ELROY ERS

Ropes to PCI Bridge Chip

Computer Technology Lab

Revision A (1.4) 1/28/2000

P/N 2AC6

1. Introduction

1.1 ERS Status

This is the near-final revision of the ELROY ERS. This revision contains mostly formatting changes from the previous version.

1.2 Objectives

The primary objectives of this document are:

- Describe the software interface and inner operation of the ELROY chip in enough detail to be used with referenced documentation for driver, firmware, diagnostic and other software development.

1.3 Feature Summary

ELROY is part of a high-performance bridge between the system bus and the PCI bus. Elroy has the following features:

Elroy Features

- Support for Turbo-Channel and Twin Turbo-Channel to provide up to 500 Mbyte of BW
- Protocol to support PCI 2.1, 1X, 2X, and 4X.
- 32-bit or 64-bit PCI Bus
- 30-66Mhz PCI Bus

2. System Address Map

This chapter describes the physical address map and then presents the rationale behind the design of range registers in ELROY that determine how transactions are routed. For the full system address map, refer to the Rope-master's (Astro) ERS.

2.1 MMIO above 4 GB

Additional memory mapped I/O is available above 4 GB. Such ranges are called "GMMIO," for Greater than 4 GB Memory Mapped I/O. If such a range is enabled, it should be placed above all main memory.

2.2 I/O port space

The I/O port space is required purely for legacy reasons. x86 processors have always had it. There are many I/O devices out there that map their registers in I/O port space instead of memory space. Generally, new I/O devices should be designed to reside in memory space, but many standard PC functions have fixed allocations in the I/O port space and those are likely to continue to exist there.

There is only 64 KB of I/O port space, so it is difficult to allocate. If a PCI device is designed so that it can be accessed through either memory space or I/O space, it should be mapped into memory space. I/O port space should be used only when there is no alternative.

Like the memory mapped I/O space, the I/O port space will also have to be distributed between different PCIs. The entire non-posted I/O port space is claimed by the rope-master and is distributed equally between all logical ropes.

2.3 Range register programming

This section described how the range registers in rope-master fit together with range registers in ELROY. The registers mentioned here are described in detail in the Rope-master's ERS and PSC chapter.

Some general rules:

- Address space assigned to unused logical ropes is wasted. Therefore, if two ropes are combined into a single 2x rope, the address space for the second rope is wasted.
- If a directed range overlaps the distributed range, the directed range takes precedence. Therefore, if a directed range is used to reassign the distributed space allocated to an unused rope, that address space is "recovered."
- It is illegal to have directed ranges overlap each other.

2.3.1 Range registers for LMMIO space

First, distributed LMMIO in the Rope-master defines a range of memory to be divided equally among all ropes. LMMIO_DIST_BASE and LMMIO_DIST_MASK specify the location and the size of the range. LMMIO_DIST_ROUTE specifies which address bits to use for selecting which rope gets a particular transaction. Space allocated to unused ropes is wasted.

To handle cases where a particular rope needs more space than is allocated by the distributed range, directed LMMIO ranges in the MBIB should be used. Each directed LMMIO range is specified by three related registers: LMMIO_DIR_BASEx, LMMIO_DIR_MASKx, and LMMIO_DIR_ROUTEx. There are four such ranges, so x varies from 0 to 3. The Base register specifies the starting address of the range while the Mask register sets the size. Finally, the Route register specifies the rope to which addresses within this range will be routed. It should be apparent that ROUTE registers of directed ranges are different from ROUTE registers of distributed ranges.

ELROY needs to know the size of the *system wide* LMMIO region. This is accomplished by setting the LMMIO_BASE and WLMMIO_BASE registers, and the LMMIO_MASK and WLMMIO_MASK registers to the same values as the LMMIO_BASE and LMMIO_MASK registers in the Rope-master respectively.

ELROY will act as the target for any PCI transactions outside this LMMIO region, since these are DMA transactions destined for the system bus. The WLMMIO and LMMIO registers were kept separate for historical reasons, and they should *always* be configured to the same values. If they are configured to different values, undefined behavior may result.

2.3.2 Range registers for GMMIO space

For GMMIO, the MBIB supports only a distributed range. It works very much like the distributed LMMIO except that it works with addresses above 4GB. ELROY has a corresponding GMMIO range that specifies how much GMMIO is allocated to that ELROY. The GMMIO Range registers should be configured in a similar fashion as for LMMIO.

2.3.3 Range registers for non-posted I/O port space

If IOS_BASE is enabled in the Rope-master, the entire 64 KB is distributed across the logical ropes. Additionally, the Rope-master has one directed I/O port space range. This can be used to recover the space allocated to an unused rope.

ELROY needs to know what portion of the I/O port space is allocated to it. The space supplied by the implied distributed range should be programmed in the IOS_BASE and IOS_MASK registers. If additional space is supplied with the directed range, EIOS_BASE and EIOS_MASK should be used to program that range as well.

I/O ports in the VGA range are always routed based on VGA related bits. i.e., VGA takes precedence over all other I/O range registers.

3. ELROY ADDRESS MAP

This chapter describes the physical address map for ELROY covering the address offsets for the configuration, address range, and status registers architected in the chip. Bit field descriptions of register are given in the chapters describing the associated functionality.

3.1 Location of ELROY configuration space

ELROY configuration space is relocatable. Its placement is determined by the ECNFG_BASE register in the Rope-master. It implements this as a distributed range, supplying 8 KB (2 functions) of address space to each ELROY. The following shows how the address space is used: (all addresses are offsets relative to ECNFG_BASE)

```
ELROY 0      0x0_0000   0x0_1FFF
ELROY 1      0x0_2000   0x0_3FFF
                        |
                        |
 ELROY 5      0x0_A000   0x0_BFFF
```

3.2 ELROY register map in the configuration space

The following table shows all the registers in ELROY's configuration space. Note that some locations are designated as reserved or unimplemented. For the unimplemented registers, writes will be bit-bucketed (with proper handshake) and reads will return 0s. For reserved registers, only 0s should be written and reads will return non-determinable data that should be ignored.The distinction is that reserved registers are reserved in the architecture and are required to behave this way. The unimplemented ones could have had other behavior.

3.2.1 Function 0 – PMC

Register Offset	Register Name	Page #
0x0000	FID (include Status and Command)	70
0x0008	FC	70
0x0010-0x002F	Reserved	
0x0030	Unimplemented	
0x0038	Reserved	
0x0040	Config Address	73
0x0048	Config Data	74
0x0050	ELROY MTLT	69
0x0058	Bus number scratch register	69
0x0060	Reserved	
0x0068	Rope parity enable bit, Loopback bit.	129
0x0070	Error Log: Address	72
0x0078	Suspend	70
0x0080-0x007F	Unimplemented	

3.2.2 Arbitration (PCI 0)

Register Offset	Register Name	Page #
0x0080	Arbitration Mask	42
0x0088	Arbitration Priority	43
0x0090	Arbitration Mode	44
0x0098	MTLT	45
0x00A0	Unimplemented	
0x00A8-0x00FF	Unimplemented	

3.2.3 PSC (PCI 0)

Register Offset	Register Name	Page #
0x0100	MOD_INFO (module_id = 5)	53
0x0108	STATUS_CTRL (hf, ce, cl, fv and rf)	53
0x0110-0x01FF	Reserved	

0x0200	LMMIO_BASE	118
0x0208	LMMIO_MASK	118
0x0210	GMMIO_BASE	116
0x0218	GMMIO_MASK	116
0x0220	WLMMIO_BASE	58
0x0228	WLMMIO_MASK	58
0x0230	WGMMIO_BASE	55
0x0238	WGMMIO_MASK	56
0x0240	IOS_BASE	60
0x0248	IOS_MASK	60
0x0250	Reserved	
0x0258	Reserved	
0x0260	EIOS_BASE	61
0x0268	EIOS_MASK	61
0x0270	Unimplemented	
0x0278	DMA Connection Control	52
0x0280-0x02FF	Unimplemented	

3.2.4 PSC

Register Offset	Register Name	Page #
0x0300	IBASE	64
0x0308	IMASK	64
0x0310	HINT_CNFG	63
0x0318-0x0378	Unimplemented	
0x0380	HINTS0	62
0x0388	HINTS1	62
0x0390	HINTS2	62
0x0398	HINTS3	62
0x03A0	HINTS4	62
0x03A8	HINTS5	62
0x03B0	HINTS6	62
0x03B8	HINTS7	62
0x03C0	HINTS8	62

0x03C8	HINTS9	62
0x03D0	HINTS10	62
0x03D8	HINTS11	62
0x03E0	HINTS12	62
0x03E8	HINTS13	62
0x03F0-0x03FF	Unimplemented	

Misc

Register Offset	Register Name	Page #
0x0608	PCI_drive	48
0x0610	Rope_config	18
0x0618	Clk_cntl	52
0x0850	See Interrupt Section	
0x0624-0x067F	Unimplemented	

3.2.5 Error

Register Offset	Register Name	Page #
0x0680	Error Configuration	39
0x0688	Error Status	25
0x0690	Error Log: Master ID	25
0x0698	Rope Error Status	25
0x06A0	Rope Error Clear	26
0x06A8-0x07F7	Unimplemented	
0x07F8	No register should be assigned to this location. The regbus controller drives the value 11'b01111111111 when the regbus is idle.	

3.2.6 Interrupt

Register Offset	Register Name	Page #
0x0800-0x0847	Information here is needed by developers, but cannot be given out. Contact pa_linux@hp.com.	
0x0848-0x0FFF	Unimplemented	

4. ELROY Reset Strategy

4.1 Reset Inputs

ELROY implements a hierarchical reset strategy. At the top of the structure is the system bus reset which resets everything that can be reset in ELROY. The software generated resets are listed in the tables below. The resets are shown below in order of most effect on the logic to least.

ELROY Reset Inputs	
pwr_rst_L	Power reset pin, low true. External input driven by Dillon, asserted at power up. Unasserted asynchronously from Dillon when power is good. Used to tristate ELROY's pads and internal tristate busses. Not used as an rope-master or ELROY functional reset.
dll_reset_L	DLL reset, low true. Resets the ELROY bus, strobe and PCI DLL's in ELROY. Unasserted asynchronously causing the DLL's to move toward lock. This input pin will likely be connected to pwr_rst_L, however it is felt to be safer to have a dedicated pin for resetting the DLL's
rope_reset	Rope command resulting from System bus RESET. This command is sent continuously across the ropes as long as RESET# is asserted on System bus. Resets everything in ELROY as well as the PCI bus.
rope_soft_reset	Rope command resulting from any of the other resets in the rope master other than RESET. Sent continuously across the ropes as long as the reset input is asserted to the rope-master rope controller. Resets all of ELROY except registers as well as the PCI bus. Causes the RF bit in ELROY to be set. The RF bit must then be cleared by software after the required 100us PCI bus reset time.
funct_reset	Function reset, high true. Software reset input generated within ELROY as a result of a write to the RF bit in the SIC register. Remains asserted until cleared by software. Must stay asserted for a minimum of 100us to meet the PCI bus reset requirement. Causes the arbiter, PSC and the PCI bus to be reset.

4.2 Rules for software resets

The following rules and guidelines should be observed by software and firmware when exercising any of the resets:

- Rope resets (whether done as function 2/3 reset or as individual rope resets) are intended to work under any conditions. It would be reasonable to use them during warm boot to ensure that any DMAs that might be in progress are stopped. These resets are also used for recovering from rope errors.
- PCI reset is only intended to work when the bus is quiescent. Asserting this reset while the bus is active is likely to put ELROY in a confused state. The only reason for making this reset software controllable is OLR. So, it is reasonable to assume that the bus will be quisced before OLR operations.

- PCI reset is different from others in that it must be cleared by software. All others finish on their own. The PCI reset was special cased in this fashion to support OLR, where the bus must be held in reset indefinitely while an OLR operation is taking place. **Resetting a rope will set PCI reset so it is necessary to clear PCI reset before proceeding.**
- When clearing PCI reset, make sure that the minimum PCI reset time is met.

4.3 Hardware behavior during software resets

Generally, software resets are treated like fatal error mode. When a rope reset is done, all previously issued PIO reads that hadn't made it back up the rope will return all 1s. Of course, previously issued PIO writes may get lost, however, since the destination device is also being reset, it is unlikely that anyone can tell if the writes were lost. And DMA reads in progress will be killed immediately. Previously posted DMA writes could also be lost. PIO transactions issued after the reset will not be lost, even if software fails to wait for the rc bit to be cleared. However, it is recommended that the rc bit be checked anyway, in case future revisions of the chip require that.

During a PCI reset, the PCI interface behaves exactly as if in fatal mode. While the bus is held in reset, PIO reads will return all 1s, PIO writes will be bit bucketed. No DMA could be taking place.

4.4 Impacts of Resets

The impacts of each of the reset inputs is shown in the table below. An X in the column indicates that the specified block is reset by the reset at the top of the column.

4.5 ELROY Reset Signals

The ELROY reset block generates the signals listed in the table below.

ELROY Reset Signals	
reg_reset	High true, asserted as a result of rope_reset command. Sets all registers to their reset state.
core_reset	High true, asserted as a result of rope_reset or rope_soft_reset command. Resets all of ELROY except registers, arbiter and the psc.
soft_reset	High true, asserts as a result of rope_soft_reset command. Causes the RF and RC bits to be set. Must be cleared by software after the required 100us PCI bus reset time.
psc_reset	High true, asserted as a result of rope_reset command, rope_soft_reset command, or funct_reset. . For a rope reset command stays asserted until rope reset command goes away. For rope_soft_reset command or funct_reset remains asserted until cleared by software. Resets arbiter and psc (except registers). This reset also clears the mask register causing the pmc to go into fatal mode.

pci_reset_L	Low true, asserted as a result of rope_reset command, rope_soft_reset command, or funct_reset. For a rope reset command stays asserted until rope reset command goes away. For rope_soft_reset command or funct_reset remains asserted until cleared by software. Reset signal to the PCI bus. Reads of the PSC SIC register return the value of this reset signal.

The table below shows where the ELROY reset signals go.

ELROY Blocks / **Reset Signal**	**reg_reset**	**core_reset**	**psc_reset**	**pci_ reset_L**
Blocks w/Registers	X			
Core (Except PSC & Arbiter)		X		
PSC & Arbiter			X	
PCI Bus				X

5. I/O Ropes

A rope is a high-speed point-to-point connection between the System Bus to Ropes bridge (rope-master) and ELROY. Up to two ropes can be bundled together for bandwidth up to 500MBytes/s.

5.1 Rope Configuration Register

The rope configuration register contains the information indicating double-wide rope communication as well as enabling two and four wide rope communication. Reads of the register return the last value written. Writes of the register only take affect after a directed reset to the affected ropes.

MSB	Rope Configuration OFFSET: 0x0610																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0																															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																							C2	Reserved					Int	Rsvd	M2
RESET INITIALIZATION																															
0																															

Rope Configuration Bit Definitions	
M2	Set to two wide rope communication
Int	Set to make unused rope1 A/D lines be inputs to be used as extra interrupts (should only be set if ELROY is in one wide rope configuration.
C2	If low, ELROY is capable of two wide communication

At reset, the rope master is configured for single-wide rope communication. To change to double-wide, the following steps must be taken:

1. Read ELROY's rope configuration register to determine if the rope is connected for two wide rope communication (bit 8 is a 1).

2. Write to ELROY's rope configuration register, setting it to 0x1. Communication with ELROY is not possible until the remaining steps are executed.

3. Write to the rope master's rope configuration register, setting it to communicate in two wide mode with the selected ELROY. See the rope controller chapter of the rope-master's ERS.

4. Write to the rf bit of the corresponding rope control register. This will reset the selected rope-master's rope controller and the ELROY below it.

5. Start a software timer to count to 100 micro seconds. This is the minimum PCI bus reset time.

5. Read the rc bit of the rope-master's rope control register to determine when the reset is complete in the rope-master. Communication with ELROY (registers only) is now possible.

7. Wait for 100 micro second timer that was started in step 5 above.

8. Write a 0 to the rf bit of the ELROY Status_Control register.

6. ELROY Clocks

6.1 Overview

ELROY operates in a single frequency domain whose clock source is also the rope strobe input, ostrb. The target frequency is 120-133.3 MHz. There are two Delay Lock Loops (DLL's) in ELROY. The main one is used to cancel clock insertion delay from the ostrb input to the intenal clock as well as generate strobes that get sent out with rope data a quarter cycle delayed from the data. The second DLL generates the PCI bus clock. This DLL is used to cancel out ELROY chip and some board insertion delay for the external PCI bus clock. Two frequencies are supported for the PCI bus clock – divide by 4 and divide by 2 of the ELROY chip clock. For the target frequency of 120-133.3 MHz, the PCI frequencies would be 30-33.3 MHz or 60-66.7 MHz.

6.2 PCI Bus Clock

ELROY's internal circuits use a signal pci_nx_edge to determine when the external and the internal clock edges align. The rising edge of the "real" pci clock will always be synchronized with a rising edge of the pci_int_clk. The frequency of the pci_int_clk will always be an integer multiple of the real PCI clock (2 or 4). The pci_nx_edge signal allows blocks that communicate with PCI to operate without knowledge of the frequency ratio between the PCI clock and pci_int_clk. The actual PCI bus clock will not be available internally to ELROY.

6.2.1 PCI_CLK_CTRL Register

MSB	PCI_CLK_CTRL Configuration OFFSET: 0x0618																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																												Dll_lock	Dll_rese	Pclk_div	Reserve
RESET INITIALIZATION																															
0																															

The operation of the PCI clock is specified by the clock control register (PCI_CLK_CTRL). The PCI_CLK_CTRL register controls the PCI clock divide ratios. The table below explains the fields in the PCI_CLK_CTRL register:

Field	bits	Description	Value
reserved	0		
pclk_div	1	Sets the divide ratio for the internal-to-external PCI clocks. The reset value is 1’b1	0 - 2:1 1 – 4:1
dll_reset	2	Causes the delay locked loop to be reset. This bit will be set to 0 at system reset.	
dll_lock	3	Read only. Indicates that the PCI DLL has stabilized.	
reserved	63:4		

Table 32. Clock Control Register Bit Definition

6.2.2 Modifying PCI Clock Divide Ratio

The pci DLL in ELROY requires a specific procedure to change the divide ratio.

1. Write a 1 to the rf bit of ELROY Status_Control register. This puts the pci bus in reset.

2. Write the new divide ratio along with a 1 in the dll_reset bit. E.g. to change from the default of 4:1 (33.3 MHz) to 2:1 (60-66.7 MHz) divide ratio, write 0x4 to the PCI_CLK_CTRL register.

3. Write again to the PCI_CLK_CTRL register with the same divide ratio value but with the dll_reset bit turned off. (0x0 in the example above). This starts the pci dll lock process.

4. Start a software timer to count 100 micro seconds. This is the pci bus minimum reset time.

5. Look for the following two conditions to both be met:

   a. 100 micro second timer has expired

   b. A read of the PCI_CLK_CTRL register returns a 1 in bit 3 indicating that the dll has stabilized.

6. Write a 0 to the rf bit of the ELROY Status_Control register. This turns off the reset to the pci bus.

7. ELROY Error Handling

7.1 ELROY Error Handling Details

ELROY performs error detection and error logging. This functionality is used in isolating the failing component. The rope-master contains an independent error register set at each of its bus ports. This register set is used for error logging.

7.2 ELROY Supportability

To allow sofware revision checking, ELROY contains chip identification and revision registers.
The rope-master also supports deconfiguration of:

Any PCI port:

- To deconfigure a PCI, it will be reset. This is accomplished through the reset function (rf) bit in the Status Information and Control (SIC) register in function 0 of that ELROY. This form of reset will put that PCI in fatal error mode, so further accesses to that PCI will not cause any action on PCI. More on fatal error mode later.

Any ELROY:

- To deconfigure an ELROY, its rope will be reset

7.3 Index of Error Codes

Error Code	Description	Page(s)
PI_D: code 0x01	PCI card requesting the bus fails to use it.	37
PI_D: code 0x0E	Any PCI device asserts LOCK#.	37
PI_D: code 0x12	ELROY Internal data	31
PI_D: code 0x13	PCI bus data error (ELROY observes PERR# asserted).	31
PI_D: code 0x14	PCI data parity	33
PI_D: code 0x15	ELROY Internal data	31
PI_D: code 0x16	PCI data parity	32
PI_D: code 0x18	ELROY internal data	34
PI_D: code 0x19	PCI bus data error.	35
PI_D: code 0x1A	(ELROY Target) ELROY signals Target-abort..	37
PI_D: code 0x1B	(ELROY Target) PCI Address / Command parity error.	36
PI_D: code 0x1C	(ELROY Initiator) PCI no DEVSEL#.	36
PI_D: code 0x1D	(ELROY Initiator) Target-abort from PCI.	36
PI_D: code 0x1F	Any PCI device asserts SERR#.	37
PI_S: code 0x01	PCI card requesting the bus fails to use it.	37
PI_S: code 0x03	PCI bus data error (ELROY observes PERR# asserted).	31
PI_S: code 0x04	PCI data parity	33
PI_S: code 0x05	ELROY Internal data	31
PI_S: code 0x06	PCI data parity	32
PI_S: code 0x08	ELROY internal data	34
PI_S: code 0x09	PCI bus data error	35

PI_S: code 0x0A	(ELROY Target) ELROY signals Target-abort..	37
PI_S: code 0x0B	(ELROY Target) PCI Address / Command parity error.	36
PI_S: code 0x0C	(ELROY Initiator) PCI no DEVSEL#.	36
PI_S: code 0x0D	(ELROY Initiator) Target-abort from PCI.	36
PI_S: code 0x0E	Any PCI device asserts LOCK#.	37
PI_S: code 0x0F	Any PCI device asserts SERR#.	37
PI_S: code 0x12	ELROY Internal data	31
RI_E: bit EI	ELROY internal data - Write Posting FIFO	34 , 33
RI_E: bit RC	Rope-master to ELROY rope command parity error	35
RI_E: bit RP	Rope-master to ELROY rope parity error	34 , 31 , 32

7.4 Error Detection

The tables below summarize the errors that Elroy detects. However, more detailed (and perhaps more accurate) information is supplied in later sections. Refer to the table on page 22 to quickly jump to specific error cases.

Legend	
unc	Uncorrectable Error
fe	Fatal Error
corr	Correctable Error

Description	**Severity / Comments**
ERR_FUNCTION	Bad parity on PCI Bus interface. (unc) Smart PCI card. (fe) Dumb PCI card.
ERR_MAP -PCI	IO_TLB entry invalid (unc) Smart card (fe) Dumb card
ERR_IMPROPER	(unc) A PCI card attempted a LOCKed transaction.
ERR_TIMEOUT -PCI	(unc) PCI master-abort – Smart card (fe) PCI master-abort – Dumb card
ERR_PARITY - PCI	Data with bad parity (unc) Smart PCI card. (fe) Dumb PCI card. Address parity (unc) Smart PCI card.

	(fe) Dumb PCI card.
ERR_ERROR -PCI	Detection of PERROR# (unc) Smart PCI card (fe) Dumb PCI card Detection of SERROR# (fe) Smart PCI card (fe) Dumb PCI card

7.5 ELROY Logging

The following are the basic error handling “rules” ELROY follows:

- Do not overwrite first error, unless a more severe subsequent error
- Log corrected errors.
- Logs must be initialized to 0 if there are no errors.
- Log enough information to isolate fault to FRU level. However, FRU isolation is not guaranteed.
- Logging must occur immediately after the error is detected
- Where applicable, log the requester, target, responder, and syndrome associated with the error.

7.5.1 Error logging registers

The rope interface in the PCI interface in ELROY will log errors in their respective Error Code registers defined below. The format of the error code register for the rope interface in ELROY is shown later. Instead of Estats, this field reports a unique error code for each error. The Estat can be determined from that error code. A, C and D bits and the severity are not be implemented either, since they too can be inferred from the error code. This approach allows firmware or diagnostic software to determine exactly which error took place. The OV bit indicates overflow. For ELROY it is set when multiple errors occur before software has had a chance to read the error code register.

The Format of the ELROY PCI error logging register is shown below.

MSB	**ELROY PCI_Error Register OFFSET: 0x0688**																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																										OV	ERR_CODE				
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The Format of the ELROY rope interface error logging register is shown below. Instead of an error code, it has individual bits for each error that it can detect, since there are only 3 possibilities.

MSB	**ELROY_Rope_Error Register OFFSET: 0x0698**																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																													RC	RP	EI
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EI: ELROY internal error. (Described later in this chapter.)
RP: Rope-master to ELROY Data parity error. (Described later in this chapter.)
RC: Rope-master to ELROY Command control parity error. (Described later in this chapter.)

7.5.2 Other logging registers

For some errors, information beyond the error code is logged. This is generally to help error isolation. The choice of what to log is specific to each type of error and is described later. The registers for logging are sprinkled through out the chip, so their descriptions are similarly spread out in the ERS. One logging register that is implemented in the error block is the Error_Log_Master_ID register. That register is described below:

MSB	**Error_Log_Master_ID Register OFFSET: 0x0690**																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															

RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved																								G	F	E	D	C	B	A	PMC
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X

PMC: Set if ELROY itself initiated the transaction that caused an error.
A: Set if bus master connected to pci_reqa_L initiated the transaction that caused an error.
B: Set if bus master connected to pci_reqb_L initiated the transaction that caused an error.
C: Set if bus master connected to pci_reqc_L initiated the transaction that caused an error.
D: Set if bus master connected to pci_reqd_L initiated the transaction that caused an error.
E: Set if bus master connected to pci_reqe_L initiated the transaction that caused an error.
F: Set if bus master connected to pci_reqf_L initiated the transaction that caused an error.
G: Set if bus master connected to pci_reqg_L initiated the transaction that caused an error.
Note: Logging of master ID is required only for some errors, as explained later. Above register contains valid information only when such an error is logged. When the register does contain valid information, only one of the bits can be set.

7.5.3 Clearing the error logs

Similarly, PCI interface errors are cleared using STATUS_CONTROL in ELROY function 0. Rope errors in ELROY are logged in the same function as PCI errors. Since STATUS_CONTROL for that function is used for PCI errors, rope errors are once again cleared with a special register like STATUS_CONTROL. The actual STATUS_CONTROL registers are described elsewhere. The rope related ones are described here.

MSB	**ELROY_Rope_Control Register OFFSET: 0x06A0**																														**LSB**
63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved																										ce	cl	Reserved			
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ce: Clear enable. Cleared by hardware whenever a new error is logged.

cl: Clears the ELROY_Rope_Error register if ce is currently set. For more information on cl and ce, please see the SPPA platform EAS.

7.6 ELROY Availability

ELROY will include an HF bit in its STATUS_CONTROL register. If this bit is set, ELROY returns a HardFail response on PIO reads as an error indication. If it is not set, -1 data is sent as an error indication. Errors are indicated when a PIO read receives bad parity data, or master-abort or target-abort. Also, if ELROY is already in Fatal Error mode when a PIO read is received, an error indication is sent.

Supported PCI card types.

Smart(HA Safe or HA Vital)

A smart card is required to detect, signal, and log any address or data parity errors. If a parity error is detected then the card is responsible not to use the data associated with the transaction. A smart card is required to perform graceful protocol completion of all transactions with error conditions; i.e. the card will not hang the bus.

PCI busses that would otherwise be considered dumb should temporarily be configured as smart during the time period when firmware is polling for card presence (i.e. the "bus walk"). This is necessary because master aborts due to absent cards will result in a PCI fatal condition in dumb mode.

Dumb (HP STD or HP Min)

A dumb card is defined by its inability to guarantee detection, signaling, and logging of address and/or data parity errors, as well as graceful completion of all transactions which include error conditions. A dumb card has the potential of using data with bad parity.

ELROY assists in error containment for dumb cards by going into fatal mode for most errors. Software must tell ELROY whether a particular PCI is dumb or smart, by programming the S bit in Error Config register. Error Config register is described in Section 7.16. If any card on a bus is dumb, software should designate that PCI as dumb. Also, the intended configuration with ELROY is to have just one PCI slot per ELROY. In that case, it may be acceptable to always mark the PCI as dumb.

ELROY PCI data error general behavior

For outbound data (i.e., PIO writes and DMA reads), the general approach is to always send data to PCI. If the data is known to be bad, it will be sent with poisoned parity. However, for PIO writes, if the error happens above ELROY, the data is discarded before it gets to PCI.

ELROY will drop DMA write data with bad parity for the initial data with bad parity and all subsequent data for that transaction. If the error took place on PCI, ELROY will signal the PCI card with PERR# for the initial data with bad parity and all subsequent data for that transaction.

If ELROY receives bad parity data on a PIO read, it will return -1 data or a HardFail response on the system bus for all data in that transaction.

ELROY will always flag bad data with bad parity to allow the PCI card to detect that.

ELROY will always detect, signal, and log data errors at PCI ports, and rope ports thus leaving a trail.

High Availability Systems

To guarantee data containment in all cases, all of the PCI cards must be at least an HA Safe card.

7.7 Fatal Mode behavior in Rope-master and ELROY

Both the rope-master and ELROY implement fatal modes. The one in the rope-master is called **Rope fatal mode**, while the one in ELROY is called **PCI fatal mode**. In the rest of this document, fatal mode, by default, refers to PCI fatal mode.

7.7.1 Rope fatal mode behavior

Rope Fatal mode is entered whenever an error is reported in RI (rope-master's rope interface error log). See the Rope-master's ERS for additional information.

Note: Rope fatal mode applies to individual rope interfaces.

Actions taken in rope fatal mode:

- Discard data for DMA writes from that rope.
- All data received after the rope goes fatal is discarded.
- Some of the data received before the rope went fatal may also be discarded.
- Stop supplying data for DMA reads.
- Return either -1 data with NormalData response or arbitrary data with HardFail response (based on the state of the HF bit) for PIO reads. Some read responses received before the rope went fatal may also turn into -1 or HardFail, per state of the HF bit. Peer to peer reads will always return a target abort on the source bus if the destination bus is in fatal mode, regardless of the state of the HF bit.
- Discard PCI data for:
  - PIO writes to memory or io port space
  - Peer to peer writes to memory or io port space
  - For non-posted IO port space writes, although the write data is discarded, a normal completion is returned to ensure that the transaction completes gracefully on the source bus.
- Allow access to all of the rope-maseter's registers.
- Registers for the ELROY below the fatal rope are not accessible, of course.
- The rope-master will put the rope in soft reset mode. This will reset the ELROY and all PCI devices below that ELROY. Also, the FIFOs in rope-master that talk to that rope only are reset by this reset. Contents of ELROY registers will be preserved. PCI has no concept of a soft reset, so PCI card registers will not be preserved.

Rope Fatal Mode Recovery

- The error logs should be cleared using the appropriate RopeN_control register described earlier.
- Flush the cache using the Flush-cache bit in the FLUSH_CTRL register.
- Purge the TLB Translation Cache using the PCOM register.
- The rope must be reset; also with the RopeN_control register. As described above, ELROY was already reset. The rope reset will bring ELROY out of reset.

- At this point, the rope fatal mode has ended. However, software should check the error logs in ELROY and take appropriate action on them before returning to normal operation.
- Some errors have the potential to corrupt registers. While this is a low probability, it would be a good practice to reinitialize ELROY registers after recovering from rope fatal mode.
- If recovery is not going to be attempted (i.e. ELROY is being deconfigured), the cache flush and TLB purge steps should still be performed

7.7.2 PCI fatal mode

PCI Fatal mode is entered whenever a fatal error is reported in PI (ELROY's PCI error log). Where error handling differs between smart and dumb PCI modes, the PI error log codes are listed separately as PI_S and PI_D respectively. Fatal errors are errors whose logging description contains 'fe' in the table. Fatal errors have log codes of 0x10 or above, non-fatal errors 0x0f or below.

Actions taken when PCI goes in fatal mode:

- When PCI goes in fatal mode, the current transactions will still be completed on PCI bus.
- Disable arbitration for that ELROY:
- Arbitration is disabled by clearing all the bits in the ELROY arbitration mask register.
- Return either –1 data with NormalData reponse or arbitrary data with HardFail response (based on the state of the HF bit) for PIO reads.
- Discard PCI data for:
- PIO writes to memory or io port space
- Peer to peer writes to memory or io port space
- For non-posted IO port space writes, although the write data is discarded, a normal completion is returned to ensure that the transaction completes gracefully on the source bus.
- Mask all interrupts from the PCI that is fatal. This is done by setting the mask bits in all entries of the I/O SAPIC in that ELROY.
- Allow access to all Rope-master and ELROY registers.
- PCI Configuration Reads and Writes will not be allowed.

The following must be done to recover from PCI fatal mode:

- Clear the error code.
- Flush the cache using the Flush-cache bit in the FLUSH_CTRL register.
- Purge the TLB Translation Cache using the PCOM register.
- Enable PIO cycles. This is done by writing a 1 to bit 0 of the Arbitration Mask register. Now, PCI configuration cycles and other PIO accesses may be performed in order to diagnose the error and take corrective action where possible.
- Enable DMA and interrupts for cards; this may or may not include the problem card.

If recovery is not going to be attempted (i.e. PCI is being deconfigured), the cache flush and TLB purge steps should still be performed.

7.8 Cache and TLB State after PCI fatal and Rope fatal

The TLB must be purged during the recovery from each rope or PCI fatal condition.

Translation cache entries fetched following a TLB miss are locked until they are either purged or used to successfully translate a cycle. A rope or PCI fatal condition can potentially leave an entry locked, reducing the useful size of the translation cache during the period after the fatal condition, before the cache purge in the recovery sequence. Since there are more TC entries than there are ropes to go fatal, this condition can never fully inhibit DMA forward progress.

When a rope or PCI goes fatal, cache lines may be left in an inconsistent state because a retried PCI cycle may never be completed.

The TLB and cache considerations described above for Rope and PCI fatal conditions also apply following a rope reset: TLB purge and cache flush precautions should be used following a rope reset, unless it can be guaranteed that all DMA traffic on a given rope was successfully quiesced before the software issued the rope reset. *The cache flush precaution should be performed after every rope reset, regardless of whether traffic was quiesced, since correct operation of the DMA read ordering scheme depends upon it.*

These precautions are not needed following a PCI reset, since PCI resets are only allowed when all traffic to or from the bus in question has been quiesced, and PCI resets do not reset the ELROY ordering counter.

7.9 Elroy Transactions

The error tables below include error logging information of the form "(PI_S: ERR_PARITY, D, corr, rs) Log code 2, TID[7:4], syndrome." The fields are:

Where detected. Valid values are:

RI_E	ELROY's rope interface
PI_S	PCI interface designated smart
PI_D	PCI interface designated dumb

Estat. This is supplied for reference only. Hardware will not log the Estat. It will log a unique error code for each individual entry below.

Additional status: A, D, or C. These indicate whether the error is on address, data or control. Hardware does not log this either. The information can be inferred for the error codes.

Severity. Values, in order of increasing severity, are: corr, unc, and fe. Hardware does not log severity explicitly, but it is implied by the error code. Hardware is aware of the severity in that it does not overwrite errors unless the subsequent error is of higher severity.
Requester/responder. Value of rq indicates that the chip logging the error was the requester in the transaction, rs indicates it was the responder. This is also inferred from error code and not actually logged. The 'Code' field is followed by a number that is unique for each error. Sometimes two numbers are assigned to the same error. This happens if the error has a different severity for smart vs. dumb PCIs. The error code is logged in the Error register. A value of 0 means that no error is logged. When an error has been logged in the Error register, it can be cleared with the SIC register. For the ELROY rope interface, a two-letter acronym identifies the bit in the error register that is set.

The remaining fields specify what other information is logged when that error takes place.

7.9.1 PIO Writes (including I/O port space)

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
Rope-master to ELROY rope parity error	OS/Driver/card	None	Hold ELROY in reset so that it stops responding on the rope. This will eventually cause the rope-master to invoke rope fatal mode (because PIO reads will not get a read return, or PIO writes will back up the CDF).	
	(RI_E: bit RP, err_parity, a/d, fe, rs?) Log:			
NOTE: Holding ELROY in reset is a distinct ELROY mode, which we may term *ELROY Rope Fatal Mode*. It is a transient rather than a steady-state condition, since it inevitably causes the rope-master to enter *Rope Fatal* mode, which in turn resets ELROY. It is used only to guarantee containment and force the rope-master into rope fatal mode. Note: The same error code can be logged during a PIO read or a DMA read				

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
ELROY Internal data	OS/Driver/card	PCI bus. Bad parity.	No Containment. PCI Fatal.	Containment.
	(PI_S: code 0x05, err_function, d, unc, rq) Log:			
	(PI_D: code 0x15, err_function, d, fe, rq) Log:			
	None	ELROY register.	Register updated, PCI goes fatal.	
	(PI_S: code 0x12, err_function, d, fe, rq) Log:			
	(PI_D: code 0x12, err_function, d, fe, rq) Log:			
Rationale: This group of errors can be caused by upstream errors. If any such error is logged, this error can be ignored. Otherwise, this error is definitely in ELROY. Either way, no additional logging is needed.				

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
PCI bus data error (ELROY observes PERR# asserted).	OS/Driver/card	PCI bus. Bad parity data.	No Containment. PCI Fatal.	Containment.
	(PI_S: code 0x03, err_error, d, unc, rq) Log:			
	(PI_D: code 0x13, err_error, d, fe, rq) Log:			
Rationale: If this error is logged, the rope-master must have sent good data and an actual parity error must have taken place on PCI. The card that detected it should set the 'Detected Parity Error' bit in its				

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
PCI Status register. This is to identify the requester and the responder.				

7.9.2 PIO reads (including I/O port space)

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
Rope-master to ELROY rope parity error	OS/Driver/card	–1 data or HardFail on System bus.	Hold ELROY in reset so that it stops responding on the rope. This will eventually cause the rope-master to invoke rope fatal mode (because PIO reads will not get a read return, or PIO writes will back up the CDF).	
	(RI_E: bit RP, err_parity, a/d, fe, rs?) Log:			
NOTE: Holding ELROY in reset is a distinct ELROY mode, which we may term *ELROY Rope Fatal Mode*. It is a transient rather than a steady-state condition, since it inevitably causes the rope master to enter *Rope Fatal* mode, which in turn resets ELROY. It is used only to guarantee containment and force the rope-master into rope fatal mode. Note: The same error code can be logged during a PIO write or a DMA read.				

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
PCI data parity	OS/Driver/card	System bus. -1 data or HardFail response. Bad PCI byte modifies all data requested in the transaction.	ELROY asserts PERR# for the initial and consecutive data words. PCI Fatal.	ELROY asserts PERR# for the initial and consecutive data words.
	(PI_S: code 0x06, err_parity, d, unc, rq) Log: address			
	(PI_D: code 0x16, err_parity, d, fe, rq) Log: address			
Note: If the transaction is a PCI config cycle, the logged address is the address of the config data register, not the value in the config address register. In other words, it is the address that software read from, not the address that appeared on PCI.				

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
ELROY internal data - Write Posting FIFO	OS/Driver/card	Rope. Bad parity data.		
	(RI_E: bit EI, err_function, d, unc, rs) Log:			
Rationale: Sending poisoned parity on the rope ensures that the rope goes fatal through ELROY to rope-master rope parity error (see below). The purpose of logging this error is to facilitate isolation. Note: This error is also listed in the DMA write table.				

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
ELROY to Rope-master rope parity error	OS/Driver/card	System bus. -1 data or HardFail response for all data requested in the transaction.	Rope fatal mode	
	(RI: code 0x11, err_parity, d, fe, rq) Log:			
Note: This error is also listed in the DMA write and DMA read tables.				

7.9.3 DMA writes

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
PCI data parity	OS/Driver/card	No data sent to the rope-master for the initial PCI data phase with bad parity and for all consecutive data words in that transaction.	ELROY asserts PERR# for initial and consecutive data words. PCI Fatal. DMA write or interrupt data discarded for any subsequent transaction.	ELROY asserts PERR# for initial and consecutive data words.
	(PI_S: code 0x04, err_parity, d, unc, rs) Log:			
	(PI_D: code 0x14, err_parity, d, fe, rs) Log:			
NOTE: In the Dumb PCI case, PCI fatal mode can't disable arbitration quickly enough to prevent another DMA write or interrupt transaction from starting on the PCI bus. Any such transaction will complete on PCI, but be discarded. *Rationale: The PCI card will set 'Data Parity Error Detected' bit in PCI status register. That identifies the requester and rope-master is known to be the responder.*				

<table>
<tr><th>Error Type</th><th>Recovery Type</th><th>Target Bus Data Value</th><th>Action (Dumb)</th><th>Action (Smart)</th></tr>
<tr><td rowspan="2">ELROY internal data - Write Posting FIFO</td><td>OS/Driver/card</td><td>Rope. Bad parity data.</td><td></td><td></td></tr>
<tr><td colspan="4">(RI_E: bit EI, err_function, d, unc, rq) Log:</td></tr>
<tr><td colspan="5">Rationale: Sending poisoned parity on the rope ensures that the rope goes fatal through ELROY to rope-master rope parity error (see below). The purpose of logging this error is to facilitate isolation.
Note: This error is also listed in the PIO read table.</td></tr>
</table>

7.9.4 DMA reads

<table>
<tr><th>Error Type</th><th>Recovery Type</th><th>Target Bus Data Value</th><th>Action (Dumb)</th><th>Action (Smart)</th></tr>
<tr><td rowspan="2">Rope-master to ELROY rope parity error</td><td>OS/Driver/card</td><td>PCI bus. Bad parity for the 32 bit word that has one or more bad parity bytes.</td><td colspan="2">Hold ELROY in reset so that it stops responding on the rope. This will eventually cause rope-master to invoke rope fatal mode (because PIO reads will not get a read return, or PIO writes will back up the CDF).</td></tr>
<tr><td colspan="4">(RI_E: bit RP, err_parity, a/d, fe, rq?) Log:</td></tr>
<tr><td colspan="5">NOTE: Holding ELROY in reset is a distinct ELROY mode, which we may term ELROY Rope Fatal Mode. It is a transient rather than a steady-state condition, since it inevitably causes the rope-master to enter Rope Fatal mode, which in turn resets ELROY. It is used only to guarantee containment and force the rope-master into rope fatal mode.
Note: The same error code can be logged during a PIO write or a PIO read.</td></tr>
</table>

<table>
<tr><th>Error Type</th><th>Recovery Type</th><th>Target Bus Data Value</th><th>Action (Dumb)</th><th>Action (Smart)</th></tr>
<tr><td rowspan="3">ELROY internal data</td><td>OS/Driver/card</td><td>PCI bus. Bad parity for the 32 bit word that has one or more bad parity bytes.</td><td>No Containment. ELROY observes PERR#.
PCI Fatal.</td><td>Containment. ELROY observes PERR#.</td></tr>
<tr><td colspan="4">(PI_S: code 0x08, err_function, d, unc, rs) Log:</td></tr>
<tr><td colspan="4">(PI_D: code 0x18, err_function, d, fe, rs) Log:</td></tr>
</table>

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
PCI bus data error.	OS/Driver/card	PCI bus. Bad parity data.	No Containment. ELROY observes PERR#. PCI Fatal.	Containment. ELROY observes PERR#.
	(PI_S: code 0x09, err_error, d, unc, rs) Log:			
	(PI_D: code 0x19, err_error, d, fe, rs) Log:			
Rationale: The PCI card will set 'Data Parity Error Detected' bit in PCI status register. That identifies the requester and ELROY is known to be the responder. No additional logging by ELROY.				

Notes:

No containment indicates that rope-master has sent the data with bad parity to the PCI card and it cannot guarantee that the card will not use the data.

7.10 Rope command errors

A number of rope errors were already described in preceding tables. However, these do not cover errors in rope commands. Rope commands pass a variety of control information between the rope-master and ELROY. An parity error in these commands is catastrophic and can not be attached to any particular transaction type. These errors are described below.

Error Type	Recovery Type	Target Bus Data Value	Action (Dumb)	Action (Smart)
Rope-master to ELROY rope command parity error	OS/Driver/card	None	Hold ELROY in reset so that it stops responding on the rope. This will eventually cause the rope-master to invoke rope fatal mode (because PIO reads will not get a read return, or PIO writes will back up the CDF).	
	(RI_E: bit RC, err_error, c, fe, rs/rq) Log:			
NOTE: Holding ELROY in reset is a distinct ELROY mode, which we may term *ELROY Rope Fatal Mode*. It is a transient rather than a steady-state condition, since it inevitably causes the rope-master to enter *Rope Fatal* mode, which in turn resets ELROY. It is used only to guarantee containment and force the rope-master into rope fatal mode.				

7.11 PCI Bus Errors

Error Type	Recovery Type	Target Bus Data Value	ELROY Action	PCI Action

<table>
<tr><th>Error Type</th><th>Recovery Type</th><th>Target Bus Data Value</th><th>ELROY Action</th><th>PCI Action</th></tr>
<tr><td rowspan="3">(ELROY Target) PCI Address / Command parity error.</td><td>OS/Driver</td><td>none. (ELROY completes with a Target-abort)</td><td>Assert SERR#.
PCI Fatal if dumb PCI.</td><td></td></tr>
<tr><td colspan="4">(PI_S: code 0x0B, err_parity, a, unc, rs) Log: Master ID (see note 1.)</td></tr>
<tr><td colspan="4">(PI_D: code 0x1B, err_parity, a, fe, rs) Log: Master ID (see note 1.)</td></tr>
<tr><td colspan="5">Note: In the case of an address parity error, the PCI spec allows either a target-abort or no response (which would lead to a master-abort). In ELROY, target-abort was an easier implementation, so that was chosen.</td></tr>
</table>

<table>
<tr><th>Error Type</th><th>Recovery Type</th><th>Target Bus Data Value</th><th>ELROY Action</th><th>PCI Action</th></tr>
<tr><td rowspan="3">(ELROY Initiator) PCI no DEVSEL#.</td><td>OS/Driver</td><td>-1 data or HardFail to System agent for Reads. Write data discarded</td><td>Master Abort. Return -1 data or HardFail for Reads.</td><td></td></tr>
<tr><td colspan="4">(PI_S: code 0x0C, err_timeout, unc, rq) Log: address</td></tr>
<tr><td colspan="4">(PI_D: code 0x1C, err_timeout, fe, rq) Log: address</td></tr>
<tr><td colspan="5">Note: If the transaction is a PCI config cycle, the logged address is the address of the config data register, not the value in the config address register. In other words, it is the address that software read from, not the address that appeared on PCI.</td></tr>
</table>

<table>
<tr><th>Error Type</th><th>Recovery Type</th><th>Target Bus Data Value</th><th>ELROY Action</th><th>PCI Action</th></tr>
<tr><td rowspan="3">(ELROY Initiator) Target-abort from PCI.</td><td>OS/Driver</td><td>-1 data or HardFail to System agent for Reads. Write data discarded.</td><td>Set ‘Received Target Abort’ bit in Status register, per PCI spec.</td><td></td></tr>
<tr><td colspan="4">(PI_S: code 0x0D, err_error, unc, rq) Log: address</td></tr>
<tr><td colspan="4">(PI_D: code 0x1D, err_error, fe, rq) Log: address</td></tr>
<tr><td colspan="5">Note: If the transaction is a PCI config cycle, the logged address is the address of the config data register, not the value in the config address register. In other words, it is the address that software read from, not the address that appeared on PCI.</td></tr>
</table>

Error Type	Recovery Type	Target Bus Data Value	ELROY Action	PCI Action

Error Type	Recovery Type	Target Bus Data Value	ELROY Action	PCI Action
(ELROY Target) ELROY signals Target-abort..	OS/Driver		Set 'Signaled Target Abort' bit in Status register, per PCI spec.	
	(PI_S: code 0x0A, err_error, unc, rq) Log:			
	(PI_D: code 0x1A, err_error, fe, rq) Log:			

Error Type	Recovery Type	Target Bus Data Value	ELROY Action	PCI Action
Any PCI device asserts LOCK#.	OS/Driver	N/A	Access completed without atomicity.	
	(PI_S: code 0x0E, err_improper, unc, rs) Log: MasterID			
	(PI_D: code 0x0E, err_improper, unc, rs) Log: MasterID			
Rationale: This is an advisory error. The OS will decide what to do about the card that tried to assert LOCK#.				

Error Type	Recovery Type	Target Bus Data Value	ELROY Action	PCI Action
Any PCI device asserts SERR#. (Possibly signaling an address parity error.)	OS/Driver	N/A	Observe SERR#. PCI Fatal if dumb.	
	(PI_S: code 0x0F, err_error, unc, rs) Log:			
	(PI_D: code 0x1F, err_error, fe, rs) Log:			
Rationale: ELROY can not differentiate between address parity errors and other SERR#s. However, if it was an address error, the card asserting SERR# should set both the 'Signaled System Error' and the 'Detected Parity Error' bits. Otherwise, only the first bit will be set. This is how software/firmware can tell these apart.				

Error Type	Recovery Type	Target Bus Data Value	ELROY Action	PCI Action
PCI card requesting the bus fails to use it.	OS/Driver	N/A	Negate GNT# to that card.	A card asserts REQ#, is given GNT# but it fails to assert FRAME# within 16 cycles.
	(PI_S: code 0x01, err_timeout, corr, rs) Log: Master ID (see note 1.)			
	(PI_D: code 0x01, err_timeout, corr, rs) Log: Master ID (see note 1.)			

Error Type	Recovery Type	Target Bus Data Value	ELROY Action	PCI Action
Rationale: PCI spec says that if a card requests the bus and is granted the bus, it must start a transaction by asserting FRAME# within 16 cycles after GNT# was asserted. If not, the card may be assumed to be broken and the arbiter may stop granting it the bus. While ELROY itself does not disable arbitration for the errant card in this situation, it is advisable for software to do so.				

Notes:

Master ID specifies which card was the bus master at the time the error took place. Master ID is generated by ELROY's internal arbiter. The Master ID will not be meaningful if an external arbiter is being used. Master ID is logged in Error_Log_Master_ID register described in section 11.4.2.

7.12 Interrupt transactions

Interrupt transactions moving through the I/O sub-system are treated like write transactions. The result is that the portion of the interrupt transaction that shows up on the data bus is parity protected and parity errors will cause bad parity to be supplied on the system bus. Generally, all the errors described in Table 7-6 (for DMA writes) can also be logged on interrupts. I.e., when a DMA write related error code is seen, it is possible that it was actually generated by an interrupt transaction.

7.13 Forcing PCI parity errors

ELROY provides a mechanism to force inbound and outbound parity errors. This can be useful in testing error behavior of ELROY and of PCI cards connected to it. There are 4 separate bits in the Error Config register (defined in Section 7.16) to introduce parity errors for the following cases:

- PIO writes (ELROY drives bad parity data). Enabled when PW bit is 1.
- PIO reads (ELROY inverts incoming parity). Enabled when PR bit is 1.
- DMA writes (ELROY inverts incoming parity). Enabled when DW bit is 1.
- DMA reads (ELROY drives bad parity data). Enabled when DR bit is 1.

Note that the rope-master can not guarantee a precise relationship between assertion of these bits and forcing of parity errors. Consider the following sequence:

- PIO write #1 (to PCI)
- Reg write to force parity error on PIO writes
- PIO write #2 (to PCI)

If this sequence is executed back to back, there is no guarantee that PIO write #1 will not get a forced parity error and that PIO write #2 will get one. The delay in having the reg write take effect is design dependent and no promises are being made at this time. To get predictable behavior, set the parity forcing bits while the bus is idle, read the Error Config register and initiate other transactions only after the register read is completed.

7.14 Error Config register

The format of the Error Config register is shown below. Its address is described in the ELROY ERS.

MSB	**Error Config Registers OFFSET: 0x0680**																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																										S	CM	DR	DW	PR	PW
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits 3 thru 0 were defined in Section 7.13. The other bits are:

- S: Smart / not Dumb bit. When this bit is a 1, the bus is considered smart. If it is 0, it is considered dumb.

- CM: Config Cycle Hard Fail Mask. When this bit is a 1, config cycles with no devsel do not cause the bus to go fatal, do not log an error, and return a -1 if it was a config cycle read. If it is a 0, ELROY will not differentiate between configuration cycles and PIO cycles with respect to the s bit and hf bit.

8. ARBITRATION

8.1 Operation Overview

ELROY's PCI arbiter will arbitrate between 8 PCI masters -- 7 external masters plus ELROY's PMC. The arbitration algorithm is based on a two priority round-robin system. Agents in the high priority class have equal access to the bus with respect to each other. Agents in the low priority class also have equal access with respect to each other. The difference between priorities is that masters within the low priority class as a group have the same access to the bus as each individual high priority agent.

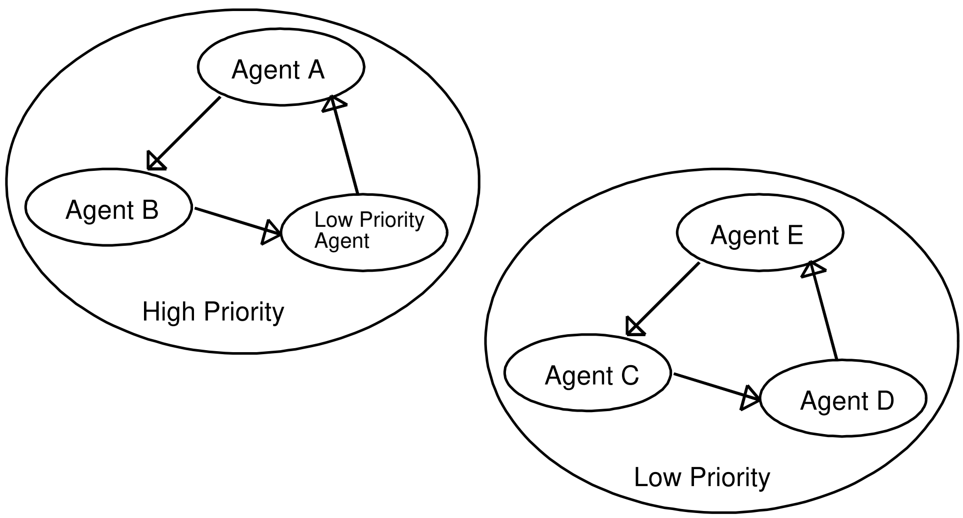

Figure 12. Example of ELROY Arbitration

In the example shown in the figure above, if all agents asserted their REQ#'s at the same time, the arbiter would assert GNT#'s in the following order: A, B, C, A, B, D, A, B, E, etc. Note that an agent will not be granted the bus if it does not request it.

Agents can be assigned either a High or Low priority by setting bits in the Arbitration Priority register. In addition, software can mask out the REQ# signal from any PCI device so that faulty cards can be prevented from mastering the bus by setting bits in the Arbitration Mask register.

There are three distinct types of cases that the arbiter has to deal with, depending on how many agents are requesting the bus at once. The first case is multiple agents requesting at once, the second only one agent requesting, and the third no agents requesting.

If multiple agents are requesting the bus then the GNT# signal to the agent granted will only stay asserted until the agent begins a transaction. Once the agent granted the bus starts a transaction the GNT# to that agent is removed and the arbiter will we grant the bus to the next agent in priority order. If the bus is idle for more than 16 clocks after GNT# has been issued, then then the arbiter assumes that card is not responding properly and it removes the GNT# to prevent the bus from hanging. The arbiter will also send the 'stall' error signal to the error block which will log the appropriate error code.

If only one agent is requesting the bus, the arbiter will keep that agents GNT# asserted as long as is it continues to request the bus to enable it to do multiple transactions. When the last agent deasserts it's REQ#, the arbiter stays parked on that agent until it finishes the transaction in progress. At that point when the PCI bus is IDLE and no agent is requesting it, the arbiter will park the bus on ELROY's PMC by asserting the PMC GNT signal internal to ELROY. The performance advantage to park on ELROY's PMC when the PCI bus is IDLE is ELROY can deassert the PMC's GNT# in the same clock cycle it asserts another master's GNT#. Normally, this could not be done when the PCI bus is IDLE because a master that has the bus parked on it could be driving the AD lines. But, since ELROY's PMC has been designed to not do this, there is no danger of bus contention.

In order to better support devices that need to preform many short transactions, a special mode of the arbiter (called Limited Unfair Arbitration) can be used to grant the bus to the same device for several consecutive PCI transactions. A Multi-transaction Latency Timer in the arbiter will control the total number of PCI clocks that these multiple transactions can take. This timer value is set by the Multi-transaction Latency Timer register. If the master deasserts REQ# or if the arbiter's Multi-transaction Latency Timer expires or if the bus is IDLE for more than 16 cycles, then the bus will be granted to the next master. Each agent can be set to normal, Limited Unfair, or Unlimited Unfair mode (to be described later) independently but there is only one latency timer that is shared between all the external masters. The PMC has it's own Latency timer implemented within it's block. This enables different timeouts between the PMC and external devices.

8.2 Arbitration Control Registers

The registers described in the following sections control the specific ELROY arbitration features described above.

8.2.1 Arbitration Mask

The arbitration mask allows software to prevent specific bus masters from requesting to master the bus. On power up, all masters will be disabled, and must be enabled by software after autoconfiguration has completed.

A mask bit set to 0 will disable arbitration for that device; a mask bit set to 1 will enable arbitration. When there is a fatal error, ELROY will go into fatal error mode where all arbitration will be disabled. When Enable Arb (1'b0) is disabled, it will enter a special mode where the PMC will throw away all the data for PIO writes and return a value of -1 for PIO reads. In this mode, registers remain accessible except no configuration cycle will be generated in the PCI bus. This special mode keeps the command data fifo empty and the software will be able to turn ELROY's arbitration back on later. This mode also disables all other mask bits while it is a zero.

<table>
<tr><td>MSB</td><td colspan="30">Arbitration Mask Register
OFFSET: 0x0080</td><td>LSB</td></tr>
<tr><td>63</td><td>62</td><td>61</td><td>60</td><td>59</td><td>58</td><td>57</td><td>56</td><td>55</td><td>54</td><td>53</td><td>52</td><td>51</td><td>50</td><td>49</td><td>48</td><td>47</td><td>46</td><td>45</td><td>44</td><td>43</td><td>42</td><td>41</td><td>40</td><td>39</td><td>38</td><td>37</td><td>36</td><td>35</td><td>34</td><td>33</td><td>32</td></tr>
<tr><td colspan="32"></td></tr>
<tr><td colspan="32">RESET INITIALIZATION</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="24">Reserved</td><td>Mask G</td><td>Mask F</td><td>Mask E</td><td>Mask D</td><td>Mask C</td><td>Mask B</td><td>Mask A</td><td>Enable Arb</td></tr>
<tr><td colspan="32">RESET INITIALIZATION</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
</table>

Field	Description	Reset Value
Reserved	Must set to 0 for writes, must be ignored for reads.	Reserved
MaskG	Mask for PCI device G (read/write).	1’b0
MaskF	Mask for PCI device F (read/write).	1’b0
MaskE	Mask for PCI device E (read/write).	1’b0
MaskD	Mask for PCI device D (read/write).	1’b0
MaskC	Mask for PCI device C (read/write).	1’b0
MaskB	Mask for PCI device B (read/write).	1’b0
MaskA	Mask for PCI device A (read/write).	1’b0
Enable Arb	If zero, disables all requests and puts PMC in fatal mode. (read/write).	1’b1

Table 25. Arbitration Mask Register

8.2.2 Arbitration Priority

The arbitration priority register allows software to individually set the priority of each PCI master controller. A priority bit set to 0 indicates a Low Priority; a priority bit set to 1 indicates a High Priority.

MSB	Arbitration Priority Register OFFSET: 0x0088																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RESET INITIALIZATION																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																								Pri G	Pri F	Pri E	Pri D	Pri C	Pri B	Pri A	Pri PMC
RESET INITIALIZATION																															
																								0	0	0	0	0	0	0	0

Field	Description	Reset Value
Reserved	Must set to 0 for writes, must be ignored for reads.	Reserved
PriG	Priority for PCI device G (read/write).	1’b0
PriF	Priority for PCI device F (read/write).	1’b0
PriE	Priority for PCI device E (read/write).	1’b0
PriD	Priority for PCI device D (read/write).	1’b0
PriC	Priority for PCI device C (read/write).	1’b0
PriB	Priority for PCI device B (read/write).	1’b0
PriA	Priority for PCI device A (read/write).	1’b0
PriPMC	Priority for ELROY’s PCI Master Controller (read/write).	1’b0

Table 26. Arbitration Priority Register

8.2.3 Arbitration Mode

The arbitration mode register allows ELROY to be configured to support Normal Arbitration and Limited Unfair Arbitration. These features allow selected devices to perform multiple transactions and allow an ELROY PCI bus to have more than the normal seven devices by using logic external to the ELROY chip.

Both modes are selected on a per agent basis. Each master’s arbitration mode is independent of the other master’s modes.

The “Arbitration Mode” is a two-bit field in the Arbitration Mode Register. See Table 28 for the proper values to put in this field.

MSB	Arbitration Mode Register OFFSET: 0x0090																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															EA	AMG		AMF		AME		AMD		AMC		AMB		AMA		AMPMC	
RESET INITIALIZATION																															
															0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Field	Description	Reset Value
Reserved	Must set to 0 for writes, must be ignored for reads.	Reserved
EA	Must be a 0.	1’b0
AMG	Arbitration Mode for PCI device G (read/write).	2’b00
AMF	Arbitration Mode for PCI device F (read/write).	2’b00
AME	Arbitration Mode for PCI device E (read/write).	2’b00
AMD	Arbitration Mode for PCI device D (read/write).	2’b00
AMC	Arbitration Mode for PCI device C (read/write).	2’b00
AMB	Arbitration Mode for PCI device B (read/write).	2’b00
AMA	Arbitration Mode for PCI device A (read/write).	2’b00
AMPMC	Arbitration Mode for ELROY’s PMC. Hard-wired to 10.	2’b10

Table 27. Arbitration Mode Register

Value	Description
00	Normal Arbitration.
01	Limited Unfair Arbitration.
10	Reserved.
11	Reserved.

Table 28. Arbitration Mode Settings

8.2.4 Multi-transaction Latency Timer

This register specifies the value of the Latency Timer used in Limited Unfair Arbitration mode. The units are in PCI clocks. The arbiter will start counting when the master has been granted the bus. The arbiter will take away grant when the timer expires and the master has started a transaction. If the timer expires in an IDLE cycle, the arbiter will wait for the master to start another transaction before giving the grant away. The 16 clock idle timer will, remove the grant if no transaction results. Setting the timer to zero has the same effect as changing the arbitration mode to normal mode, i.e. the arbiter will take away grant as soon as the master started a transaction.

MSB	Multi-transaction Latency Timer OFFSET: 0x0098																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																								Latency Timer							
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x00							

Field	Description
Read Only	Return 0's on Reads, Writes have no effect.
Latency Timer	In units of PCI clocks. Initialized to 0 after Reset (read/write).

Table 29. Arbiter Multi-transaction Latency Timer

9. PCI INTERFACE

9.1 Overview

ELROY's PCI bus is a 64 bit bus capable of running at either 30-33 or 60-66MHz. PCI protocol specifies that the data bus width is negotiated on a per transaction basis. So, the transactions could be either 32 bit or 64 bit. ELROY's PCI is compatible with both 5V and 3.3V signaling environments specified in the PCI spec. This is achieved by having the output drivers always run off 3.3V supply but making the pad circuitry tolerant of 5V inputs. ELROY has an integrated arbitration controller (arbiter) that will support up to 7 bus masters with no additional external logic.

See the PCI Master Controller, PCI Slave Controller , PCI Arbiter, and Interrupt chapters for more detailed information on ELROY's PCI interface.

9.2 Features

- Ropes to PCI bridge
- 32 bit or 64 bit operation.
- 30-33 or 60-66 MHZ operation
- 3V output drivers with 5V tolerant pads
- Programmable drive strength in the PCI output drivers (to allow improved electrical characteristics)
- Support for 64 bit addressing
- Compliance to **PCI Local Bus Specification**, revision 2.1
- Flexible pre-fetch hint table for optimal DMA performance
- Parity on internal data path, including fifos
- Medium speed decoder (Fast decoder when master is 64 bit and sends 64 bit address)

9.3 Arbitration Features

- Arbitration for 7 masters (in addition to ELROY itself)
- Any PCI Master can be granted the bus for consecutive transactions (using a Latency Timer in the Arbiter)
- Programmable arbitration priority for each Master (High and Low)

9.4 Transaction Types

PCI Bus Command	PCI Transaction	ELROY Master Controller Generates	ELROY Slave Controller Responds
0000	Interrupt Acknowledge	yes	no
0001	Special Cycle	no	no
0010	I/O Read	yes	yes[1]
0011	I/O Write	yes	yes[1]
0100	Reserved	no	no
0101	Reserved	no	no
0110	Memory Read	yes	yes
0111	Memory Write	yes	yes
1000	Reserved	no	no
1001	Reserved	no	no
1010	Configuration Read	yes	no
1011	Configuration Write	yes	no
1100	Memory Read Multiple	no	yes
1101	Dual Address Cycle	yes	yes
1110	Memory Read Line	no	yes
1111	Memory Write and Invalidate	no	yes

[1] This is not supported and should be disabled. See section 15.3.2 for details.

Table 2. PCI Transaction Support

9.5 Unsupported PCI Signals

The following signals are not supported by ELROY:

- LOCK# (The signal exists, but it doesn't lock anything. If a PCI card asserts this signal ELROY will log an error. See the errors chapter for details.)
- SBO#
- SDONE

9.6 Selection of PCI clock frequency

After a platform reset, the PCI bus will be started at 30-33 MHz. If appropriate, it should be switched to 60-66 MHz. The algorithm for deciding that depends on whether the slot is OLR capable or not. For an OLR capable slot, do the following:

1. Read the state of M66EN from OLAR_CNTL register.
2. If the bit is 1, switch the bus to 60-66 MHz using the CLK_CNTL register described next.

For non-OLR capable slots, do the following:

1. At power up, start the bus at 30-33 MHz.
2. Probe the configuration space to find out which slots are populated. For slots that are populated, find out if the device is 60-66 MHz capable.
3. If all devices are 60-66 MHz capable, switch to 60-66 MHz.

Caveat: There could be system specific considerations here. For example, because of bus length, a particular bus may be incapable of running at 60-66 MHz even if the devices plugged into it are 60-66 MHz capable. Firmware picking the frequency may need system specific knowledge to make this determination.

9.7 Programmable drive strength

To get the best signal integrity for signals driven by ELROY, the drive strength of the PCI outputs in ELROY is programmable. It is controlled through the PCI_drive register described below. The board designers must determine the appropriate value for the PCI_drive register of each ELROY. Firmware should initialize this value after a platform reset. Higher numbers correspond to a stronger drive strength.

MSB	**PCI_drive Register OFFSET:0x0608**																														LSB
63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved																													PCI_drive		
RESET INITIALIZATION																															
00000000																													3'b111		

10. PCI Slave Controller (PSC)

10.1 Operation Overview

The PCI Slave Controller does the control for ELROY responding as a PCI target as well as the control for inbound data. PCI target transactions include DMA, peer-to-peer, and interrupts. The PSC writes all inbound data into the Write Posting FIFO (WPF). For exmaple, DMA write data and PIO read return data are handled by the PSC. It also is responsible for recording errors so the operating system can take appropriate recovery steps.

The PSC also supports a HINT lookup table. Bits in this table are sent across the rope to the cache along with the read or write address to give the cache hints on the best prefetch and/or flush policy to use for each PCI device. The operation of the HINT look-up table is described in this chapter. The use of the HINTS is described in the rope-master's cache chapter.

10.2 Major Sub-blocks

10.2.1 Address Decoder

The address decoder will determine when the PSC should assert the PCI signal DEVSEL# to claim an access to this target. Accesses that are claimed include Memory Space and I/O Space transactions. (I/O space is not currently supported and should be disabled in the PCI control register).

10.2.1.1 Memory Space

PCI memory space includes system memory and memory-mapped I/O. Although ELROY can master memory-mapped I/O transactions, it is not capable of being the target for memory-mapped I/O (PCI to PCI). ELROY assumes that all memory-mapped I/O that takes place is destined to the same PCI bus, and hence it does not need to intervene. The only memory space transactions that are claimed by the PSC are accesses to main memory (DMA) and transaction based interrupts.

Range registers are used to define the regions of memory out of the 64-bit address space that correspond to memory-mapped I/O. Transactions in this memory range are not claimed by the PSC. All remaining memory transactions are assumed to be DMA and are claimed by the PSC. The start of these memory regions must be naturally aligned to memory based on the size of the region. For example, an 8 MB region must be aligned on an 8 MB boundary. The smallest size a memory region can be is 1 MB. Therefore, the 20 least significant bits in the range registers are ignored. See 3.6 for more detailed information on these registers.

Two pairs of range registers will define the memory space for all PCI busses in the system. One maps the 32-bit memory-mapped I/O space, and the other maps the 64-bit space (see section 5.1 for details on the memory map). In addition, one PCI bus in the system may be configured to decode the 128K VGA frame buffer region from 640K to 768K for compatibility with obsolete operating systems and software. A configuration register will be used to enable decoding of this range.

10.2.1.2 I/O Space

ELROY does not support being a target for I/O space transactions. The I/O range should always be disabled in the PCI control register. Enabling it will result in undefined behaviour.

10.2.1.3 Parity Generator/Error Log

The parity generator is responsible for computing parity on the AD[31:0] + C/BE[3:0] lines (PAR) and on the AD[63:32] + C/BE[7:4] lines (PAR64). PAR and PAR64 are driven by the same agent that drove the AD[63:0] lines. If a parity error was detected, PERR must be driven by the receiving agent for data transfers. SERR must be driven by any agent that notices a parity error during the address phase.

This block is also responsible for recording that a PERR or SERR occurred so that the operating system can take whatever steps are necessary for error containment and recovery. This error reporting will be implemented as an error register that will record the type of error that occurred and the address of the current transaction.

Byte parity is passed along all the internal paths of the rope-master. Therefore, this block will compute valid parity for each PCI data byte received to pass onto the write posting FIFO. In order to not propagate bad data, any data received from PCI with bad parity will be “poisoned” with bad parity when passed onto the write posting FIFO. Likewise, data received from the system bus with bad parity will be passed onto PCI with bad parity. PCI cards are responsible for not using data received with bad parity.

10.2.1.4 Hint lookup table

The hint lookup table provides the rope-master and ELROY hardware with performance optimizing hints about the current PCI transaction. This used to be a TLB function in the original definition, but is now a part of the PSC. More on this later.

10.3 PSC Registers

This section contains all the PSC’s registers.

All registers are equal to 0 after a reset. Reserved fields must be set to 0 for Writes and ignored for reads.

10.3.1 DMA Connection Control Register

The DMA Connection Control Register controls how the PSC behaves during DMA Read transactions when the requested data is not in the cache. It contains the ODC and ILT fields.

The ODC field (Optional Disconnect Control) determines how the PSC responds to DMA read attempts. On the first DMA read attempt, the rope-master will check its cache to see if the data is present. If it is not, then the rope-master will send a RETRY command down the rope to ELROY. This RETRY command is an *optional* retry (also known as an optional disconnect), meaning that ELROY does not have to retry the transaction on PCI (it may choose to remain connected).

There are three possible settings for ODC. The first is to ignore the optional disconnect. This setting has the possibility of violating the PCI rev 2.1 requirement that host bus bridges only insert 32 wait states before returning the requested data. The rational behind this requirement is that the PCI bus should not be unnecessarily tied up by wait states. Of course, this should not be a concern on single slot busses.

A second ODC setting is to disconnect only if some other master is requesting the bus. This has the advantage that the PSC will not keep retrying the DMA read until the data is available in the cache. In addition, it does not suffer from the disadvantage that another PCI device will have to unnecessarily wait for the data to arrive.

The third ODC setting is to always disconnect in response to a rope RETRY command. This will insure that the PSC does not violate the latency rule of PCI rev 2.1, but provides no guarantee of forward progress.

The ILT (Initial Latency Timer) controls the maximum number of PCI wait states added for DMA reads. The only possible settings for this are 16 wait states, and 32 wait states. Alternatively, the ITL may be disabled by clearing the ILTE bit.

The SLT (subsequent latency timer) bits control the maximum number of PCI wait states added for non-initial data phases of DMA Reads. After the first DMA Read data phase completes, if the data for the next data phase is not present in the cache, this field controls how long the PSC will wait for the data to arrive. (Note that if prefetching is disabled for this transaction, then the PSC will disconnect immediately.)

For example, if the SLT is set to 7, the PSC will wait for 8 PCI clocks before disconnectiong. If the data arrives on the 8^{th} clock or sooner, the PSC will return the data and not disconnect. Note that after the data arrives in the cache, it still takes another PCI clock to get the data out onto the PCI bus, so if the data arrives on the last possible clock (8 in the example above), then another wait state is added. This means that the number of wait states is equal to (SLT+1) if the data did not arrive just in time, and (ITL+2) if it did. Therefore, to meet the PCI rev 2.1 requirement that non initial data phases have no more then 8 wait states, set the SLT to 6 or less.

DMA Connection Control Register
OFFSET: 0x0278

MSB																															LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																ODC		Reserved								ILTE	ILT	SLT			
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Register 14-1 DMA Connection Control

Field	Description
Reserved	Must be set to 0 for writes, must be ignored for reads.
ODC	Optional Disconnect Control. Response to the optional disconnect command received by the PSC over the rope is: 00 Ignore optional disconnect 01 Always disconnect 10 Disconnect only if some master other than the current is requesting the bus (including the PMC) 11 Undefined
ILTE	Initial Latency Timer Enable. If set to 1, the Initial Latency Timer is enabled; if set to 0 it is disabled. Set this field to 1 to be compliant with PCI rev 2.1. However, setting it to 1 provides no forward progress gaurentee.
ILT	Initial Latency Timer. Controls the maximum number of PCI wait states from the assertion of FRAME to the first data transfer of the transaction. If this bit is set to 1, the Initial Latency Timer is set to 32; if set to 0 it is set to 16.

SLT	Subsequent Latency Timer. Controls the maximum number of PCI wait states in DMA Read data phases after the first data phase of the transaction. This timer is always enabled. Set this field to 6 or less to be compliant with PCI rev 2.1.

10.3.2 Module Information Register (MOD_INFO)

The MOD_INFO register is read-only and is defined as follows:

MSB	MOD_INFO Register OFFSET: 0x0100																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																Module ID															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Register 10-2 MOD_INFO Register

Bits 15:0 define the Module ID. The value of this field is 0x0005.

10.3.3 Status Information and Control Register (STATUS_CONTROL)

This register is used to reset the PCI bus, control Elroy's response to error conditions in PIO Reads, clear the error log, and activate the VGA address range on this PCI.

MSB	STATUS_CONTROL Register OFFSET: 0x0108																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved for status																															Rc
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Reserved for Control																									Hf	Ce	Cl	Ve	Unused		Rf
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register 10-3 STATUS_CONTROL Register

Field	Description
Status	
Rc	Reset complete. This read only bit equals 1 while the PCI bus is held in reset. (See rf bit below.)
Control	
Hf	HardFail response control. If hf = 1, HardFail response is returned on system bus if there is an error on PIO Reads (such as parity error, master-abort, target-abort, or the rope-master is in fatal mode). If hf = 0, then a -1 is returned instead.
Ce	Clear Error Log Enable. Clearing of error logs is enabled if ce = 1. If ce = 0, error log clearing is disabled. Whenever a new error is logged by this function, the ce bit is reset to 0.
cl	Clear Error Log. If a 1 is written to the cl bit and ce = 1, then the error log is cleared. Otherwise, no action is taken.
ve	VGA Enable. If ve = 1, VGA frame buffer and IOP space accesses will never be claimed by ELROY, regardless of range register programming. This bit must be set if a VGA device is located on this PCI bus. Only one

	ELROY in the system should have its ve bit set. If ve = 0, ELROY may claim VGA space, depending on range register programming.
rf	Reset Function. Writing a 1 to this bit resets the PCI bus. Bus will be held in reset until a subsequent write sets this bit to 0. This reset is intended to be used for OLR operations only. (See the reset chapter for more information.)

10.3.4 Address Range Registers

Address range registers are split into BASE and MASK pairs. The MASK register defines what bits in the BASE register are compared with the incoming address. If this compare is equal, then the range hits. Unimplemented bits are hardwired to 0 (reads return 0, writes have no effect). Note that for an address range to match, the most significant unimplemented bits must all be equal to 0 (all the bits from bit 63 through the start of the base_address or address_mask bits in the incoming address must be equal to zero).

Bits marked RE are the Range Enables. The address range is ignored if RE = 0. When RE = 1, then incoming addresses are compared with the address range register pair.

10.3.4.1 Memory Mapped I/O Space Ranges

10.3.4.1.1. 64 Bit Memory Mapped I/O Space -- WGMMIO

MSB	WGMMIO Base Register OFFSET: 0x0230																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																				Base_address											
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															re
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register 14-4 WGMMIO Base

MSB	WGMMIO Mask OFFSET: 0x0238																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																				Address Mask											
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register 14-5 WGMMIO Mask

MSB	GMMIO BASE OFFSET: 0x0210																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																				Base Address											
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Base Address						Reserved																									re
RESET INITIALIZATION																															
X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register 14-3 GMMIO BASE

MSB	GMMIO MASK OFFSET: 0x0218																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																				Address Mask											
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Address Mask						Unimplemented																									
RESET INITIALIZATION																															
X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register 14-4 GMMIO MASK

10.3.4.1.2. Whole 32 Bit Memory Mapped I/O Space -- WLMMIO

MSB	**WLMMIO Base OFFSET: 0x0220**																														LSB
63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Base Address											Reserved																			re
RESET INITIALIZATION																															
1	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register 14-6 WLMMIO Base

MSB	**WLMMIO Mask OFFSET: 0x0228**																														LSB
63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Address_mask											Reserved																			
RESET INITIALIZATION																															
1	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register 14-7 WLMMIO Mask

MSB	LMMIO Base OFFSET: 0x0200																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Base address															Reserved															re
RESET INITIALIZATION																															
1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register 14-3 LMMIO BASE

MSB	LMMIO Mask OFFSET: 0x0208																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Address Mask															Reserved															
RESET INITIALIZATION																															
1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register 14-4 LMMIO Mask

10.3.4.1.3. Interrupt Space

Any access to memory space in the fixed range 0xFEE0 0000 - 0xFEEF FFFF will be treated as a transaction-based interrupt.

Read attempts from PCI to this range will result in a Target Abort.

10.3.4.1.4. VGA Frame Buffer Space

The VGA frame buffer space is a fixed 128 k region from 640 k to (768 k - 1). The only PCI bus in the system that will claim this space is the one with the VGA Enable bit set. This PCI bus will also claim accesses to VGA I/O Space (see Section 8.3.3.2.3).

10.3.4.2 I/O Space Ranges

Note that since 64 bit addressing to IOP space in not supported by PCI 2.1, the upper 64 bits of these registers are meaningless.

10.3.4.2.1. Local I/O Port Space -- IOS

MSB	IOS BASE OFFSET: 0x0240																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																Base Address								Reserved							re
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0

Register 14-8 IOS BASE

MSB	IOS MASK OFFSET: 0x0248																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																Address Mask								Reserved							
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0

Register 14-9 IOS MASK

10.3.4.2.2. Extra Local I/O Port Space -- EIOS

MSB	EIOS Base OFFSET: 0x0260																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																Base Address								Reserved							re
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0

Register 14-10 EIOS BASE

MSB	EIOS MASK OFFSET: 0x0268																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																Address Mask								Reserved							
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0

Register 14-11 EIOS MASK

10.3.4.2.3. VGA I/O Port Space

The VGA I/O space is the fixed I/O space region from 0x03B0 through 0x03DF and the 10 bit aliases. Also, even if the VGA I/O Space overlaps the I/O space of another PCI bus, only the PCI bus with its VGA Enable bit set will respond to these ranges.

10.3.5 Hint Lookup Table format

There are 14 HINTSx registers, labeled HINTS0 through HINTS13. HINTS0 and HINTS1 are used by the bus master connected to the first REQ# input of the Elroy arbiter. (If recommendations in the "system design considerations" chapter are followed, this will be device # 0.) Similarly, HINTS2 and HINTS3 are associated with the second REQ# input, and so on for all 7 REQ# inputs. This normally allows for 2 sets of hints for each master. However, if 3 or fewer masters are supported on a bus, 4 sets of hints can be supplied for each master. To do this, every other REQ# input should be used. Then, HINTS0 through HINTS3 can be associated with the first REQ# input, HINTS4 through HINTS7 with the third REQ# input, and so on.

One or two PCI address bits control which of the 2 or 4 hint sets is used for any given transaction. HINT_CNFG register, defined below, controls which address bits do this if the address was **not** translated by the TLB. If it is translated, then the 2 address bits immediately above the bits that form the offset within the IOV space are used to select the hints. For instance, if the I/O PDIR backed space is 1MB, then bits 21 and 20 would be hint select bits.

The format of each entry in the hints look up table is shown below:

MSB	**Hints Look Up Table OFFSET: 0x0380 – 0x03E8**																														LSB
63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved		WH			RH									LH									MH								
RESET INITIALIZATION																															
0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Register 144-12 HINTS Look-up Table

Note that although ELROY implements the hint lookup table, the hints are interpreted by the rope-master. Refer to the appropriate ERS for how the hints are interpreted. ELROY does select a field of the hint set to send upstream, based on which PCI command was used. The fields are:

WH: Write Hints. Selected when a Memory Write or Memory Write Invalidate command is used on PCI.
RH: Read Hints. Selected for a PCI Memory Read command.
LH: Read Line Hints. Selected for a PCI Memory Read Line command.
MH: Read Multiple Hints. Selected for a PCI Memory Read Multiple command.
Bits 63:30 are unused

10.3.6 HINT_CNFG

HINT_CNFG controls which bits of the PCI address are used to select the desired hint set.

MSB	HINT_CNFG OFFSET: 0x0310																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																								A1_sel				A0_sel			
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Register 104-13 HINT CNFG

The fields are:

1. A1_sel: Identifies which PCI address bit should be the hint select MSB.
2. A0_sel: Identifies which PCI address bit should be the hint select LSB.

The following encoding apply to both fields: 0 selects bit 32, 1 selects bit 33, … , 13 selects bit 45. Values 14 and 15 will force that hint select bit to 0.

The address bits used to index into the hint look-up table are masked off before the physical address is sent over the rope.

10.4 Direct Virtual Index (DVI) mode

PCX-W needs virtual indexes to ensure coherent DMA. The virtual index is normally supplied by the I/O TLB. However, it can also be supplied as a part of the PCI address. This is called the DVI mode.

An address is a DVI address if bit 54 or bit 47 of the address is set. If bit 54 is set, virtual index is encoded in bits 53:46 of the PCI address. If bit 54 is not set and bit 47 is set, virtual index is encoded in bits 46:39. These PCI address bits are sent over the Rope with the rest of the PCI physical address. The remaining bits contain the actual physical address and the hint select bits described earlier. All the DVI bits and hint bits are masked off to arrive at the real physical address.

10.5 PDIR IOVA Base and Mask Registers (IBASE, IMASK) (Read/Write):

MSB	IBASE Register OFFSET:0x300																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IBASE										Reserved																					re
RESET INITIALIZATION																															
X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MSB	IMASK Register OFFSET:0x308																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMASK										Reserved																					
RESET INITIALIZATION																															
X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

These registers are used to program the location and the size of the IOV space. IBASE sets the starting address for the IOV space. Bit 0 of IBASE is “range enable” (re). The entire IOV space is disabled if this bit is false. This is a good way to completely disable the translation cache functionality. The size is communicated by IMASK. IMASK is programmed with 1’s for those the bits corresponding to the bits of IBASE that actually define the location of this space. Lower order bits, which correspond to an index within this space, are masked off with 0s. For instance, if this space is 16 MB, so that the largest valid index is 0x00ff_ffff, then IMASK should be 0xff00_0000. A PCI address is considered to fall in the IOV space if the following is true:

((PCI_address & IMASK) == IBASE) & IBASE.re

The rope-master also implements IBASE and IMASK registers. The following rules apply to the rope master and all the ELROYs below it:

1. All IBASE registers must have the same value
2. All ELROY IMASK registers must have the same value
3. The rope-maseter's IMASK should be (ELROY IMASK >> 2). A more accurate description would be (ELROY IMASK >> 2) & 0xFFF0_0000, but the masking is not really needed because hardware ignores unused bits.

The last rule is needed so that ELROY will decode 4 times as much IOV space as the rope-master which allows ELROY to use the 2 MSB of the IOV space address as an index into the hint registers.

11. PCI Master Controller (PMC)

11.1 Operation Overview

The PCI master controller generates transactions to targets on the PCI bus (PIO reads and writes). It is also responsible for recording errors so the operating system can take appropriate recovery steps.

Because system bus transactions are of fixed length, but PCI transactions are variable length, logic in the PMC will keep track of the address of each 32 or 64 bit word of data transmitted over PCI so that it can continue the transaction later if it did not complete. Since the largest amount of data that can be sent in a single system-bus transaction is one cacheline, coalescing of write posted data will also be done. In order to maximize the bandwidth of PCI, coalescing will be preformed by the rope-master before the data is passed to Elroy.

Memory Write transactions to sequential addresses can be coalesced. PCI spec refers to this as "Combining." Whenever a transaction is written into the Command/Data FIFO, the ISC keeps track of the next address that would be written if Combining took place. When another transaction comes in, if the new address matches the stored address, only the data word is written into the FIFO. This will naturally cause the PMC to treat the two transactions as a single burst. On PCI, any burst can be terminated prematurely by the target, so the PMC must remember the address where it should restart the burst next time. This logic also covers the case where ISC writes in only a data value because it can be coalesced, but by the time the PMC sees it, the previous transaction had already been completed. Coalescing only works when the incoming transaction supplies more than 8 bytes. I.e., single word transactions will not get coalesced.

ELROY will never generate Memory Write Invalidate (MWI) transactions on PCI. (ELROY accepts MWI, but that's discussed in the DMA section.)

11.1.1 ELROY Behavior on PCI

When ELROY is attempting a 64 bit or smaller transaction, it will not assert REQ64# on PCI, since there would only be a performance hit from doing that negotiation. This means an eight byte transaction will always be mastered as two four byte transactions, even if the target is a 64 bit target.

ELROY only generates Memory Read commands as a master. It will not generate Memory Read Line or Memory Read Multiple. ELROY does understand and take advantage of these commands as a slave.

11.1.2 Latency Timer and Consecutive Transactions

The PMC will have the ability to perform several consecutive transactions. This requires special support in the PCI arbiter, since no PCI master can begin a transaction if its GNT# is not asserted. Elroy's Arbiter will be designed to grant the PCI bus to the PMC as long as its REQ# is asserted. To prevent the PMC from starving other devices on the PCI bus, it will have a Latency Timer that keeps counting during the consecutive transactions.

The PMC's Latency Timer will begin counting when it starts a PCI transaction after its GNT# makes a deasserted to asserted transition. This timer will continue to count through consecutive transactions until the timer expires or the PMC deasserts GNT#. When the Latency Timer does expire, the PMC will deassert REQ# until its GNT# is deasserted. By doing this, the PMC will be able to continue past its Latency Timer expiring if no other masters are requesting the bus without preventing any other master from being granted the bus as soon as they need it. Once its GNT# has been deasserted, the PMC must end the current PCI burst, and is free to re-assert its REQ#.

11.1.3 Arbiter Considerations

In order for the PMC to be able to perform several consecutive transactions and be able to continue transactions past its Latency Timer expiring, the PCI arbiter must support both these features:

1. Once the PMC wins arbitration, the arbiter must keep the PMC's GNT# asserted as long as its REQ# is asserted.
2. The arbiter must do either one of the following ("parking" refers to which agent gets GNT# when no one is requesting the bus):
   - park the bus on the PMC
   - park the bus on the last master until the bus goes IDLE, and when the bus is IDLE, the bus must be parked on the PMC

The PMC can still function in a system where the arbiter only allows the PMC to perform one PCI transaction per arbitration cycle. However, the following performance considerations apply when the PMC is the only master ready to start a PCI transaction:

1. If the arbiter does not park the bus on the PMC when it deasserts REQ#, then the PMC will not be able to continue a PCI burst after its Latency Timer has expired.
2. If the arbiter does not park the bus on the PMC when the PCI bus is IDLE, then the PMC will not be able to perform fast back-to-back transactions after its Latency Timer has expired. In this situation, the PMC will need to re-arbitrate for the bus to start its next transaction -- wasting PCI bandwidth.

11.2 Major Sub-blocks

11.2.1 System-Bus Transaction to PCI Transaction Conversion

This block is responsible for generating appropriate PCI transactions to retrieve or store the information received from an agent on the System bus. The types of PCI transactions generated can be found in Table 4.

PCI transaction generated	Under these conditions
Memory Read/Write	System bus memory read/write
I/O Read/Write	System bus I/O space read/write
Configuration Read/Write	Generated by accesses to CONFIG_ADDRESS and CONFIG_DATA registers
Dual Address Cycle	Whenever accessing a target outside the first 4 GB of memory space

Table 4. PCI transactions generated by the PMC

This block will also keep track of the current address of each word of data sent over PCI. If a transaction ends before all the data the system bus expected to transfer has been sent, then another PCI transaction that picks up where the last one left off will be generated. If a transaction is Retried on PCI, the PMC will continue to issue that transaction until it has completed successfully. However, in such a situation, the PMC will switch back and forth between posted writes from the Command/Data FIFO and non-posted transactions from the delayed request queue. Therefore, indefinite retries of transactions from one FIFO do not interfere with completion of transactions from the other FIFO.

11.2.2 64 to 32 bit converter

The 64 to 32 bit converter functions similarly to the 32 to 64 bit converter of the PSC -- conditionally allowing the upper 32 bit word to shift into the lower 32 bits so data from Elroy's 64 bit FIFO can be sent to 32 bit devices. The PMC will know the PCI target is 64 bit capable when it asserts ACK64# at the same time it asserts DEVSEL#.

11.3 Parity Generator/Error Log

The parity generator is responsible for computing parity on the AD[31:0] + C/BE[3:0] lines (PAR) and on the AD[63:32] + C/BE[7:4] lines (PAR64). PAR and PAR64 are driven by the same agent that drove the AD[63:0] lines. If a parity error was detected, PERR must be driven by the receiving agent for data transfers. SERR must be driven by any agent that notices a parity error during the address phase.

This block is also responsible for recording that a PERR or SERR occurred so that the operating system can take whatever steps are necessary for error containment and recovery. This error reporting will be implemented as an error register that will record the type of error that occurred and the address of the current transaction.

Byte parity is passed along all the internal paths of Elroy. Therefore, this block will compute valid parity for each PCI data byte received to pass onto the write posting FIFO. In order to not propagate bad data, any data received from PCI with bad parity will be "poisoned" with bad parity when passed onto the write posting FIFO.

11.3.1 Registers

Registers 0x2050 (for PCI0, 0x3050 for PCI1) is an 8 bit value which sets the PCI Master multi-transaction timeout value. The PCI Master controller is allowed to keep its REQ line asserted for this many PCI clock cycles while it has been granted the bus. After this counter has expired, the master controller will de-assert its REQ line, but will continue to use the bus until it loses GNT.

MSB	ELROY Multi-transaction timeout counter OFFSET:0x0050																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																								Req Timeout							
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MSB	FW Scratch register OFFSET:0x0058																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																Scratch R/W register															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

11.3.2 PCI Command and Status Registers

This register behaves like the “Control” and “Status” registers specified in the PCI spec revision 2.1, sections 6.2.2 and 6.2.3 with two exceptions. Bit 2 of the PCI Control Register is hardwired to a 1 to allow booting off of PCI. This is not explicitly permitted by the PCI spec. In addition, B.7 0 has a reset state of 1 instead of 0.

Note: The I/O range in the PCI Control register should always be disabled (since peer-to-peer is not supported between PCI busses).

MSB	**FID Function Identification Register** **OFFSET: 0x0000**																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Status																Control															
RESET INITIALIZATION																															
0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Function ID (Read only)																Vendor ID (Read Only)															
RESET INITIALIZATION																															
0x1054																0x103CH															

The Function ID register is described in the SPPA Platform EAS Section 8.3. Also see the PCI spec section 6.2

MSB	**Function Class Register** **OFFSET: 0x0008**																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Bist								Header Type								Lat Timer								Line_size							
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Class Code (Read Only)																								Revision (Read Only)							
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Rev number							

Line size: 8 bit R/W register.

Lat timer: Latency timer as described in PCI 2.1 spec.

Header Type: Read Only? . Always returns 0x00

Bist: Built in self test: Always returns 0x00

<table>
<tr><th>MSB</th><th colspan="30">Misc Control Register
OFFSET: 0x0068</th><th>LSB</th></tr>
<tr><td>63</td><td>62</td><td>61</td><td>60</td><td>59</td><td>58</td><td>57</td><td>56</td><td>55</td><td>54</td><td>53</td><td>52</td><td>51</td><td>50</td><td>49</td><td>48</td><td>47</td><td>46</td><td>45</td><td>44</td><td>43</td><td>42</td><td>41</td><td>40</td><td>39</td><td>38</td><td>37</td><td>36</td><td>35</td><td>34</td><td>33</td><td>32</td></tr>
<tr><td colspan="23">Reserved</td><td colspan="9">Reserved</td></tr>
<tr><th colspan="32">RESET INITIALIZATION</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="21">Reserved</td><td>Loopback</td><td>Reserved</td><td>RPCE</td><td colspan="8">Reserved</td></tr>
<tr><th colspan="32">RESET INITIALIZATION</th></tr>
<tr><td colspan="9">0x17</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table>

The RPCE (Rope Parity Check Enable) bit is used to control the way the ropes controller behaves in response to rope parity errors. If this bit is set, then a ropes command parity error will result in ELROY going into fatal mode. If this bit is not set, then the ropes controller will ignore all command parity errors. The default value for this bit is 0. Software should set this bit as part of ELROY initialization.

The loopback bit is used to put ELROY into a diagnostic mode called Loopback Mode. Normally, ELROY has internal sanity checking that will prevent the PSC from responding to transactions driven by the PMC, even if the range registers are configured that way. The Loopback Mode disables this sanity checking. This allows for diagnostics where a transaction can be sent down the rope, mastered by the PMC, claimed by the PSC, and sent back up the rope. Note that the range registers in the rope-master and ELROY must be configured correctly for this to occur.

MSB	**Error Log Address Register** **OFFSET: 0x0070**																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register contains the address of the transaction the PMC was trying to handle when it encountered an error.

MSB	**Suspend Register** **OFFSET: 0x0078**																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

There is no data associated with the suspend register. A read from this register puts ELROY in a state where no delayed transactions are processed by the PMC. Arbitration is disabled (no other agent can get the bus) and interrupts are prevented from being sent up the wpf.

A write to this register takes ELROY out of the above mode, or out of the "lock" mode used to implement a series of locked transactions. The lock mode is entered by doing a PIO read flagged with a lock.

11.4 Accessing PCI Configuration Space

PCI configuration cycles are generated by using the CONFIG_ADDRESS and CONFIG_DATA registers. To create a configuration cycle, software first writes to the CONFIG_ADDRESS register, defined in Table 3. This write identifies which PCI bus, which device on the bus, and which function (of a multi-function device) the configuration cycle will target. Once CONFIG_ADDRESS has been set, a Read or Write to CONFIG_DATA will generate the Configuration Space access. Byte Enables used to Read/Write to CONFIG_DATA will be used in the Configuration Read/Write as well. After the read or write to the CONFIG_DATA register, A register read (probably to the CONFIG_ADDRESS register) should be done to make sure the configuration cycle has completed before writing a new address to the CONFIG_ADDRESS register. If several writes are done to the CONFIG_DATA register without reading the CONFIG_ADDRESS register, the DRQ (Delay Resource Queue) in ELROY will overflow. Care should be taken to ensure that only one process is allowed to access this pair of registers at a time.

When a configuration cycle is targeting a device that is directly on ELROY's PCI, a type 0 configuration cycle needs to be generated. To do so, use bus number of 0. Setting bus number 0 is merely an indication that a type 0 configuration cycle needs to be generated. If the CONFIG_ADDRESS "Bus Number" is greater then 0, then a Type 1 cycle will be generated.

MSB	**CONFIG Address OFFSET: 0x0040**																														LSB
63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved								Bus Number								Device Number					Functio n Number			Register number						0	0
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Description
Reserved	Must set to 0 for writes, must be ignored for reads.
Bus Number	Identifies the PCI bus number the configuration access is intended for.
Device Number	Identifies the physical PCI device number the configuration access is intended for.
Function Number	Identifies the function number within a physical PCI device.
Register Number	Identifies the word address of the target function's configuration register.
Bits 1,0	Read only bits. Returns 00 on reads.

MSB	CONFIG Data OFFSET: 0x0048																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
RESET INITIALIZATION																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data																															
RESET INITIALIZATION																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

11.4.1 Generation of IDSEL lines.

The IDSEL lines to selected PCI slots come from AD lines that are run through a resistor to decouple the IDSEL load from the AD line. The IDSEL signals are generated during a type 0 configuration cycle by decoding the "Device Number" (bits 11-15 of the Configuration Address register and driving AD bit number (16+Device Number) to a "1" while driving the rest of bits 16-31 to a "0". If Device Number is more than 15, none of the lines 16-31 will be driven to a "1". To allow the IDSEL lines to be resistivly decoupled from the associated AD line, the address will be driven on the AD lines for 3 PCI cycles before "Frame#" is asserted. This will allow 45nsec for the RC time constant to reach the desired level in a 66MHz system before "Frame#" is asserted, and 90nsec in a 30-33MHz system.

12. INTERRUPTS

Information in this chapter cannot be given out, but will be needed by developers. Contact pa_linux@hp.com for more information.

13. ELROY Revision History

Information in this section is needed by developers, but cannot be given out. Contact pa_linux@hp.com for more information.