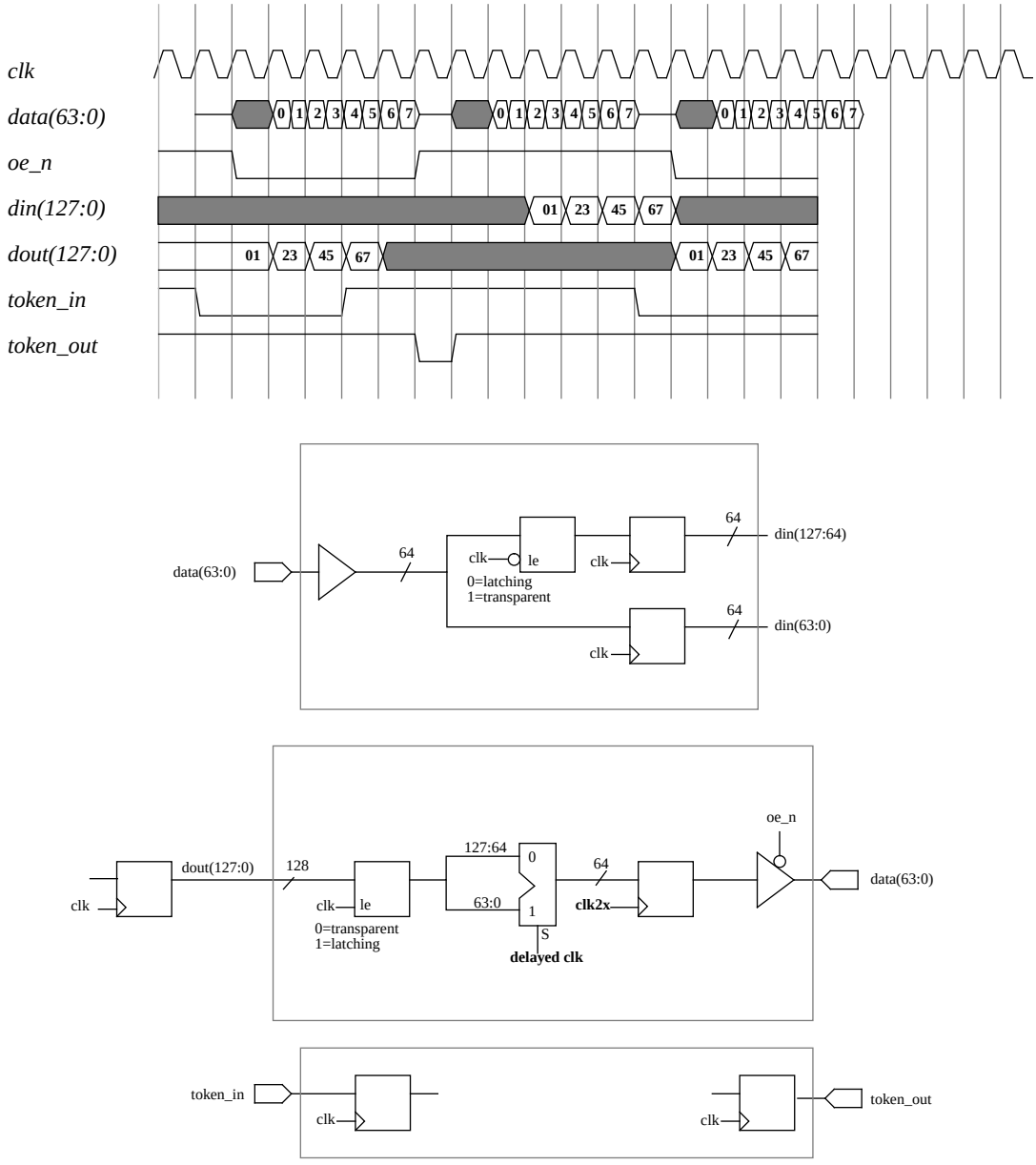
Note that for DMA writes, the memory request is issued after the last data has been written into the 16x256 write buffer. On DMA reads, the data may be transferred as soon as it arrives from the memory arbiter, since the memory data rate is twice the GBE bus interface data rate.

4.5.5 Clocking scheme

The following logical circuits are used to clock data in and out of GBE.

4.5.6 DAC / flat panel interface

GBE contains a programmable dot clock PLL, which generates a pixel clock up to 140 MHz. This clock drives the pixel pipeline and the video



timing controller. The pixel clock is output from GBE on the pclk_out pin. The interface to the DAC consists of pclk_out, red[7:0], grn[7:0], blu[7:0], blank_n and sync_n. In addition, hdrv and vdrv provide separate monitor sync signals. An input status bit is provided for the sense_n output of the DAC. This allows software to detect whether the monitor is plugged in. The pixel clock can be driven externally or from the internal pixel PLL

The flat panel display requires a horizontal and vertical sync signal which is active for the entire blanking period. In order to simultaneously drive the CRT and flat panel displays, GBE produces fp_hdrv and fp_vdrv specifically for the flat panel. In addition, an fp_de (flat panel display enable) signal is produced, which functions as a flat panel blanking signal but with slightly different timing than the CRT blank_n.

CHAPTER 5

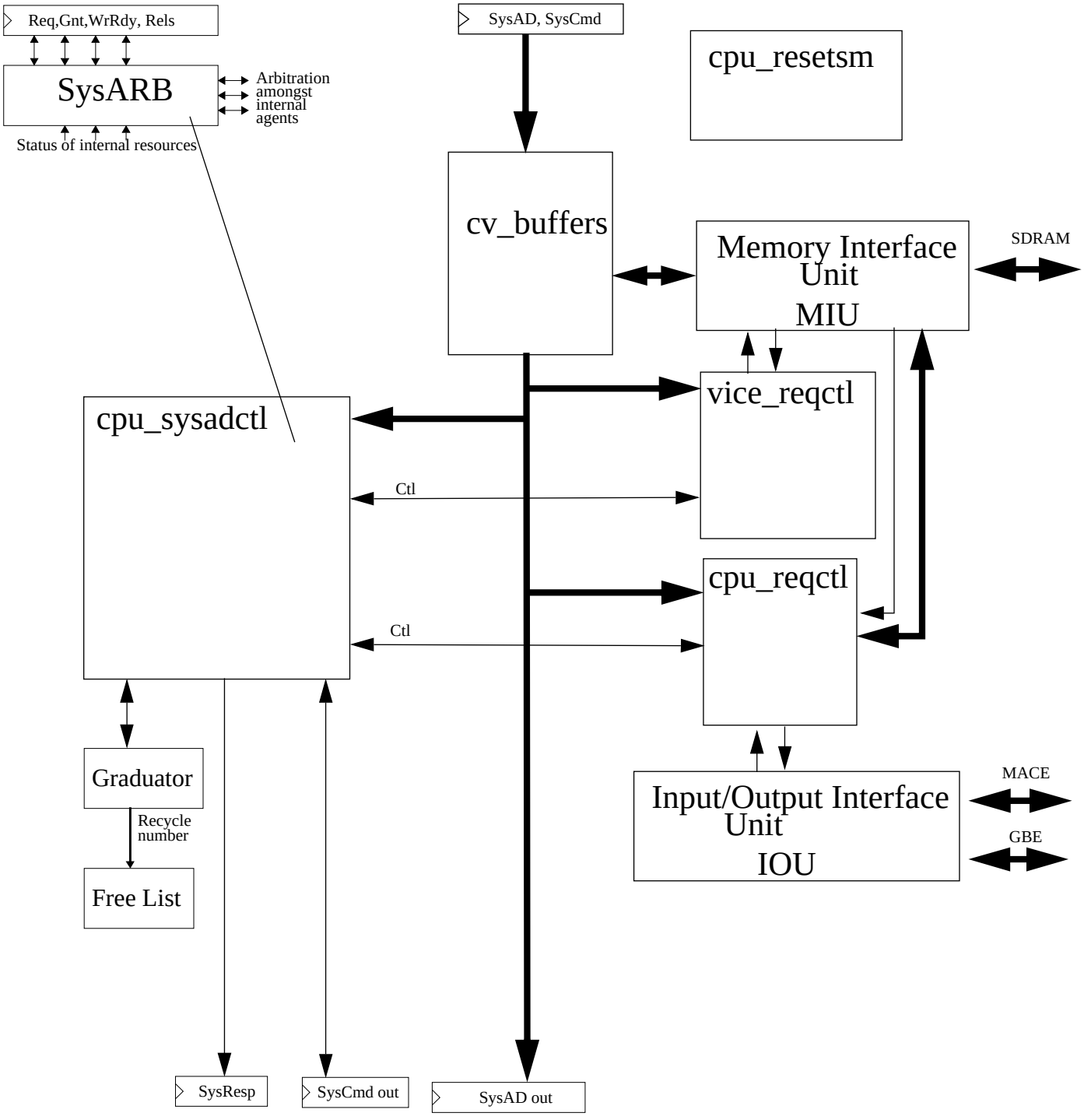
Processor Interface Unit and Its Operations

Note: Editorial comments like this are in normal size font but in bold face. They should eventually be eliminated in later drafts of this spec.

5.1 Introduction and Block Diagram

The Processor Interface Unit block diagram is shown in the Figure on the next page.

Figure 5-1 Processor Interface Unit Block Diagram



The main functions of this unit are:

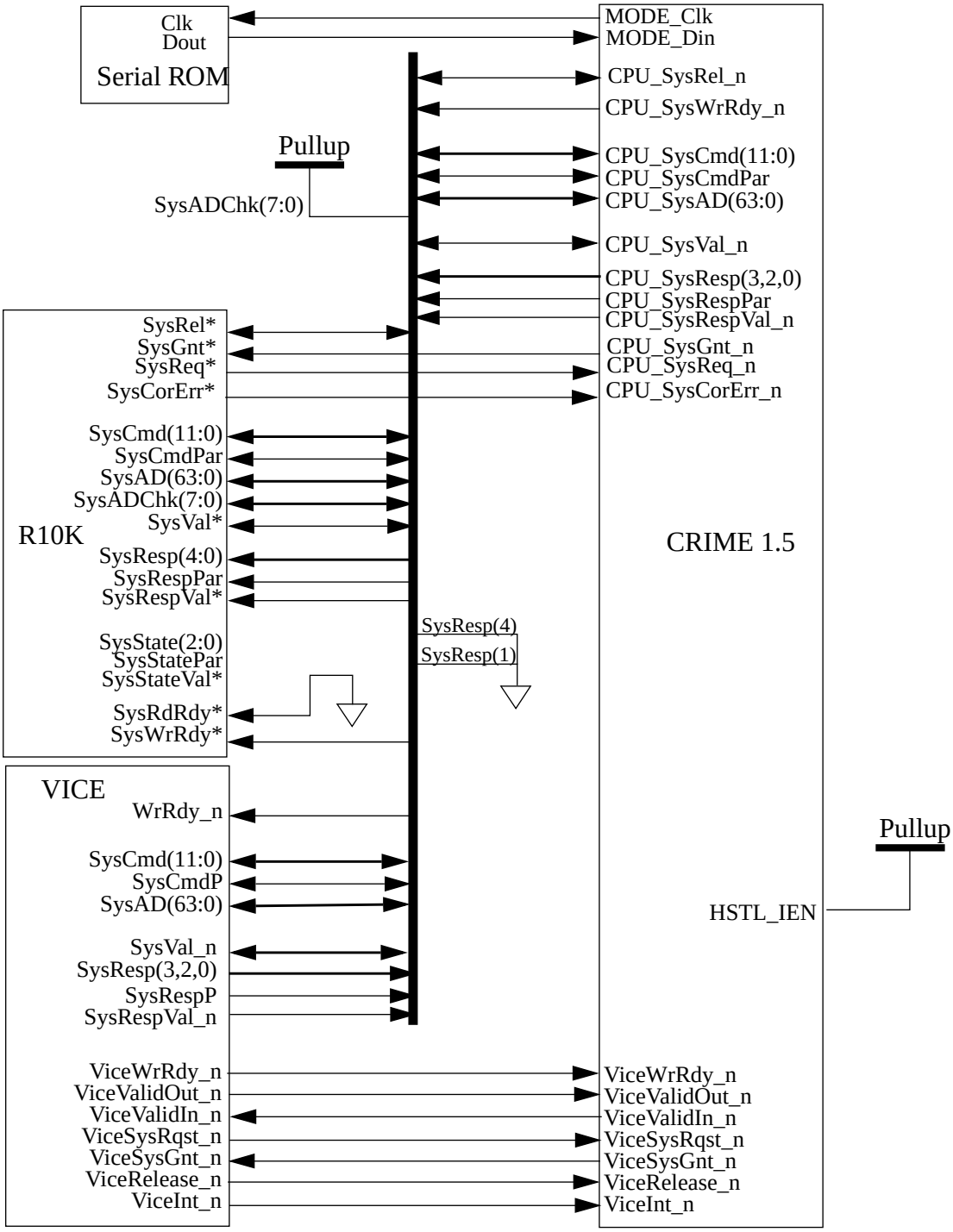
1. SysAD bus arbitration
2. Monitor, buffer and respond to processor requests
3. Monitor, buffer and respond to VICE requests

PIU will be designed to support 1 R10K/R12K processor and the VICE processor on the SysAD. Coherent IO will not be supported in CRIME 1.5.

5.2 R10K/R12K connection to CRIME

CRIME supports one R10K or one R12K processor. The VICE chip is supported with either of these processors.

Figure 5-2 R10K/R12K CRIME - VICE Interface



5.2.1 Supported R10K/R12K Bus Requests

Processor eliminate requests are not supported in CRIME and should be disabled in the R10K/R12K processor by de-asserting the PrcElmReq mode bit at initialization time.

CRIME supports Processor Requests per Table 5-1.

Table 5-1 Processor Requests support by CRIME 1.5

Processor Request	CRIME Response
coherent block read shared	"Juice Fix" ERR response if > address range programmed by CPU Interface Control register. ACK with data if < address range.
coherent block read exclusive	"Juice Fix" ERR response if > address or range programmed by CPU Interface Control register. ACK with data if < address range.
noncoherent block read	ACK with data.
double/single/partial-word read	ACK with data.
block write	CRIME Accepts data.
double/single/partial-word write	CRIME Accepts data
upgrade	ACK with data - Treated as normal read.
eliminate	Not supported. No response. Not placed in write buffer queue or popped from queue with no adverse affect. Left to implementation choice.

5.2.2 External Requests

The only External Request that CRIME will initiate is an interrupt request.

5.2.3 Speculative Load Workaround for CRIME 1.5

There is a scheme to use the coherent SysCMD identifier to identify the speculative loads that occur because of the R10K micro-architecture that was designed for the current O2-R10K product. This scheme is more fully explained in [13] *Moosehead R10000 Kernel Software* but some key hardware issues have been extracted here for convenience.

An additional resource for this information comes from the RTL of Juice 2 vhdl from

"/hosts/wain.esd/vault1/d1/moosehead/subsystem/juice2/vhdl"

From [13] *Moosehead R10000 Kernel Software*

Hardware impact summarized as:

1. Software configures cacheable TLB entries to use "noncoherent" access mode

2. K0SEG to use "coherent" access mode

3. Hardware "rejects" K0SEG (coherent) references from 8Meg-512Meg

4. DMA buffers NOT allocated in the first 8 Meg of memory.

What Juice Does

5. For coherent block reads from 8Meg to 512Meg Juice sets a flag called i_cmd_err. This produces a SysResp of ERR when Juice performs the external completion response.

Details: (name in parenthesis is the vhdl file for reference)

(t5_dec.vhd)

Juice sees the SysCMD(7:6) = 00 indicating coherent block read during a processor address cycle specified by SysCMD(11) = 0 and SysVal* = 0. AND Juice sees HIAD = '1' which means that the address has at least one bit set in bits 28 downto 23 of the address. (range from 8MEG to 512MEG to match spec above)

Details of address decode

(had_chk.vhd)

HIAD <= AD(5) or AD(4) or AD(3) or AD(2) or AD(1) or AD(0);

(../ecs/juice_ctl.vhd)

Port Map (AD(5 downto 0)=>T5_AD_I(28 downto 23), HIAD=>HIAD);

When Juice sees this condition, it sets a signal called “i_cmd_err” <= ‘1’which in turn drives a signal called CMD_ERR which is fed to t5_rsp.vhdand causes Juice to perform an “external completion response” of ERR. This means it sets SysResp(1:0) to “01” and places the request number in SysResp(4:2). Juice does NOT perform the read, nor does it return any data to the R10000 with SysVal.

A coherent block read from < 8 Meg will be performed by Juice and an external response of ACK will be supplied along with the data. A non-coherent block read from ANY range of memory will be performed by Juice and an external response of ACK will be supplied along with the data.

What does this mean for Crime 1.5?:
==========

Per the request of the DSD software team, Crime 1.5 will be made programmable to allow the range of coherent K0SEG to be adjustable from 8Meg to 32Meg in 4Meg increments. This means that HIAD needs to be smarter and must compare the address bits from 28 down to 23 and also look at a register that covers the choices of address range. In fact CRIME should decode the entire address range of bits 39:23 for this implementation.

Address (or any bit set in Address bits 39:26)

25 24 23 22 K0SEG access ignored for coherent read

0 0 1 0	> 8 Meg
0 0 1 1	> 12 Meg
0 1 0 0	> 16 Meg
0 1 0 1	> 20 Meg
0 1 1 0	> 24 Meg
0 1 1 1	> 28 Meg
1 0 0 0	> 32 Meg

Since Vice is non-cacheable and is outside of this address space, VICE does not have to worry about fielding speculative loads from it's address space.

Behavior of Crime 1.5 is similar to Juice in that it returns an ERR external completion response when it detects a K0SEG access above 8 Meg (or 12 Meg, or 16Meg per the choices above).

5.2.4 CRIME 1.5 Signal Pin Accounting Summary

Added Pins = 17

Removed Pins = 17

Pin Change 0

5.2.4.1 Pins added to CRIME 1.5 for R10K/R12K support

CPU_SysCmd(11:9)
CPU_SysVal_n
CPU_SysGnt_n
CPU_SysRel_n
CPU_SysReq_n
CPU_SysResp(3,2,0)
CPU_SysRespPar
CPU_SysRespVal_n
CPU_Clk_n,
CPU_SysCorErr_n
VRefSys
Mode_Clk
Mode_Din

New pins on CRIME 1.5 = 17

Note that some number of VcqSys pins will be used to support the HSTL logic of CRIME at a ratio of 1 VcqSys pin for each 6 outputs. This should be a total of about 120/6 = 20 pins. These may be able to be exchanged with other power pins presently used on the package.

5.2.4.2 Pins removed from CRIME 1.1 because no R5K/RM7K support

CPU_SysADC(7:0)
CPU_ValidIn_n
CPU_ValidOut_n

CPU_ExtRqst_n
CPU_ScWord(1:0),
CPU_ScDOE_n
CPU_ScMatch
CPU_ScTCE_n
CPU_ColdReset_n
Pins removed CRIME = 17

5.3 Processor Interface Assumptions

The R10K/R12K processor may have the mode bits set as follows.

a. SysClkDiv is set to 0 or 1, i.e., Divide by 2 or 3 respectively.
b. SCBlkSize is set to 1, i.e., 32-word secondary cache block size.
c. PrcReqMax is set to 3, for 4 outstanding request per processor.
d. CohPrcReqTar is set to 0, i.e., coherency disabled.
e. ReplUnowned is set to 0, i.e., disabled

See Chapter 8 for detailed mode bit settings.

The SysAD bus implementation in CRIME is not supporting the ECC protection bits for SysAD(63:0).

5.4 SysAD Arbitration

The SysAD arbitration protocol follows that described in the section System Interface Arbitration Rules in R10K/R12K Spec.

After initialization, CRIME asserts SysWrRdy_n, by driving it low. It also asserts the SysAD bus grant signal to the R10K/R12K processor.

During operation, SysWrRdy_n will be deasserted and reasserted as the relevant internal resources run out and get freed, and depending on the relative priority if there are other requests / responses vying for the SysAD (See the priority list below).

SysRdRdy_n does not need to be throttled based on the number of outstanding requests, since the latter will be automatically throttled by the

R10K/R12K processor as it observes that there are no more request numbers. CRIME does not implement SysRdRdy_n.

During normal operation, if it does not need to use the bus itself, and there are no bus requests from other processors, CRIME always parks the grant to the processor. However, when the bus is parked on the processor, a few cycles before any read response is ready in CRIME, CRIME will start arbitrating for the bus in order that the response can be returned with no arbitration delay. This reduces the read response latency for processor requests.

5.4.1 Suggested SysAD Arbitration Priority

A. intwrite - Crime writes interrupt register in R10K
B. crimersp2vice - Crime returns dma data to Vice
C. vicedma2crime - Vice granted bus to perform dma to crime
D. vicersp2cpu - Vice responds with read request data for the R10K
E. r10KSysRqst - Crime grants bus to R10K
F. crimersp2cpu - Crime responds with read request data for R10K

When there is no requestor for the SysAD bus, the R10K should be granted the bus to reduce latency for R10K load/store operations.

Note that the order of B and C above are not that important as the CRIME request queues cannot accept more requests from VICE until the last request has been processed or returned (DMA READ) to VICE.

Another performance issue is to set the read request queue so that if only 2 read requests are in the input queue, and no data is back yet from the memory controller (or PIO source), the bus can be given to the R10K if SysRqst_n is asserted, so that it can provide more requests while Crime is processing the requests outstanding. This keeps the queue from the R10K full to keep the round trip latency less visible (ie. don't let the pipeline drain).

5.4.2 SysAD Arbitration Justification

Interrupts should be low latency and high priority. Vice is important, since when it runs, it is usually a real time application and should be provided with low latency. Since Vice is sort of throttled to be ~150 MBytes/sec because of the memory arbiter in CRIME it can be high pri-

ority but will still leave lots of bandwidth for the CPU which will run at a lower priority.

5.5 Latency of Processor to Memory Block Read

5.5.1 Best Case Read

The following is a cycle by cycle (SysClk period) latency analysis of a best case processor to memory block read. Best case assumptions of circumstances and state are:

- No other reads ahead of this one in CRB.
- Read is non-coherent --- interventions to other processor take time
- MIU is not servicing a processor write or an IO request.
- CRIME acquires the SysAD before the first data reaches CRIME's SysAD output flip-flops.

The cycle count is as follows:

sck1 processor read request on sysad
sck2 request in sysad input flops
sck3 request in input command queue

mck1 sync
mck2 sync
mck3 command queue not empty
mck4 memory interface unit request
mck5 memory interface unit grant
mck6-12 Memory Interface Unit Page Miss Access
mck13 memory data returned to read resp. buffer
mck14 read resp buffer not empty

sck4 sync
sck5 sync
sck6 state machine initiates sysad response
sck7 flop data out onto sysad

The total latency is thus 7 sysad cycles at 10 ns each plus 14 memory cycles at 15 ns each, = 70 + 210= 280 ns. This makes a very pessimistic estimate that synchronizer delays are two cycles. On average, a synchronizer delay is 1.5 cycles when crossing clock domains of random relative phase.

Table 5-2 Crime 1.5 predicted latencies without GBE collision

Processor Speed	SysAD Clock	7 SysAD cycles	14 mclk cylces	1st Data Delivered	15 SysAD cycles rest of cache line	Cache Line Delivered
175 MHz	87.5 MHz	80 nsec	210 nsec	290 nsec	171 nsec	461 nsec
195 MHz	97.5 MHz	72 nsec	210 nsec	282 nsec	154 nsec	436 nsec
225MHz	90 MHz	78 nsec	210 nsec	288 nsec	166 nsec	454 nsec
250 MHz	100 MHz	70 nsec	210 nsec	280 nsec	150 nsec	430 nsec
300 MHz	100 MHz	70 nsec	210 nsec	280 nsec	150 nsec	430 nsec
400 MHz	100 MHz	70 nsec	210 nsec	280 nsec	150 nsec	430 nsec

5.5.2 Typical Read

For average performance calculations, add in a collision with GBE half the time, colliding at the mid-point of the GBE's 23 clock cycle. This adds .5 x .5 x 23 mclk = ~6 mclk of latency on average to the processor read.

Factoring that into the table above changes the column labeled14mck from 210 to 300 nsec which then adds 90 nsec to the Subtotal and Total column as per below:

Table 5-3 Crime 1.5 predicted latencies with GBE average collision

Processor Speed	SysAD Clock	7 SysAD cycles	14 mclk cylces	1st Data Delivered	15 SysAD cycles rest of cache line	Cache Line Delivered
175 MHz	87.5 MHz	80 nsec	300 nsec	380 nsec	171 nsec	551 nsec
195 MHz	97.5 MHz	72 nsec	300 nsec	372 nsec	154 nsec	526 nsec
225MHz	90 MHz	78 nsec	300 nsec	378 nsec	166 nsec	544 nsec
250 MHz	100 MHz	70 nsec	300 nsec	370 nsec	150 nsec	520 nsec
300 MHz	100 MHz	70 nsec	300 nsec	370 nsec	150 nsec	520 nsec
400 MHz	100 MHz	70 nsec	300 nsec	370 nsec	150 nsec	520 nsec

5.6 PIO Client Interface

Table 5-4 CPU to Client Signals

Signal	Direction	Description
xx_piord	from cpu to client	indicates that there is a pio read on the xx_pioaddr bus
xx_piowr	from cpu to client	indicates that there is a pio write on the xx_pioaddr and xx_piodataout bus
xx_piowrdy	from client to cpu	indicates that the client can accept another write
xx_piomask	from cpu to client	8 bit byte mask
xx_piodataout	from cpu to client	64 bit data bus
xx_pioaddr	from cpu to client	? bit address bus
xx_piodatain	from client to cpu	64 bit data bus
xx_piordrdy	from client to cpu	indicates that there is a read response on io_piodatain

Figure 5-3 64 bit PIO Write to I/O

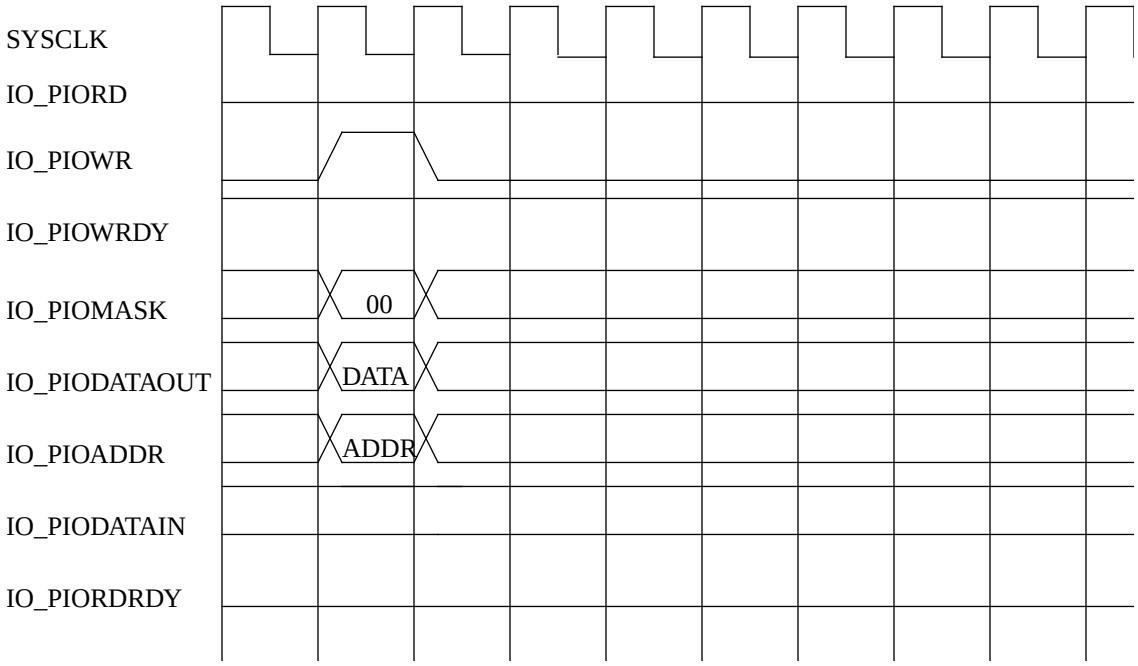


Figure 5-4 Back to Back 64 bit PIO Writes to I/O with Flow Control

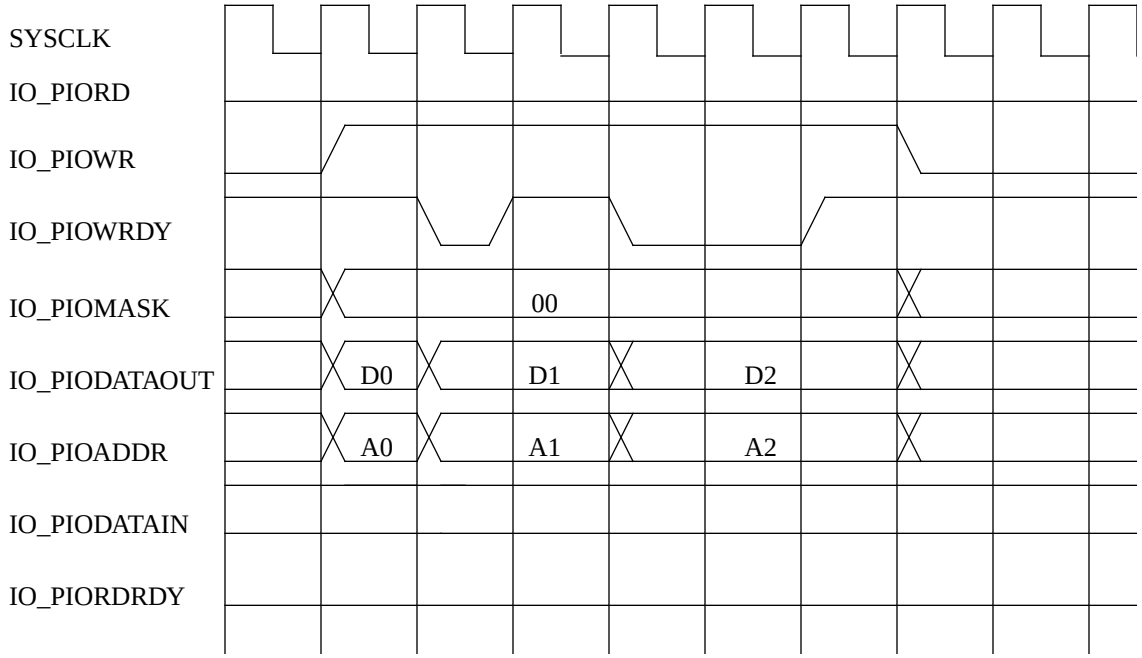
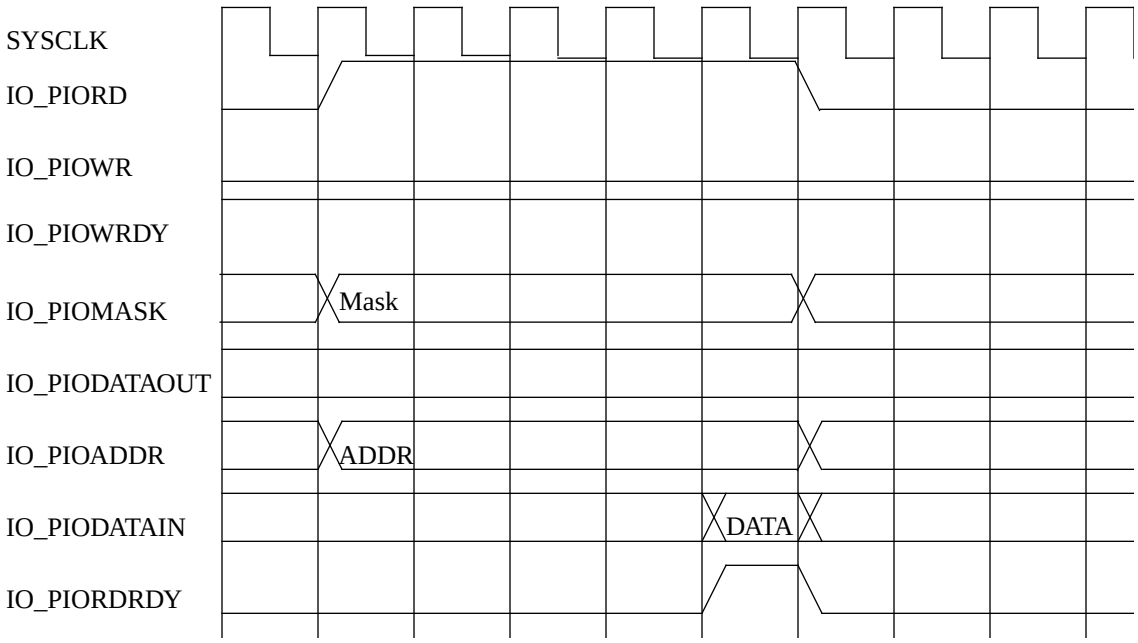


Figure 5-5 PIO Read to I/O



NOTE: Each PIO client must use XX_PIORD as an active high signal that enables XX_PIODATAOUT. If XX_PIORD is ‘0’, XX_PIODATAOUT is tristated. If XX_PIORD is ‘1’, XX_PIODATAOUT is connected. This is necessary because all PIO clients share the same data output bus, and only one of the PIO clients can be bus master at any given time.

The format of the byte mask is:

Bit 0

- 1 - Bits 7 downto 0 of xx_piodataout are not valid
- 0 - Bits 7 downto 0 of xx_piodataout are valid

...

Bit 7

- 1 - Bits 63 downto 56 of xx_piodataout are not valid
- 0 - Bits 63 downto 56 of xx_piodataout are valid

5.7 Interrupts

Interrupts are indicated through the *Interrupt Status* register. This register is read-only and is the logical OR of the *Hardware Interrupt* register and the *Software Interrupt* register. The *Hardware Interrupt* register collects interrupts from the Crime subsystems (CPU interface, memory controller, rendering engine), GBE, Mace, and Vice. The *Software Interrupt* register is a read/write register which the host can use for software generated interrupts.

The *Interrupt Enable* interrupt is a read/write register for enabling and disabling interrupts to the host. The *Interrupt Status* register is logically AND-ed with the *Interrupt Enable* register to produce an interrupt for each of the 32 bits in the register.

These individual interrupts are grouped together (OR-ed) and assigned to the processor levels as shown in the table below. The CRIME ASIC performs an interrupt access to the R10K/R12K when it senses a change (assertion OR de-assertion) on one of these interrupt bits.

Note that the CPU_SysCorErr_n has been added to level 6 with other interface errors.

Table 5-5 CRIME Interrupt Assignments

Bit	Description	Map to R10K IP#
31	VICE interrupt	2
30	CPU_SysCorErr interrupt Note: Formerly Software interrupt 2 which was not used in Crime 1.1 (we think!)	6
29	Software interrupt 1 CRM_INT_SOFT1 from crime.h	5
28	Software interrupt 0 CRM_INT_SOFT0 from crime.h	4
27	RE idle interrupt - level sensitive CRM_INT_RE5 from crime.h	2

Table 5-5 CRIME Interrupt Assignments

Bit	Description	Map to R10K IP#
26	RE FIFO full interrupt - level sensitive CRM_INT_RE4 from crime.h	2
25	RE FIFO empty interrupt - level sensitive CRM_INT_RE3 from crime.h	2
24	RE idle interrupt - edge sensitive CRM_INT_RE2 from crime.h	2
23	RE FIFO full interrupt - edge sensitive CRM_INT_RE1 from crime.h	2
22	RE FIFO empty interrupt - edge sensitive CRM_INT_RE0 from crime.h	2
21	Memory error interrupt CRM_INT_MEMERR from crime.h	6
20	CPU interface error interrupt CRM_INT_CRMERR from crime.h	6
19	GBE Interrupt #3 GBE_INT_GBE3 from crime.h	3
18	GBE Interrupt #2 GBE_INT_GBE2 from crime.h	3
17	GBE Interrupt #1 GBE_INT_GBE1 from crime.h	3
16	GBE Interrupt #0 GBE_INT_GBE3 from crime.h	3
15	MACE PCI Interrupt Input #7 MACE_PCI_SHARED2 from crime.h	3
14	MACE PCI Interrupt Input #6 MACE_PCI_SHARED1 from crime.h	3
13	MACE PCI Interrupt Input #5 MACE_PCI_SHARED0 from crime.h	3

Table 5-5 CRIME Interrupt Assignments

Bit	Description	Map to R10K IP#
12	MACE PCI Interrupt Input #4 MACE_PCI_SLOT2 from crime.h	3
11	MACE PCI Interrupt Input #3 MACE_PCI_SLOT1 from crime.h	3
10	MACE PCI Interrupt Input #2 MACE_PCI_SLOT0 from crime.h	3
9	MACE PCI Interrupt Input #1 (SCSI controller 1) MACE_PCI_SCSI1 from crime.h	3
8	MACE PCI Interrupt Input #0 (SCSI controller 0) MACE_PCI_SCSI0 from crime.h	3
7	MACE PCI Error Conditions MACE_PCI_BRIDGE from crime.h	6
6	MACE Peripheral Controller, Audio Interrupts MACE_PERIPH_AUD from crime.h	5
5	MACE Peripheral Controller, Misc. keyboard/mouse/timer MACE_PERIPH_MISC from crime.h	2
4	MACE Peripheral Controller, Serial/Parallel Interrupts MACE_PERIPH_SERIAL from crime.h	2
3	MACE Fast Ethernet Interface MACE_ETHERNET from crime.h	2
2	MACE Video Output Channel MACE_VID_OUT from crime.h	3
1	MACE Video Input Channel #2 MACE_VID_IN2 from crime.h	3
0	MACE Video Input Channel #1 MACE_VID_IN1 from crime.h	3

5.7.1 R10K Interrupt Map

For convenience, the mapping of the R10K Interrupt Cause Register, with the IP level and the SysAD bit fields that CRIME must write to effect a given interrupt are included in the Table below.

Table 5-6 R10K Interrupt Relationship

IP Level	Interrupt Cause Register Bit	SysAD(4:0) Field to Interrupt	SysAD(20:16) Write Enable Mask
IP[2]	10	SysAD[0]	SysAD[16]
IP[3]	11	SysAD[1]	SysAD[17]
IP[4]	12	SysAD[2]	SysAD[18]
IP[5]	13	SysAD[3]	SysAD[19]
IP[6]	14	SysAD[4]	SysAD[20]

5.7.2 VICE Interrupts

Bit 31 of the *Interrupt Status* register is a registered version of the vice_int_n pin which is a 100 Mhz signal. Please refer to the VICE specification for the procedure to clear an interrupt, and the format of the interrupt status register in VICE. Bit 31 of the *Interrupt Status* register is read only.

5.7.2.1 RE Interrupts

Bits 27:22 of the *Interrupt Status* register are set by the rendering engine. To clear an interrupt, software must clear the condition that caused the interrupt.

5.7.2.2 CPU Error Interrupt

Bit 21 of the *Interrupt Status* register indicates that a CPU transaction error has occurred. The *Error Status* register indicates the type of error.

5.7.2.3 Memory Error Interrupt

Bit 20 of the *Interrupt Status* register indicates that a memory ECC error has occurred.

5.7.3 GBE Interrupts

GBE updates bits 19:16 of the *Interrupt Status* register by sending Crime 4 interrupt set signals across the GBE/Crime interconnect. If an interrupt set signal is asserted, the GBE interface logic on Crime asserts for one cycle the appropriate gbe_set_int(3:0) signal that goes to the *Interrupt Status* register. A GBE interrupt is cleared by writing a 0 to the appropriate bit of the Crime *Hardware Interrupt* register. Please note that the GBE interrupts are edge sensitive, not level sensitive.

5.7.4 MACE Interrupts

MACE updates bits 15:0 of the *Interrupt Status* register by sending Crime a 16 bit data and mask over the MACE/Crime interconnect. The figure below is MACE interrupt 0 which is bit 0 of the Interrupt Status register. Bit 15:1 use the same logic. Please refer to the MACE specification for the procedure to clear an interrupt, and the format of the interrupt status register in MACE. Bits15:0 of the *Interrupt Status* register are read only.

5.7.5 Interrupt Structure

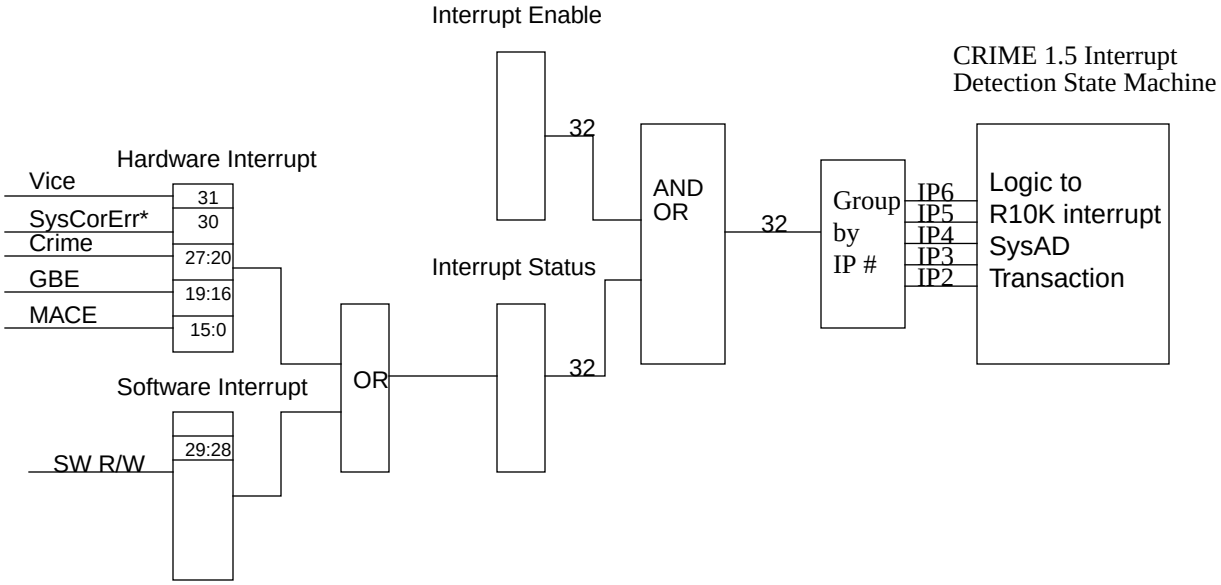
The *Hardware Interrupt* Register is set by the various hardware devices in the Moosehead system. The *Software Interrupt* Register can be used to emulate any of the Hardware Interrupts and it also provides for two software only interrupts. The *Interrupt Status* Register can be read by software to see what individual interrupts actually caused the interrupt on one of the R10K IP levels. The *Interrupt Enable* Register can be used to disable any single interrupt inside of CRIME.

The IP3-IP7 level R10K interrupts cannot be disabled as a group inside of CRIME as that feature can be accomplished using the masks inside the R10K processor.

Whenever a new interrupt is received by CRIME, the R10K Interrupt Register must be written by the CRIME interrupt detection State Machine to inform the processor that an interrupt has occurred.

Refer to Figure 5-6 for a graphical relationship of the different Interrupt Registers.

Figure 5-6 Interrupt Register Relationship



5.7.6 Edge vs. Level Interrupts

Edge interrupts are “latched” inside of CRIME such that when the source of the interrupt is removed, CRIME continues to remember the interrupt in the *Hardware Interrupt Status Register.* The Hardware Interrupt Register must be written to clear this “latched” interrupt as part of the interrupt handler. If CRIME has no other interrupt source on that level once the latched interrupt is cleared, CRIME will perform a SysAD Interrupt Write to deassert that particular IP level. If the Source (RE, GBE, Software Interrupt) is still asserted at that time, CRIME will immediately re-interrupt the R10K processor again.

Level interrupts are not latched inside of CRIME. When a level interrupt is asserted by the Source (RE, Memory Controller, MACE) then CRIME will perform a SysAD Interrupt Write Cycle to assert the interrupt inside of the R10K processor. When the level interrupt is deasserted by the

Source, CRIME will perform a SysAD Interrupt Write Cycle to deassert the interrupt inside of the R10K.

5.8 Timers

5.8.1 WatchDog Timer

Refer to Table 5-14 for the details of the Watch Dog Timer.

5.8.2 Crime Timer

Refer to Table 5-15 for the details of the Crime Timer.

5.9 Register Map

The Crime CPU interface registers (address 1400 0000 through 1400 4048) can only be accessed with double word operations.

The same restriction applies to the Memory Control Register Addresses and the Rendering Engine Addresses?

Table 5-7 CPU Interface Register Address Map

Addr	Function
1400 0000	ID-Revision
1400 0008	CPU Interface Control
1400 0010	Interrupt Status
1400 0018	Interrupt Enable
1400 0020	Software Interrupt
1400 0028	Hardware Interrupt
1400 0030	Watchdog Timer
1400 0038	Crime Timer
1400 0040	CPU Error Address
1400 4048	CPU Error Status **Note: This register moved to separate 16K page so that it can be mapped to user programs while not exposing the other registers in CRIME 1.5 as they can be mapped to a different page. In CRIME 1.1 this register appears at 0x1400 0048.**
1400 0200	Memory Control Register Base Address
1500 0000	Rendering Engine Register Base Address

ID-Revision Register should have bits 3:0 (the revision number) hard-wired to a mux in the RTL and placed on one level of metal so that metal mask revisions of the chip can easily incorporate a revision number change.

Note:
Petty Crime had an ID value of 0xA and a revision number of 0
Crime 1.1 had an ID value of 0xA and a revision number of 1

Table 5-8 ID-Revision Register

Bits	Function	Read/ Write	Reset Value
7:4	ID value	R	B
3:0	revision number	R	1

Table 5-9 CPU Interface Control Register

Bits	Function	R/ W	Reset Value
13	R5000 SysADC check control bit **NOT USED IN CRIME 1.5** 0 - disable parity generation from Crime to R5000 1 - enable parity generation from Crime to R5000	R/W	0
12	CRIME SysADC check control bit **NOT USED IN CRIME 1.5** 0 - don’t check SysADC coming in from R5000 1 - check SysADC on data cycles	R/W	0
11	Cold Reset 0 - no action 1 - reset system with power-on reset Set by S/W or watchdog timer, cleared by H/W	R/W	0

Table 5-9 CPU Interface Control Register

Bits	Function	R/ W	Reset Value
10	Soft Reset **NOT USED IN CRIME 1.5** 0 - Normal mode 1 - CPU_SysRst_n asserted for R10K/R12K Set by S/W or watchdog timer, cleared by H/W	R/W	0
9	Watchdog timer enable 0 - disabled 1 - enabled	R/W	0
8	Endianness 0 - Little Endian 1 - Big Endian	R	NA
7:5	"JUICE" Mode - Perform Load if below memory range specified. Ignore Load if "coherent" and above memory range. Perform all non-coherent loads regardless of memory range. 111- Ignore for coherent load above 32 Meg 110- Ignore for coherent load above 28 Meg 101- Ignore for coherent load above 24 Meg 100 - Ignore for coherent load above 20 Meg 011- Ignore for coherent load above 16Meg 010- Ignore for coherent load above 12 Meg 001- Ignore for coherent load above 8 Meg (Present Juice setting is 8 Meg) 000- No Fix implemented, ignore all coherent loads	R/W	001
4:0	Reserved	R	0

See Section 5.7 "Interrupts" and Section 5.7.5 "Interrupt Structure" for a description of the different Interrupt Registers, their functions and structure.

Table 5-10 Interrupt Status Register

Bits	Function	R/W	Reset Value
31	VICE interrupt	R	0
30	CPU_SysCorErr interrupt	R	0
29:28	Software interrupts	R	0
27	RE idle interrupt - level sensitive	R	0
26	RE FIFO full interrupt - level sensitive	R	0
25	RE FIFO empty interrupt - level sensitive	R	0
24	RE idle interrupt - edge sensitive	R	0
23	RE FIFO full interrupt - edge sensitive	R	0
22	RE FIFO empty interrupt - edge sensitive	R	0
21	Memory error interrupt	R	0
20	CPU interface error interrupt	R	0
19:16	GBE interrupts - edge sensitive	R	0
15:0	MACE interrupts	R	0

Table 5-11 Interrupt Enable Register

Please refer to the Interrupt Status Register to see which bits correspond to actual interrupts..

Bits	Function	Read/ Write	Reset Value
31:0	0 - interrupt disabled 1 - interrupt enabled	R/W	0

Table 5-12 Software Interrupt Register

Please refer to the Interrupt Status Register to see which bits correspond to actual interrupts.

Bits	Function	Read/ Write	Reset Value
31:30	0 - no interrupt 1 - interrupt	R/W	0
29:28	software interrupt bits 1:0 0 - no interrupt 1 - interrupt	R/W	0
24:0	0 - no interrupt 1 - interrupt	R/W	0

Table 5-13 Hardware Interrupt Register

Bits	Function	R/W	Reset Value
31	VICE interrupt	R	0
30	CPU_SysCorErr - edge sensitive	R/W	0
29:28	Software interrupts	R/w	0
27	RE idle interrupt - level sensitive	R	0
26	RE FIFO full interrupt - level sensitive	R	0
25	RE FIFO empty interrupt - level sensitive	R	0

Table 5-13 Hardware Interrupt Register

Bits	Function	R/W	Reset Value
24	RE idle interrupt - edge sensitive	R/W	0
23	RE FIFO full interrupt - edge sensitive	R/W	0
22	RE FIFO empty interrupt - edge sensitive	R/W	0
21	Memory error interrupt	R	0
20	CPU interface error interrupt	R	0
19:16	GBE interrupts - edge sensitive	R/W	0
15:0	MACE interrupts	R	0

Table 5-14 Watchdog Timer Register

See description below for how the watchdog timer works in CRIME 1.1. For CRIME 1.5 the behavior will be the same except that the system will always perform a Cold Reset after the second counter overflow, since CRIME 1.5 does not support a Warm Reset.

Bits	Function	Read/ Write	Reset Value
20	Watchdog soft (warm reset) Time Out 1 - Watchdog timed out 0 - normal mode	R/W	0
19:0	Watchdog value	R/W	0

Analysis courtesy Minghua:

bit 20 : Watchdog soft(warm reset) Time Out

1 - Watchdog timed out
0 - normal mode

bit 19-0 : Watchdog value

The description should be :

If the watchdog timer is not cleared by software for 0.5 second, it will initiate a soft reset. If it has still not been cleared after another 0.5 second, it will initiate a hard reset. Note that this is the behavior for CRIME 1.1. For CRIME 1.5 a hard reset will occur at 1.0 second with no activity except an internal flag set at 0.5 seconds.

The watchdog timer counter will increase 1 every 64-cycles of CRIME timer (running on 66.7MHz). When the watchdog timer bit 19 = 1, it will generate a flag to initiate a warmreset and reset the watchdog timer counter. At this time, the watchdog timer counter will start counting from 0 until bit 19 = 1 again. If at this time, the warmreset flag is still not cleared by software, a coldreset will be initiate.

So the time calculation should be :

2^19 * (64 * 1/66.7MHz) = 0.5 Second

Table 5-15 Crime Time Register

Bits	Function	Read/ Write	Reset Value
31:0	Crime timer value	R/W	0

Table 5-16 CPU Error Address Register

The address of a CPU error is recorded in the event of SysAD Parity Error, Invalid Address.

Bits	Function	R/W	Reset Value
31:0	Address[33:2] of CPU PIO error	R	0

Table 5-17 CPU/VICE Error Status Register

The bits in the *CPU/VICE Error Status* register are set by the hardware and cleared by software.

Bits	Function	R/W	Reset Value
8:3	Address[39:34] of CPU PIO error **New in CRIME 1.5 if it can be implemented**	R	XX
2	CPU illegal address	R/W	0
1	VICE write parity error **No longer supported now that SysADC(7:0) are not used.**	R/W	0
0	CPU write parity error **No longer supported now that SysADC(7:0) are not used**.	R/W	0

CHAPTER 6

Memory Controller

6.1 Definition of Memory Controller Terms

Arbitration Slot - An arbitration slot is a series of requests from the same memory client.

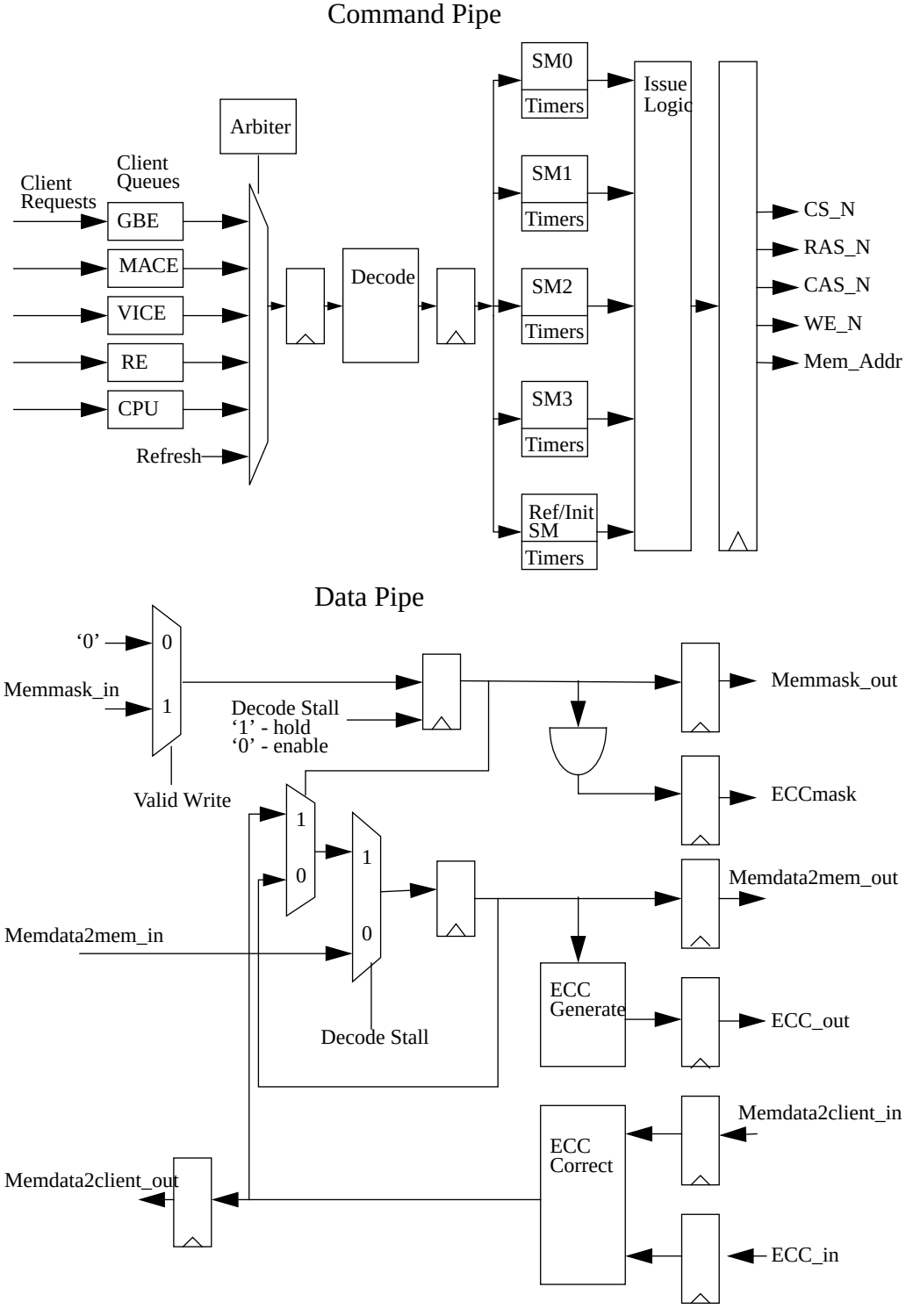
SDRAM page and page crossings - A SDRAM is made up of rows and columns of memory cells. A row of memory cells is referred to as a page. A memory cell is accessed with a row address and column address. When a row is accessed, the entire row is placed into latches, so that subsequent accesses to that row only require the column address. Accesses to the same row are referred to as page accesses. Before accessing a new row, referred to as a page crossing, a precharge command must be issued. The precharge command places the active row back into the array. After the precharge command, the row address and then the column address must be issued. As you can see, crossing a page is expensive and should be avoided whenever possible.

SDRAM bank - A SDRAM bank is an array of memory cells. Each SDRAM component has two internal banks and allows one row from each bank to be active at a time.

6.2 Introduction

The Memory Controller (MC) is the interface between the GBE, MACE, VICE, RE, CPU (clients) and the main memory. Clients make read and write requests to the main memory through the MC. The MC converts the requests into the appropriate control sequences and passes the data between the clients and the main memory. The MC contains two pipeline structures, one for commands and another for data. The request pipe has three stages, arbitration, decode and issue/state machine. The data pipe has only one stage, ECC. Requests and data flow through the pipes in the following manner. Clients place their requests in a queue. The arbitration logic looks at all of the requests at the top of the client queues and decides which request to start through the pipe. From the arbitration stage, the request flows to the decode stage. During the decode stage, information about the request is collected and passed onto the issue/state machine stage. Based on this information, one of the four state machines sequences through the proper commands for the main memory. The later portion of the issue stage decodes the state of the state machine into control signals that are latched and then sent across to the main memory. The MC starts the data through the data pipe when a request reaches the decode stage. The data is held in a flop and flows through the ECC stage and out to the main memory until the request has been retired from the request pipe. A block diagram of the MC is shown in FigureFigure 1, “MC Block Diagram,” on page 3. In the following sections, the MC interfaces are described, followed by a detailed explanation of each stage in the pipes.

Figure 6-1 MC Block Diagram



6.2.1 Client Interface

The client interface contains the following signals:

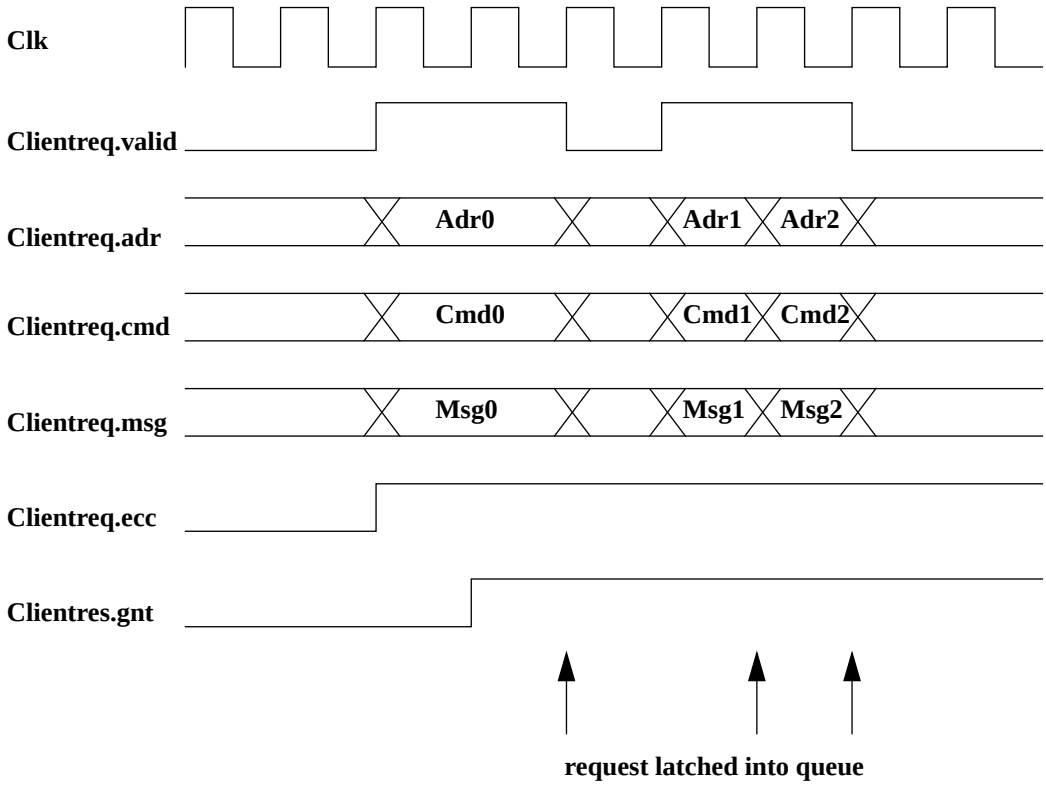
Table 6-1 Client Interface Signals

Signal	Crime Pin Name	# of Bits	Direction	Description
clientreq.cmd	internal only	3	in	type of request - 1 - read 2 - write 4 - rmw
clientreq.adr	internal only	25	in	address of request
clientreq.msg	internal only	7	in	message sent with request
clientreq.valid	internal only	1	in	1 - valid 0 - not valid
clientreq.ecc	internal only	1	in	1 - ecc is valid 0 - ecc not valid
clientres.gnt	internal only	1	out	1 - room in client queue 0 - no room
clientres.wrrdy	internal only	1	out	1 - MC is ready for write data 0 - MC not ready for write data
clientres.rdrdy	internal only	1	out	1 - valid read data 0 - not valid read data
clientres.oe	internal only	1	out	1 - enable client driver 0 - disable client driver
clientres.rdmsg	internal only	7	out	read message sent with read data
clientres.wrmsg	internal only	7	out	write message sent with wrrdy
memdata2mem_in	internal only	256	in	memory data from client going to main memory
memmask_in	internal only	32	in	memory mask from client going to main memory 0 - write byte 1 - don’t write byte memmask_in(0) is matched with memdata2mem_in(7:0) and so on.

Signal	Crime Pin Name	# of Bits	Direction	Description
memdata2client_out	internal only	256	out	memory data from main memory going to the client

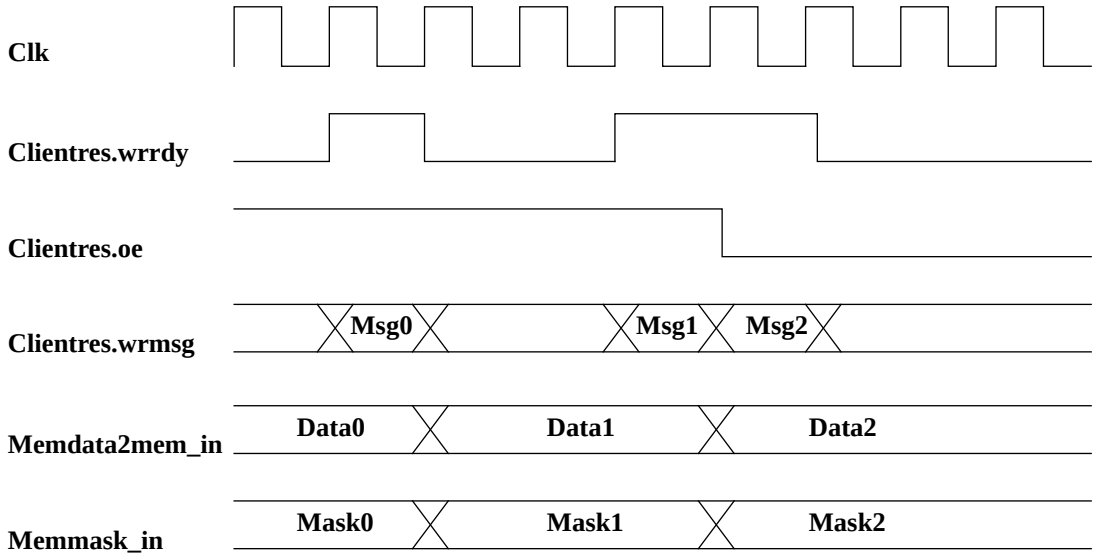
A client makes a request to the MC by asserting clientreq.valid while setting the clientreq.adr, clientreq.msg, clientreq.cmd and clientreq.ecc lines to the appropriate values. If there is room in the queue, the request is latched into the client queue. Only two of the clients, RE and Mace, use clientreq.msg. The message specifies which subsystem within Mace or RE made the request. When an error occurs, this message is saved along with other pertinent information to aid in the debug process. For the RE, the message is passed through the request pipe and returned with the clientreq.wrrdy signal for a write request or with the clientreq.rdrdy signal for a read request. RE uses the information contained in the message to determine which RE queue to access. Please refer to the following figure.

Figure 6-2 Client Request



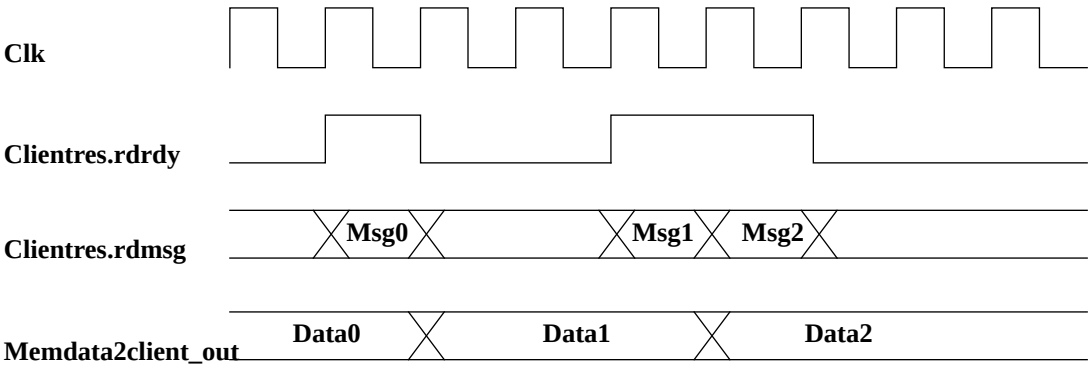
The data for a write request is not latched with the address and request. Instead, the data, mask and message are latched when the MC asserts the clientreq.wrrdy indicating that the request has reached the decode stage of the request pipe. Because the client queues are in front of the request pipe, there is not a simple relationship between the assertion clientreq.gnt and clientreq.wrrdy. Clientreq.msg is only valid for the RE and Mace. The MC asserts the clientreq.oe signal at least one cycle before the assertion of clientreq.wrrdy. Clientreq.oe is latched locally by the client and is used to turn on the client's memory data bus drivers. Please refer to Figure 3, "Client Write Data," on page 8.

Figure 6-3 Client Write Data



The read data is sent to the client over the memdata2client_out bus. When clientres.rdrdy is asserted, the data and message are valid. Please refer to Figure 4, “Client Read Data,” on page 8.

Figure 6-4 Client Read Data



6.2.2 Memory Interface

The memory interface contains the following signals:

Table 6-2 Memory Interface Signals

Signal	Crime Pin Name	# of Bits	Direction	Description
memwrite	mem_dir	1	out	controls direction of SDMUX chips - default to write
memdata2mem_out	mem_data	256	out	memory data from client going to main memory
memdata2client_out	internal only	256	out	memory data from main memory going to the client
memmask_out	mem_mask	32	out	memory mask from client going to main memory
memdataoe	internal only	3	out	enable memory data bus drivers
ecc_out	mem_ecc	32	out	ecc going to main memory
eccmask	mem_eccmask	32	out	ecc mask going to main memory
mem_addr	mem_addr	14	out	memory address
ras_n	mem_ras_n	1	out	row address strobe
cas_n	mem_cas_n	1	out	column address strobe
we_n	mem_we_n	1	out	write enable
cs_n	mem_cs(3:0)_n	8	out	chip selects

The data and mask are latched in the data pipe and flow out to the main memory on memmask_out and memdata2mem_out. From the data and mask, the ECC and ECC mask are generated and sent to the main memory across eccmask and ecc_out. The memdataoe signal is used to turn on the memory bus drivers. Data and ECC from the main memory come in on the memdata2client_in and ecc_in busses. The ECC is used to deter-

mine if the incoming data is correct. If there is a one bit error in the data, the error is corrected, and the corrected data is sent to the client. If there is more than one bit in error, the cpu is interrupted, and incorrect data is returned to the client. Ras_n, cas_n, we_n and cs_n are control signals for the main memory. Please refer to the following figures for an example of a main memory read and write to a new page. A read or write operation to the same SDRAM page is the same as the operation shown in the following figures, except a same page operation does not need the precharge and activate cycles.

Figure 6-5 Main Memory Write

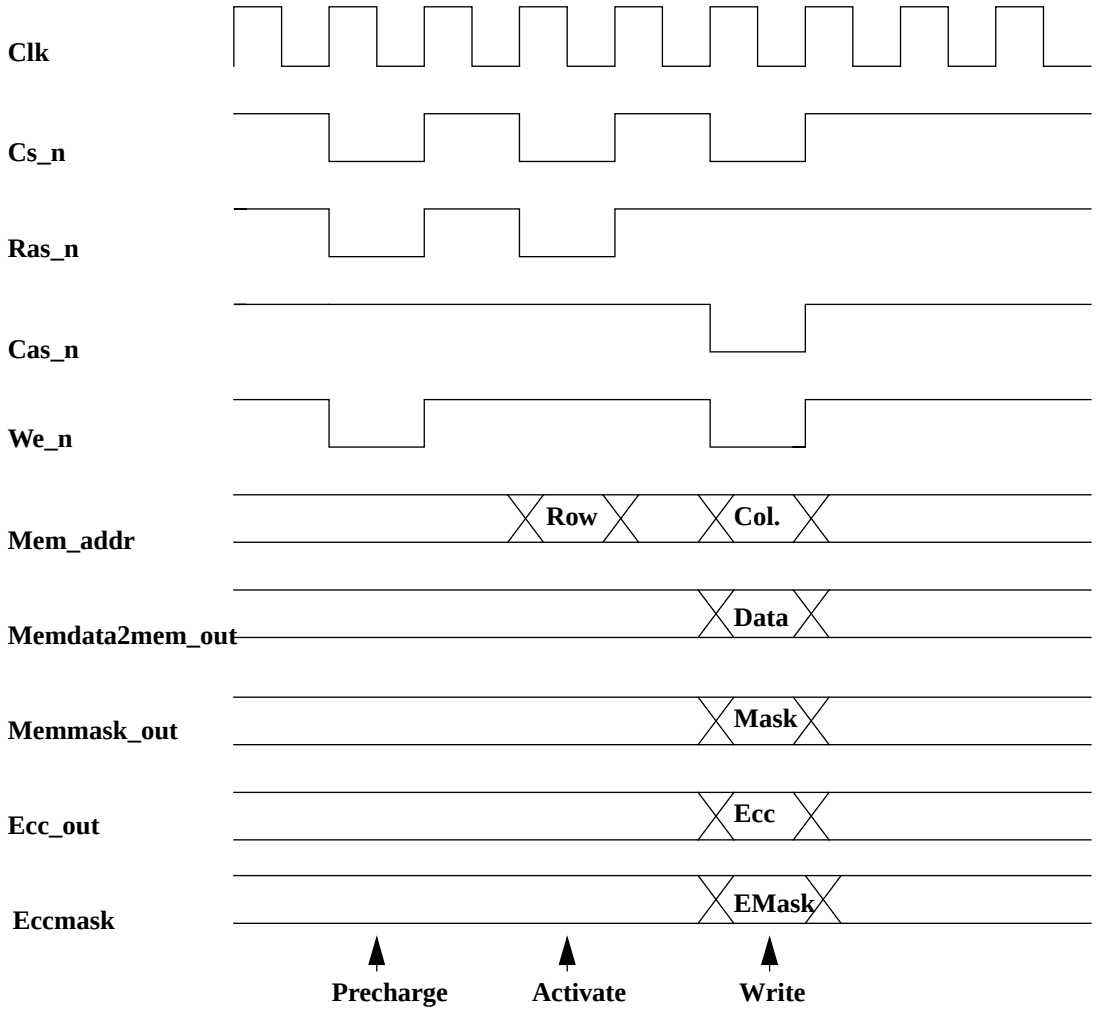
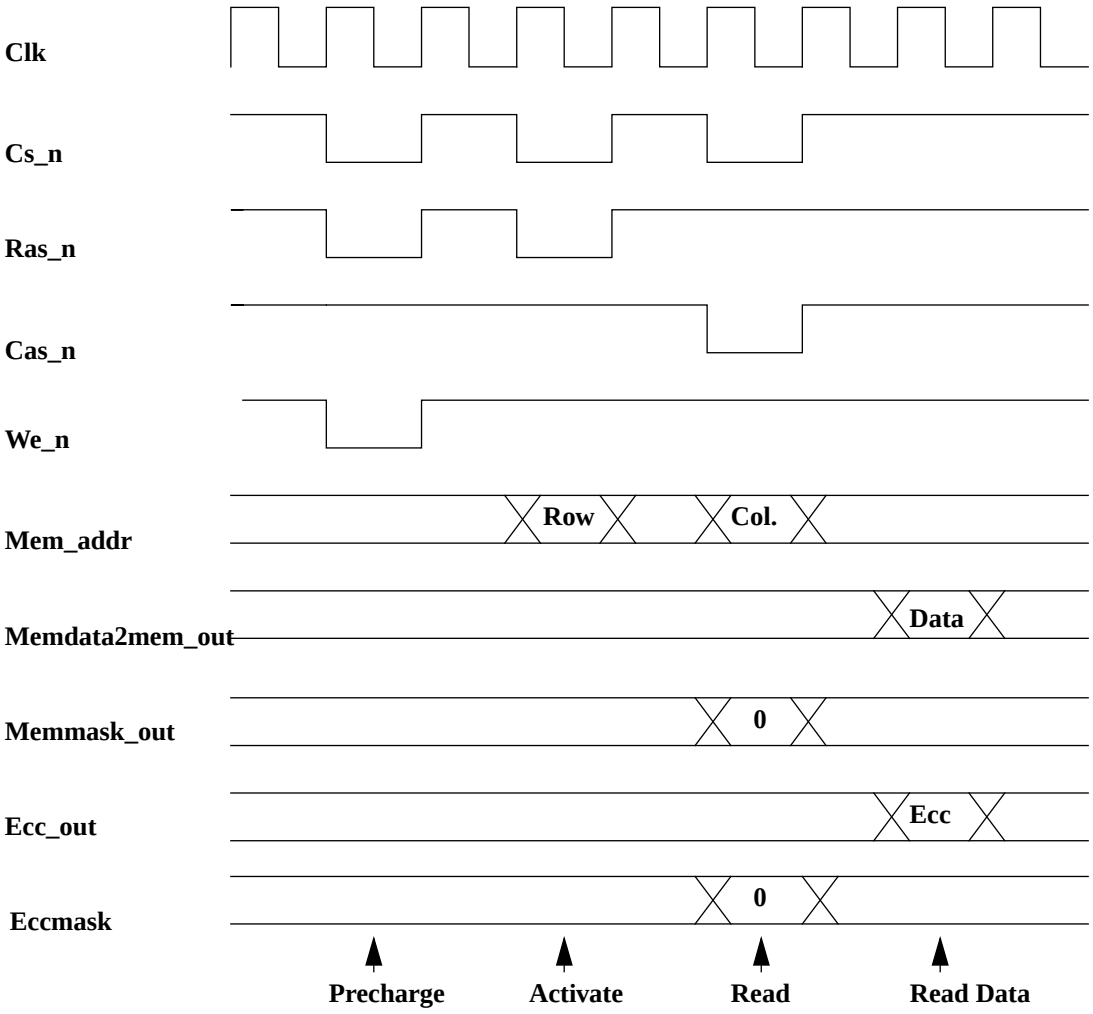


Figure 6-6 Main Memory Read



6.2.3 Request Pipe

The request pipe is the control center for the memory controller. Client requests are placed in one end of the pipe and come out the other side as memory commands. The client queues are at the front of the pipe, followed by the arbitration, then the decode, and finally the issue/state machine. If there is room in their queue, a client can place a request in it. The arbitration logic looks at all of the requests at the top of the client

queues and decides which request to start through the request pipe. From the arbitration stage, the request flows to the decode stage. During the decode stage, information about the request is collected and passed onto the issue/state machine stage. Based on this information, a state machine determines the proper sequence of commands for the main memory. The later portion of the issue stage decodes the state of the state machine into control signals that are latched and then sent across to the main memory. A request can sit in the issue stage for more than one cycle. While a request sits in the issue/state machine stage, the rest of the request pipe is stalled. Each stage of the request pipe is discussed in detail in the following sections.

6.2.3.1 Client Queue

All of the clients have queues, except for Refresh. A refresh request is guaranteed to retire before another request is issued, so a queue is not necessary. The five client queues are simple two-port structures with the clients on the write side and the arbitration logic on the read side. If there is space available in a client queue, indicated by the assertion of clientres.gnt, a client can place a request into its queue. A client request consists of an address, a command (read, write or read-modify-write), a message, an ECC valid and a request valid indication. If both clientreq.valid and clientres.gnt are asserted, the request is latched into the client queue. If the pipeline is not stalled, the arbitration logic looks at all of the requests at the top of the client queues and determines which request to pop off and pass to the decode stage of the request pipe.

Because there is a request queue between the client and the arbiter, the clientres.gnt signal does not indicate that the request has retired. The request still needs to go through the arbitration process. To put it another way, client A might receive the clientres.gnt signal before client B, but if client B has a higher priority, its request might retire before the request from client A.

6.2.3.2 Arbiter

As stated above, the arbiter determines which client request to pass to the decode stage of the request pipe. This decision process has two steps. The first step is to determine if the arbitration slot for the current client is over or not. An arbitration slot is a series of requests from the same client. The

number and type of requests allowed in one arbitration slot varies. The following table lists what each client can do in an arbitration slot.

Table 6-3 Requests allowed in an Arbitration Slot

Client	Possible Operations
GBE	<= 16 memory word read with no page crossings <= 16 memory word write with no page crossings
VICE	<= 8 memory word read with 1 page crossings <= 8 memory word write with 1 page crossings 1 read-modify-write operation
RE, CPU, MACE	<= 8 memory word read with no page crossings <= 8 memory word write with no page crossings 1 read-modify-write operation
Refresh	refresh 2 rows

Based on state for the current arbitration slot and the next request from the current slot owner, the arbiter determines if the arbitration slot should end or not. If not, the request from the client who owns the current arbitration slot is passed to the decode stage. If the current arbitration slot is terminated, the arbiter uses the results from an arbitration algorithm to decide which request to pass to the decode stage. The arbitration algorithm ensures that GBE gets 1/2 of the arbitration slots, MACE 1/4, VICE 1/8, RE 1/16, CPU 1/32 and refresh 1/64.

Predicting the average bandwidth for each client is difficult, but the worst-case slot frequency per client can be calculated. The first step is to determine the maximum number of cycles that each client can use during

an arbitration slot. The following table shows the number of cycles associated with each type of operation.

Table 6-4 4 Maximum Cycles for a Memory Operation

Operation	Command Sequence	# of Cycles
8 Word Read	P X A X R0 R1 R2 R3 R4 R5 R6 R7	12
8 Word Write	P X A X W0 W1 W2 W3 W4 W5 W6 W7	12
Read-Modify-Write	P X A X R0 X X X X X X W0	12
8 Word Vice Read with page crossing	P X A X R0 X X P X A X R1 R2 R3 R4 R5 R6 R7	18
2 Row Refresh	P X Ref X X X X X Ref X X X X X	14

Key -
P - Precharge
X - Dead Cycle
A - Activate
R0 - Read Word 0
W0 - Write Word 0
Ref - Refresh

The following table lists the maximum number of cycles for each of the

Table 6-5 Maximum # Cycles per Slot

Client	**Operation**	**# of Cycles**
GBE	16 memory word read or write	20 cycles
CPU,RE,MACE	8 memory word read or write	12 cycles
VICE	8 memory word read or write 1 page crossings	18 cycles
Refresh	refresh 2 rows	14 cycles

memory clients.

Finally, slots per second for each client can be calculated. If all of the clients are requesting all of the time, every client will get a turn after 64 slots. Lets refer to this as a “round”. In that round, GBE gets 32 out of the

64 slots, MACE gets 16 out of the 64 slots etc., so a round takes 32*20 + 16*12 + 8*18 + 4*12 + 2*12 + 14 =1062 cycles.

Table 6-6 Slot Frequency for Each Client

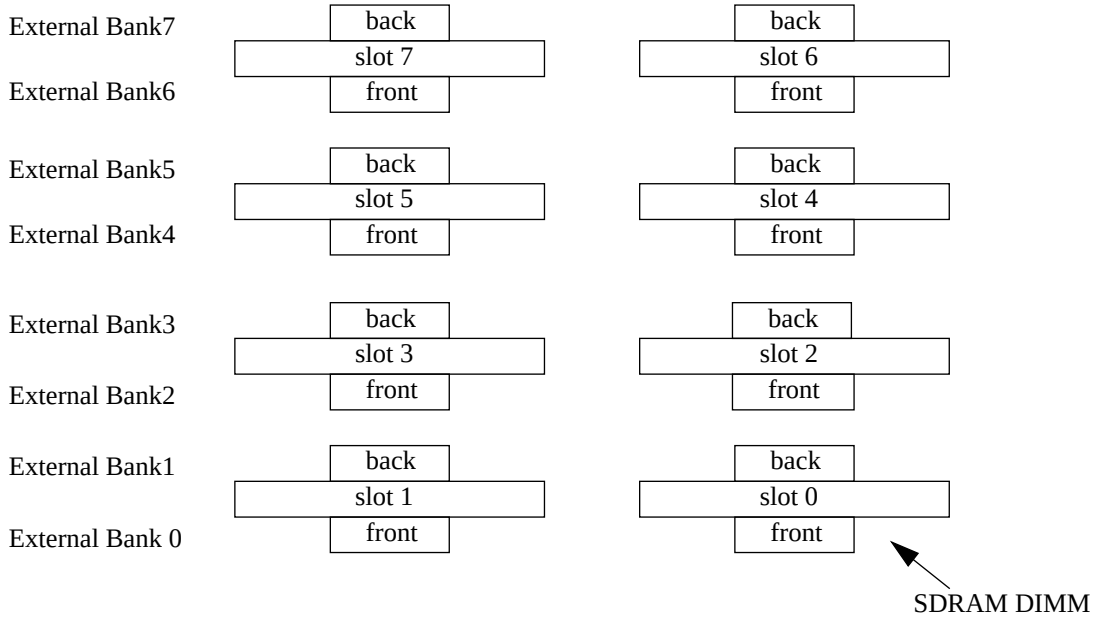
Client		Bandwidth f Slot is Fully Utilized
GBE	32 slots/15.93 us slot/0.50 us	1 GB/sec
MACE	slot/1.00 us	256 MB/sec
VICE	slot/2.00 us	128 MB/sec
RE	slot/4.00	64 MB/sec
CPU	slot/8.00 us	32 MB/sec
Refresh	slot/16.00 us	NA

6.2.3.3 Decode Logic

The decode logic receives requests from the arbiter. Based on state maintained from the previous requests and information contained in the current request, the decode logic determines which memory bank to select, which of the four state machines in the next stage will handle the request, and whether or not the current request is on the same page as the previous request. This information is passed to the issue/state machine stage.

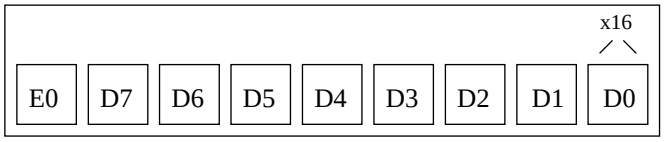
Before continuing the discussion of the decode logic, the memory system is explained so that the details of the decode logic are easier to understand. The memory system is made up of 8 slots. Each slot can hold one SDRAM DIMM. A SDRAM DIMM is constructed from 1Mx16 or 4Mx16 SDRAM components and populated on the front only or the front and back side of the DIMM. Two DIMMs are required to make an external SDRAM bank. 1Mx16 SDRAM components construct a 32 Mbyte external bank, while 4Mx16 SDRAM components construct a 128 Mbyte external bank. The memory system can range in size from 32 Mbytes to 1 Gbyte. Please refer to the figure below

Figure 6-7 Memory System



External bank =
2 SDRAM Front Only DIMMs =
256 bits of data and 32 bits of ECC

Front side of a SDRAM DIMM
8 data chips = 128 bits
1 ECC chip = 16 bits



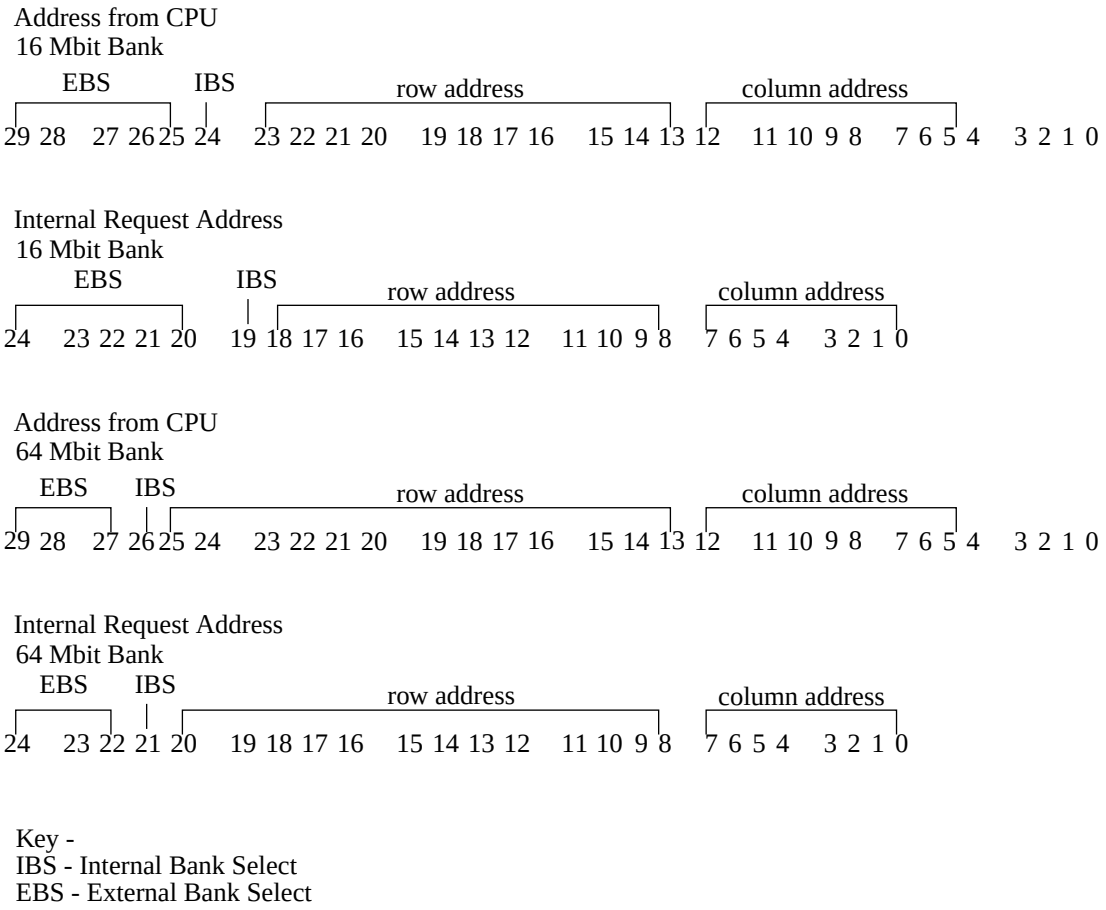
Each SDRAM component has two internal banks, hence two possible open pages. The maximum number of external banks is 8 and the maximum number of internal banks is 16. The memory controller only supports 4 open pages at a time. This issue will be discussed in detail later in this section.

We will now look at the decode logic in more detail. During initialization, software probes the memory to determine how many banks of memory are present and the size of each bank. Based on this information, the software programs the 8 bank control registers. Each bank control register (please refer to the register section) has one bit that indicates the size of the bank and 5 bits for the upper address bits of that bank. Software

must place the 64 Mbit external banks in the lower address range followed by any 16 Mbit external banks. This is to prevent gaps in the memory. The decode logic compares the upper address bits of the incoming request to the 8 bank control registers to determine which external bank to select. The number of bits that are compared is dependent on the size of the bank. For example, if the bank size is 64 Mbit, the decode logic compares bits 24:22 of the request address to bits 4:2 of the bank control register. If there is a match, that bank is selected. Each external bank has a separate chip select. If an incoming address matches more than one bank's control register, the bank with the lowest number is selected. If an incoming address does not match any of the bank control registers, a memory address error occurs. When an error occurs, pertinent information about the request is captured in error registers and the processor is interrupted -- if the MC interrupt is enabled. The request that caused the error is still sent to the next stage in the pipeline and is processed like a normal request, but the MC deasserts all of the external bank selects so that the memory operation doesn't actually occur. Deasserting the external bank selects is also done when bit 6 of the RE message is set. The RE sets this bit when a request is generated using an invalid TLB entry.

Although the MC can handle any physical external bank configuration, we recommend that external bank 0 always be filled and that the external banks be placed in decreasing density order (for example a 64 Mbit exter-

nal bank in bank 0 and a 16 Mbit external bank in bank 2). Please refer to the following figure.



The previous paragraph describes how the decode logic determines what external bank to select. This paragraph describes the method for determining page crossings and which bank state machine to use in the next stage of the pipeline. The row address, along with the internal and external bank bits for previous requests, are kept in a set of registers which are referred to as the row registers. Each row register corresponds to a bank state machine. There are four row registers (hence four bank state machines), so the decode logic can kept track of up to four open pages. The decode logic compares the internal/external bank bits of the new request with the four row registers. If there is a match, then the bank state machine corresponding to that row register is selected. If the new request does not match any of the row registers, one of the row registers is selected and the register is updated with the new request information. If the internal/external bank bits match one of the row registers and the row bits

of the new request match the row bits in that register, then request is on the same page otherwise it is not.

6.2.3.4 State Machines and Issue Logic

The decode logic passes the request along with the external bank selects, state machine select and same page information to the issue/state machine stage. The selected bank state machine sequences through the proper states, while the issue logic decodes the state of the bank state machine into commands that are sent to the SDRAM DIMMS. In addition to the four bank state machines, there is a state machine dedicated to refresh and initialization operations. The initialization/refresh state machine sequences through special states for initialization and refresh while the four bank state machines are forced to an idle state. The bank state machines and the initialization/refresh state machine are discussed in more detail in the following sections.

6.2.3.5 Bank State Machines

The four bank state machines operate independently, subject only to conflicts for access to the control, address, and data signals. The bank state machines default to page mode operation. That is, the autoprecharge commands are not used, and the SDRAM bank must be explicitly precharged whenever there is a non-page-mode random reference. The decode state passes the request along with the page information to the selected state machine which sequences through the proper states. At certain states, interval timers are started that inhibit the state machine from advancing to the next state until the SDRAM minimum interval requirements have been met. The bank state machines operate on one request at a time. That is, a request sequences through any required precharge and activation phases and then a read or write phase, at which point it is considered completed and the next request initiated. Finally, the state of the four bank state machines is decoded by the issue logic that generates the SDRAM control signals

There are several SDRAM parameters that the state machines must obey. These parameters vary slightly from vendor to vendor, but to simplify the state machines, the most common parameters were chosen and hard coded into the interval timers. Any SDRAM that is not compatible with the parameters listed in the following table is not supported.

Tr2rp and Tr2w are additional timing parameters that explicitly define the interval between successive read, write, and precharge commands. These

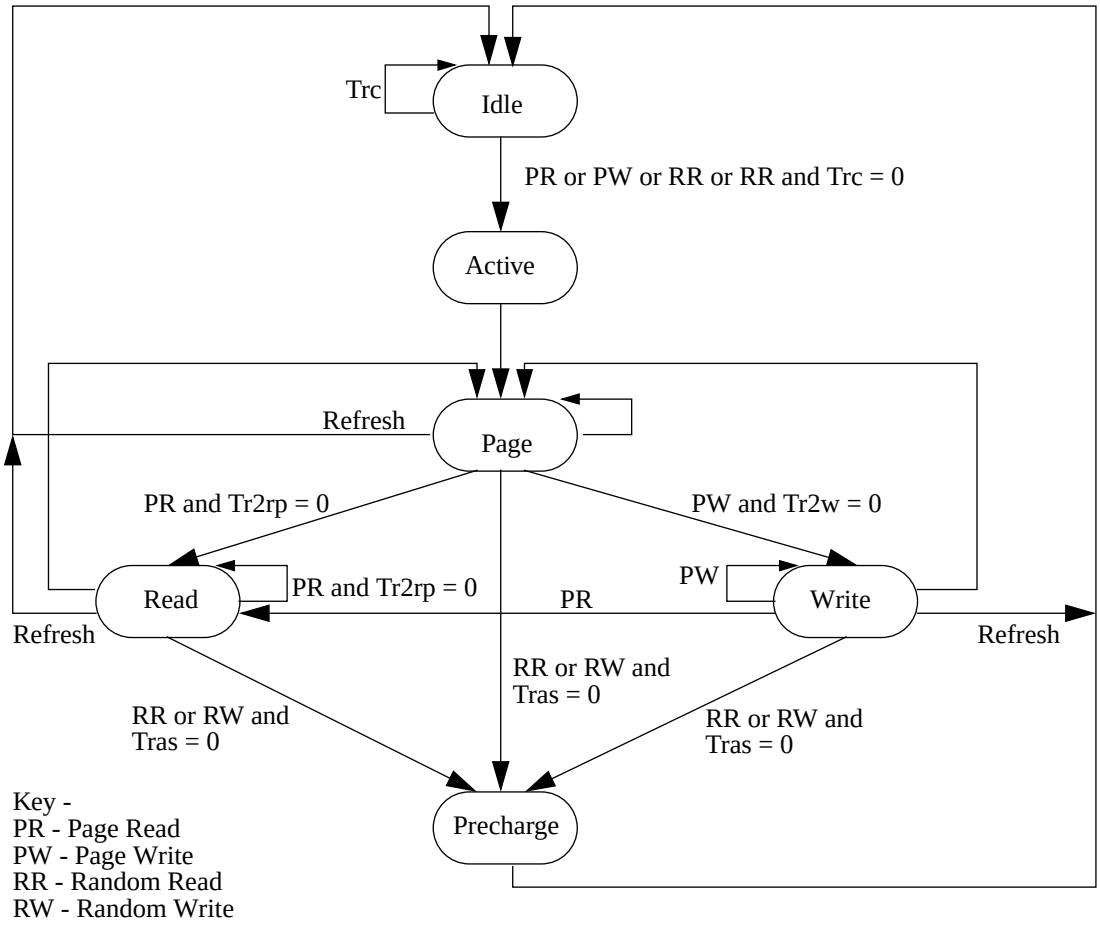
parameters insure that successive commands do not cause conflicts on the data signals. While these parameters could be derived internally by a state machine sequencer, they are made explicit to simplify the state machines and use the same timer paradigm as the SDRAM parameters.

Table 6-7 SDRAM Parameters (Banks A and B are in the same external bank. Bank C is in a different external bank)

Parameter	Value	Description
Trc	7	Activate bank A to Activate bank A
Tras	5	Activate bank A to Precharge bank A
Trp	2	Precharge bank A to Activate bank A
Trrd	2	Activate bank A to Activate bank B
Trcd	2	Activate bank A to Read bank A
Twp	1	Datain bank A to Precharge bank A
Tr2rp	2	Read bank A to Read or Precharge bank C
Tr2w	6	Read bank A to Write bank A

The flow diagram for the bank state machines is shown in Figure 8, “Bank State Machine Flow Diagram,” on page 21. As you can see from the diagram, Trp, Trrd and Trcd are enforced by design. The Trc, Tras, Tr2rp and Tr2w parameters have a timer for each of the four bank state machines. The Tr2rp and Tr2w timers are common to all of the four bank state machines, because they are used to prevent conflicts on the shared data lines.

Figure 6-8 Bank State Machine Flow Diagram



6.2.3.6 Initialization/Refresh State Machine

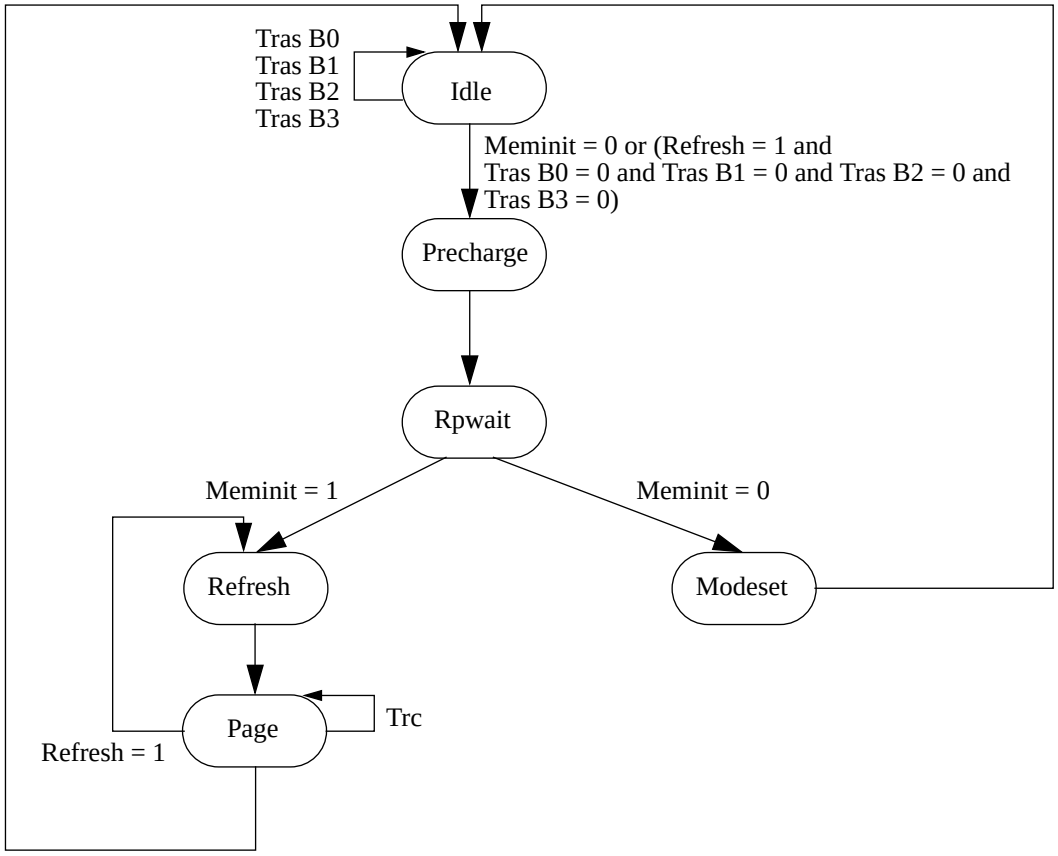
As the name suggests, the initialization/refresh state machine has two functions, initialization and refresh. The initialization procedure is discussed first, followed by the refresh.

After a reset, the initialization/refresh state machine sequences through the SDRAM initialization procedure, which is a precharge to all banks, followed by a mode set. The issue stage decodes the state of the initialization/refresh state machine into commands that are sent the SDRAM. After the mode set, the software will not touch the memory for at least 120 us so that 8 refresh cycles occur. The mode set command programs the SDRAM mode set register to a CAS latency of 2, burst length of 1 and a sequential operation type.

The SDRAM requires that 4096 refresh cycles occur every 64 ms. In order to comply with this requirement, there is a refresh client with a timer. The timer sends out a signal every 27 us which causes the refresh client to make a request to the arbiter. The arbiter treats refresh just like all the other clients. When the arbiter determines that the time for the refresh slot has come, the arbiter passes the refresh request to the decode stage. The decode stage invalidates all of the row registers and passes the request onto the state machine/issue stage. When a bank state machine sees that it is a refresh request, it goes to its idle state. The initialization/refresh state machine sequences through the refresh procedure which is a precharge to all banks followed by two refresh cycles. A refresh command puts the SDRAM in the automatic refresh mode. An address counter, internal to the device, increments the word and bank address during the refresh cycle. After a refresh cycle, the SDRAM is in the idle state, which means that all the pages are closed. This is why it is important that the bank state machines are forced to the idle state and the row registers are invalidated during a refresh request.

The initialization/refresh state machine is very similar in structure to the bank state machines and has timers to enforce SDRAM parameters. A Trc timer is used to enforce the Trc requirement between refresh cycles, and the outputs from the bank Tras timers are used to ensure that the “precharge all” command does not violate Tras for any of the active banks.

Figure 6-9 Initialization/Refresh State Machine Flow Diagram



6.2.4 Data Pipe

The main functions of the data pipe is to move data between the client and main memory, perform ECC operations and to merge new byte data from a client with old data from memory during a read-modify-write operation. Each of these functions is described in detail in the following sections.

6.2.4.1 Data Flow

The data pipe has one stage which is in lock-step with the last stage of the request pipe. When a write request reaches the decode stage, the request pipe asserts clientres.wrrdy. The clientres.wrrdy signal indicates to the client that the data on the Memdata2mem_in bus has been latched into

the ECC stage of the data pipe. The data is held in the ECC stage and flows out to the main memory until the request is retired in the request pipe. Please refer to figure Figure 1, “MC Block Diagram,” on page 3 for a block diagram of the data pipe.

Incoming read data is latched in the data pipe, flows through the ECC correction logic and then is latched again before going on the Memdata2client_out bus. The request pipe knows how many cycles the main memory takes to return read response data. When the read response data is on the Memdata2client_out bus, the request pipe asserts clientres.rdrdy.

6.2.4.2 ECC Logic

The ECC codes chosen for the memory are “single error correcting” and “double error detecting” codes. Eight bits of ECC code cover 64 bits of data. Since the memory word is 256 bits, the ECC logic is composed of four identical generate, correct and detect units. Each unit handles 64 bits of data. Please refer to for a high level view of the ECC logic.

During the ECC stage of the data pipe, ECC is generated and sent to the memory along with the write data. When read response data is returned, the ECC logic generates a syndrome based on the incoming ECC and data. This syndrome is sent to the detect and correct units. Based on the syndrome, the correct unit corrects the incoming data if there is a single bit error. In parallel, the detect unit determines if there is a single or double bit error. If there is an error, all of the pertinent information is stored in the MC register file and the processor is informed via an interrupt -- if the MC interrupt is enabled. If there is an error that can not be corrected, the ECC logic asserts the error signal that is returned to the client along with the incorrect read response data.

ECC is always generated on writes, but a client can choose to turn off the detection and correction units on a read by setting the ECC valid bit to ‘0’ in their read request. If the ECC valid bit is ‘0’, the syndrome for the incoming data is forced to zero, so that data is not corrected and any errors in the data are not detected.

The ECC check bit and replacement registers are used to verify the ECC logic. After any read, the ECC check bit register contains the ECC check bits read from memory. If the “use ECC replacement” bit is set in the MC control register, the value in the ECC replacement register is sent to the main memory as the ECC check bits instead of the ECC check bits gener-

ated by the hardware. The ECC replacement register is only 8 bits wide. The 8 bits are replicated 4 times to generate 32 bits of ECC for the 256 bit memory word. Software can use the ECC replacement register to place bad ECC in main memory.

6.2.4.3 Read-Modify-Write Operation

Read-modify-write operations are necessary when a client wants to write less than 64 bits to an area of memory that is protected by ECC. When the arbiter sees that the request is a read-modify-write command, it breaks the command into a read request followed by a write request. The read and write requests flow down the pipe in the normal fashion. The Tr2w timer is adjusted, so that the write is delayed in the ECC stage until the read data is returned. When read response data appears, the data pipe uses the mask, provided by the client, to determine which bytes of the read data to merge with the new write data from the client. ECC is generated for the resulting data, and both flow out to the memory.

One thing to note: during any write operation, even during a read-modify-write operation, the client data mask is sent unaltered to the memory. What this means is the client data mask prevents the merged corrected read data from being written to the memory. Which means that read-modify-write operations should not be used to scrub memory. The purpose of a read-modify-write operation is to generate ECC on the merged data -- not to rewrite or clean the data that was already in the memory.

6.2.5 Error Handling

There are two types of errors that can occur in the MC, a memory address error and an ECC error. A memory address error occurs when a request address does not match any of the bank control registers. An ECC error occurs when bad data or ECC check bits are read from memory and ECC is enabled. When either type of error occurs, information about the request that caused the error is captured in the *Memory Error Status* register. This information includes which client made the request, the message (if the error was caused by the RE or Mace), and what type of error occurred. The address of the error is captured in the *Memory Error Address* register. The error registers are updated in the following fashion.

First hard error - Both the *Memory Error Status* register and the *Memory Error Address* register are updated.

Second hard error - The multiple error bit in the *Memory Error Status* register is set.

First soft error and no hard errors - Both the *Memory Error Status* register and the *Memory Error Address* register are updated.

Memory address error and no ECC errors - Both the *Memory Error Status* register and the *Memory Error Address* register are updated.

When an ECC error occurs, the syndrome is captured in the *ECC Syndrome* register. For more information about the ECC syndrome, please refer to the “R4000 Users Manual”.

6.2.6 Signals

Table 6-8 Misc. Signals

Signal	Crime Pin Name	# of Bits	Direction	Description
clk	sys_clk2x	1	in	clock
reset_n	internal only	1	in	reset

Table 6-9 Memory Interface Signals

Signal	Crime Pin Name	# of Bits	Direction	Description
memwrite	mem_dir	1	out	controls direction of SDMUX chips - default to write
memdata2mem_out	mem_data	256	out	memory data from client going to main memory
memdata2client_out	internal only	256	out	memory data from main memory going to the client
memmask_out	mem_mask	32	out	memory mask from client going to main memory
memdataoe	internal only	3	out	enable memory data bus drivers
ecc_out	mem_ecc	32	out	ecc going to main memory
eccmask	mem_eccmask	32	out	ecc mask going to main memory
mem_addr	mem_addr	14	out	memory address
ras_n	mem_ras_n	1	out	row address strobe
cas_n	mem_cas_n	1	out	column address strobe
we_n	mem_we_n	1	out	write enable
cs_n	mem_cs(3:0)_n	8	out	chip selects

Table 6-10 Client Interface Signals

Signal	Crime Pin Name	# of Bits	Direction	Description
clientreq.cmd	internal only	3	in	type of request - 1 - read 2 - write 4 - rmw
clientreq.adr	internal only	25	in	address of request
clientreq.msg	internal only	7	in	message sent with request
clientreq.valid	internal only	1	in	1 - valid 0 - not valid
clientreq.ecc	internal only	1	in	1 - ecc is valid 0 - ecc not valid
clientres.gnt	internal only	1	out	1 - room in client queue 0 - no room
clientres.wrrdy	internal only	1	out	1 - MC is ready for write data 0 - MC not ready for write data
clientres.rdrdy	internal only	1	out	1 - valid read data 0 - not valid read data
clientres.oe	internal only	1	out	1 - enable client driver 0 - disable client driver
clientres.rdmsg	internal only	7	out	read message sent with read data
clientres.wrmsg	internal only	7	out	write message sent with wrrdy
memdata2mem_in	internal only	256	in	memory data from client going to main memory
memmask_in	internal only	32	in	memory mask from client going to main memory 0 - write byte 1 - don’t write byte memmask_in(0) is matched with memdata2mem_in(7:0) and so on.

Table 6-10 Client Interface Signals

Signal	Crime Pin Name	# of Bits	Direction	Description
memdata2client_out	internal only	256	out	memory data from main memory going to the client

Table 6-11 PIO Interface Signals

Signal	Crime Pin Name	# of Bits	Direction	Description
piowr	internal only	1	in	1 - write 0 - no write
piord	internal only	1	in	1 - read 0 - no read
pioaddr	internal only	5	in	PIO address
piordrdy	internal only	1	out	1 - read response data is valid 0 - read response data is not valid
piowrrdy	internal only	1	out	1 - ready for a write 0 - not ready for a write
piodata	internal only	64	inout	PIO data

The PIO interface is described in the cpu interface document.

6.2.7 Registers

The MC registers can only be accessed with double word operations.

Table 6-12 MC Register Address Mapr

Addr	Function
1400 0200	MC Status/Control
1400 0208	Bank 0 Control
1400 0210	Bank 1 Control
1400 0218	Bank 2 Control
1400 0220	Bank 3 Control
1400 0228	Bank 4 Control
1400 0230	Bank 5 Control
1400 0238	Bank 6 Control
1400 0240	Bank 7 Control
1400 0248	Refresh Counter
1400 0250	Memory Error Status
1400 0258	Memory Error Address
1400 0260	ECC Syndrome Bits
1400 0268	ECC Generated Check Bits
1400 0270	ECC Replacement Check Bits

Table 6-13 MC Status/Control Register

Address	crime base + 0x200

Bits	Function	Read/ Write	Reset Value
1	Use *ECC Replacement* register for ECC check bits instead of hardware generated ECC check bits. 0 - Normal mode 1 - Use *ECC Replacement* register	R/W	0
0	ECC enable 1 - ECC enabled 0 - ECC disabled	R/W	0

Table 6-14 Bank 0-7 Control Register

Address	crime base + 0x208 to 0x240

Bits	Function	Read/ Write	Reset Value
8	SDRAM size 0 - 16 Mbit 1 - 64 Mbit	R/W	0
7:5	Reserved	R	0
4:0	This field is compared with the request address to determine which of the 8 external banks to select. 16 Mbit - bits 4:0 are compared with request address bits 29:25. 64 Mbit - bits 4:2 are compared with request address bits 29:27.	R/W	bank 0 - 0 bank 1 - 1 bank 2 - 2 bank 3 - 3 bank 4 - 4 bank 5 -5 bank 6 - 6 bank 7 - 7

Table 6-15 Refresh Counter Register

Address	crime base + 0x248

Bits	Function	Read/ Write	Reset Value
10:0	Refresh count value	R/W	count value

Table 6-16 Memory Error Status Register

The bits in the *Memory Error Status* register are set by the hardware and cleared by software. A memory error sets the memory error interrupt bit in the *Hardware Interrupt* register. All bits are active high.

Address	crime base + 0x250

Bits	Function	Read/ Write	Reset Value
27	Invalid Memory Address during a RMW	R/W	0
26	Invalid Memory Address during a write	R/W	0
25	Invalid Memory Address during a read	R/W	0
24	Memory ECC read error during a read-modify-write operation	R/W	0
23	Memory ECC read error	R/W	0
22	Multiple hard errors	R/W	0
21	Hard error	R/W	0
20	Soft error	R/W	0
19	Reserved	R/W	0
18	CPU access	R/W	0
17	VICE access	R/W	0
16	GBE access	R/W	0
15	RE access	R/W	0
14:8	RE source ID	R/W	0
7	MACE access	R/W	0
6:0	MACE source ID	R/W	0

Table 6-17 Memory Error Address Register.

Address	crime base + 0x258

Bits	Function	Read/ Write	Reset Value
29:0	Address of error	R	0

Table 6-18 ECC Syndrome Bits

Address	crime base + 0x260

Bits	Function	Read/ Write	Reset Value
31:24	ECC syndrome bits for bits 255:192 of the 256 bit memory word	R	0
23:16	ECC syndrome bits for bits 191:128 of the 256 bit memory word	R	0
15:8	ECC syndrome bits for bits 127:64 of the 256 bit memory word	R	0
7:0	ECC syndrome bits for bits 63:0 of the 256 bit memory word	R	0

Table 6-19 ECC Generated Check Bits

Address	crime base + 0x268

Bits	Function	Read/ Write	Reset Value
31:24	ECC check bits for bits 255:192 of the 256 bit memory word	R	0
23:16	ECC check bits for bits 191:128 of the 256 bit memory word	R	0
15:8	ECC check bits for bits 127:64 of the 256 bit memory word	R	0
7:0	ECC check bits for bits 63:0 of the 256 bit memory word	R	0

Table 6-20 ECC Replacement Check Bits

If the *Use Replacement Check* bit is set in the *Memory Controller Status/ Control* register, the bits from the following register are used as the ECC check bits instead of the hardware generated check bits.

Address	crime base + 0x270

Bits	Function	Read/ Write	Reset Value
7:0	This byte is replicated four times to create the 32 check bits for the 256 bit memory word.	R/W	0

CHAPTER 7 Rendering Engine

7.1 Rendering Engine Overview

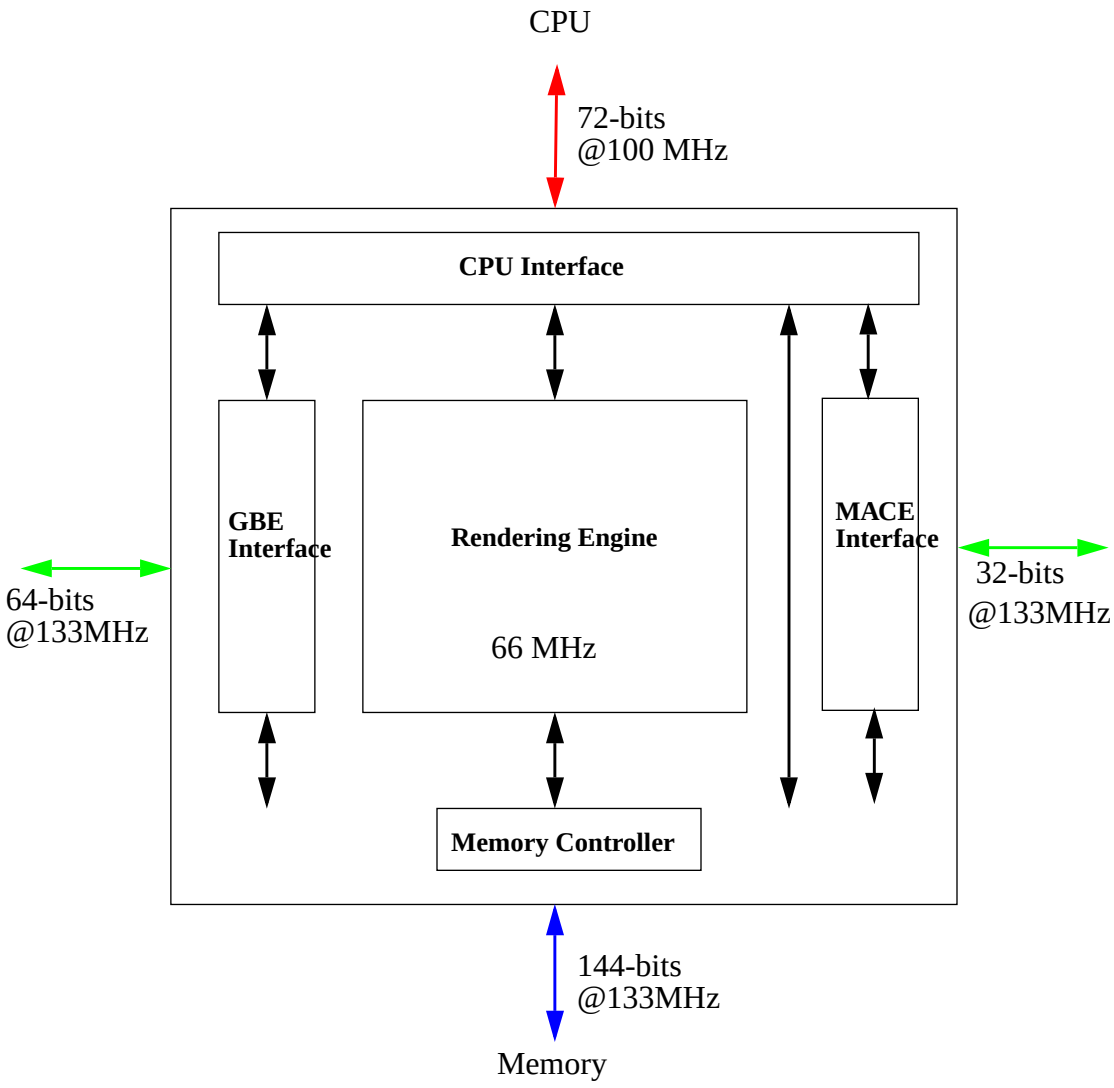
To be updated to more completely describe the CRIME 1.1 Rendering Engine functionality.

No changes to the Rendering Engine are anticipated for the CRIME 1.5 revision.

7.2 Introduction

The CRIME Rendering Engine is a 2D and 3D graphics coprocessor which accelerates X and OpenGL rasterization for Moosehead. It receives rendering parameters from the host and renders directly to framebuffers stored in system memory. The rendering engine is integrated into the CRIME ASIC and is tightly connected to the CPU interface and memory controller subsystems also in CRIME. A high level block diagram of CRIME is shown in Figure 1.

Figure 7-1 CRIME Block Diagram



The rendering engine supports the following features and functions:

- 8-bit, 12-bit color index pixels and 8-bit, 16-bit, and 32-bit RGB(A) pixels
- 64k x 64k pixel address space for X and 4k x 4k pixel address space for OpenGL
- Virtual to physical address translation buffers for rendering and access to dynamically allocated pixel buffers in system memory
- Rasterization of X and OpenGL points, lines, triangles, and rectangles as basic rendering primitives with 6-bit subpixel precision for OpenGL rasterization
- Window-relative addressing and window clipping support through screen masks and clip ID's
- OpengGL scissor testing
- Line and rectangle stippling and patterning
- Gouraud shading of line and triangle primitives
- Texture mapping - 1D and 2D, mip-mapped, filtered
- Fog interpolation and application
- Line antialiasing coverage generation and general coverage application
- OpenGL alpha test functions
- OpenGL alpha blending functions and ops
- Dithering for 8-bit and 16-bit RGB pixels
- Logic ops
- Color buffer plane masking
- 24-bit depth buffer interpolation and depth testing
- 8-bit stencil buffer testing
- Pixel DMA with format conversion and integral zooms through the rendering pipeline
- Memory bandwidth block clear and copy operations

Moosehead has no dedicated graphics memory. All rendering engine pixel buffers (color, depth, stencil, texture) are allocated in system memory and the rendering engine accesses these pixel buffers across Moosehead's high-bandwidth memory data bus. Further, the rendering engine implements a virtual framebuffer rendering model which allows pixel buffers to be allocated in tiles which may be scattered throughout system memory.

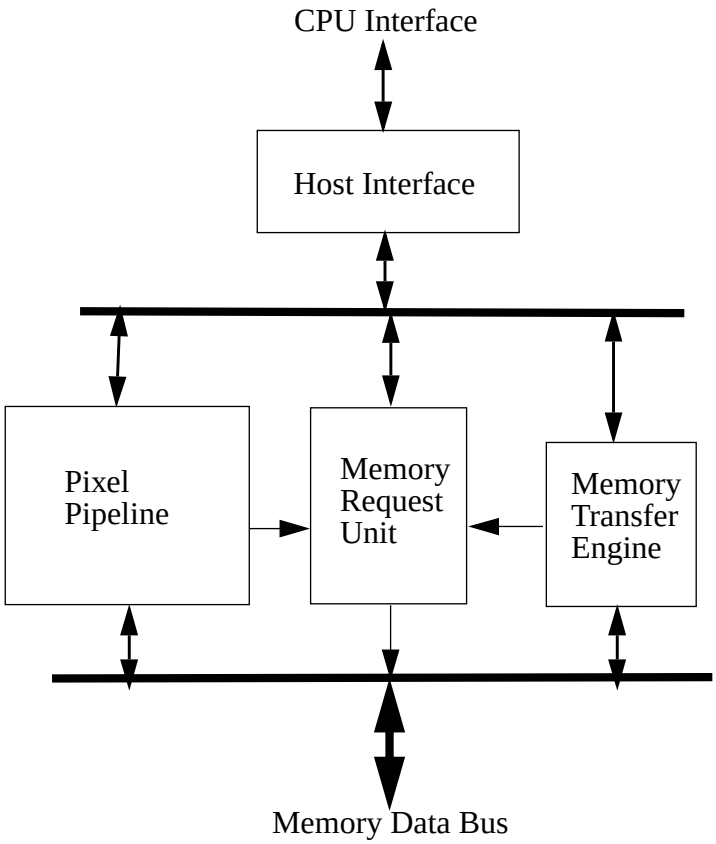
System memory is implemented using synchronous DRAM (SDRAM) accessed via a 256-bit wide memory data bus cycled at 66MHz. Therefore, the Moosehead memory system provides a peak data bandwidth of 2.133 Gb/s.

The rendering engine is also cycled at 66MHz and operates synchronously to the memory system. The rendering engine shares the memory system with the other Moosehead memory system clients, namely the CPU, the graphics back-end (GBE) subsystem, the MACE subsystem, and the VICE coprocessor. The rendering engine issues its memory access requests to the Moosehead memory controller which also resides in CRIME. The memory controller arbitrates requests from the various memory clients and handles the memory bus protocol. Since the rendering engine shares the memory system with other clients, rendering engine performance will vary as a function of the load on the memory system.

The rendering engine accelerates only the rasterization of geometric primitives into the frame buffer. All operations prior to rasterization, such as vertex transforms, lighting, texture coordinate assignments, and primitive plane equation set-up, must be performed by the host processor. The rendering engine does not perform any aspect of the the image display process, which involves fetching and translating visible pixels from memory for display refresh. In Moosehead, image display is controlled entirely by the graphics back-end (GBE) subsystem, which .

The rendering engine is logically partitioned into four major functional units: the *host interface*, the *pixel pipeline*, the *memory transfer engine*, and the *memory request unit*. The *host interface* controls reads and writes from the host to the programming interface registers. The *pixel pipeline* implements the rasterization set-up and rendering pipeline to the framebuffer. The *memory transfer engine* performs memory bandwidth byte-aligned clears and copies on both linear buffers and framebuffers. The *memory request unit* arbitrates between requests from the pixel pipeline and performs the virtual to physical address translation and queues up memory requests to be issued to the memory controller. Figure 2 shows the high-level partioning of the rendering engine.

Figure 7-2 Rendering engine block diagram



7.3 Programming Interface Specification

The rendering engine presents the host with a set of 32-bit or 64-bit programming registers. The host must directly write to these registers to program rendering engine state or initiate rendering operations.

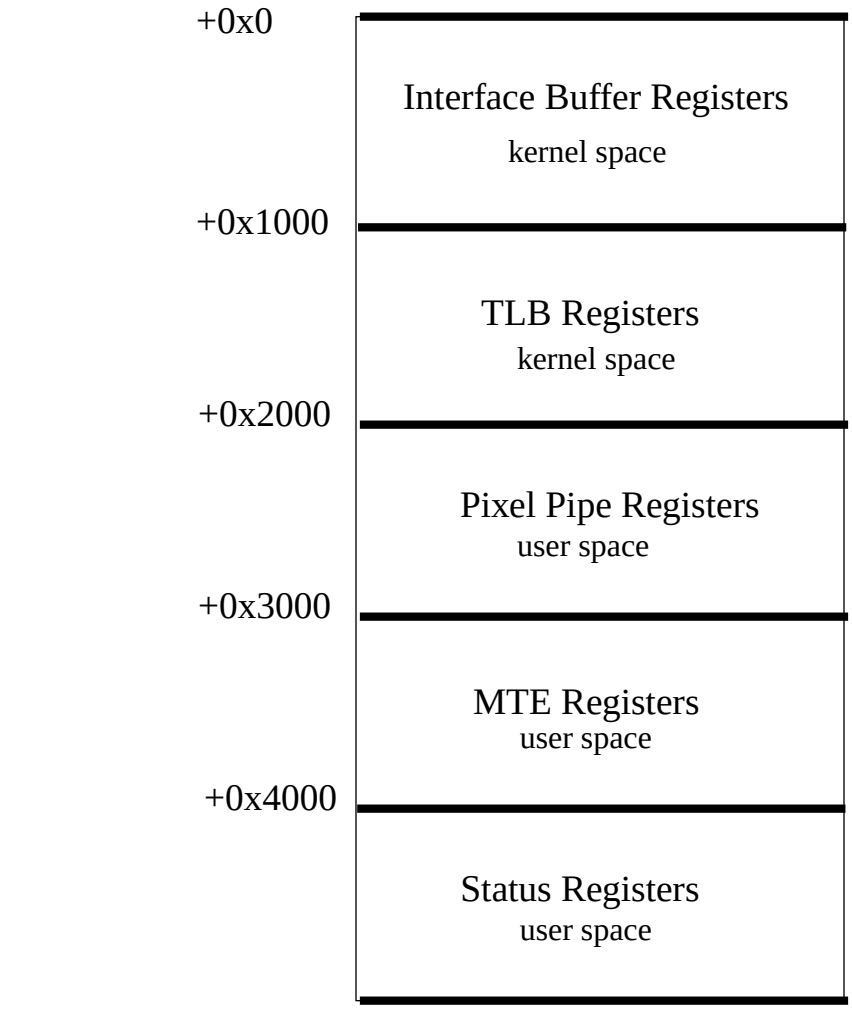
7.3.1 Register Definitions

The rendering engine registers are grouped into five logical sets. These register sets are:

- Interface buffer registers
- TLB registers
- Pixel pipeline registers
- MTE registers
- Status register

Each register set is placed within its own 4kb address space.

Figure 7-3 Rendering engine address space partitioning



7.3.1.1 Interface Buffer Registers

Table 7-1 **Interface Buffer register address map**

Offset	Register	Size	R/W	Description
0x0	IntfBuf.data[128]	64	R/W	interface buffer register data RAM

Table 7-1 Interface Buffer register address map

Offset	Register	Size	R/W	Description
0x200	IntfBuf.addr[128]	64	R/W	interface bufferregister address RAM
0x400	IntfBuf.ctl	32	W	interface buffer control parameters

7.3.1.2 TLB Registers

The TLB performs the translation from framebuffer, texture, or linear virtual addresses to physical memory addresses. The TLB registers hold the base addresses of the 64kb tiles and 4kb linear pages.

Table 7-2 TLB register address map

Offset	Register	Size	Type	R/W	Description
0x0	TLB.fbA[64]	64	G	R/W	framebuffer 64kb tile TLB A
0x200	TLB.fbB[64]	64	G	R/W	framebuffer 64kb tile TLB B
0x400	TLB.fbC[64]	64	G	R/W	framebuffer 64kb tile TLB C
0x600	TLB.tex[28]	64	G	R/W	texture map 64kb tile TLB
0x6e0	TLB.cid[4]	64	G	R/W	clip ID 64kb tile TLB
0x700	TLB.linearA[16]	64	G	R/W	linear 4kb page TLB A
0x780	TLB.linearB[16]	64	G	R/W	linear 4kb page TLB B

7.3.1.3 Pixel Pipeline Registers

The pixel pipe registers specify and control the operation of the pixel rasterization pipeline. There are two classes of pixel pipe registers - global and non-global. Global registers are not updated until all pixels are flushed from the pipeline to memory. Non-global registers are loaded as soon as the beginning of the pipeline is idle. Typically, the global registers set-up drawing context which would apply to multiple primitives and the non-global registers would typically be updated per primitive.

Table 7-3 Pixel Pipeline register address map

Offset	Register	Size	Type	R/W	Description
0x000	BufMode.src	32	G	R/W	pixel transfer source color buffer and pixel formats
0x008	BufMode.dst	32	G	R/W	destination color buffer and pixel formats
0x010	ClipMode	32	G	R/W	window clipping mode bits
0x018	DrawMode	32	G	R/W	rendering mode enable bits
0x020	ScrMask[0]	64	G	R/W	framebuffer-relative (x_{min},y_{min}) and (x_{max},y_{max}) of screen mask 0
0x028	ScrMask[1]	64	G	R/W	framebuffer-relative (x_{min},y_{min}) and (x_{max},y_{max}) of screen mask 0
0x030	ScrMask[2]	64	G	R/W	framebuffer-relative (x_{min},y_{min}) and (x_{max},y_{max}) of screen mask 0
0x038	ScrMask[3]	64	G	R/W	framebuffer-relative (x_{min},y_{min}) and (x_{max},y_{max}) of screen mask 0
0x040	ScrMask[4]	64	G	R/W	framebuffer-relative (x_{min},y_{min}) and (x_{max},y_{max}) of screen mask 0
0x048	Scissor	64	G	R/W	window-relative (x_{min},y_{min}) and (x_{max},y_{max}) of GL scissor rectangle
0x050	WinOffset.src	32	G	R/W	source framebuffer window offset (x_{offset},y_{offset}) for translation from window-relative to framebuffer-relative coordinates
0x058	WinOffset.dst	32	G	R/W	destination framebuffer window offset (x_{offset},y_{offset}) for translation from window-relative to framebuffer-relative coordinates
0x060	Primitive	32	NG	W	geometric primitive op-code
0x070	Vertex.X[0]	32	NG	W	(x_0,y_0) vertex for X primitive
0x074	Vertex.X[1]	32	NG	W	(x_1,y_1) vertex for X primitive

Table 7-3 Pixel Pipeline register address map

Offset	Register	Size	Type	R/W	Description
0x078	Vertex.X[2]	32	NG	W	(x_2,y_2) vertex for X primitive
0x080	Vertex.GL[0].x	32	NG	W	x_0 vertex coordinate for GL primitive. 13.6 fixed point value.
0x084	Vertex.GL[0].y	32	NG	W	y_0 vertex coordinate for GL primitive. 13.6 fixed point value.
0x088	Vertex.GL[1].x	32	NG	W	x_1 vertex coordinate for GL primitive. 13.6 fixed point value.
0x08c	Vertex.GL[1].y	32	NG	W	y_1 vertex coordinate for GL primitive 13.6 fixed point value.
0x090	Vertex.GL[2].x	32	NG	W	x_2 vertex coordinate for GL primitive 13.6 fixed point value.
0x094	Vertex.GL[2].y	32	NG	W	y_2 vertex coordinate for GL primitive 13.6 fixed point value.
0x0a0	PixelXfer.src.addr	32	NG	W	pixel transfer framebuffer or linear source buffer start address
0x0a8	PixelXfer.src.xStep	32	NG	W	pixel transfer source buffer x-direction step size
0x0ac	PixelXfer.src.yStep	32	NG	W	pixel transfer source buffer y-direction step size
0x0b0	PixelXfer.dst.linAddr	32	NG	W	pixel transfer linear destination buffer starting address
0x0b4	PixelXfer.dst.linStride	32	NG	W	pixel transfer linear destination buffer stride offset
0x0c0	Stipple.mode	32	NG	R/W	stipple mode
0x0c4	Stipple.pattern	32	NG	R/W	32-bit stipple pattern
0x0d0	Shade.fgColor	32	NG	W	flatshade or foreground color
0x0d8	Shade.bgColor	32	G	R/W	background color for opaque stippling
0x0e0	Shade.R_s	32	NG	W	initial red component value for color interpolation 9.12 two’s complement fixed point value

Table 7-3 Pixel Pipeline register address map

Offset	Register	Size	Type	R/W	Description
0x0e4	Shade.G_s	32	NG	W	initial green component value for color interpolation 9.12 two's complement fixed point value
0x0e8	Shade.B_s	32	NG	W	initial blue component value for color interpolation 9.12 two's complement fixed point value
0x0ec	Shade.A_s	32	NG	W	initial alpha component value for color interpolation 9.12 two's complement fixed point value
0x0f0	Shade.drdx	32	NG	W	dR/dx slope for color interpolation 9.12 two's complement fixed point value
0x0f4	Shade.dgdx	32	NG	W	dG/dx slope for color interpolation 9.12 two's complement fixed point value
0x0f8	Shade.drdy	32	NG	W	dR/dy slope for color interpolation 9.12 two's complement fixed point value
0x0fc	Shade.dgdy	32	NG	W	dG/dy slope for color interpolation 9.12 two's complement fixed point value
0x100	Shade.dbdx	32	NG	W	dB/dx slope for color interpolation 9.12 two's complement fixed point value
0x104	Shade.dadx	32	NG	W	dA/dx slope for color interpolation 9.12 two's complement fixed point value
0x108	Shade.dbdy	32	NG	W	dB/dy slope for color interpolation 9.12 two's complement fixed point value
0x10c	Shade.dady	32	NG	W	dA/dy slope for color interpolation 9.12 two's complement fixed point value
0x110	Texture.mode	32	G	R/W	texturing mode bits
0x118	Texture.format	64	G	R/W	texture map coordinate formats
0x120	Texture.SQ_s	64	NG	W	initial s/w value for homogeneous texture coordinate interpolation 36.12 two's complement fixed point value

Table 7-3 Pixel Pipeline register address map

Offset	Register	Size	Type	R/W	Description
0x128	Texture.TQ_s	64	NG	W	initial t/w value for homogeneous texture coordinate interpolation 36.12 two's complement fixed point value
0x130	Texture.Q_s	32	NG	W	initial 1/w value for homogeneous texture coordinate interpolation 18.12 two's complement fixed point value
0x138	Texture.dsqdx	64	NG	W	d(s/w)/dx slope for homogeneous texture coordinate interpolation 36.12 two's complement fixed point value
0x140	Texture.dsqdy	64	NG	W	d(s/w)/dx slope for homogeneous texture coordinate interpolation 36.12 two's complement fixed point value
0x148	Texture.dtqdx	64	NG	W	d(t/w)/dx slope for homogeneous texture coordinate interpolation 36.12 two's complement fixed point value
0x150	Texture.dtqdy	64	NG	W	d(t/w)/dx slope for homogeneous texture coordinate interpolation 36.12 two's complement fixed point value
0x158	Texture.dqdx	32	NG	W	d(1/w)/dx slope for homogeneous texture coordinate interpolation 18.12 two's complement fixed point value
0x15c	Texture.dqdy	32	NG	W	d(1/w)/dx slope for homogeneous texture coordinate interpolation 18.12 two's complement fixed point value
0x160	Texture.borderColor	32	G	R/W	texture border color
0x168	Texture.envColor	32	G	R/W	texture environment color
0x170	Fog.color	32	G	R/W	RGB fog color

Table 7-3 Pixel Pipeline register address map

Offset	Register	Size	Type	R/W	Description
0x178	Fog.F_s	32	NG	W	initial fog blending factor value 9.12 two's complement fixed point value
0x180	Fog.dfdx	32	NG	W	df/dx slope for fog blending factor interpolation 9.12 two's complement fixed point value
0x188	Fog.dfdy	32	NG	W	df/dy slope for fog blending factor interpolation 9.12 two's complement fixed point value
0x190	Antialias.line	32	NG	W	slope of ideal line and ideal line starting minor coordinate for antialiasing 2.12 two's complement fixed point value for both slope and ideal line coordinate
0x194	Antialias.cov	32	NG	W	start and end endpoint coverage values for antialiasing 7-bit coverage values
0x198	AlphaTest.	32	G	R/W	alpha test function
0x1a0	Blend.constColor	32	G	R/W	blend RGBA constant color
0x1a8	Blend.func	32	G	R/W	source and destination blending functions
0x1b0	LogicOp	32	G	R/W	logic op value
0x1b8	ColorMask	32	G	R/W	color buffer plane mask
0x1c0	Depth.func	32	G	R/W	depth test function
0x1c8	Depth.Z_s	64	NG	W	initial z value for depth interpolation 25.12 two's complement fixed point value
0x1d0	Depth.dzdx	64	NG	W	dz/dx slope for depth interpolation 25.12 two's complement fixed point value
0x1d8	Depth.dzdy	64	NG	W	dz/dy slope for depth interpolation 25.12 two's complement fixed point value
0x1e0	Stencil.mode	32	G	R/W	stencil test parameters

Table 7-3 Pixel Pipeline register address map

Offset	Register	Size	Type	R/W	Description
0x1e8	Stencil.mask	32	G	R/W	stencil buffer bit mask
0x1f0	PixPipeNull	32	NG	W	null register
0x1f8	PixPipeFlush	32	G	W	pipeline flush command

7.3.1.4 BufMode Registers

Table 7-4 BufMode.src register format

Bits	Name	Description
31:13	**reserved**	
12:10	**bufType**	Indicates whether the source buffer is a tiled or linear buffer and which TLB maps the source buffer. 0: tiled buffer - framebuffer TLB A 1: tiled buffer - framebuffer TLB B 2: tiled buffer - framebuffer TLB C 3: reserved 4: linear buffer - linear TLB A 5: linear buffer - linear TLB B 6: reserved
9:8	**bufDepth**	Indicates the source buffer word depth, which must be greater than or equal to the source buffer pixel depth.. 0: 8-bit buffer 1: 16-bit buffer 2: 32-bit buffer
7:4	**pixType**	Indicates the source buffer pixel format. 0: CI 1: RGB 2: RGBA 3: ABGR 4-14: reserved 15: YCrCb
3:2	**pixDepth**	Indicates the source buffer pixel depth, which must be less than or equal to the source buffer word depth.. 0: 8-bit 1: 16-bit 2: 32-bit
1	**doublePix**	Indicates that each word of the source buffer is split into a front and back half for double buffering
0	**doublePixSel**	If doublePix is set, this bit indicates which half of the buffer word is selected for drawing.

Table 7-5 BufMode.dst register format

Bits	Name	Description
31:13	**reserved**	
12:10	**bufType**	Indicates whether the source buffer is a tiled or linear buffer and which TLB maps the source buffer. 0: tiled buffer - framebuffer TLB A 1: tiled buffer - framebuffer TLB B 2: tiled buffer - framebuffer TLB C 3: reserved 4: linear buffer - linear TLB A 5: linear buffer - linear TLB B 6: reserved
9:8	**bufDepth**	Indicates the buffer word depth. 0: 8-bit buffer 1: 16-bit buffer 2: 32-bit buffer
7:4	**pixType**	Indicates the source or destination pixel format. 0: CI 1: RGB 2: RGBA 3: ABGR 4-14: reserved 15: YCrCb (BufMode.src only)
3:2	**pixDepth**	Pixel depth. 0: 8-bit 1: 16-bit 2: 32-bit
1	**doublePix**	Indicates that each word of the color buffer is split into a front and back half for double buffering
0	**doublePixSel**	If doublePix is set, this bit indicates which half of the buffer word is selected for drawing.

7.3.1.5 ClipMode Register

Table 7-6 ClipMode register format

Bits	Name	Description
31:12	**reserved**	
11	**enCid**	0/1: disable/enable clip ID testing
10	**cidMapSel**	0/1: disable/enable clip ID testing
9	**enScrMask0**	0/1: disable/enable screen mask 0 testing
8	**enScrMask1**	0/1: disable/enable screen mask 1 testing
7	**enScrMask2**	0/1: disable/enable screen mask 2 testing
6	**enScrMask3**	0/1: disable/enable screen mask 3 testing
5	**enScrMask4**	0/1: disable/enable screen mask 4 testing
4	**scrMaskMode0**	0: pass only if pixel is outside screen mask 0 1: pass only if pixel is inside screen mask 0
3	**scrMaskMode1**	0: pass only if pixel is outside screen mask 1 1: pass only if pixel is inside screen mask 1
2	**scrMaskMode2**	0: pass only if pixel is outside screen mask 2 1: pass only if pixel is inside screen mask 2
1	**scrMaskMode3**	0: pass only if pixel is outside screen mask 3 1: pass only if pixel is inside screen mask 3
0	**scrMaskMode4**	0: pass only if pixel is outside screen mask 4 1: pass only if pixel is inside screen mask 4

7.3.1.6 DrawMode Register

Table 7-7 DrawMode register format

Bits	Name	Description
31:24	**reserved**	
23	**enNoConflict**	0/1: enable/disable read/write coherency conflict checking
22	**enGL**	0: disable GL mode / enable X mode 1: enable GL mode / disable X mode
21	**enPixelXfer**	0/1: disable/enable pixel pipeline transfers
20	**enScissorTest**	0/1: disable/enable scissor testing
19	**enLineStipple**	0/1: disable/enable line stippling
18	**enPolyStipple**	0/1: disable/enable polygon stippling
17	**enOpaqStipple**	0/1: disable/enable opaque stippling
16	**enShade**	0/1: flatshade/smooth shade
15	**enTexture**	0/1: disable/enable texture mapping
14	**enFog**	0/1: disable/enable fog
13	**enCoverage**	0/1: disable/enable coverage application for points and rectangles
12	**enAntialiasLine**	0/1: disable/enable line antialiasing
11	**enAlphaTest**	0/1: disable/enable alpha test
10	**enBlend**	0/1: disable/enable alpha blending
9	**enDither**	0/1: disable/enable RGB dithering
8	**enLogicOp**	0/1: disable/enable logic ops
7	**enColorMask**	0/1: disable/enable color buffer bit masking
6:3	**enColorByteMask**	0/1: disable/enable color buffer byte masking
2	**enDepthTest**	0/1: disable/enable depth test
1	**enDepthMask**	0/1: disable/enable depth buffer masking
0	**enStencilTest**	0/1: disable/enable stencil test

7.3.1.7 Primitive Register

Table 7-8 Primitive register format

Bits	Name	Description
31:24	**opCode**	Indicates type of geometric primitive to be rendered through the pixel pipeline 0: point 1: line 2: triangle 3: rectangle 4-255: reserved
23:19	**reserved**	
18	**lineSkipLastEP**	Indicates whether the ending vertex pixel of an X line should be rendered or skipped 0: render line endpoint pixel 1: skip line endpoint pixel
17:16	**edgeType**	Indicates the triangle or rectangle traversal direction during rasterization 0: traverse left to right, bottom to top 1: traverse right to left, bottom to top 2: traverse left to right, top to bottom 3: traverse right to left, top to bottom
15:0	**lineWidth**	Indicates the subpixel line width/2

7.3.1.8 WinOffset Register

Table 7-9 WinOffset register format

Bits	Name	Description
31:16	**x**	x-coordinate offset for translation from window-relative to framebuffer-relative coordinates

Table 7-9 WinOffset register format

Bits	Name	Description
15:0	**y**	x-coordinate offset for translation from window-relative to framebuffer-relative coordinates

7.3.1.9 Scissor Register

Table 7-10 Scissor register format

Bits	Name	Description
63:48	**min.x**	Window x-coordinate of left inclusive edge of scissor rectangle
47:32	**min.y**	Window y-coordinate of top inclusive edge of scissor rectangle
31:16	**max.x**	Window x-coordinate of right exclusive edge of scissor rectangle
15:0	**max.y**	Window y-coordinate of bottom exclusive edge of scissor rectangle

7.3.1.10 ScrMask Registers

Table 7-11 ScrMask register format

Bits	Name	Description
63:48	**min.x**	Framebuffer x-coordinate of left inclusive edge of scissor rectangle
47:32	**min.y**	Framebuffer y-coordinate of top inclusive edge of scissor rectangle
31:16	**max.x**	Framebuffer x-coordinate of right exclusive edge of scissor rectangle
15:0	**max.y**	Framebuffer y-coordinate of bottom exclusive edge of scissor rectangle

7.3.1.11 Stipple.mode Register

Table 7-12 Stipple.mode register format

Bits	Name	Description
31:29	**reserved**	
28:24	**index**	starting 32-bit stipple pattern index [0-31]
23:20	**reserved**	
20:16	**maxIndex**	maximum 32-bit stipple pattern index [0-31]
15:8	**repeatCnt**	pattern index repeat count [0-255]
7:0	**maxRepeat**	maximum pattern index repeat count [0-255]

7.3.1.12 Shade Registers

Bits	Name	Description

7.3.1.13Texture Registers

Table 7-13 Texture.mode register

Bits	Name	Description
31:25	**reserved**	
24	**tiled**	Indicates that the texture map is in the subtiled format 0: not subtiled texture map 1: subtiled texture map
23:20	**texelType**	Indicates type of texel 0: reserved 1: RGB 2: RGBA 3: reserved 4: RGBA4 5: ALPHA 6: INTENSITY 7: LUMINANCE 8: LUMINANCE_ALPHA
19:18	**texelDepth**	Indicates texel depth 0: reserved 1: 16-bit texel 2: 32-bit texel
17:14	**mapLevel**	Indicates base texture map level
13:10	**maxLevel**	Indicates maximum mipmap level
9:7	**minFilter**	texture minification filter mode 0: NEAREST 1: LINEAR 2: NEAREST_MIPMAP_NEAREST 3: LINEAR_MIPMAP_NEAREST 4: NEAREST_MIPMAP_LINEAR 5: LINEAR_MIPMAP_LINEAR
6	**magFilter**	Indicates texture magnification filter mode 0: NEAREST 1: LINEAR

Table 7-13 Texture.mode register

Bits	Name	Description
5:4	**wrapS**	Indicates wrap mode for s coordinate 0: CLAMP 1: REPEAT 2: CLAMP_TO_EDGE 3: CLAMP_TO_BORDER
3:2	**wrapT**	Indicates wrap mode for t coordinate 0: CLAMP 1: REPEAT 2: CLAMP_TO_EDGE 3: CLAMP_TO_BORDER
1:0	**func**	Indicates texture application function 0: MODULATE 1: DECAL 2: BLEND 3: REPLACE

Table 7-14 Texture.format register

Bits	Name	Description
63:48	**reserved**	
47:44	**uShift**	
43:40	**vShift**	
39:36	**uWrapShift**	
35:32	**vWrapShift**	
31:16	**uIndexMask**	
15:0	**vIndexMask**	

7.3.1.14Antialias Registers

Bits	Name	Description
31:16	**slope**	slope of the line as a 2.14 two's complement fixed point value
15:0	**ideal**	ideal line starting minor coordinate as a 2.14 two's complement fixed point value

7.3.1.15AlphaTest Register

Bits	Name	Description
31:12	**reserved**	
11:8	**func**	alpha test function: 0: NEVER 1: LESS 2: EQUAL 3: LEQUAL 4: GREATER 5: NOTEQUAL 6: GEQUAL 7: ALWAYS
7:0	**ref**	alpha test comparison reference value

7.3.1.16Blend Registers

Bits	Name	Description
31:12	**reserved**	

Bits	Name	Description
11:8	**op**	blend op: 0: ADD 1: MIN 2: MAX 3: SUBTRACT 4: REVERSE_SUBTRACT
7:4	**src**	source blend function: 0: ZERO 1: ONE 2: DST_COLOR 3: ONE_MINUS_DST_COLOR 4: SRC_ALPHA 5: ONE_MINUS_SRC_ALPHA 6: DST_ALPHA 7: ONE_MINUS_DST_ALPHA 8: CONSTANT_COLOR 9: ONE_MINUS_CONSTANT_COLOR 10: CONSTANT_ALPHA 11: ONE_MINUS_CONSTANT_ALPHA 12: SRC_ALPHA_SATURATE
3:0	**dst**	destination blend function: 0: ZERO 1: ONE 2: SRC_COLOR 3: ONE_MINUS_SRC_COLOR 4: SRC_ALPHA 5: ONE_MINUS_SRC_ALPHA 6: DST_ALPHA 7: ONE_MINUS_DST_ALPHA 8: CONSTANT_COLOR 9: ONE_MINUS_CONSTANT_COLOR 10: CONSTANT_ALPHA 11: ONE_MINUS_CONSTANT_ALPHA

7.3.1.17LogicOp Register

Bits	Name	Description

7.3.1.18Depth Registers

Bits	Name	Description

7.3.1.19Stencil Registers

Bits	Name	Description

Bits	Name	Description

7.3.2 Memory Transfer Engine Registers

Table 7-15 MTE Registers

Offset	Register	Size	Type	R/W	Description
0x00	MTE.mode	32	G	R/W	MTE transfer mode
0x08	MTE.byteMask	32	G	R/W	32-bit byte mask
0x10	MTE.stippleMask	32	NG	W	32-bit stipple mask
0x18	MTE.fgValue	32	G	R/W	clear value
0x20	MTE.src0	32	G	R/W	source buffer framebuffer or linear start address
0x28	MTE.src1	32	G	R/W	source buffer framebuffer or linear start address
0x30	MTE.dst0	32	NG	W	destination bufferframebuffer or linear start address
0x38	MTE.dst1	32	NG	W	destination buffer framebuffer or linear end address

Table 7-15 MTE Registers

Offset	Register	Size	Type	R/W	Description
0x40	MTE.srcYStep	32	G	R/W	source framebuffer y stride value
0x48	MTE.dstYStep	32	G	R/W	destination framebuffer y stride value
0x70	MTE.null	32	NG	W	null register
0x78	MTE.flush	32	G	W	MTE flush command

7.3.2.1 MTE.mode Register

Bits	Name	Description
31:12	**reserved**	
11	**opCode**	Indicates MTE operation: 0: clear destination buffer 1: copy from source buffer to destination buffer
10	**enStipple**	0/1: enable/disable stipple pixel mask application during clear operations
9:8	**pixDepth**	Pixel depth used 0: 8-bit 1: 16-bit 2: 32-bit 3: reserved
7:5	**srcBufType**	Indicates whether the source buffer is a tiled or linear buffer and which TLB maps the source buffer. 0: tiled buffer - framebuffer TLB A 1: tiled buffer - framebuffer TLB B 2: tiled buffer - framebuffer TLB C 3: tiled buffer - texture TLB 4: linear buffer - linear TLB A 5: linear buffer - linear TLB B 6: tiled buffer - clipID TLB 7: reserved

Bits	Name	Description
4:2	**dstBufType**	Indicates whether the destination buffer is a tiled or linear buffer and which TLB maps the destination buffer. 0: tiled buffer - framebuffer TLB A 1: tiled buffer - framebuffer TLB B 2: tiled buffer - framebuffer TLB C 3: tiled buffer - texture TLB 4: linear buffer - linear TLB A 5: linear buffer - linear TLB B 6: tiled buffer - clipID TLB 7: reserved
1	**srcECC**	perform ECC when reading from source buffer
0	**dstECC**	perform ECC when writing to destination buffer

7.3.3 Host Interface

7.3.3.1 Interface Buffer

To balance the host register write issue rate with the rate at which the rendering engine can retire register writes, in which the host issues writes faster than the rendering engine retires them, host writes to rendering engine registers are stored in a ring buffer called the *host interface buffer*. read out of the interface buffer in FIFO order.

It is important that the interface buffer not overflow when the host issues writes faster than the rendering engine can retire them. The host can avoid overflows by examining the interface buffer FIFO level indicated in the intfBufLevel field Status register to determine if the buffer has sufficient space to hold the parameters for the next set of primitives. However, the performance overhead of this check may be prohibitive for some primitives. For this reason, the host interface supports a mechanism to avoid buffer overflows which requires less monitoring by the host. First, if the number of entries in the buffer exceeds that indicated in the *stallLevel* field of the IntfBufCtl register, the host interface logic stalls host writes for the number of 66MHz cycles indicated in the *stallCycle* field of the IntfBufCtl register. If the level rises beyond that indicated in the *intLevel* field of the IntfBufCtl register, an interrupt is issued to the

host. The interrupt handler must then either wait for the level to fall to some acceptable level by polling the Status register or enable the interface buffer empty interrupt.

The host indicates to the rendering engine that it has completed loading the pertinent parameters for executing a primitive by offsetting its last register write by START_OFFSET bytes. Subsequent writes from the host do not affect the execution of committed primitives. All primitives execute in the order they are committed and execute atomically, i.e. they run to completion and cannot be interrupted.

Therefore, parameters which typcially do not change across multiple primitives would typically not need to be re-loaded per primitive.

7.3.3.2 Context Switching

7.3.4 Pixel Buffer Allocation

In Moosehead, a framebuffer comprises a set of rectangular pixel buffers statically or dynamically allocated in system memory. These pixel buffers may include multiple color buffers, a depth buffer, and a stencil buffer. The rendering engine has no inherent notion of whether a color buffer is on-screen or off-screen. A framebuffer may be shared by multiple windows or may be allocated per window or pixmap. The rendering engine supports rendering to framebuffers of up to 2048x2048 pixels.

7.3.4.1 Framebuffer Tiling

Pixel buffers are partitioned into and allocated as 64kb *tiles* which may be allocated non-contiguously throughout system memory. These tiles are arranged as arrays of 128x128x32-bit pixels, 256x128x16-bit pixels or 512x128x8-bit pixels packed into a physically contiguous 64kb address space. Tiles must be begin on 64kb aligned address. Pixel buffers must be made up of an integral number of tiles. For example, a 200x200 pixel buffer would require four 128x128 pixel tiles. Figure 4 illustrates framebuffer tiling.

There are two motivations for framebuffer tiling. One is that tiles provide a convenient unit for framebuffer memory allocation. By allowing tiles to be scattered throughout memory, tiling makes the amount of memory which must be contigously allocated manageable. Additionally, tiling provides a means of reducing the amount of system memory consumed by framebuffers.

For example, a 1024x1024 virtual framebuffer consisting of front and back RGBA buffers and a depth buffer would consume 12Mb of memory if fully resident. However, if each 1024x1024 buffer were partitioned into 64 128x128 tiles of which only four tiles contained non-occluded pixels, only memory for those visible tiles would need to be allocated. In this case, only 3Mb would be consumed.

Figure 7-4 Framebuffer tiling

7.3.4.2 Framebuffer Address Translation

Since framebuffer tiles may be scattered throughout memory, the RE supports a framebuffer address translation buffer (TLB) to translate framebuffer (x,y) addresses into physical memory addresses. This TLB is loaded by the kernel with the base physical memory addresses of the tiles which compose a color buffer and the stencil-depth buffer of a framebuffer.

The framebuffer TLB must have enough entries to hold the tile base physical memory addresses of a 2048x2048 pixel color buffer and a 2048x2048 pixel stencil-depth buffer. Therefore, the TLB has 256 entries for color buffer tiles and 256 entries for stencil-depth buffer tiles.

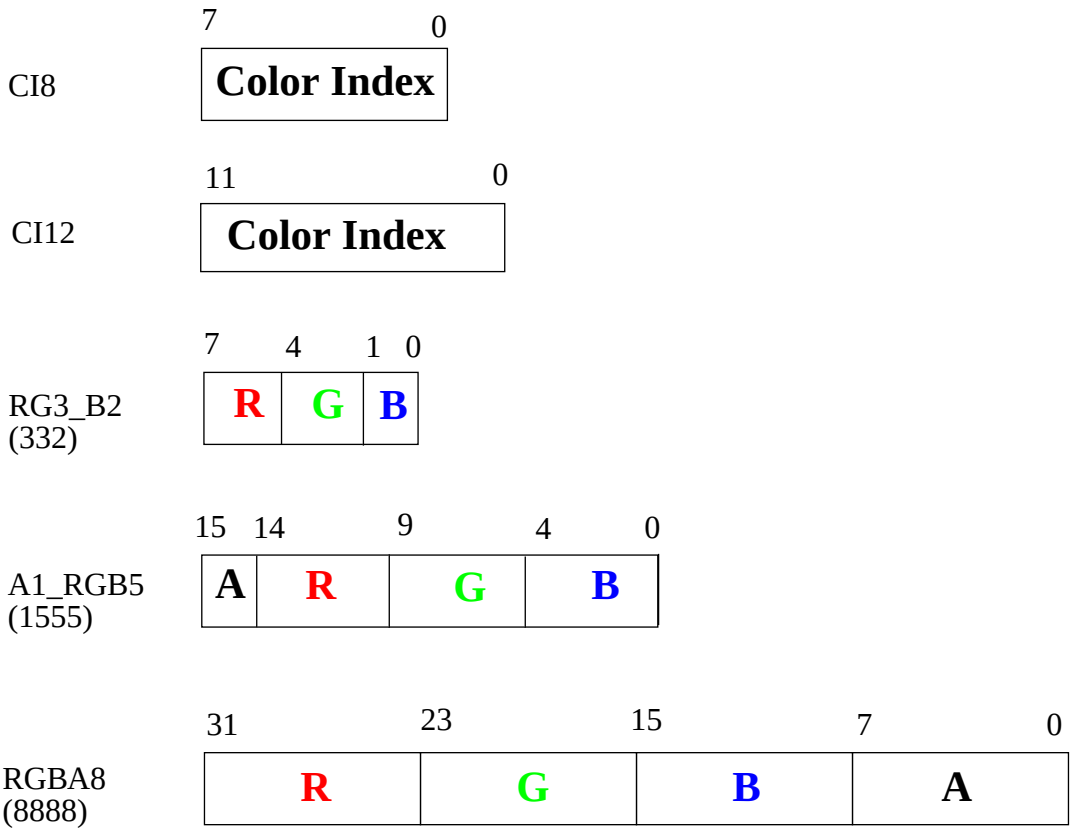
TBD

7.3.4.3 Color Buffers and Pixels

The rendering engine supports rendering to color buffers of depth 8, 16, or 32-bits. Although a framebuffer may comprise multiple color buffers (front, back, left, right, auxiliary, overlay), the rendering engine has no inherent notion of the type of the color buffer to which it is rendering.

The rendering supports 8-bit and 12-bit color index (CI) pixels and 8-bit, 16-bit and 32-bit RGB(A) pixels. The possible constants specifying the color pixel formats and their corresponding color pixel formats are shown in Figure 5.

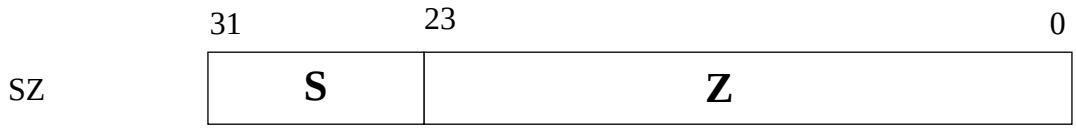
Figure 7-5 Color pixel formats



7.3.4.4 Stencil-Depth Pixels and Buffers

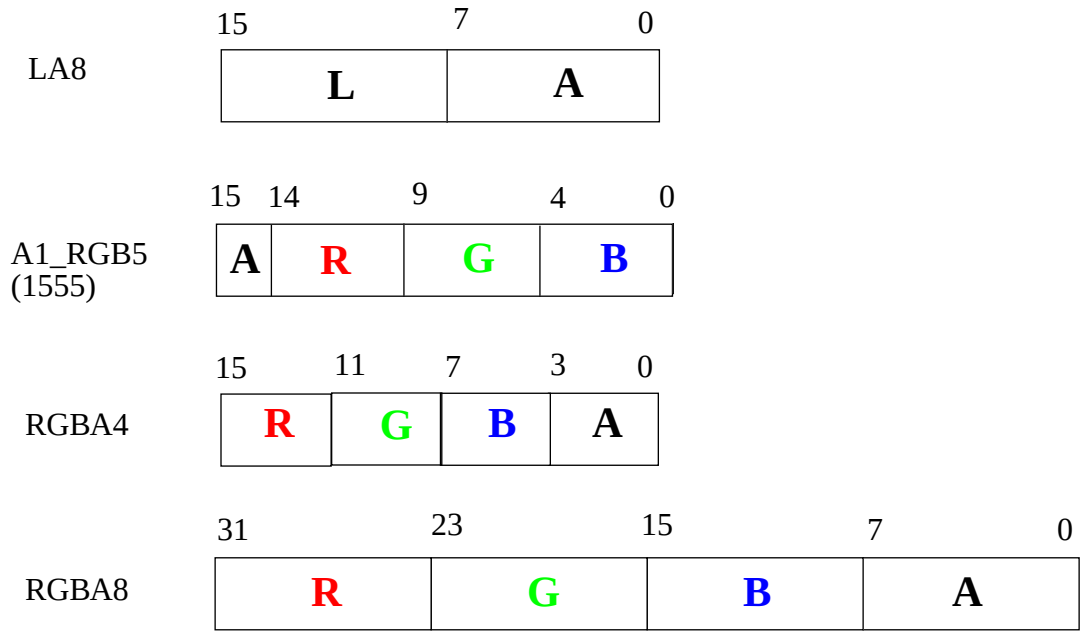
The rendering engine supports 8-bit stencil pixels and 24-bit depth pixels, represented by S and Z, respectively. S and Z pixels are packed into a 32-bit SZ pixel as shown in Figure 6.

Figure 7-6 SZ pixel format



7.3.4.5 Texels and Texture Maps

Figure 7-7 Texel formats



7.3.5 Pixel Rasterization Pipeline

7.3.5.1 Rasterize

Point, line, triangle and rectangle primitives are efficiently and rigorously point sampled and traversed by the rasterize unit. This unit iterating and evaluating a set of linear *edge functions* defined by the primitive. The rasterize unit traverses all primitives in unit steps in either the x or y direction and after each step issues *stepping* commands which indicate traversal information to the other stages of the pixel pipeline.

7.3.5.2 Edge Functions

This section presents the arithmetic the rendering engine performs to point sample and traverse a primitive. We begin with the equation of the line defined by the points (x_0,y_0) and (x_1,y_1):

$$y = dy/dx * x + b \quad \text{(EQ 1)}$$

where $dx = (x_1-x_0)$ and $dy = (y_1-y_0)$. We can re-write (EQ 1) in the form:

$$E(x,y) = A*x + B*y + C = 0 \quad \text{(EQ 2)}$$

where $A = dy$, $B = -dx$, $C = b*dx$. We refer to E(x,y) as an *edge function*. By making the following substitutions

$x = x_s + m$

$y = y_s + n$

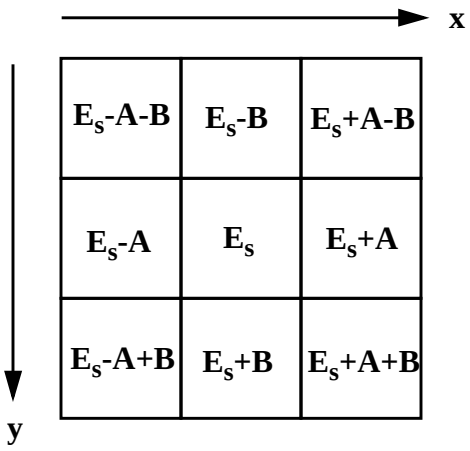
$E_s = E(x_s,y_s)$ where (x_s,y_s) is some arbitrary starting point

the edge function can be re-written in the form:

$$E(x,y) = E(x_s + m, y_s + n) = E_s + A*m + B*n \quad \text{(EQ 3)}$$

It is clear from this form of the edge function equation that after computing E_s, only addition/subtraction of A or B is required to evaluate E(x,y) after each unit step in the +/- x or y direction, respectively. This *edge function arithmetic* is illustrated in Figure 8.

Figure 7-8 Edge function arithmetic



To determine E_s, we first recall that $E(x_0,y_0) = 0$ since (x_0,y_0) is a point on the line. Then, we make the following substitutions:

$$E_s = E(x_s,y_s) =$$

$$E(x_0 + (x_s-x_0), y_0 + (y_s-y_0)) =$$

$$E(x_0,y_0) + A*(x_s-x_0) + B*(y_s-y_0) =$$

$$A*(x_s-x_0) + B*(y_s-y_0)$$

So, we find that

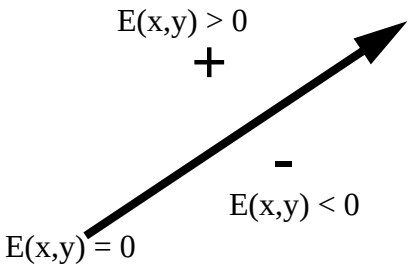
$$E_s = A*(x_s-x_0) + B*(y_s-y_0) \quad \text{(EQ 4)}$$

TBD

7.3.6 Line and Triangle Scan Conversion

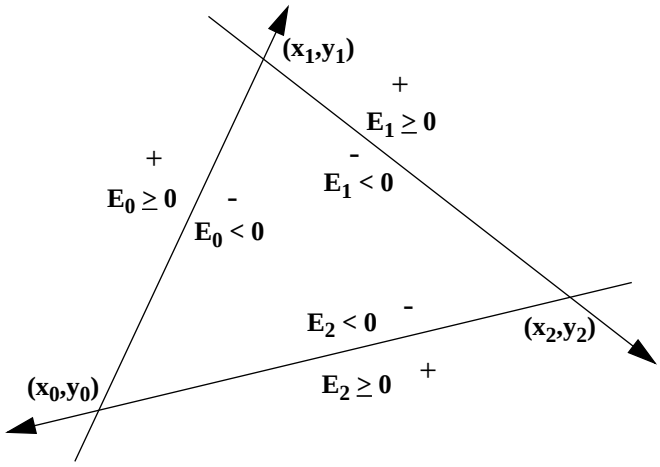
This section explains how the edge functions are used to point sample and traverse the geometric primitive. First, we note that a line divides a plane into two *half-planes*. If E(x,y) represents the edge function of the line, then it can be shown that E(x,y) > 0 for all points in one half-plane which we label **+**, and E(x,y) < 0 for all points in the other half-plane which we label **-**. For convenience, we define that points on the line E(x,y) = 0 belong to the **+** half-plane. Therefore, simply by evaluating the sign of the edge function E(x,y) of a particular line at any point (x,y), we can exactly determine on which side of the line that point resides.

Figure 7-9 Half-planes



Each edge of a polygon defines a line and so each edge divides a plane into its own + and - half-planes. If the edges of an N-sided convex polygon are directed clock-wise around the interior of the polygon, then the intersection of all N of the - half-planes is exactly the set of points which are interior or on the edges of the polygon. We refer to this set of points in the + half-planes as being *inside* the polygon. Therefore, if $E_i(x,y)$ represents the edge function at (x,y) for edge i of an N-sided convex polygon, then (x,y) is exactly inside the polygon if and only if $E_i(x,y) >= 0$ for i = 0 to N-1. This is illustrated in Figure 10.

Figure 7-10 Triangle repesented as the intersection of three half-planes



Once we have computed the value of the edge functions at any point (x,y), we can determine if points at unit steps in the +/- x or y directions from (x,y) are inside the polygon by using edge function arithmetic to compute the edge function values and then evaluating their signs. Therefore, by traversing the polygon in unit steps in the +/- x or y direction, we can use edge function arithmetic to rigorously point sample the polygon.

TBD

Figure 7-11 Primitive traversal using edge functions

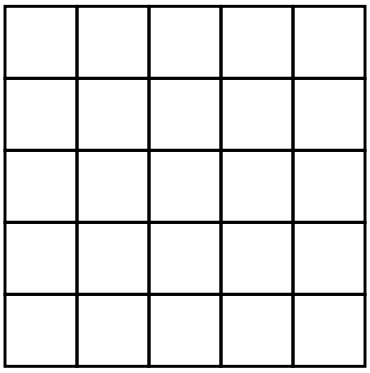
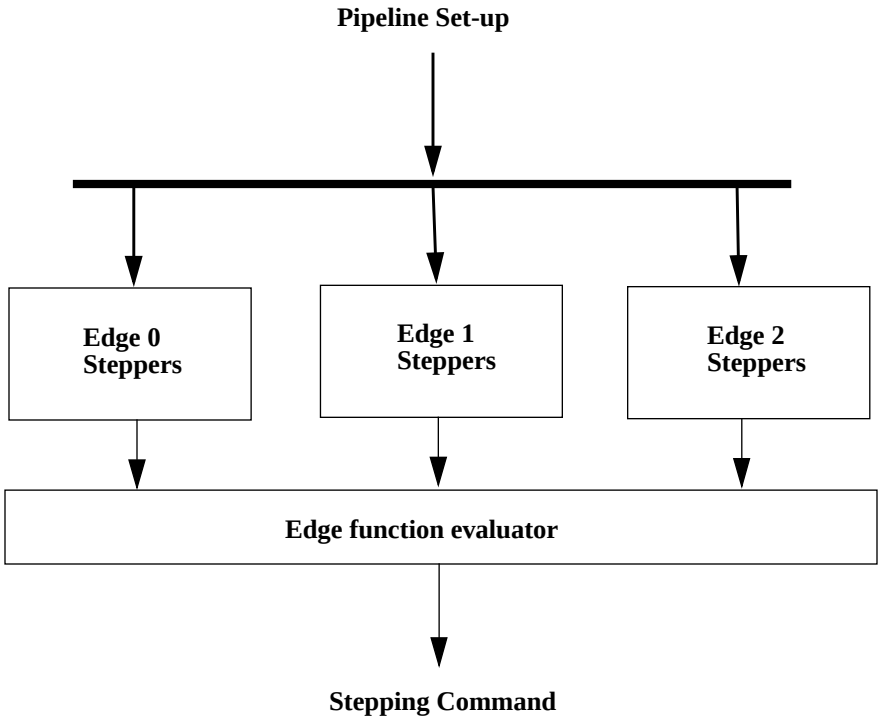


Figure 7-12 Line and triangle rasterizer



7.3.6.1 Geometric Primitives

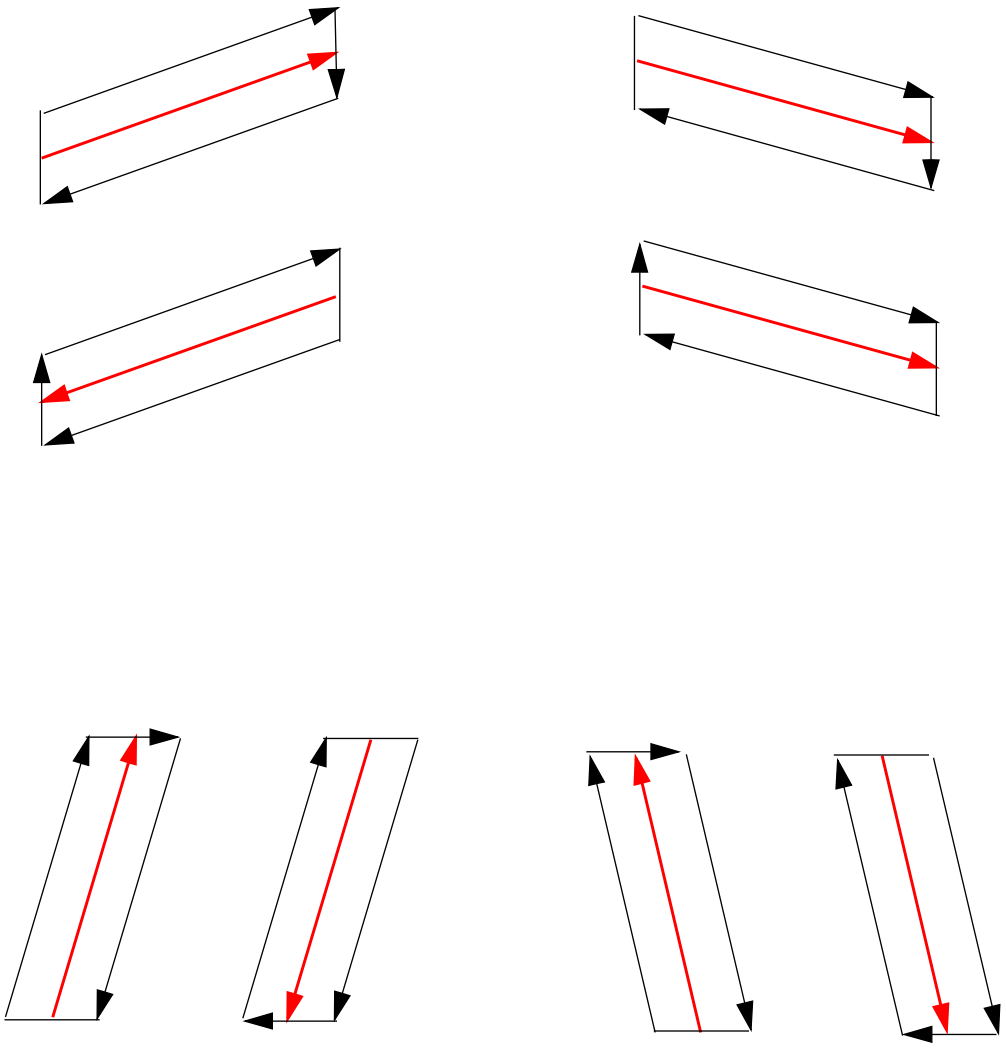
TBD

The pixel pipeline rasterizes point, line, triangle, and rectangle primitives. The type of primitive to be rasterized is indicated through the opCode field of the Primitive register. The possible constants specifying which primitive to rasterize are OPCODE_POINT, OPCODE_LINE, OPCODE_TRI, and OPCODE_RECT corresponding to point, line, triangle and rectangle primitives, respectively.

Geometric primitives are specified to the pixel pipeline by their (x,y) vertices. Primitives are rasterized differently depending on whether the rendering engine is in X mode or OpenGL mode.

1. Point Primitive
2. Line Primitive

Figure 7-13 Line traversal



3. Triangle Primitive

4. Rectangle Primitive

(x,y) window coordinates of the primitive into the **Rasterize.vertexX** registers for X primitives or the **Rasterize.vertexGL** registers for OpenGL primitives.

In X mode, the host specifies coordinates as 16-bit unsigned values in the range [0, 2^{16}). In GL mode, the host specifies coordinates as 13.4 unsigned fixed point values in the range [0,2^{13}).

In both X and GL mode, coordinate (0,0) corresponds to the upper left corner of the framebuffer. This is consistent with its location in the X Windows coordinate system but not with the OpenGL coordinate system, which places coordinate (0,0) at the lower left corner of the framebuffer. Therefore, the OpenGL viewport matrix must be conditioned to transform vertices to the X Windows coordinate system. Pixels are assumed to be square and are sampled at their upper left corner.

7.3.7 Plane Equation Arithmetic

This section presents the arithmetic that performed in various stages of the pixel pipeline to linearly interpolate a fragment parameter z(x,y) of a primitive along the plane defined by the primitive's (x,y,z) vertex coordinates. The parameter z may represent R, G, B, or A color components, homogeneous texture coordinates, the fog blending factor, or the depth value. We begin with the *plane equation*

$$z(x,y) = A*x + B*y + C \qquad \text{(EQ 5)}$$

By making the following substitutions

$x = Xs + m$

$y = Ys + n$

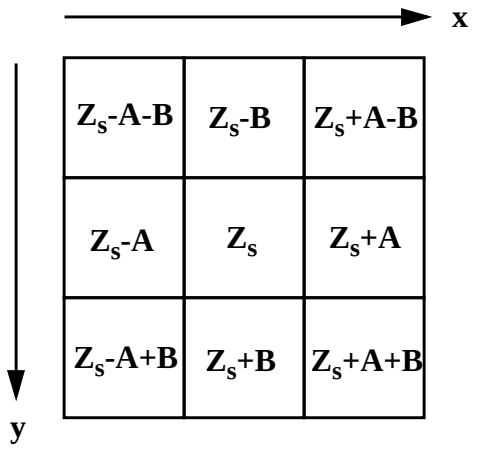
$Zs = z(Xs,Ys)$ where (Xs,Ys) is some arbitrary starting point

the plane equation can be re-written in the form

$$z(Xs + m, Ys + n) = Zs + A*m + B*n \qquad \text{(EQ 6)}$$

It is clear from this form of the plane equation that given Zs, only addition/subtraction of A (dz/dx) or B (dz/dy) is required to evaluate z(x,y) after each unit step in the +/- x or y direction, respectively. This *plane equation arithmetic* is illustrated in Figure 14. Note that plane equation arithmetic has the same form as the edge function arithmetic presented earlier.

Figure 7-14 Plane equation arithmetic



The functional unit that performs plane equation arithmetic is a *stepper.* Steppers are essentially adder/subtractor units which respond to stepping commands from the **Rasterize** stage.

In order for the rendering engine to perform plane equation arithmetic, the host must compute and load the plane equation parameters A, B and Zs. A and B can be determined by solving the following system of simultaneous equations:

z0 = z(x0,y0) = A*x0 + B*y0 + C

z1 = z(x1,y1) = A*x1 + B*y1 + C

z2 = z(x2,y2) = A*x2 + B*y2 + C

where (x0,y0,z0), (x1,y1,z1), and (x2,y2,z2) are the vertex coordinates of the primitive.

The solution yields:

$$A = ((z1-z0)*(y1-y0) - (z2-z1)*(y2-y1)) / t \quad \text{(EQ 7)}$$

$$B = ((z1-z0)*(x1-x0) - (z2-z1)*(x2-x1)) / t \quad \text{(EQ 8)}$$

where

$$t = ((y1-y0)*(x2-x1) - (y2-y1)*(x1-x0)) \quad \text{(EQ 9)}$$

Zs can be determined as follows:

Zs = z(Xs,Ys) =

z(x0 + (Xs-x0), y0 + (Ys-y0)) =

z(x0,y0) + A*(Xs-x0) + B*(Ys-y0) =

z0 + A*(Xs-x0) + B*(Ys-y0)

So, we find that

$$Zs = z0 + A*(Xs-x0) + B*(Ys-y0) \quad \text{(EQ 10)}$$

7.3.7.1 Flatshading and Smooth Shading

At the start of the pixel rasterization pipeline, each fragment of the primitive is assigned a color index or RGB(A) value. Primitives are either *flatshaded* or *smooth shaded* (or Gouraud shaded).Triangle and line primitives are flatshaded if DrawMode.enShade = 0 and smooth shaded otherwise. Point and rectangle primitives are always flatshaded.

1. Flatshading:

When flatshading, the value in the Shade.fgColor register is assigned to all fragments of the primitive. For color index flatshading, the Shade.fgColor register should be loaded with the least significant bit-aligned color index flatshade value. If an 8-bit pixel depth is indicated through the BufMode.dst.pixelDepth field, then a CI8 value should be loaded into Shade.fgColor; if a pixel depth of 16-bits or 32-bits is indicated, then a CI12 value should be loaded into Shade.fgColor. For RGBA flatshading, an RGBA8 value should be loaded into the Shade.fgColor register, regardless of pixel depth.

2. Smooth Shading:

When smooth shading, the color value assigned to the fragment is determined by linearly interpolating the color index or R, G, B, and A values using the plane equation arithmetic described in Section 7.3.7. Therefore, smooth shading requires the host to compute the plane equation arithmetic parameters c_s, dc/dx, and dc/dy, where c represents color index or R,G,B,A values to be interpolated.

The smooth shade R,G,B,A values are iterated in parallel using four 21-bit two's complement fixed point steppers. The host should load the R,G,B,A plane equation registers with 9.12 two's complement values. The 12-bits of fraction are included to maintain precision across a 2k x 2k pixel address range. The 8-bit R,G,B,A values assigned to the frag-

ment are generated by truncating the fractional part of the iterated 9.12 values and then clamping the resulting 9-bits to the 8-bit range [0,255]. The clamping hardware evaluates bits 7 and 8 of the 9-bit truncated value to determine whether the color has underflowed ([8:7] = ‘11’) or overflowed ([8:7] = ‘10’). This works under the reasonable assumption that iterated values of fragments *inside* the primitive will not overflow or underflow by more than 127.

Color index smooth shading values are iterated on the alpha component stepper. For 12-bit color index iteration, the host should load the alpha component plane equation registers with 12.8 two’s complement fixed point values. For 8-bit color index iteration, the host should load the alpha component plane equation registers with 8.12 two’s complement fixed point values. The color index value assigned to the fragment are generated by simply truncating the fractional part of the iterated value. In accordance with OpenGL, color index values are not clamped prior to being assigned to the fragment.

As stated, the fractional portion of the iterated smooth shade values are truncated, not rounded. However, by adding 0.5 to each initial color value, the host can ensure that the hardware will assign properly rounded color values.

TABLE 16.

Shade.r	9.12	r_s
Shade.g	9.12	g_s
Shade.b	9.12	b_s
Shade.a	9.12	a_s
Shade.drdx	9.12	dr/dx
Shade.drdy	9.12	dr/dy
Shade.dgdx	9.12	dg/dx
Shade.dgdy	9.12	dg/dy
Shade.dbdx	9.12	db/dx
Shade.dbdy	9.12	db/dy
Shade.dadx	9.12	da/dx
Shade.dady	9.12	da/dy

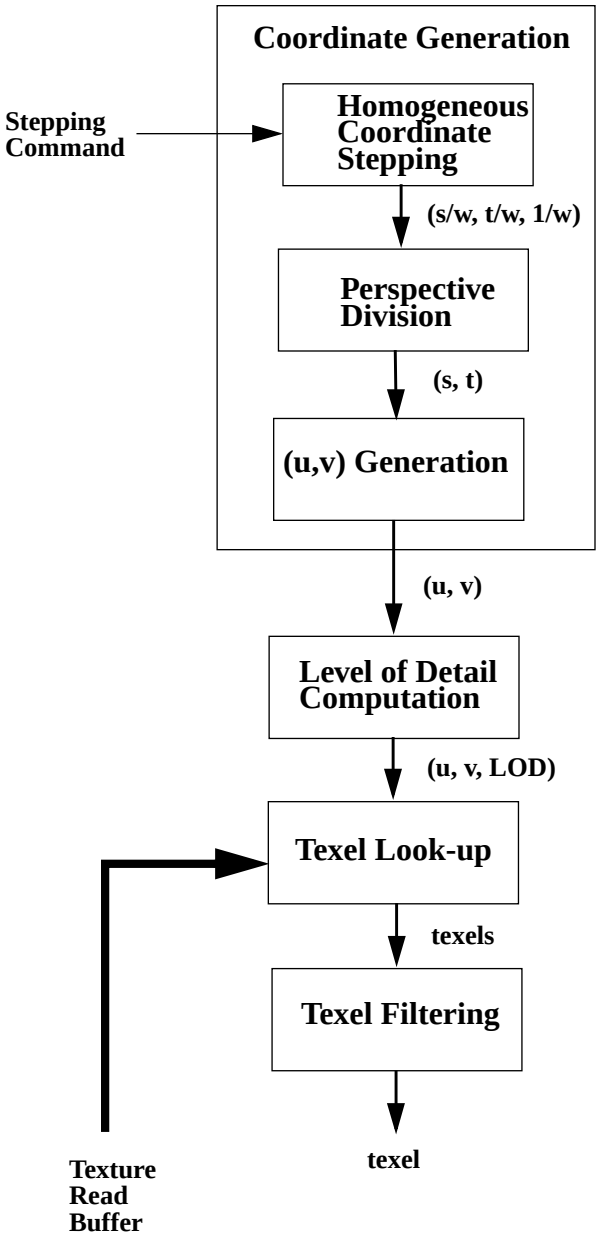
7.3.7.2 Texture Mapping

The rendering engine supports OpenGL conformant 1-D and 2-D mip-mapped texture mapping. In particular, it supports the following texture features:

- 1kx1k to 1x1 texel texture maps
- 16-bit and 32-bit texel formats
- Clamped and repeating textures
- Nearest and linear mipmapped and non-mipmapped filters
- Dynamic texture map level of detail computation
- OpenGL texture application functions

7.3.7.3 Texel Generation

Figure 7-15 Texel generation pipeline



TBD

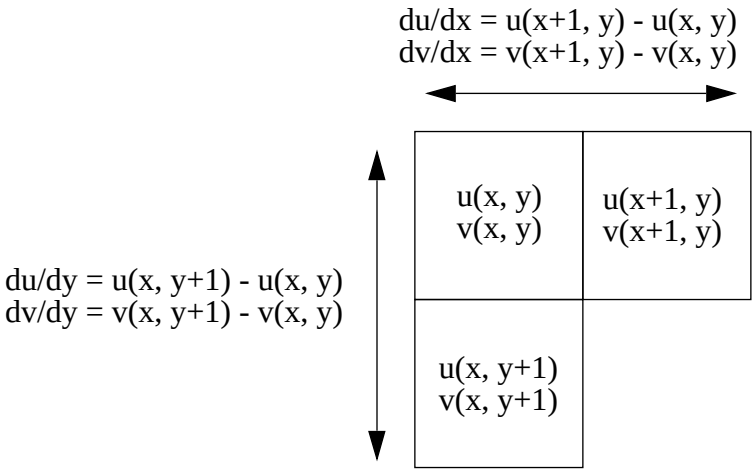
7.3.7.4 Coordinate Generation

Table 7-17 Texture Coordinate Generation

TexGen.sq	36.12	sq_s
TexGen.tq	36.12	tq_s
TexGen.q	18.12	q_s
TexGen.dsqdx	36.12	dsq/dx
TexGen.dszdy	36.12	dsq/dy
TexGen.dtqdx	36.12	dtq/dx
TexGen.dtqdy	36.12	dtq/dy
TexGen.dqdx	18.12	dq/dx
TexGen.dqdy	18.12	dq/dy

7.3.7.5 Mip-Map Level of Detail

Figure 7-16 Mip-map level of detail computation



$rho = max(du/dx, dv/dx, du/dy, dv/dy)$

$LOD = \log_2(rho)$

7.3.7.6 Texel Look-up

TBD

7.3.7.7 Minification and Magnification Filtering

TBD

7.3.7.8 Texture Application

If texture application is enabled, the fragment's RGBA color assigned during shading is modulated or blended with the generated texel's color. It applies only for RGB(A) pixels. Texture application is enabled or disabled by setting the *enTexture* field of the **DrawMode** register to ENABLE or DISABLE, respectively.

7.3.7.9 Fogging

If fog is enabled, the fragment's RGB color after texture application is modulated using a linearly interpolated fog factor. Fog blends the fragment's R, G, and B components with the corresponding components of the fog color according to the formula

$$f*(C_r-F_c) + F_c \quad \text{(EQ 11)}$$

where f is a blending factor in the range [0,1], F_c is the R, G, or B fog color, and C_r is the fragment's incoming R, G, or B color. The RGB fog color is indicated through the **Fog.color** register. The A component is unchanged by fog.

The fog blending factor f is linearly interpolated along the plane defined by the primitive's (x,y,f) vertex coordinates. This interpolation is performed using the plane equation arithmetic described in Section 7.3.7.

1. This is an equivalent but less computationally expensive expression of the formula given in the OpenGL specification:

$$f*C_r + (1-f)*F_c$$

Therefore, fog application requires the plane equation arithmetic parameters f_s, df/dx, and df/dy, where f is the fog blending factor.

The fog blending factor is applied as a 1.8 unsigned fixed point value clamped to the range [0,1.0]. To maintain precision, a 1.20 two's complement fixed point stepper is used to interpolate the blending factor. The lowest 12-bits of the stepper value are then truncated The fractional part of this 9.12 value is then truncated and then clamping them to the 8-bit range [0,255]. If this value is greater than 128, it is incremented by one so that

Table 7-18 Fog Blending Factors

Fog.F_s	1.20	f_s
Fog.dfdx	1.20	df/dx
Fog.dfdy	1.20	df/dy

7.3.7.10 Antialiasing

Antialiasing operates on the fragment's alpha value after fogging. Antialiasing applies only for point and line primitives and only for RGB(A) pixels. If antialiasing is enabled, a coverage value is computed for the fragment and used it to modulate the fragment's alpha value.

TBD

If antialiasing is enabled for a line primitive, a line width of two is assumed.

Coverage values are computed using a 128x8 ROM coverage table indexed by the three upper bits of the absolute value of the slope of the line and the 4-bit subpixel distance between the minor coordinates of the sample point and ideal line. This coverage value is modulated by an endpoint coverage value obtained from a 128x4-bit ROM endpoint coverage table indexed by the upper bits of the slope and the subpixel distance between the If the fragment is considered an endpoint fragment, Then, an endpoint filter is applied to the indexed coverage value if the distance between the major coordinates of the sample point and either of the ideal line's end-

points is less than one fragment width. The resulting coverage value is a 0.8 fixed point value corresponding to the floating point range [0, 1). The product of the coverage value and the fragment's alpha value is the fragment's new alpha value.

Antialias computes the ideal line's minor coordinate iteratively using the line slope value which the host loads into the Antialias.slope register. The slope is defined as dy/dx for x-major lines and dx/dy for y-major lines. To maintain precision, the host must specify the slope as a 2.12 two's complement fixed point value in the range [-1.0, 1.0].

Since antialias modulates only the alpha value of the fragment, the host must enable blending and specify an appropriate blending function to modulate the fragment's RGB value.

7.3.7.11 Alpha Test

The alpha test is performed on the fragment's alpha value after antialiasing coverage application. It is enabled/ disabled through the DrawMode.enAlphaTest bit and applies only for RGB(A) pixels. If enabled, the alpha test discards the fragment if the comparison between the fragment's alpha value and an 8-bit reference value fails.

The possible constants specifying the alpha test function and their corresponding functions are shown in Table 7-19. The alpha test function and 8-bit reference value are indicated through the AlphaTest.func and AlphaTest.ref fields.

Value	Function
NEVER	false
LESS	alpha < ref
EQUAL	alpha == ref
LEQUAL	alpha <= ref
GREATER	alpha > ref
NOTEQUAL	alpha != ref
GEQUAL	alpha >= ref
ALWAYS	true

Table 7-19 Alpha test functions

7.3.7.12 Alpha Blend

Alpha blending is performed on the fragment's RGB(A) color after anti-aliasing coverage application. It is enabled/disabled through the enBlend bit of the DrawMode register and applies only for RGB(A) pixels. If alpha blending is enabled, the source RGBA components are combined with their corresponding destination RGBA components according to the formula

$$C_s*S \text{ OP } C_d*D \qquad \text{(EQ 12)}$$

where C_s and C_d are source and destination components, S and D are source and destination blending factors, and OP is the blend op. Both the source and destination blending factors are RGBA quadruplets applied to the source and destination components, respectively. The source and destination blending factor quadruplets are computed according to the src and dst fields of the Blend.func register. The possible constants for computing the source and destination blending factor quadruplets and the blending factors they compute are shown in Table 20 and Table 21.

Table 7-20 Source blending factors

Value	Source Blend Factor
ZERO	(0, 0, 0, 0)
ONE	(1, 1, 1, 1)
DST_COLOR	(R_d, G_d, B_d, A_d)
ONE_MINUS_DST_COLOR	(1, 1, 1, 1) - (R_d, G_d, B_d, A_d)
SRC_ALPHA	(A_s, A_s, A_s, A_s)
ONE_MINUS_SRC_ALPHA	(1, 1, 1, 1) - (A_s, A_s, A_s, A_s)
DST_ALPHA	(A_d, A_d, A_d, A_d)
ONE_MINUS_DST_ALPHA	(1, 1, 1, 1) - (A_d, A_d, A_d, A_d)
SRC_ALPHA_SATURATE	
CONSTANT_COLOR	(R_c, G_c, B_c, A_c)

Table 7-20 Source blending factors

Value	Source Blend Factor
ONE_MINUS_CONSTANT_COLOR	(1, 1, 1, 1) - (R_c, G_c, B_c, A_c)
CONSTANT_ALPHA	(A_c,A_c, A_c, A_c)
ONE_MINUS_CONSTANT_ALPHA	(1, 1, 1, 1) - (A_c, A_c, A_c, A_c)

Table 7-21 Destination blending factors

Value	Destination Blend Factor
ZERO	(0, 0, 0, 0)
ONE	(1, 1, 1, 1)
SRC_COLOR	(R_s, G_s, B_s, A_s)
ONE_MINUS_SRC_COLOR	(1, 1, 1, 1) - (R_s, G_s, B_s, A_s)
SRC_ALPHA	(A_s, A_s, A_s, A_s)
ONE_MINUS_SRC_ALPHA	(1, 1, 1, 1) - (A_s, A_s, A_s, A_s)
DST_ALPHA	(A_d, A_d, A_d, A_d)
ONE_MINUS_DST_ALPHA	(1, 1, 1, 1) - (A_d, A_d, A_d, A_d)
CONSTANT_COLOR	(R_c, G_c, B_c, A_c)
ONE_MINUS_CONSTANT_COLOR	(1, 1, 1, 1) - (R_c, G_c, B_c, A_c)
CONSTANT_ALPHA	(A_c,A_c, A_c, A_c)
ONE_MINUS_CONSTANT_ALPHA	(1, 1, 1, 1) - (A_c, A_c, A_c, A_c)

Table 7-22 Blend op values and their corresponding blend ops.

Value	Blend Op
ADD	C_s*S + C_d*D
SUBTRACT	C_s*S - C_d*D
REVERSE_SUBTRACT	C_d*D - C_s*S
MIN	min(C_s, C_d)

Table 7-22 Blend op values and their corresponding blend ops.

Value	Blend Op
MAX	$\max(C_s, C_d)$

7.3.7.13Dithering

TBD

7.3.7.14Logic Ops

Logic ops are enabled throught the enLogicOp bit of the DrawMode register. If enabled, a bit-wise logical operation is performed between the fragment's source color value after dithering and the corresponding destination color value read from the framebuffer. Logiccops apply to both color index and RGBA pixels. The logical operation performed is indicated through LogicOp register.

Table 7-23 Logic op values & their corresponding logical operations applied bitwise on the source (s) and destination (d) colors.

Value	Operation
CLEAR	0
AND	s & d
AND_REVERSE	s & ~d
COPY	s
AND_INVERTED	~s & d
NOOP	d
XOR	s ^ d
OR	s | d
NOR	~(s | d)
EQUIV	~(s ^ d)

Table 7-23 Logic op values & their corresponding logical operations applied bitwise on the source (s) and destination (d) colors.

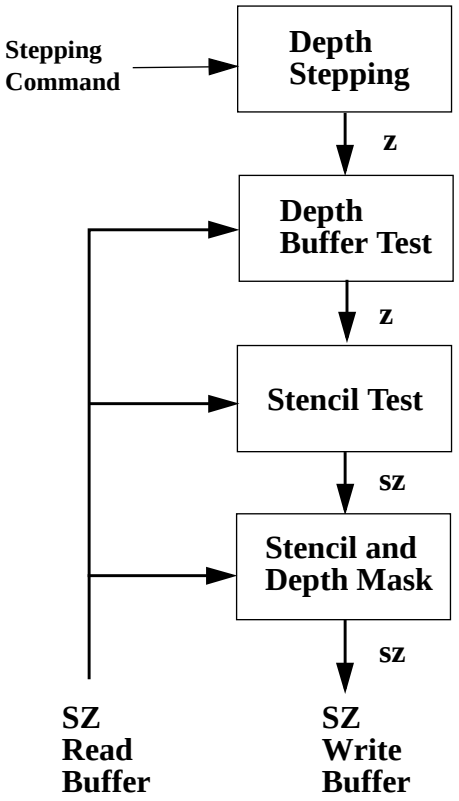
Value	Operation
INVERT	~d
OR_REVERSE	s | ~d
COPY_INVERTED	~s
OR_INVERTED	~s | d
NAND	~(s & d)
SET	1

7.3.7.15 Color Mask

If enabled, the color buffer plane mask controls which bits of the fragment's color are written to the color buffer. This plane mask can be applied to both CI and RGB(A) pixels.

The color buffer plane mask is indicated through the **ColorMask** register. Where a 1 appears in the mask, the corresponding bit in the color buffer is updated; where a 0 appears, the corresponding bit is not updated. Because the Moosehead memory system has no direct write-per-bit capability, plane masking is implemented using a read-modify-write operation.

Figure 7-17 SZ pipeline flow



7.3.7.16 Depth Stepping

The depth value z assigned to the fragment is determined by linearly interpolating depth values along the plane defined by the primitive's (x,y,z) vertex coordinates. The depth value is linearly interpolated using the plane equation arithmetic described in Section 7.3.7. Therefore, depth value interpolation requires the plane equation arithmetic parameters z_s, dz/dx, and dz/dy, where z is the depth value to be interpolated.

To maintain precision, a 25.12 two's complement fixed point stepper is used to interpolate 24-bit depth values. The 24-bit depth value assigned to the fragment is generated by truncating the fractional part of this 25.12 value and then clamping it to the 24-bit range $[0,2^{24})$. To assign properly rounded depth values, the host should adjust each initial depth value z_s by 0.5.

The depth plane equation parameters must be specified as two's complement numbers. They are loaded into the following registers:

Table 7-24 Depth plane registers

Depth.z	25.12	Z_s
Depth.dzdx	25.12	dz/dx
Depth.dzdy	25.12	dz/dy

7.3.7.17 Depth Test

The depth buffer test operates on the depth value assigned to the fragment. If enabled, the depth buffer test discards the fragment if a comparison between the fragment's depth value z_s and the value of the corresponding depth buffer pixel z_d value is false. The depth buffer test is enabled or disabled by setting the *enDepthTest* field of the **DrawMode** register to ENABLE or DISABLE, respectively.

The possible constants specifying the depth buffer test function and their corresponding functions are shown in the table below. The depth buffer test function is indicated through the **Depth.func** register.

Table 7-25 Depth function values and their corresponding comparison functions.

Value	Function
NEVER	false
LESS	$z_s < z_d$
EQUAL	$z_s == z_d$
LEQUAL	$z_s <= z_d$
GREATER	$z_s > z_d$
NOTEQUAL	$z_s != z_d$
GEQUAL	$z_s >= z_d$
ALWAYS	true

7.3.7.18 Stencil Test

If enabled, the stencil test discards the fragment if a comparison between the stencil value and an 8-bit reference value is false. The stencil test is enabled or disabled by setting the *enStencilTest* field of the **DrawMode** register to ENABLE or DISABLE, respectively.

TBD

Table 7-26 Stencil functions & their corresponding comparison functions.

Value	Function
NEVER	false
LESS	s < ref
EQUAL	s == ref
LEQUAL	s <= ref
GREATER	s > ref
NOTEQUAL	s != ref
GEQUAL	s >= ref
ALWAYS	true

7.3.7.19 Stencil Op

Table 7-27 Stencil ops values & their corresponding operations.

Value	Op
ZERO	s = 0
KEEP	s = s
REPLACE	s = ref
INCR	s++
DECR	s--
INVERT	s = ~s

CHAPTER 8

System Initialization and Reset

8.1 Introduction

The R10K/R12K processor supports two types of the reset sequences: Power-on reset and Cold Reset. Soft Reset for the R10K/R12K processor family is not supported. (Note: Crime 1.1 supported a Warm Reset for the R5K, but that is because the R5K has deterministic state after Warm Reset, which is not always so for the R10K/R12K.)

Power-on reset allows >1 msec for the power supplies to stabilize before asserting DCOK. CRIME then performs the defined sequence to assert SysRst*, load the Mode word into the R10K/R12K with SysRespVal*, and finally to deassert SysRst*.

Cold Reset assumes that the power is stable and so DCOK remains asserted to the R10K/R12K. CRIME then performs the same sequence as a Power-on reset.

Soft Reset assumes that the power is stable and that all the phase lock loops are stable. No Mode word is loaded, only the SysRst* pin is asserted for > 16 Clocks. Note that CRIME 1.5 does NOT support this mode.

8.2 Power-on Reset Sequence

A detailed description of the Power-on Reset sequence for the R10K processor can be found in the R10K User's Manual. Refer to Chapter 8 Section 8.2 in that manual for a functional timing diagram.

It is anticipated that the Power-on Reset sequence for the R12K will be identical.

8.3 Cold Reset Sequence

A detailed description of the Cold Reset sequence for the R10K processor can be found in the R10K User's Manual. Refer to Chapter 8 Section 8.3 in that manual for a functional timing diagram.

It is anticipated that the Cold Reset sequence for the R12K will be identical.

A Cold Reset Sequence can be initiated through Software by programming the CPU Interface Control Register described in Chapter 5.

8.4 Soft Reset Sequence

Soft Reset Sequence will not be supported by CRIME 1.5 as the state of the R10K is often un-recoverable when a Soft Reset Sequence is initiated.

8.5 Reading the Mode bits

CRIME reads a serial ROM on the CPU board as part of the Power-on and Cold Reset Sequences. In this ROM are 64 bits that comprise the Mode bit pattern required by the R10K/R12K processor for the Mode initialization sequence.

8.5.1 Mode bit sequence

These bits are stored in the ROM in big endian bit format so that the 64 bit pattern is more readable to humans looking at them with editors and ROM programmers.

This means that bit 0 read from the serial ROM is bit 63 of the SysAD Mode word, bit 1 is bit 62 etc.

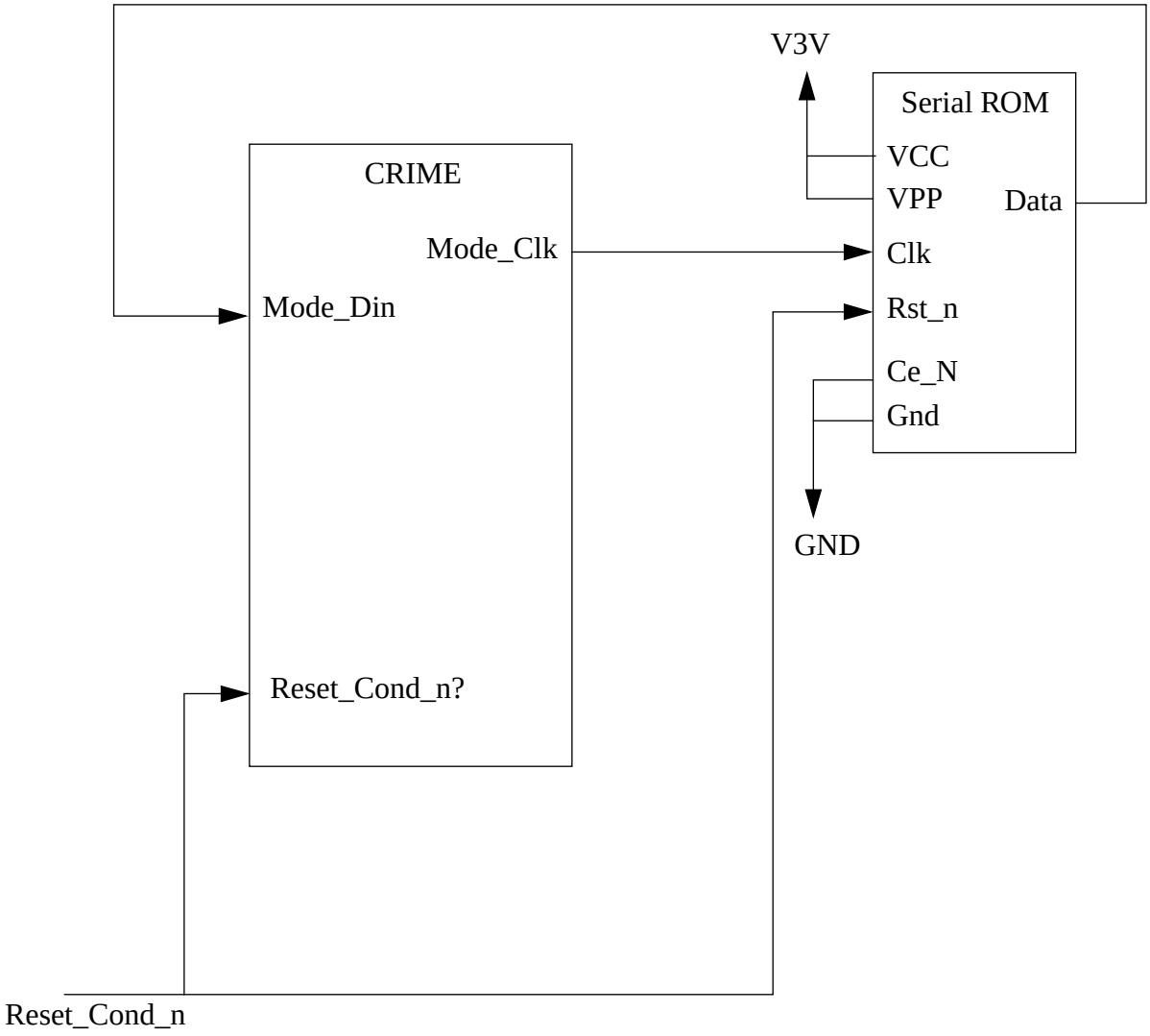
The mode bits are loaded from a serial port on CRIME comprised of the MODE_CLK (output) and MODE_DIN (input) pins.

During the design implementation, should additional mode bits be required for the use by the CRIME chip, they could be added to the protocol here.

The diagram shows the serial ROM that would reside on the CPU board with the R10K or R12K and the VICE chip. The ROM should reside there so that different speed processors can have the ROM programmed for the correct clock divisors.

8.5.2 Mode bit values

The list of the R10K mode bits is shown below. The bits and the default (or recommended operating) states for an icehouse system are indicated in the table for ease of reading. The function and the encoding of the bits can be found in the *R10K User's Manual*.



Note: this input to CRIME and the Serial ROM must be active for Power On Reset and Cold Reset

Figure 8-1 Serial ROM Connection diagram

Table 8-1: R10K Mode bits

SysAD bit	Name	Remarks
2:0	Kseg0CA	Default 3: Cacheable nonchoherent
4:3	DevNum	Default 0: Processor 0
5	CohPrcReqTar	Default 0: Coherency disabled
6	PrcElmReq	Default 0: Disable issuing of processor eliminate requests.
8:7	PrcReqMax	Default 3: 4 Outstanding processor requests
12:9	SysClkDiv	Default 3: SysClk is PClk divided by 2 (200MHz) 4: SysClk is PClk divided by 2.5 (250 MHz) 5: SysClk is PClk divided by 3 (300 MHz)
13	SCBlkSize	Default 1: 32 words.
14	SCCorEn	Default 0: Secondary cache access always through the corrector. ???????
15	MemEnd	Default 1: Big Endian.
18:16	SCSize	Default 0x1: 1MB.
21:19	SCClkDiv	Default 0x2: Divide by 1.5.
24:22	Reserved	Default 0x0.
28:25	SCClkTap	Default 0x0. Will try out in the system design.
29	Reserved	
30	ODrainSys	Default 1: Open-drain (To be reviewed).
31	CTM	Default 0: Test Mode disabled.
63:32	Reserved	

8.6 Quick Boot Sequence

NOT Supported

8.7 Pin Reset States

Needs to be updated with CRIME Signals

Table 8-2: Pin Reset States

Pins	Reset State	Remarks
SysPLL_CtrOut	undefined	
MemPLL_CtrOut	undefined	
SysCmd(11:0)	tristate	
SysCmdPar	tristate	
SysAD(63:0)	tristate	
SysAD_Chk(7:0)	tristate	
SysVal_n	tristate	
SysRdRdy_n	negated	
SysWrRdy_n	negated	
SysGntP(3:0)_n	negated	
SysRel_n	tristate	
SysResp(4:0)	11111	For R10K, the reset state is a don’t-care.
SysRespPar	0	From odd parity generator.
SysRespVal_n	negated	
MemData(63:0)	tristate	
MemDataChk(7:0)	tristate	
MemAddr(12:0)	0x0	**This may be changed to invalid address.**
MemBS	0	This is arbitrarily chosen. It can be changed to match the actual design.

Table 8-2: Pin Reset States

Pins	Reset State	Remarks
MemRAS_n	negated	
MemCAS_n	negated	
MemWE_n	negated	
MemCKE	asserted	
MemCS_n(15:0)	negated	
MemDir	0	
MemGS	0	
InitDCOK	negated	
InitSysReset_n	asserted	
InitColdReset_n	asserted	
InitSW_Reset_n	tristate	
HrtPLL_RstOut_n	asserted	
InitModeClk	0	
InitSR_SH	0	
Trig	0	
TstDO	0	

8.8 System Board Initialization Note

During the design review for CRIME 1.5 it was noted that the CPU board layout should include mounting pads and space to mount a reset switch and de-bounce circuitry that could be used in the lab to drive the NMI pin of the R10K/R12K for lab debug purposes.

CHAPTER 9 Open Issues & Decisions

Open issues are documented here. The decisions on recent open issues are documented here for the sake of reference and archiving

9.1 Open Issue list

9.1.1 128 byte line size for R10K/R12K

A singe read or write request from a R10K/R12K processor can produce up-to 4 memory requests for the internal CRIME Memory Interface Unit (MIU).

NOTE: The final decision for cache line size will be made based on available silicon area, as the die will remain the same size as CRIME 1.1. As such my take on this would be to shoot for a 64 byte cache line size which will make the layout more likely to succeed.

The read buffer in CRIME should be 256 bytes to keep performance high and control complexity low.

Because a single memory clock at 66 MHz produces data for 4 SysAD cycles at 100 MHz it is theoretically possible to sustain the read bandwidth of a single R10K/R12K processor with a read buffer that is smaller than 256 bytes. The present CRIME 1.1 part has a read buffer of 32 bytes. A buffer of 128 bytes while seemingly an increase of 4x die area would in fact be less than that since the buffer is presently implemented with flip-flops and would be moved to a latch based design for larger implementations.

9.2 Decisions History

This section contains the original “todo” list compiled by James Tornes at the outset of the “icehouse” project in February of 1997. Further decisions since then are included. For issues that have been deferred as an implementation detail, a notation has been made to that effect.

9.2.1 4 outstanding reads from processor

a. Add 3 more entries to present 3 entry request queue.
b. Larger read response buffer see 9.1.1 "128 byte line size for R10K/R12K" to hold 2-128 byte cache lines.
c. Handshake required to acknowledge when entries are freed from the CPU clock domain to the memory clock domain. Not required in present 1 outstanding read design.
d. No ECC check bits will be generated for R10K/R12K SysAD transactions. No ECC check bits will be checked on SysAD transactions from the R10K/R12K. Interrupt Acknowledge cycles do not require ECC.
e. Process all processor requests in order, but still need to keep transaction number for 4 outstanding requests to return them to the R10K/R12K.
f. A “graduator” for read responses and logic to decide which of the 2-128 byte read response buffers drives the internal system response bus is required. The arbiter requirement is eliminated if a single 128 byte response buffer is implemented at the expense of more complex control logic between the PIU state machine that submits read requests to the internal CRIME MIU.
g. RdRdy_n will not be implemented in CRIME because the internal CRIME command buffer will always be able to accept

4 read requests. This is consistent with the first version of CRIME which did not implement RdRdy_n because the command buffer could always accept 1 possible read request from the R5K processor.

9.2.2 Add speculation work around for R10K

The Juice method was selected for this fix. See Chapter 5 for details.

9.2.3 Increase write buffer size

CRIME 1.1 contains eight 32 byte entries. This is 256 bytes. A 50% increase in write buffer size would be a buffer of 384 bytes. This would allow 3 cache lines to be resident in the write buffer at any time.

The control logic to inform the processor on the SysAD bus that CRIME is NOT ready to accept additional writes (WrRdy_n asserted) will be enhanced to monitor the command queue of writes AND the data buffer level of writes. This logic is not that complicated as it previously existed in a version of the CRIME PIU.

One method to accomplish this is to assert WrRdy_n anytime the write data buffer contains 256 bytes. This would leave a 128 byte cushion for colliding with a write cycle that would be max size (full cache line).

There is no advantage to increasing the write command queue depth beyond 8 since 8 double word writes still take 256 bytes in the write data buffer if they are to the same address (example is the graphics fifo).

The algorithm then for WrRdy_n assertion is 256 bytes in the data buffer OR the high water level water mark in the write command queue (say 5 which I think is the existing water mark in CRIME).

9.2.4 SysAD arbitration

9.2.4.1 SysAd arbiter to R10K/R12K & VICE

The arbitration scheme for the SysAD bus between CRIME and an R10K is different than with the R5K.

A new arbitration block will be created to support the R10K/R12K protocol. VICE is an added complexity to this protocol. The existing VICE CRIME “SysAD DMA” method {patent pending :-) }will be preserved for all VICE <-> CRIME SysAD activity.

VICE read responses in an R10K/R12K implementation will follow a protocol proposed by Minghua Lin. This protocol requires that VICE ask the CRIME chip for the bus with the usual ViceSysRqst_n signal and wait for a grant from CRIME with the ViceSysGnt_n signal prior to responding to an R10K/R12K read request. This method will prevent VICE from driving the bus if the R10K/R12K is producing more SysAD activity and will also prevent VICE from colliding with CRIME read responses intended for the R10K/R12K.

A suggested enhancement to this protocol would be to have VICE assert ViceSysRqst_n for a single cycle when the intended VICE request for the SysAD bus is for the sole intention of a read response to the R10K/R12K. This would be a performance enhancement, since CRIME could then grant the bus to VICE even when the CRIME-VICE dma buffers are full in the CRIME chip.

9.2.5 R10K reset sequence

This sequence involves a protocol involving the SysAD bus rather than dedicated pins as in the R4x000 and R5K implementations. As such, the external agent is typically the device that handles this protocol as the R10K/R12K is initialized.

Two pins will be added to the connector between the CPU module and the System board in the O2 implementation. The serial ROM will reside on the CPU board so that it is present for the R10K/R12K implementation and can send data to CRIME. CRIME will in turn support the R10K/R12K initialization protocol across the SysAD bus using the information from the CPU board serial ROM.

9.2.6 SyaAD I/O buffers

a. The SysAD Interface pins of CRIME and VICE will support HSTL similar to the Heart chip in the Octane implementation.
b. A differential PECL clock input will be added to CRIME and VICE.
c. VICE <-> CRIME interconnections that are unique to the VICE CRIME protocol will remain as LVTTL signals.

9.2.7 VICE

VICE will remain on the SysAD bus in the icehouse architecture.

VICE will support the R10K/R12K protocols.

VICE will support SysAD I/O buffers similar to the Heart chip in the Octane implementation.

VICE will support a PECL clock.

VICE will follow the protocol extensions for the R10K/R12K read response as outlined in 9.2.4 "SysAD arbitration"

9.2.8 Uncache accelerated support

Uncached accelerated writes are not supported to I/O.

Uncached accelerated writes are supported to Memory as they are very useful for memory management (buffer initialization, 64K page to 4K page transfers).

9.2.9 Remove 1 pipeline stage for read responses

This is an attempt to minimize memory latency for Unix processor reads from memory. Any implementation techniques that further the goal of minimizing memory read latency, while preserving system functionality (GBE must send pixels to the display, JPEG must work for VICE, MACE must be allowed to move 3 video streams in real-time) are strongly encouraged.

9.2.10 R10K Interrupt Cycle

The R10K/R12K has a single clock period interrupt cycle specified in the SysAD protocol. CRIME will support this as the mechanism to communicate interrupts to the R10K/R12K.

9.2.11 R10K/R12K 40 bit addressing

CRIME will support 40 bit address checking in the R10K/R12K implementation mode to prevent address aliasing.

9.2.12 Bus Errors

Bus errors will be generated for all cache reads not to memory address space.

9.2.13 R5K/RM7K Stream Buffers

Stream buffers will not be supported in this design as the R5K/RM7K are not supported by CRIME 1.5.

9.2.14 CRIME bus error on separate interrupt level?

CRIME 1.5 will support all 5 interrupt levels specified in the MIPS interrupt registers of the R10K/R12K.

The 32 interrupt bits in CRIME that presently are logically ored together (after compared with their individual mask bits) will be collected into 5 separate groups. This will allow the kernel a performance gain when managing interrupts. The present implementation requires many PIO read and write cycles to CRIME to perform rudimentary Software Protection Level (SPL) emulation using the CRIME 1.1 implementation.

9.2.15 CPU_SYSCORERR_N

This output pin from the R10K/R12K processor will identify when a correctable error has occurred on any of the internally ECC protected data paths which include all caches and tags.

CRIME will not implement a counter each time one of these occur, as the frequency for a passable part is a couple of occurrences over several hours, which can be monitored by Software without a hardware counter.

CRIME will generate an interrupt each time one of these occur.

This will be used by the kernel software to monitor the frequency of internal R10K/R12K correctable errors. The value of this is to find failures in R10K/R12K processors in the manufacturing process and in the installed base that would otherwise be masked by the correcting circuitry inside these processors.

9.2.16 CRIME Interrupt Levels

The grouping of the 32 CRIME interrupts into 5 Processor interrupts has been finalized.

This is per Chris Johnson’s analyses with one addition. The new CPU_SysCorErr* interrupt has been added to level 7 to be grouped with interface error interrupts.

IP1 - softint0

IP2 - softint1

IP3 - /* SPL3 / SPL5 */

CRM_INT_VICE 0x80000000
CRM_INT_RE5 0x08000000
CRM_INT_RE4 0x04000000
CRM_INT_RE3 0x02000000
CRM_INT_RE2 0x01000000
CRM_INT_RE1 0x00800000
CRM_INT_RE0 0x00400000
MACE_PERIPH_MISC 0x00000020
MACE_PERIPH_SERIAL 0x00000010
MACE_ETHERNET 0x00000008

IP4 - /* SPLHINTR */

CRM_INT_GBE3 0x00080000
CRM_INT_GBE2 0x00040000
CRM_INT_GBE1 0x00020000
CRM_INT_GBE0 0x00010000
MACE_PCI_SHARED2 0x00008000
MACE_PCI_SHARED1 0x00004000
MACE_PCI_SHARED0 0x00002000
MACE_PCI_SLOT2 0x00001000
MACE_PCI_SLOT1 0x00000800
MACE_PCI_SLOT0 0x00000400
MACE_PCI_SCSI1 0x00000200
MACE_PCI_SCSI0 0x00000100
MACE_VID_OUT 0x00000004
MACE_VID_IN_2 0x00000002
MACE_VID_IN_1 0x00000001

IP5 - /* SPL6 */

CRM_INT_SOFT0 0x10000000

IP6 - /* SPL65 */

CRM_INT_SOFT2 0x40000000
CRM_INT_SOFT1 0x20000000
MACE_PERIPH_AUDIO 0x00000040

IP7 - /* SPL7 */

CRM_INT_MEMERR 0x00200000
CRM_INT_CRMERR 0x00100000
MACE_PCI_BRIDGE 0x00000080

IP8 -

CPU count/compare int

Note that Software numbers interrupts from IP[1] through IP[8] while the R10K document and Chapter 5 of this specification number them IP[1] through IP[7].

9.3 Revisions

REV 0.3 6/2/97

All Figure and Table Numbers preceeded by Chapter Number and restarted in each chapter. Table of Contents modified to show chapter number preceeding each page number within a chapter. This will better preserve Figure, Table and Page Numbers for future revisions.

Ch1 - Vice 1.5 Spec added to list of references.

Ch2 - HSTL_IEN pin added for HSTL test requirement.
First pass at Mechanical Drawing for 584+ TBGA Package.

Ch5 - SysAD arbitration priority added.

Interrupts that are edge triggered are really “level latched” in Crime 1.1. Writing the Hardware Interrupt register is necessary to reset “level latched” interrupts. Also, interrupts mapped to R10K/R12K IP levels go from IP2-IP6 instead of IP3-IP7 as in REV 0.2 of this specification. Fixed error in “Crime Interrupt Assignments” Table that had MACE interrupts incorrectly named. CPU_SysCorErr_n interrupt added to various Interrupt registers as it was overlooked in Rev 0.2 of this specification.

CPU Error Status register moved from 0x1400 1048 to 0x1400 4048 to be on separate 16K page as requested at previous design review. The 0x1400 1048 address was incorrectly assigned as this is only 4K away from the other CRIME CPU registers.

HSTL_IEN put pin added to Figure 5-2.
Watchdog Timer Documented by Minghua

Ch 8 - Soft/Warm Reset not supported for R10K/R12K. Crime 1.1 does support Soft/Warm Reset for the R5K.

REV 0.2 5/14/97

Notes from Design Review of 4-30-97 for CRIME 1.5 Specification

1.0 Major Issues Not Yet Covered

- Interrupt Map, revise per Chris Johnson's Map.

Update Section 5 and Section 9

- Exceptions Listed - Section 5 **OPEN**
- Juice Functionality Documented in Spec

Preserve Memory Maps of CRIME 1.1

2.0 Open Software Issues

- CPU_SysCorErr_n add to interrupt register at bit 30
Previously occupied by Software Interrupt 2.

3.0 Issues found in detail review of

Chapters 2,5,9 (and some input on 8 too)

Chapter 2 Device Interface Description
=========

-Remove Godzilla / Heart references
-Document HSTL 1A driver type for SysAD

- Logic Levels added to Chapter 2 Table for all signal pins
-Physical Packaging Diagram updated to CRIME 1.5 package
- Table 1 CPU_SysCorErr_n added to pin list
- Table 1 SysClk1x and SysClk2x missing from pin list
  Also some confusion in the text in re-naming these pins MemClk. Remove reference to MemClk.
- Table 1 Phase Lock Loop pins for CPU_CLK missing from pin list

Chapter 5 Processor Interface Unit and Its Operations
=========

-Figure 5-1 Block Diagram SysCorErr* added to R10K and CRIME
SysCmdVal* should be called SysVal* (R10K) SysVal_n (CRIME)

-Section 5.2.1 R10K/R12K Bus Request Supported - Elaborate
Always return exclusive when other types of requests submitted.

-Section 5.2.2.1 Number of VcqSys pins is HSTL/6 => 120/6 = 20
May borrow some of these from existing 3.3V pins on package.

-Section 5.3 Processor Interface Assumptions
Highlight that SysAD is running WITHOUT ECC.

-Section 5.5 Add typical latency number assuming collision with GBE
-Figure 5-15 Remove second label of Figure 5-1 on same page.

-Section 5.7 Interrupts
2nd paragraph is hold over from R5K interrupt pin. Rephrase to indicate R10K interrupt cycle on SysAD by CRIME 1.5. Algorithm to decide when to update R10K based on external interrupt changes.

-Section 5.8 Timers
Add description of WatchDog Timer and Crime Timer

-Section 5.9 Register Map
Table 15 - Register Address Map

Software would like Crime Timer to be on separate 16K page so that it can be mapped to user process without exposing other registers in Crime. 0x1400 0038 => 0x1400 1038? Chris Johnson to verify that this Crime Timer is the same that we expose to user programs.

Table 16 - ID register - check value. Add note to make a hard mux in layout so that revisions that are mask changes only

can easily change the revision number field.

Table 17 - CPU Interface Control Register

Move JUICE Mode (bits 15:14) to bits field 7:5
Change range from 0/8/16/32 to 0/4/8/12/16/20/24/28/32 MEG
=> Bit field increases from 2 to 3.

Bit 10 Soft Reset Not supported for R10K as it was on R5K.
Add “NOT USED IN CRIME 1.5”

Table 18 - Interrupt Status Register bits 20 and 21 swapped.

THIS WAS NOT AN ERROR!

The VHDL, the spec and the Software Defines all agree that the bit positions below are correct.
Memory error interrupt (21)
CPU interface error (20)

Table 21 - Same comment as 18

Tables 18-21 Add description of how interrupt registers function:

Interrupt Enable Register must set bit to a 1 to allow the interrupt to cause an interrupt on that bit to produce a physical interrupt to the processor.

Interrupts that were enabled and created an interrupt can be read in the Hardware Interrupt Register.

Interrupts that occur can also be read in the Interrupt

Table 24 CPU Error Address Register

Contains bits 33:2 of the address for CPU PIO errors. Software would like bits 39:34 captured as well. May be put in

the CPU/VICE Error Status Register or a new register added.

Chapter 8 System Initialization and Reset
=========

Reminder to leave pads and room for switch and de-bounce circuitry on the CPU Board Layout to drive the NMI pin of R10K/R12K (HSTL Level?) for lab debug purposes.

Chapter 9 Open Issues and Decisions
=========

All Remove reference to R5K / RM7K

9.1.2 Update CRIME Interrupt Assignments

9.2.5 SysAD Arbitration Document priority scheme for releasing bus back to R10K with multiple reads in CRIME queue. James suggested perhaps a 2 deep read queue and then give R10K back bus for another if it is requesting, in order to keep the queue filled.

9.2.7 SysAD I/O buffers - No more LVTTL on SysAD

9.2.9 Uncached accelerated support - Add info that identical not supported.

9.2.14 Stream Buffers, remove since no R5K.

9.2.15 CRIME Bus error on SPL7 Includes MEMERR and CRMERR

4.0 Other Issues

Sections 1-12, 2-7 Remove R5K/RM7K references

Check entire Spec for R5K/RM7K references and HSTL and LVTL I/O cell confusion on SysAD

PIU / MIU Flow Control Documented Chapter 5 **OPEN**

Remove Support for Little Endian

CRIME/VICE Protocol Documented Chapter 5 - See VICE Spec

Final decision for cache line size 64/128 will be made
based on physical design constraints.

Soft Reset support not required - T5 state on Soft Reset not predictable.

Cold Reset will continue to be supported.