DIGITAL Semiconductor 21174 Core Logic Chip

Technical Reference Manual

Order Number: EC–R12GC–TE

Revision/Update Information: This is a revised, preliminary document. It supersedes the *DIGITAL Semiconductor 21174 Core Logic Chip Technical Reference Manual*, EC–R12GB–TE.

Preliminary

Digital Equipment Corporation
Maynard, Massachusetts

http://www.digital.com/semiconductor

October 1997

Contents

Figures

Tables

Preface

Purpose and Audience

The *DIGITAL Semiconductor 21174 Core Logic Chip Technical Reference Manual* describes the operation of the DIGITAL Semiconductor 21174 core logic chip (also referred to as the 21174). This manual is for designers who use the 21174.

Manual Organization

This manual contains the following chapters, appendixes, and an index.

- Chapter 1, Introduction, includes a general description of the 21174. It also provides an overview of the workstation configuration.
- Chapter 2, Internal Architecture, provides the physical layout of the 21174 and describes each of the input and output signals.
- Chapter 3, Pinout, provides the pin layout of the 21174 and describes each of the input and output signals.
- Chapter 4, Register Definitions, provides a complete list of the 21174 registers.
- Chapter 5, Register Descriptions, provides a complete bit description of the 21174 registers.
- Chapter 6, System Address Space, describes the organization of the system address space and shows the methods used to translate 21164 and PCI addresses.
- Chapter 7, Electrical Specifications, lists the dc electrical specifications for the 21174.
- Chapter 8, Mechanical and Thermal Specifications, lists and illustrates the mechanical and thermal specifications of the 21174.
- Appendix A, 21174 DMA Page Boundary Solution, provides the files and the code necessary to manage PCI DMA reads that cross 8K page boundaries.

- Appendix B, 21174 DMA Lock Solution, explains how to manage inappropriate LOCK commands issued by the 21164 to the CMD bus.
- Appendix C, AlphaPC 164LX Layout Design Rules, contains the module design layout rules.
- Appendix D, Support, Products, and Documentation, contains information about technical support and ordering information.

Conventions

This section describes the abbreviation and notation conventions used throughout this manual.

Bit Notation

Multiple bit fields are shown as extents (see Extents).

Caution

Cautions indicate potential damage to equipment or loss of data.

Data Units

Table 1 defines the data unit terminology used throughout this manual.

Table 1 Data Units

Term	Words	Bytes	Bits	Other
Byte	1	—	8	—
Word	1	2	16	—
Tribyte	—	3	24	—
Longword	2	4	32	—
Quadword	4	8	64	—
Octaword	8	16	128	Single read fill; that is, the cache space that can be filled in a single read access. It takes two read accesses to fill one Bcache line (see Hexword).
Hexword	16	32	256	Cache block, cache line. The space allocated to a single cache block.

Extents

Extents are specified by a pair of numbers in angle brackets (<>) separated by a colon (:) and are inclusive. For example, bits <7:3> specifies an extent including bits 7, 6, 5, 4, and 3.

Logic Levels

The values 1, 0, and X are used throughout the manual. A 1 signifies a logic high, a 0 signifies a logic low, and an X signifies a don’t care (1 or 0) which can be determined by the system designer.

Must Be Zero

Fields specified as must be zero (MBZ) must never be filled by software with a nonzero value. If the processor encounters a nonzero value in a field specified as MBZ, a reserved operand exception occurs.

Note

Notes emphasize particular information.

Numbering

All numbers are decimal or hexadecimal unless otherwise specified. In cases of ambiguity, a subscript indicates the radix of nondecimal numbers. For example, 19 is decimal, but 19_{16} and 19A are hexadecimal.

Processor Chip Type

All references to the 21164 microprocessor specifically refer to the version of the DIGITAL Semiconductor 21164 in 0.35-µm CMOS and to the DIGITAL Semiconductor 21164PC in 0.35-µm CMOS.

Ranges

Ranges are specified by a pair of numbers separated by two periods (..) and are inclusive. For example, a range of integers 1..4 includes the integers 1, 2, 3, and 4.

Register and Memory Figures

Register figures have bit and field position numbering starting at the right (low-order) and increasing to the left (high-order). Memory figures have addresses starting at the top and increasing toward the bottom.

All shaded bits and bit fields in the figures are reserved, and software drivers should write only 0 to these bits and bit fields.

Register figures and tabulated descriptions have a mnemonic that indicates the bit or field characteristic as described in Table 2.

Table 2 Register Field Notation

Notation	Description
RW	A read-write bit or field. The value can be read and written by software, microcode, or hardware.
RO	A read-only bit or field. The value can be read by software, microcode, or hardware. The bit is written by hardware. Software or microcode write operations to this bit are ignored.
WO	A write-only bit. The value can be written by software and microcode. The bit is read by hardware. Read operations to this bit by software or microcode return an UNPREDICTABLE result.
WZ	A write-only bit or field. The value can be written by software or microcode. The bit is read by hardware. Read operations to this bit by software or microcode return a zero.
WC	A write-to-clear bit. The value can be read by software or microcode. Software or microcode write operations with a 1 to this bit cause the bit to be cleared by hardware. Software or microcode write operations with a 0 to this bit do not modify the state of the bit.
RC	A read-to-clear field. The bit is written by hardware and remains unchanged until the bit is read. The bit can be read by software or microcode, at which point hardware can write a new value into the field.

Other register fields that are unnamed may be labeled as specified in Table 3.

Table 3 Unnamed Register Field Notation

Notation	Description
0	A 0 in a bit position indicates a register bit that is read as a 0 and is ignored on a write operation.
1	A 1 in a bit position indicates a register bit that is read as a 1 and is ignored on a write operation.
x	An x in a bit position indicates a register bit that does not exist in hardware. The value is UNPREDICTABLE when read and is ignored on a write operation.

Should Be Zero

Fields specified as should be zero (SBZ) should be filled by software with a zero value. These fields may be used at some future time. Nonzero values in SBZ fields produce UNPREDICTABLE results.

Signal Name References

Signal names are printed in boldface, lowercase type. Mixed-case and uppercase signal naming conventions are ignored. These two examples illustrate the conventions used in this document:

- **MEM_WE_L[1]** is shown as **mem_we_l<1>**
- **TEST_MODE[1]** is shown as **test_mode<1>**

UNPREDICTABLE and UNDEFINED Definitions

Results specified as UNPREDICTABLE may vary from moment to moment, implementation to implementation, and instruction to instruction within implementations. Software can never depend on results specified as UNPREDICTABLE.

Operations specified as UNDEFINED may vary from moment to moment, implementation to implementation, and instruction to instruction within implementations. The operation may vary from nothing to stopping system operation. UNDEFINED operations must not cause the processor to hang, that is, reach a state from which there is no transition to a normal state where the machine can execute instructions.

Note the distinction between results and operations. Nonprivileged software cannot invoke UNDEFINED operations.

Warning

Warnings provide information to prevent personal injury.

1 Introduction

The DIGITAL Semiconductor 21174 core logic chip (also referred to as the 21174) is a single-chip core logic PCI-to-21164 interface for low-cost workstations. It provides an inexpensive memory, cache, and PCI controller for uniprocessor workstations. The 21174 may be used with 21164PC devices and with 21164 devices that support a clock frequency equal to or greater than 400 MHz. These types of devices are referred to as 21164 in this manual.

1.1 21174 Features

The 21174 has the following features:

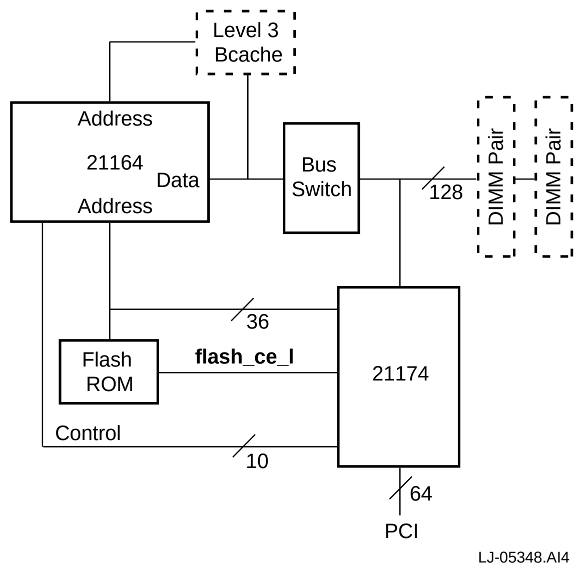
- Synchronous dynamic RAM (DRAM) memory controller
- Supports optional Bcache (level 3 cache)
- Supports 64-bit PCI at 33 MHz
- 64 interrupts through external shift register
- 32 general-purpose inputs through external shift register
- 32 general-purpose outputs through external shift register
- 3.3–V design
- Quadword ECC support, longword parity, or no parity on system and memory data buses
- Onchip phase-locked loop (PLL)
- Direct attachment of flash ROM
- Startup from flash ROM
- Compact design, complete interface in single 474-pin ball grid array (BGA)

- 1000 MB/s peak memory bandwidth
- Glueless workstation memory controller

1.2 21174 System Configuration

Figure 1–1 shows the 21174 used in a system configuration.

Figure 1–1 System Configuration



See Appendix C for additional details on designing a system using two DIMM pairs. Appendix C also contains key module layout design rules. It is crucial that these rules are followed when designing a system.

2 Internal Architecture

The 21174 provides an interface between three units — memory, the PCI bus, and the 21164 (along with flash ROM). Figure 2–1 shows the internal components of the 21174 and the three internal units: memory controller, PCI interface, and the interface to the 21164 flash ROM.

2.1 Memory Controller

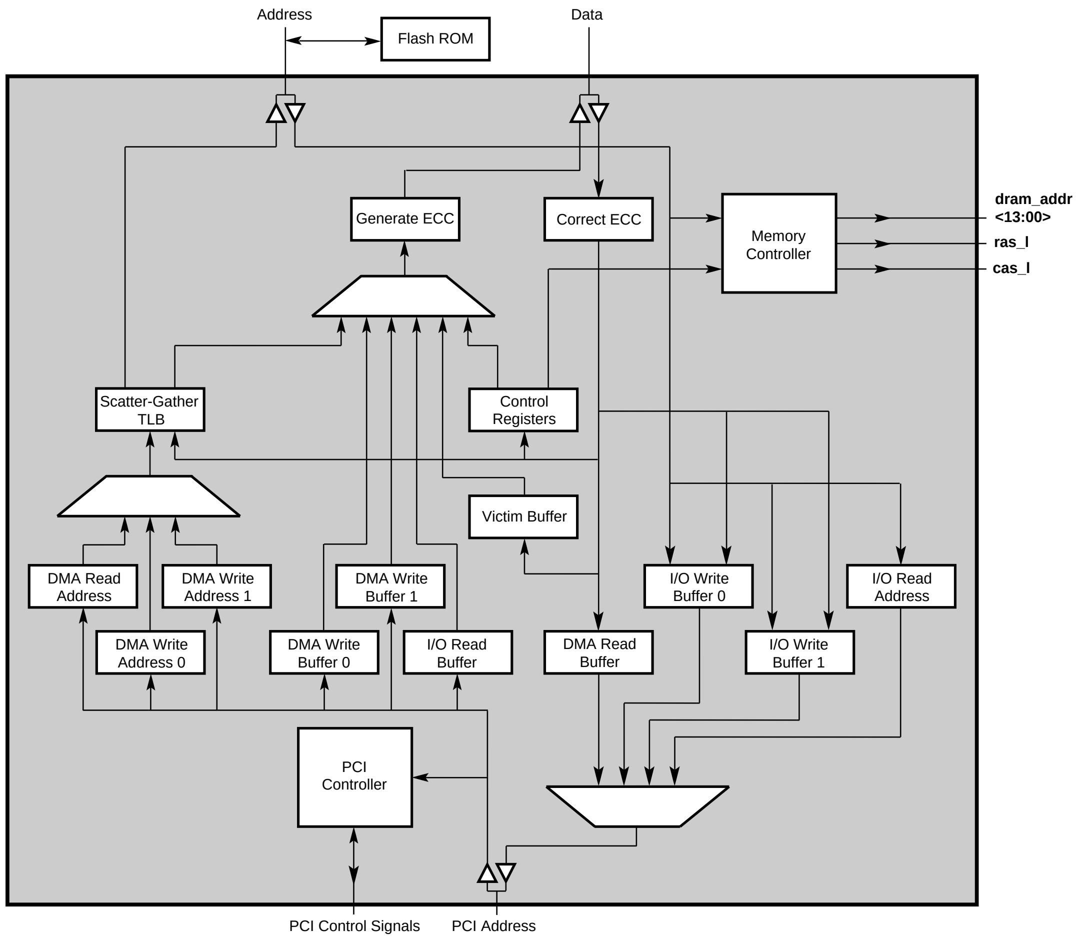
The 21174 memory controller provides clocks, data, address, and control to the memory unit. When power is turned on, the memory controller gathers information from the memory banks, and then uses that information to initialize and configure the memory unit.

2.1.1 Memory Sequencers

The memory sequencer permits up to two partially overlapping memory transactions to be active at any given time. The sequencer is internally implemented in two parts; the master sequencer and the data transfer machine. The master sequencer initiates all memory operations and I/O operations. It generates all control signal timings for the SDRAMs. When a memory operation has progressed to the point where a column access has started, the master sequencer hands off the transaction to the data transfer machine. The data transfer machine controls the four data cycles that complete the transaction. In most cases, the master sequencer is ready to start a new transaction as soon as the old transaction has been handed off to the data transfer machine.

Memory Controller

Figure 2–1 21174 Block Diagram



2.1.2 DMA Read Transaction

A DMA read transaction consists of an optional scatter-gather translation lookaside buffer (TLB) lookup (and TLB refill, if needed), followed by a probe transaction to the 21164, followed by either a cache read from 21164 or a read from the memory. Figure 2–2 shows a flow diagram of a DMA read transaction. The length of the DMA read transaction is determined by the prefetch logic associated with the scatter-gather TLB.

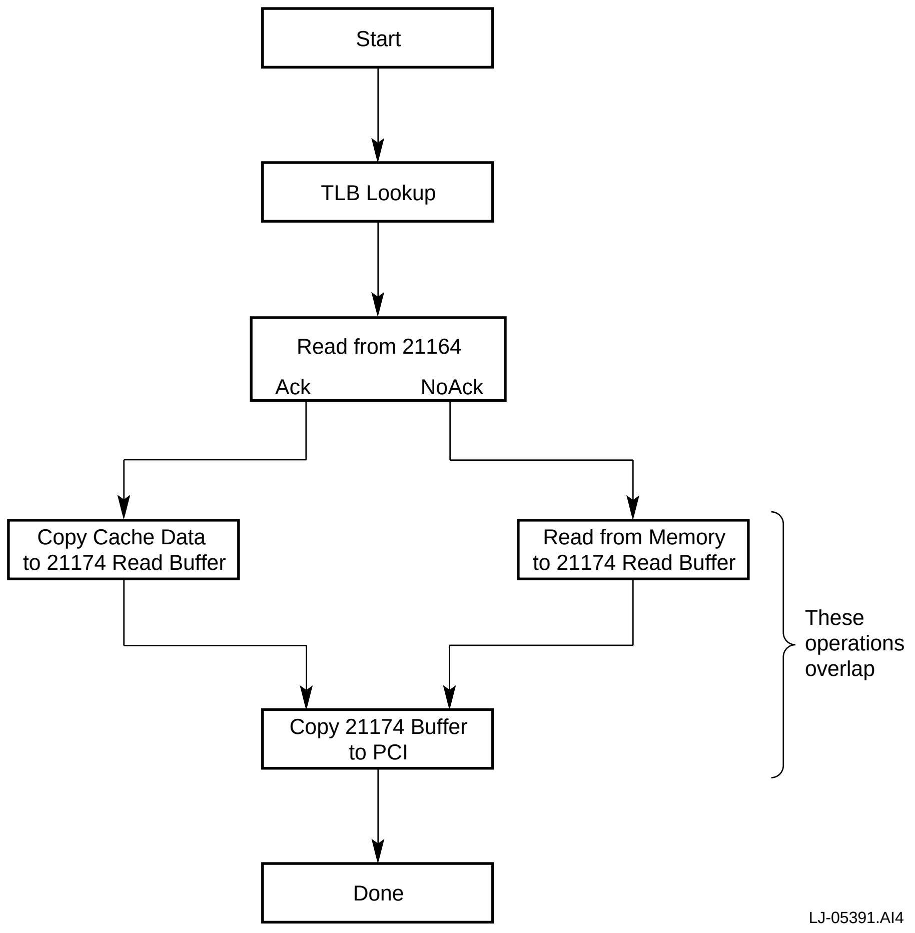
If the scatter-gather lookup misses in the TLB, an additional cache probe, and possibly a memory read transaction, must be performed to fill the TLB before completing the main probe and memory transaction. The DMA read address is also compared to the victim buffer addresses. If there is a victim buffer hit, data from the victim buffer is substituted for the memory read transaction data.

2.1.2.1 PCI DMA Page Boundary Problem

PCI DMA reads that attempt to cross 8K page boundaries cause data corruption problems.

See Appendix A for the DMA page boundary solution.

Figure 2–2 DMA Read Transaction Flow Diagram



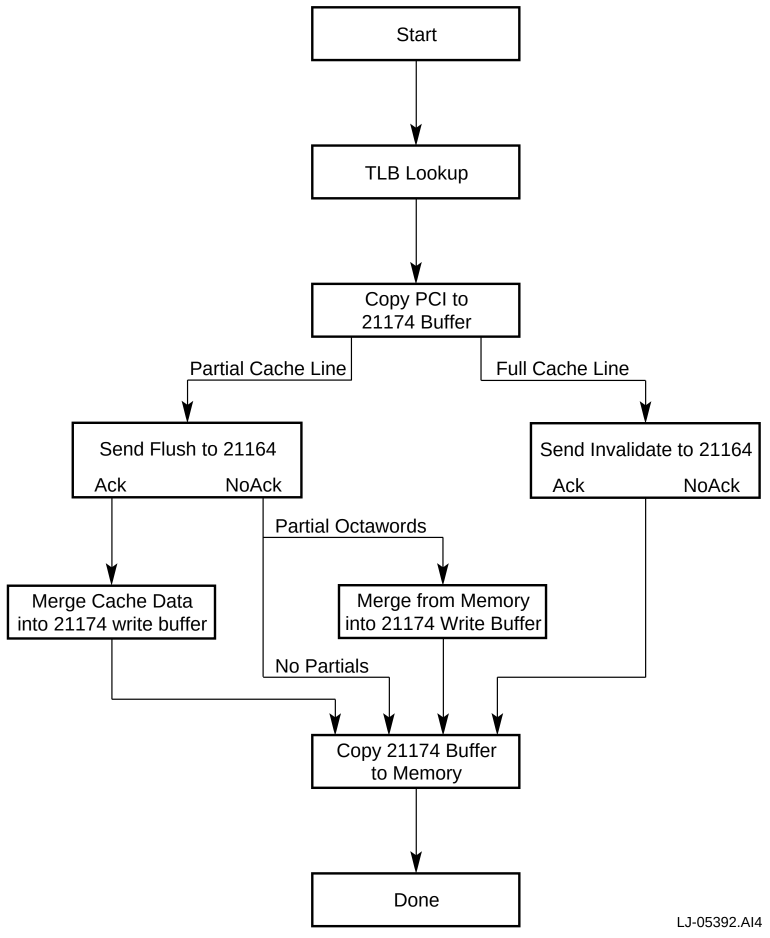
2.1.3 DMA Write Transaction

A DMA write transaction consists of an optional scatter-gather TLB lookup (and TLB refill, if needed), followed by a flush transaction to the 21164, followed by a write to the memory. Figure 2–3 shows a flow diagram of a DMA write transaction.

If the scatter-gather lookup misses in the TLB, an additional cache probe, and possibly a memory read, must be performed to fill the TLB before completing the main probe and memory transaction.

The physical DMA address is also compared to the victim buffer addresses. If there is a victim buffer hit, the victim buffer data is merged with the write buffer before the write is performed, and the victim buffer is invalidated.

Figure 2–3 DMA Write Transaction Flow Diagram



2.1.4 21174 DMA Lock Problem

The 21164 sometimes issues LOCK commands on the CMD bus. The 21174 treats the LOCK command as a no-op command and goes back to idle. This does not actually clear the LOCK command. Thus, the process repeats indefinitely, blocking DMA requests that may be waiting for service.

See Appendix B for the solution to this problem.

2.1.5 Minimum Memory Activation Period

Several cases where the memory controller activates the memory arrays but does not perform a read or write operation are described here:

- If the victim buffer is hit during a 21164 read transaction, the memory controller performs a complete memory read operation and then discards the data before taking the data from the victim buffer.
- If the cache is hit during a DMA read transaction, the memory controller may have activated the memory arrays but does not perform a read operation. The 21164 takes the data from the cache while the memory controller completes the cache read cycles before returning to the idle state. In the worst case, the following state sequence occurs.

State	SELECT State	Number of Cycles
DMA_RD_PROBE	SELECT is asserted.	*n* cycles
DMA_RD_SCACHE_DATA	SELECT is deasserted.	2 cycles
DMA_RD_CACHE_DATA	SELECT is deasserted.	4 cycles
IDLE	—	—

  This state sequence implies that the memory arrays will be activated for a minimum of 6 (5) cycles, because the soonest that a new SELECT can occur is during the IDLE cycle.

- The DMA_RD_PROBE state deasserts SELECT when the next state is DMA_RD_SCACHE_DATA or DMA_RD_CACHE_DATA. This adds one state, which provides an additional margin, extending the minimum guarantee for the activated memory arrays to 7 (6) cycles. This is enough time for all currently available or proposed SDRAMs.

- During a DMA write transaction, even if there is a cache hit or victim buffer hit, there is a write-back to the selected memory location, so the memory arrays cannot become stranded in the activated state.

2.2 Memory Banks

The 21174 memory controller supports up to two banks, i.e. four DIMM slots, of synchronous DRAM memory. Each bank can have two sub-banks. The memory banks may be of different sizes and speeds, but the two sub-banks within a bank must be identical.

See Figure 1–1 for a typical system configuration.

The 21174 reads the I^{2}C control register (see Section 5.9.9), by way of the I^{2}C bus, to check for the absence or presence of memory DIMMs. Startup code will read the information from each DIMM. Each DIMM must contain the proper I^{2}C ROM.

2.2.1 Refresh

The operation of memory refresh is controlled by using the refresh timing register. All banks of memory are refreshed simultaneously.

2.2.2 Error Checking and Correction

The 21174 operates in one of two software selectable modes — ECC or PCA56 longword parity. Use PYXIS_CTRL[ECC_CHK_EN] to enable ECC checking or use PYXIS_CTRL1[LW_PAR_MODE] to enable PCA56 longword parity mode.

Initialization firmware must check the memory banks and the 21164 to determine whether parity or ECC bits are present in the memory and whether the 21164 will support ECC. The firmware then establishes the state of the error checking and error code generation based on this criteria. If the memory does not provide ECC/parity bits, then all memory error checking should be turned off.

If the 21174 is operated with error checking enabled, the memory should be initialized by firmware to contain good ECC or parity in all locations.

If the 21174 is operating in ECC mode and an ECC error occurs on a DMA read transaction or I/O write transaction, the ECC error is corrected. If the ECC error is not correctable, the DMA read or I/O write transaction will not complete. In either case, appropriate error status bits are set and the 21174 error interrupt is asserted if error reporting is enabled. The interrupt service routine must clear the error status bits to deassert the error interrupt.

If a parity error is detected on I/O read transaction data or DMA write transaction data, the operation will complete. The appropriate error bits are set and the error interrupt is asserted if error reporting is enabled.

2.2.3 DRAM Initialization

After power-up, the memory must be activated. The activation requires at least two refresh cycles before and after writing data (in hexadecimal) from the memory control register to the DIMMs. The algorithm is:

1. Write 634_{16} to the global timing register.
2. Write 80E0 to the refresh timing register (refresh width = 6, refresh interval = 5, and force refresh asserted).
3. Wait approximately 300 ns.
4. Write 80E0 to the refresh timing register (refresh width = 6, refresh interval = 5, and force refresh asserted).
5. Wait approximately 300 ns.
6. Write 3A0001 to the memory control register (3A is memory specific and the 1 is mode register set).
7. Write 80E0 to the refresh timing register (refresh width = 6, refresh interval = 5, and force refresh asserted).
8. Wait approximately 300 ns.
9. Write 80E0 to the refresh timing register (refresh width = 6, refresh interval = 5, and force refresh asserted).
10. Wait approximately 300 ns.

Executing this algorithm wakes up memory and sets the DRAM mode registers. This sequence configures the memory SDRAMs to the burst length, wrap type, and latency mode. This sequence is required by the SDRAMs and is not part of the system memory configuration sequence. Prior to accessing the memory, ensure that the memory is configured correctly, that PCA56 longword parity mode or ECC mode is set up properly, and that all of memory is written to initialize the memory error detection fields prior to enabling error checking. Write to memory only if error checking is to be performed.

Note: The SROM code is responsible for setting up the memory system. In most cases the memory system will be initialized prior to transferring control out of the initialization code.

2.3 PCI Interface

The PCI interface is 64 bits wide and supports operation at up to 33 MHz. Operation of the PCI is always synchronous to **sys_clk**. The PCI interface can be operated at a clock frequency ratio of 2:1 relative to the 21164 **sys_clk** frequency. The operating frequencies are listed in Table 2–1.

Table 2–1 PCI Operating Frequencies

CPU Frequency	sys_clk Ratio	sys_clk Frequency	sys_clk Cycle Time	PCI Frequency	PCI/sys_clk Ratio
466 MHz	7	66.0 MHz	15.00 ns	33.0 MHz	2:1
533 MHz	8	66.0 MHz	15.00 ns	33.0 MHz	2:1
600 MHz	9	66.0 MHz	15.00 ns	33.0 MHz	2:1

2.3.1 Scatter-Gather Map

The TLB provides eight scatter-gather map entries. Scatter-gather operation is enabled by setting the appropriate enable bit in the window base registers.

2.3.2 DMA Read Prefetch

The scatter-gather map TLB entries in the 21174 each contain a prefetch length field. The prefetch length field indicates the total number of 128-bit memory cycles to be fetched to satisfy the request. The prefetcher always fetches exactly the number of cycles specified in the prefetch length field. When the data is exhausted, the 21174 disconnects.

When a TLB entry is first loaded, the prefetch length is set to the value given in the appropriate read type field in the control register (PYXIS_CTRL). If the appropriate USE_HISTORY bit is set and the initiator disconnects without accepting all available prefetched data, the prefetch length is set to the number of memory cycles needed to supply the data transferred. During subsequent write transactions, the previous length is used to determine the prefetch length. See the description of the control register (PYXIS_CTRL) in Section 5.1.3.

2.3.3 DMA Write Buffer

A 2-entry write buffer is provided for DMA write data. Each entry is 64 bytes in length. When both buffers are in use, 21174 issues a retry response to DMA write transactions.

At the beginning of a PCI DMA write transaction, an unused write buffer is allocated for the transaction. The write data is aligned within the write buffer based on the low-order bits of the address. Address mask information is updated during the transaction. If the transaction reaches the end of the cache line, the 21174 disconnects. After a write buffer entry has been written, the 21174 schedules a memory write transaction or a memory read-modify-write transaction, depending on the state of the mask bits in the write buffer, to copy the data to memory.

When a PCI read transaction is initiated from the 21164, the 21174 ensures that the DMA write buffers are flushed to memory before returning the read data to the 21164.

2.3.4 DMA Write Buffer Merging

During DMA write transactions, byte mask information is received from the PCI bus. Depending on the alignment of the starting address and the state of the byte masks, it may be necessary to read quadwords from memory and merge bytes into the DMA quadwords before writing the DMA data to memory.

After a DMA write buffer in the 21164 is filled, the 21174 determines how many memory cycles are needed to complete the write transaction. The 21174 also determines if it is necessary to merge data within any quadword or if an empty quadword exists between quadwords that contain data. If either condition exists, a read-modify-write transaction sequence is scheduled instead of the simple DMA write transaction sequence performed for a normal aligned DMA. The read-modify-write transaction sequence is performed in the following order:

1. All quadwords of the DMA transaction are read from memory across **d<127:0>** into the 21174 chip. The bytes not selected by the DMA transaction are merged into the DMA write buffers.
2. All quadwords of the DMA transaction are written back to memory.

2.3.5 I/O Write Buffer

A 2-entry write buffer is provided for PCI write transactions originated from the 21164. Consecutive entries may be merged under optimal circumstances.

2.3.6 Configuration Cycles and Special Cycles

Configuration cycles and special cycles are generated in compliance with the *PCI Local Bus Specification, Version 2.1*. See Sections 6.9 and 6.10 for more information.

2.4 Flash ROM Interface

A flash ROM can be directly attached to the **addr<39:4>** signal lines. The address and data bits of the flash ROM are connected to the **addr<39:4>** pins as shown in Table 2–2.

Table 2–2 Flash ROM Pin Assignment

21174 Pin	Flash ROM Pin	21174 Pin	Flash ROM Pin
addr<39>	OE	**addr<16>**	A<3>
addr<31>	A<18>	**addr<15>**	A<2>
addr<30>	A<17>	**addr<14>**	A<1>
addr<29>	A<16>	**addr<13>**	A<0>
addr<28>	A<15>	**addr<12>**	A<B>
addr<27>	A<14>	**addr<11>**	D<7>
addr<26>	A<13>	**addr<10>**	D<6>
addr<25>	A<12>	**addr<9>**	D<5>
addr<24>	A<11>	**addr<8>**	D<4>
addr<23>	A<10>	**addr<7>**	D<3>
addr<22>	A<9>	**addr<6>**	D<2>
addr<21>	A<8>	**addr<5>**	D<1>
addr<20>	A<7>	**addr<4>**	D<0>
addr<19>	A<6>	**flash_ce_l**	ce_l
addr<18>	A<5>	**flash_we_l**	we_l
addr<17>	A<4>	—	—

The flash ROM can be read and written through the address range selected in the flash ROM control register. After reset, the flash ROM is at location 0. The 21174 supports cache fills and noncacheable reads from the flash ROM. For example, the 21174 will perform multiple read transactions to the flash ROM to assemble full octawords. The processor can start executing directly from the flash ROM if the 21164 is configured to start from cache misses rather than from SROM.

For flash ROMs smaller than 64MB, the high-order address bits can be left unconnected. This results in aliasing of the flash ROM throughout the flash ROM address space. It is also possible to attach multiple flash ROMs, or other devices, using the control bits and high-order address bits to drive a decoder. In this case, buffers may be needed to limit the loading for each signal within **addr<39:4>**.

If the flash ROM is accessed through cache fills (for example, through one of the windows in cacheable space), an unwanted parity error on the address bus (**addr<39:4>** and **cmd<2:0>**) may be generated unless the 21164 Bcache control register bit, BC_CONTROL[DIS_SYS_PAR], is set. Setting the DIS_SYS_PAR bit will disable parity errors during any type of flash ROM transactions. Unwanted parity errors may also be inhibited by setting BC_CONTROL[EI_DIS_ERR]. The EI_DIS_ERR bit is initialized to 1 during 21164 reset, so in the 21164 initial state, no unwanted parity errors will be generated.

On a system with a Bchache, if a private cache write is overlapped with the beginning of a fill from the flash ROM, with the fill approximately asserted on the second cycle of the private write, the system will appear to hang. This probably affects cacheable flash ROM fills and has only been seen on an I-stream miss at the end of a flash-to-memory copy. A workaround has been incorporated into the SROM code to add additional timing that prevents the system from hanging. I-stream and D-stream fills from cache should be limited to SROM operation. All other accesses to the flash ROM should be done using the I/O space assigned to flash memory space.

2.5 Auto DACK

The 21164 supports an “Auto DACK” feature to accelerate data bus transfers. When this feature is enabled by setting the AUTO_DACK bit in the 21164 BC_CONTROL register, the 21164 assumes that the final two DACKs of a data transfer are contiguous. The 21174 supports this mode of operation.

2.6 Dummy Memory

A dummy memory block is provided to facilitate flushing of 21164 Scache and Bcache. Read transactions to this 4-GB block of dummy memory region causes a value of 0 to be returned. Write transactions to this memory region result in nonexistent memory traps. The dummy memory region is the last 4GB of cached memory address space starting at E.0000.0000.

The dummy memory region can be effectively used to implement a small memory area at power-up, using the processor's Dcache (level 1) and Scache (level 2). If the 21164 is set up to access memory within the dummy area, fills will be served by the dummy area, but after the fill, the cache will work properly to serve those memory addresses. Use only a memory area that will fit within the Dcache and Scache, because a victim ejection will cause a machine check. This same technique can be used to emulate a much larger memory using the optional Bcache (level 3). It is also possible to use the flash ROM address spaces to emulate memory, but the fills from this space are much slower because they require sequential flash ROM transactions. In either case, avoid sharing cache address space with the instruction stream to prevent inadvertent victim ejections.

2.7 Interrupts

Interrupts and general-purpose inputs are acquired through a free-running 64-bit external shift register. Typically, the shift register is implemented with one or more 74HC165 chips. The shift register does not need to be fully implemented if fewer than 64 bits of interrupts and general-purpose inputs are needed. The **int_sr_load_l** signal is asserted low to load the interrupts into the shift register. The **int_sr_clk** signal clocks the shift register contents into the 21174 through the **int_sr_in** pin. The shift register clock rate depends on the CPU speed. See Table 2–3 to determine the shift register rate and latency time. If fewer than 64 inputs are needed, the interrupt latency can be reduced by writing a smaller value into the IRQ_COUNT field of the INT_CNFG register.

Table 2–3 Shift Register Rates and WC Latency Times

CPU Speed	System Clock Rate	Shift Register Rate	WC Latency
466 MHz	66.67 MHz	16.67 MHz	3.84 usec (approx.)
533 MHz	66.67 MHz	16.67 MHz	3.84 usec (approx.)
600 MHz	66.67 MHz	16.67 MHz	3.84 usec (approx.)

The normal active state of interrupts is active low. The INT_HILO register is provided to allow for devices that are active high, such as the 82378 ISA bridge. The register provides for eight devices that can be made active high. Setting a bit in this register causes the active state of the interrupt to be changed from active low to active high.

The state of each interrupt or input can be read through the interrupt request register. The state of the interrupts will persist in the interrupt register for up to 3 µsec after the interrupt has been deasserted at the shift register input. If the interrupt bit in the interrupt request register is not promptly cleared, a second interrupt might be taken before the shift register scans the deasserted value into the interrupt request register. For this reason, interrupts latched in the interrupt request register can be reset individually by writing a 1 to the bit to be cleared. This immediately clears the bit to avoid taking a second interrupt. It is also acceptable to write 1 to all interrupt bits within the interrupt request register.

The interrupt mask register provides individual mask bits for each interrupt. The bits used for general-purpose inputs should be masked using this register.

In normal operation, all external interrupt sources are routed to **irq<1>_h** on the 21164. In some instances, it may be necessary to route the sources to **irq<0>_h** or **irq<3:2>_h**. The 21174 provides for eight external interrupt sources that can be routed to different 21164 interrupt request lines. See the description of the interrupt routine select register (INT_ROUTE) in Section 5.9.4 for a description of this function.

2.8 General-Purpose Inputs and Outputs

General-purpose inputs can be configured using the interrupt's external shift register. When the interrupt's external shift register is used for general-purpose inputs, the interrupt enable bits should be deasserted.

General-purpose outputs can be implemented with one or more 74HC595 chips. The contents of the general-purpose output register are continuously transferred to the shift register. The worst case delay for output posting is approximately 3 µsec, or less if the IRQ_COUNT field of the INT_CNFG register is programmed to reduce the length of the shift register cycle time.

2.9 Programmed 21164 Reset

The 21164 processor can be restarted by setting the DO_RESET in the power control register. This is usually done to change the 21164 frequency or the **sys_clk** divider ratio. The **sys_clk** divider ratio can be set in the CSR_CLOCK_DIVIDE and CSR_PCLK_DIVIDE fields of the clock control register. While **dc_ok** is asserted, the initial value for this field is loaded from pull-up and pull-down resistors attached to the pins. This field can be written under program control, and the new value is used during any subsequent 21164 reset.

The duration of a programmed 21164 reset can be controlled by writing to the RESET_PULSE_WIDTH field in the power-down timing register.

2.10 Clock

This section explains the internal clock PLL and the DRAM clock aligner.

2.10.1 Clock PLL

An onchip PLL generates an internal clock at two times the **sys_clk** rate. This clock is used to derive all other clocks. The fast clock is divided again by two or three to make a **sys_clk** replica clock that is used to clock the internal 21174 logic. The fast clock is divided by 4 to generate the PCI clocks.

2.10.2 DRAM Clock Aligner

A precision clock aligner generates clocks for the SDRAMs. The clock aligner allows the module designer to position the external clocks accurately within the overall clock cycle to maximize the margins for setup and hold times for the various system components.

The clock aligner consists of a group of 128 delay elements. Some portion of the 128 delay elements can be bypassed by way of the clock control register (CCR) bits <31:24>. The delay of each element is nominally 150 ps, but the delay can vary with process, supply voltage, and operating temperature.

To calibrate the delay for any given operating point, a phase comparator is provided to compare a dummy copy of the DRAM clock to the input **sys_clk**. The dummy copy can be externally delayed through module etch to create additional lead time between the DRAM clocks and the **sys_clk**. For example, if the DRAM clocks are to lead **sys_clk** by 3 ns, the dummy feedback clock etch should be laid out with 3 ns of additional module etch beyond the etch length needed to match the DRAM clock distribution etch.

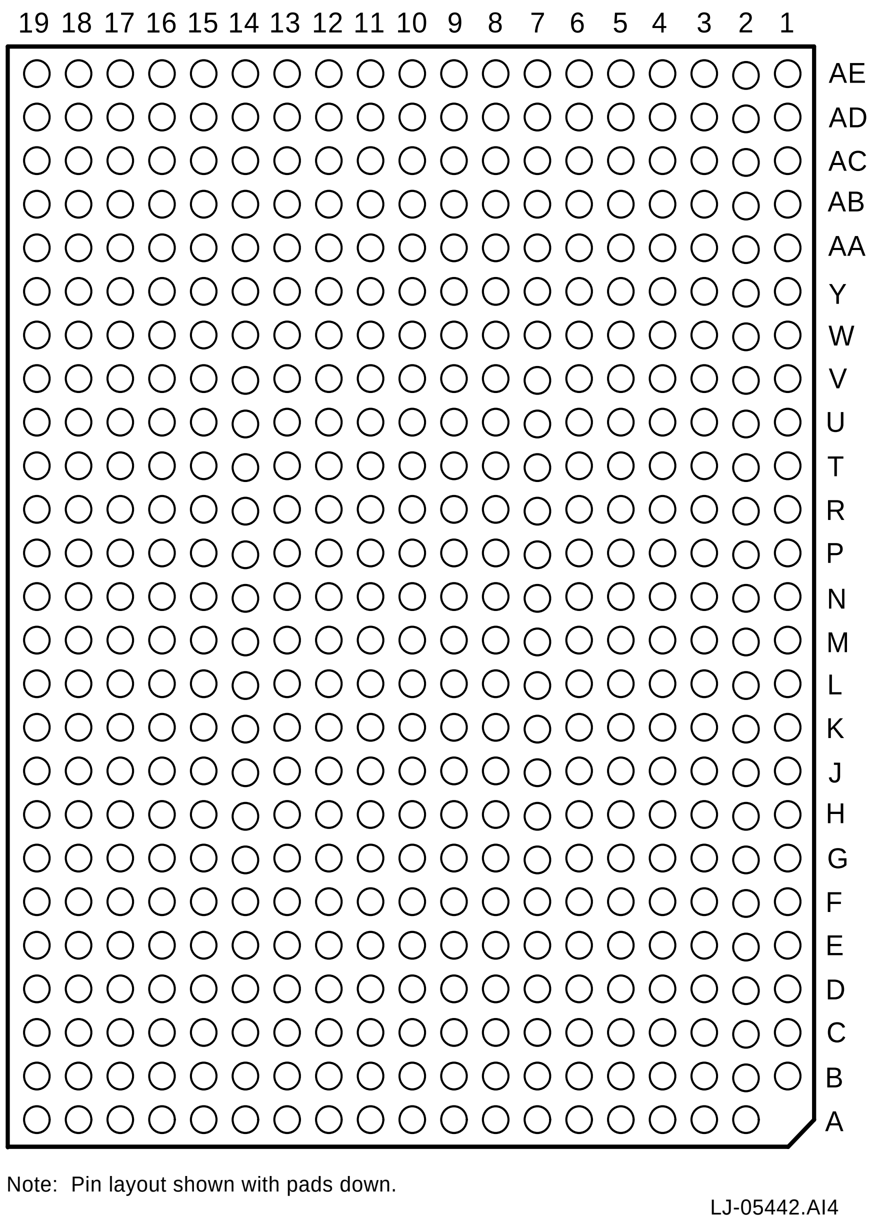
3 Pinout

This chapter describes the 21174 signals and pinouts in the following tables:

- Table 3–1 lists the 21174 pins in alphanumeric order.
- Table 3–2 lists the power and ground pins.
- Table 3–3 lists the 21174 signals in alphanumeric order.
- Table 3–4 describes the signals.

Figure 3–1 shows the physical pin layout of the 474-pin 21174 BGA.

Figure 3–1 21174 BGA Pin Assignment (Pads Down)



3.1 Pin List (Alphanumeric)

Table 3–1 lists the 21174 pins in alphanumeric order (I/O pins are bidirectional).

Table 3–1 Pin List (Alphanumeric) *(Sheet 1 of 8)*

Pin	Signal Name	Type	Pin	Signal Name	Type
A02	**cmd<1>**	I/O	A03	**cache_isolate<1>**	Output
A04	**mem_cs_l**	Input	A05	**int4_valid<2>**	Input
A06	**sram_clk_en<0>**	Output	A07	**int4_valid<0>**	Input
A08	**we_l**	Output	A09	**fill_id**	Output
A10	**cs1a_l**	Output	A11	**dack**	Output
A12	**cs1b_l**	Output	A13	**addr_res<1>**	Input
A14	**data<127>**	I/O	A15	**flash_ce_l**	Output
A16	**data<95>**	I/O	A17	**data<63>**	I/O
A18	**data<31>**	I/O	AA01	**ad<47>**	I/O
AA03	**ad<46>**	I/O	AA04	**ad<58>**	I/O
AA05	**ad<35>**	I/O	AA07	**ad<17>**	I/O
AA09	**ad<10>**	I/O	AA11	**par64**	I/O
AA13	**serr_l**	Input	AA15	**data<103>**	I/O
AA16	**data<69>**	I/O	AA17	**data<37>**	I/O
AA19	**data<4>**	I/O	AB01	**ad<43>**	I/O
AB02	**ad<44>**	I/O	AB03	**ad<38>**	I/O
AB04	**ad<30>**	I/O	AB05	**ad<34>**	I/O
AB06	**ad<22>**	I/O	AB07	**ad<18>**	I/O
AB08	**ad<14>**	I/O	AB09	**ad<11>**	I/O
AB10	**devsel_l**	I/O	AB11	**perr_l**	I/O
AB12	**spare1**	N	AB13	**data<99>**	I/O
AB14	**data<100>**	I/O	AB15	**data<101>**	I/O
AB16	**data<102>**	I/O	AB17	**data<68>**	I/O
AB18	**data<36>**	I/O	AB19	**data<3>**	I/O

Pin List (Alphanumeric)

Table 3–1 Pin List (Alphanumeric) *(Sheet 2 of 8)*

Pin	Signal Name	Type	Pin	Signal Name	Type
AC01	**ad<42>**	I/O	AC02	**ad<39>**	I/O
AC03	**ad<37>**	I/O	AC05	**ad<25>**	I/O
AC06	**ad<23>**	I/O	AC06	**ad<23>**	I/O
AC07	**ad<19>**	I/O	AC09	**pci_cbe_l<3>**	I/O
AC10	**irdy_l**	I/O	AC11	**pci_cbe_l<0>**	I/O
AC13	**req_l**	Output	AC14	**data<98>**	I/O
AC15	**test_out**	Output	AC17	**data<67>**	I/O
AC18	**data<35>**	I/O	AC19	**data<2>**	I/O
AD01	**ad<41>**	I/O	AD02	**ad<40>**	I/O
AD03	**ad<33>**	I/O	AD04	**ad<28>**	I/O
AD05	**ad<26>**	I/O	AD07	**ad<20>**	I/O
AD08	**ad<15>**	I/O	AD09	**ad<12>**	I/O
AD11	**par**	I/O	AD12	**data<97>**	I/O
AD13	**gnt_l**	Input	AD15	**data<65>**	I/O
AD16	**data<66>**	I/O	AD17	**data<33>**	I/O
AD18	**data<34>**	I/O	AD19	**data<1>**	I/O
AE02	**ad<36>**	I/O	AE03	**ad<31>**	I/O
AE04	**ad<29>**	I/O	AE05	**ad<27>**	I/O
AE06	**ad<24>**	I/O	AE07	**ad<21>**	I/O
AE08	**pci_cbe_l<5>**	I/O	AE09	**pci_cbe_l<4>**	I/O
AE10	**pci_cbe_l<6>**	I/O	AE11	**frame_l**	I/O
AE12	**trdy_l**	I/O	AE13	**sram_clk_in**	Input
AE14	**data<96>**	I/O	AE15	**rst_l**	Output
AE16	**data<64>**	I/O	AE17	**data<32>**	I/O
AE18	**data<0>**	I/O	B01	**addr_h<21>**	I/O
B02	**addr_h<20>**	I/O	B03	**data_bus_req**	Output

Table 3–1 Pin List (Alphanumeric) *(Sheet 3 of 8)*

Pin	Signal Name	Type	Pin	Signal Name	Type
B04	**sram_addr<5>**	Output	B05	**dram_addr<12>**	Output
B07	**dram_addr<5>**	Output	B08	**dram_addr<8>**	Output
B09	**drive_tag_ctl_l**	I/O	B11	**cas_l**	Output
B12	**dram_clk<8>**	Output	B13	**dram_clk<11>**	Output
B15	**dram_clk<4>**	Output	B16	**data<126>**	I/O
B17	**data<94>**	I/O	B18	**data<62>**	I/O
B19	**data<30>**	I/O	C01	**addr_h<19>**	I/O
C02	**addr_h<18>**	I/O	C03	**cmd<2>**	I/O
C05	**victim_pending**	Input	C06	**sram_addr<4>**	Output
C07	**int4_valid<1>**	Input	C09	**fill_error**	Output
C10	**dram_cke**	Output	C11	**fill**	Output
C13	**addr_res<0>**	Input	C14	**data<125>**	I/O
C15	**pll_lock**	Output	C17	**data<93>**	I/O
C18	**data<61>**	I/O	C19	**data<29>**	I/O
D01	**addr_h<17>**	I/O	D02	**addr_h<16>**	I/O
D03	**cmd<3>**	I/O	D04	**cache_isolate<2>**	Output
D05	**idle_bc**	Output	D06	**dram_addr<9>**	Output
D07	**bank01_l**	Output	D08	**dram_addr<6>**	Output
D09	**dram_addr<2>**	Output	D10	**dram_addr<0>**	Output
D11	**ras_l**	Output	D12	**dram_clk<10>**	Output
D13	**dram_clk<2>**	Output	D14	**dram_clk<7>**	Output
D15	**data<124>**	I/O	D16	**data<92>**	I/O
D17	**data<91>**	I/O	D18	**data<60>**	I/O
D19	**data<28>**	I/O	E01	**addr_h<15>**	I/O
E03	**addr_h<14>**	I/O	E04	**addr_h<13>**	I/O
E05	**cmd<0>**	I/O	E07	**int4_valid<3>**	Input

Pin List (Alphanumeric)

Table 3–1 Pin List (Alphanumeric) *(Sheet 4 of 8)*

Pin	Signal Name	Type	Pin	Signal Name	Type
E09	**tag_dirty**	Output	E11	**cack**	Output
E13	**flash_we_l**	Output	E15	**data<123>**	I/O
E16	**data<90>**	I/O	E17	**data<59>**	I/O
E19	**data<27>**	I/O	F01	**addr_h<12>**	I/O
F02	**addr_h<11>**	I/O	F03	**cpu_reset_l**	Output
F04	**addr_h<37>**	I/O	F05	**addr_h<9>**	I/O
F06	**addr_bus_req**	Output	F07	**bank23_l**	Output
F08	**dram_addr<11>**	Output	F09	**dram_addr<1>**	Output
F10	**cs2a_l**	Output	F11	**dram_clk<12>**	Output
F12	**dram_clk<0>**	Output	F13	**dram_clk<1>**	Output
F14	**data<122>**	I/O	F15	**data<121>**	I/O
F16	**data<88>**	I/O	F17	**data<89>**	I/O
F18	**data<58>**	I/O	F19	**data<26>**	I/O
G01	**addr_h<8>**	I/O	G02	**addr_h<7>**	I/O
G03	**addr_h<6>**	I/O	G05	**addr_h<30>**	I/O
G07	**sram_clk_en<1>**	Output	G09	**dram_addr<3>**	Output
G11	**dqm**	Output	G13	**data<120>**	I/O
G15	**data<119>**	I/O	G17	**data<57>**	I/O
G18	**data<25>**	I/O	G19	**data<24>**	I/O
H01	**sram_fill_clk<2>**	Output	H02	**addr_h<4>**	I/O
H03	**sram_fill_clk<1>**	Output	H04	**addr_h<39>**	I/O
H05	**addr_h<38>**	I/O	H06	**addr_h<10>**	I/O
H07	**addr_h<36>**	I/O	H08	**cache_isolate<0>**	Output
H09	**dram_addr<13>**	Output	H10	**cs2b_l**	Output
H11	**dram_clk<3>**	Output	H12	**data<118>**	I/O
H13	**data<117>**	I/O	H14	**data<116>**	I/O

Table 3–1 Pin List (Alphanumeric) *(Sheet 5 of 8)*

Pin	Signal Name	Type	Pin	Signal Name	Type
H15	**data<86>**	I/O	H16	**data<87>**	I/O
H17	**data<56>**	I/O	H18	**data<23>**	I/O
H19	**data<22>**	I/O	J01	**addr_h<35>**	I/O
J03	**addr_h<34>**	I/O	J05	**addr_h<33>**	I/O
J07	**addr_h<32>**	I/O	J09	**dram_addr<7>**	Output
J11	**dram_clk<9>**	Output	J13	**data<115>**	I/O
J15	**data<85>**	I/O	J17	**data<55>**	I/O
J19	**data<21>**	I/O	K01	**sys_reset_l**	Output
K02	**addr_cmd_par**	I/O	K03	**dram_clk_in**	Input
K04	**addr_h<31>**	I/O	K05	**sys_clk**	Input
K06	**addr_h<5>**	I/O	K07	**addr_h<29>**	I/O
K08	**bank45_l**	Output	K09	**dram_addr<10>**	Output
K10	**cs0a_l**	Output	K11	**dram_clk<5>**	Output
K12	**data<114>**	I/O	K13	**data<82>**	I/O
K14	**data<83>**	I/O	K15	**data<84>**	I/O
K16	**data<54>**	I/O	K17	**data<53>**	I/O
K18	**data<52>**	I/O	K19	**ecchi<7>**	I/O
L01	**cpu_pwr_en**	Output	L02	**addr_h<28>**	I/O
L03	**addr_h<27>**	I/O	L05	**addr_h<26>**	I/O
L07	**bank67_l**	Output	L09	**dram_addr<4>**	Output
L11	**dram_clk<6>**	Output	L13	**data<113>**	I/O
L15	**data<81>**	I/O	L17	**data<51>**	I/O
L18	**data<20>**	I/O	L19	**data<19>**	I/O
M01	**test_mode<0>**	Input	M02	**alt_clk**	Input
M03	**test_ri**	Input	M04	**addr_h<25>**	I/O
M05	**sram_fill_clk<0>**	Output	M06	**addr_h<24>**	I/O

Pin List (Alphanumeric)

Table 3–1 Pin List (Alphanumeric) *(Sheet 6 of 8)*

Pin	Signal Name	Type	Pin	Signal Name	Type
M07	**addr_h<23>**	I/O	M08	**fan_on**	Output
M10	**cs0b_l**	Output	M12	**data<112>**	I/O
M13	**data<80>**	I/O	M14	**data<50>**	I/O
M15	**data<49>**	I/O	M16	**data<48>**	I/O
M17	**data<18>**	I/O	M18	**data<17>**	I/O
M19	**ecchi<6>**	I/O	N01	**pll_fixed_vdd**	Input
N03	**bypassn**	Input	N05	**irq<3>**	I/O
N07	**addr_h<22>**	I/O	N09	**fan_high**	Output
N11	**ecchi<5>**	I/O	N13	**ecchi<4>**	I/O
N15	**data<16>**	I/O	N17	**ecchi<3>**	I/O
N19	**ecchi<2>**	I/O	P01	**dc_ok**	Input
P02	**halt_irq**	I/O	P03	**int_sr_in**	Input
P04	**dimm_sda**	I/O	P05	**clk_in**	Input
P06	**dimm_scl**	I/O	P07	**ad<63>**	I/O
P08	**irq<2>**	I/O	P10	**pci_cbe_l<7>**	I/O
P12	**ecchi<1>**	I/O	P13	**data<47>**	I/O
P14	**data<15>**	I/O	P15	**data<14>**	I/O
P16	**ecchi<0>**	I/O	P17	**ecclo<5>**	I/O
P18	**ecclo<6>**	I/O	P19	**ecclo<7>**	I/O
R01	**irq<1>**	I/O	R02	**clk<6>**	Output
R03	**ad<59>**	I/O	R05	**clk<3>**	Output
R07	**clk<1>**	Output	R09	**ad<1>**	I/O
R11	**req64_l**	I/O	R13	**data<46>**	I/O
R15	**data<45>**	I/O	R17	**data<44>**	I/O
R18	**data<13>**	I/O	R19	**ecclo<4>**	I/O
T01	**pll_avdd**	Input	T02	**pwr_fail_irq**	I/O

Table 3–1 Pin List (Alphanumeric) *(Sheet 7 of 8)*

Pin	Signal Name	Type	Pin	Signal Name	Type
T03	**int_sr_load_l**	Output	T04	**clk<5>**	Output
T05	**irq<0>**	I/O	T06	**ad<54>**	I/O
T07	**clk<0>**	Output	T08	**ad<3>**	I/O
T09	**ad<2>**	I/O	T10	**ad<0>**	I/O
T11	**data<43>**	I/O	T12	**data<42>**	I/O
T13	**data<79>**	I/O	T14	**data<78>**	I/O
T15	**ecclo<3>**	I/O	T16	**data<12>**	I/O
T17	**data<11>**	I/O	T18	**ecclo<2>**	I/O
T19	**ecclo<1>**	I/O	U01	**ad<60>**	I/O
U03	**clk<2>**	Output	U05	**clk<4>**	Output
U07	**ad<45>**	I/O	U09	**ad<4>**	I/O
U11	**data<111>**	I/O	U13	**data<41>**	I/O
U15	**data<77>**	I/O	U17	**data<9>**	I/O
U19	**data<10>**	I/O	V01	**test_di1**	Input
V02	**ad<56>**	I/O	V03	**test_di2**	Input
V04	**int_clk**	Output	V05	**ad<62>**	I/O
V06	**ad<51>**	I/O	V07	**ad<48>**	I/O
V08	**ad<7>**	I/O	V09	**ad<5>**	I/O
V10	**pci_cbe_l<1>**	I/O	V11	**data<109>**	I/O
V12	**data<110>**	I/O	V13	**data<73>**	I/O
V14	**data<74>**	I/O	V15	**data<75>**	I/O
V16	**data<76>**	I/O	V17	**data<40>**	I/O
V18	**data<8>**	I/O	V19	**data<7>**	I/O
W01	**ad<52>**	I/O	W02	**ad<55>**	I/O
W03	**ad<57>**	I/O	W05	**gp_sr_out**	Output
W07	**ad<8>**	I/O	W09	**ad<6>**	I/O

Pin List (Alphanumeric)

Table 3–1 Pin List (Alphanumeric) *(Sheet 8 of 8)*

Pin	Signal Name	Type	Pin	Signal Name	Type
W11	**ack64_l**	I/O	W13	**data<108>**	I/O
W15	**data<71>**	I/O	W17	**data<72>**	I/O
W18	**data<39>**	I/O	W19	**data<6>**	I/O
Y01	**ad<50>**	I/O	Y02	**ad<49>**	I/O
Y03	**mchk_irq**	I/O	Y04	**ad<61>**	I/O
Y05	**ad<53>**	I/O	Y06	**ad<32>**	I/O
Y07	**ad<16>**	I/O	Y08	**ad<13>**	I/O
Y09	**ad<9>**	I/O	Y10	**pci_cbe_l<2>**	I/O
Y11	**stop_l**	I/O	Y12	**data<104>**	I/O
Y13	**data<105>**	I/O	Y14	**data<106>**	I/O
Y15	**data<107>**	I/O	Y16	**data<70>**	I/O
Y17	**ecclo<0>**	I/O	Y18	**data<38>**	I/O
Y19	**data<5>**	I/O			

Table 3–2 lists the 21174 power and ground pins.

Table 3–2 Power and Ground Pin List *(Sheet 1 of 2)*

Signal	PGA Location							
+3V	W16	AA6	AA10	AA14	AC4	AC8	AC12	AC16
	R12	R16	U2	U6	U10	U14	U18	W4
	W8	W12	M9	M11	N2	N6	N14	N18
	P9	P11	R4	R8	G16	J2	J6	J10
	J14	J18	L4	L8	L12	L16	C4	C8
	C12	C16	E6	E10	E14	G4	G8	G12
GND	AE1	AE19	W6	W10	W14	AA2	AA8	AA12
	AA18	AD6	AD10	AD14	N10	N12	N16	R6
	R10	R14	U4	U8	U12	U16	G14	J4

Table 3–2 Power and Ground Pin List *(Sheet 2 of 2)*

Signal	PGA Location							
	J8	J12	J16	L6	L10	L14	N4	N8
	A19	B6	B10	B14	E2	E8	E12	E18
	G6	G10	—	—	—	—	—	—

3.2 Signal List (Alphanumeric)

Table 3–3 lists the 21174 signals in alphanumeric order.

Table 3–3 Signal List (Alphanumeric) *(Sheet 1 of 8)*

Name	Pin	Type	Name	Pin	Type
ack64_l	W11	I/O	**ad<0>**	T10	I/O
ad<1>	R09	I/O	**ad<2>**	T09	I/O
ad<3>	T08	I/O	**ad<4>**	U09	I/O
ad<5>	V09	I/O	**ad<6>**	W09	I/O
ad<7>	V08	I/O	**ad<8>**	W07	I/O
ad<9>	Y09	I/O	**ad<10>**	AA09	I/O
ad<11>	AB09	I/O	**ad<12>**	AD09	I/O
ad<13>	Y08	I/O	**ad<14>**	AB08	I/O
ad<15>	AD08	I/O	**ad<16>**	Y07	I/O
ad<17>	AA07	I/O	**ad<18>**	AB07	I/O
ad<19>	AC07	I/O	**ad<20>**	AD07	I/O
ad<21>	AE07	I/O	**ad<22>**	AB06	I/O
ad<23>	AC06	I/O	**ad<24>**	AE06	I/O
ad<25>	AC05	I/O	**ad<26>**	AD05	I/O
ad<27>	AE05	I/O	**ad<28>**	AD04	I/O
ad<29>	AE04	I/O	**ad<30>**	AB04	I/O
ad<31>	AE03	I/O	**ad<32>**	Y06	I/O
ad<33>	AD03	I/O	**ad<34>**	AB05	I/O

Signal List (Alphanumeric)

Table 3–3 Signal List (Alphanumeric) *(Sheet 2 of 8)*

Name	Pin	Type	Name	Pin	Type
ad<35>	AA05	I/O	**ad<36>**	AE02	I/O
ad<37>	AC03	I/O	**ad<38>**	AB03	I/O
ad<39>	AC02	I/O	**ad<40>**	AD02	I/O
ad<41>	AD01	I/O	**ad<42>**	AC01	I/O
ad<43>	AB01	I/O	**ad<44>**	AB02	I/O
ad<45>	U07	I/O	**ad<46>**	AA03	I/O
ad<47>	AA01	I/O	**ad<48>**	V07	I/O
ad<49>	Y02	I/O	**ad<50>**	Y01	I/O
ad<51>	V06	I/O	**ad<52>**	W01	I/O
ad<53>	Y05	I/O	**ad<54>**	T06	I/O
ad<55>	W02	I/O	**ad<56>**	V02	I/O
ad<57>	W03	I/O	**ad<58>**	AA04	I/O
ad<59>	R03	I/O	**ad<60>**	U01	I/O
ad<61>	Y04	I/O	**ad<62>**	V05	I/O
ad<63>	P07	I/O	**addr_h<4>**	H02	I/O
addr_h<5>	K06	I/O	**addr_h<6>**	G03	I/O
addr_h<7>	G02	I/O	**addr_h<8>**	G01	I/O
addr_h<9>	F05	I/O	**addr_h<10>**	H06	I/O
addr_h<11>	F02	I/O	**addr_h<12>**	F01	I/O
addr_h<13>	E04	I/O	**addr_h<14>**	E03	I/O
addr_h<15>	E01	I/O	**addr_h<16>**	D02	I/O
addr_h<17>	D01	I/O	**addr_h<18>**	C02	I/O
addr_h<19>	C01	I/O	**addr_h<20>**	B02	I/O
addr_h<21>	B01	I/O	**addr_h<22>**	N07	I/O
addr_h<23>	M07	I/O	**addr_h<24>**	M06	I/O
addr_h<25>	M04	I/O	**addr_h<26>**	L05	I/O

Table 3–3 Signal List (Alphanumeric) *(Sheet 3 of 8)*

Name	Pin	Type	Name	Pin	Type
addr_h<27>	L03	I/O	**addr_h<28>**	L02	I/O
addr_h<29>	K07	I/O	**addr_h<30>**	G05	I/O
addr_h<31>	K04	I/O	**addr_h<32>**	J07	I/O
addr_h<33>	J05	I/O	**addr_h<34>**	J03	I/O
addr_h<35>	J01	I/O	**addr_h<36>**	H07	I/O
addr_h<37>	F04	I/O	**addr_h<38>**	H05	I/O
addr_h<39>	H04	I/O	**addr_bus_req**	F06	Output
addr_cmd_par	K02	I/O	**addr_res<0>**	C13	Input
addr_res<1>	A13	Input	**alt_clk**	M02	Input
bank01_l	D07	Output	**bank23_l**	F07	Output
bank45_l	K08	Output	**bank67_l**	L07	Output
bypassn	N03	Input	**cache_isolate<0>**	H08	Output
cache_isolate<1>	A03	Output	**cache_isolate<2>**	D04	Output
cack	E11	Output	**cas_l**	B11	Output
clk<0>	T07	Output	**clk<1>**	R07	Output
clk<2>	U03	Output	**clk<3>**	R05	Output
clk<4>	U05	Output	**clk<5>**	T04	Output
clk<6>	R02	Output	**clk_in**	P05	Input
cmd<0>	E05	I/O	**cmd<1>**	A02	I/O
cmd<2>	C03	I/O	**cmd<3>**	D03	I/O
cpu_pwr_en	L01	Output	**cpu_reset_l**	F03	Output
cs0a_l	K10	Output	**cs0b_l**	M10	Output
cs1a_l	A10	Output	**cs1b_l**	A12	Output
cs2a_l	F10	Output	**cs2b_l**	H10	Output
dack	A11	Output	**data<0>**	AE18	I/O
data<1>	AD19	I/O	**data<2>**	AC19	I/O

Signal List (Alphanumeric)

Table 3–3 Signal List (Alphanumeric) *(Sheet 4 of 8)*

Name	Pin	Type	Name	Pin	Type
data<3>	AB19	I/O	**data**<4>	AA19	I/O
data<5>	Y19	I/O	**data**<6>	W19	I/O
data<7>	V19	I/O	**data**<8>	V18	I/O
data<9>	U17	I/O	**data**<10>	U19	I/O
data<11>	T17	I/O	**data**<12>	T16	I/O
data<13>	R18	I/O	**data**<14>	P15	I/O
data<15>	P14	I/O	**data**<16>	N15	I/O
data<17>	M18	I/O	**data**<18>	M17	I/O
data<19>	L19	I/O	**data**<20>	L18	I/O
data<21>	J19	I/O	**data**<22>	H19	I/O
data<23>	H18	I/O	**data**<24>	G19	I/O
data<25>	G18	I/O	**data**<26>	F19	I/O
data<27>	E19	I/O	**data**<28>	D19	I/O
data<29>	C19	I/O	**data**<30>	B19	I/O
data<31>	A18	I/O	**data**<32>	AE17	I/O
data<33>	AD17	I/O	**data**<34>	AD18	I/O
data<35>	AC18	I/O	**data**<36>	AB18	I/O
data<37>	AA17	I/O	**data**<38>	Y18	I/O
data<39>	W18	I/O	**data**<40>	V17	I/O
data<41>	U13	I/O	**data**<42>	T12	I/O
data<43>	T11	I/O	**data**<44>	R17	I/O
data<45>	R15	I/O	**data**<46>	R13	I/O
data<47>	P13	I/O	**data**<48>	M16	I/O
data<49>	M15	I/O	**data**<50>	M14	I/O
data<51>	L17	I/O	**data**<52>	K18	I/O
data<53>	K17	I/O	**data**<54>	K16	I/O

Table 3–3 Signal List (Alphanumeric) *(Sheet 5 of 8)*

Name	Pin	Type	Name	Pin	Type
data<55>	J17	I/O	**data<56>**	H17	I/O
data<57>	G17	I/O	**data<58>**	F18	I/O
data<59>	E17	I/O	**data<60>**	D18	I/O
data<61>	C18	I/O	**data<62>**	B18	I/O
data<63>	A17	I/O	**data<64>**	AE16	I/O
data<65>	AD15	I/O	**data<66>**	AD16	I/O
data<67>	AC17	I/O	**data<68>**	AB17	I/O
data<69>	AA16	I/O	**data<70>**	Y16	I/O
data<71>	W15	I/O	**data<72>**	W17	I/O
data<73>	V13	I/O	**data<74>**	V14	I/O
data<75>	V15	I/O	**data<76>**	V16	I/O
data<77>	U15	I/O	**data<78>**	T14	I/O
data<79>	T13	I/O	**data<80>**	M13	I/O
data<81>	L15	I/O	**data<82>**	K13	I/O
data<83>	K14	I/O	**data<84>**	K15	I/O
data<85>	J15	I/O	**data<86>**	H15	I/O
data<87>	H16	I/O	**data<88>**	F16	I/O
data<89>	F17	I/O	**data<90>**	E16	I/O
data<91>	D17	I/O	**data<92>**	D16	I/O
data<93>	C17	I/O	**data<94>**	B17	I/O
data<95>	A16	I/O	**data<96>**	AE14	I/O
data<97>	AD12	I/O	**data<98>**	AC14	I/O
data<99>	AB13	I/O	**data<100>**	AB14	I/O
data<101>	AB15	I/O	**data<102>**	AB16	I/O
data<103>	AA15	I/O	**data<104>**	Y12	I/O
data<105>	Y13	I/O	**data<106>**	Y14	I/O

Signal List (Alphanumeric)

Table 3–3 Signal List (Alphanumeric) *(Sheet 6 of 8)*

Name	Pin	Type	Name	Pin	Type
data<107>	Y15	I/O	**data<108>**	W13	I/O
data<109>	V11	I/O	**data<110>**	V12	I/O
data<111>	U11	I/O	**data<112>**	M12	I/O
data<113>	L13	I/O	**data<114>**	K12	I/O
data<115>	J13	I/O	**data<116>**	H14	I/O
data<117>	H13	I/O	**data<118>**	H12	I/O
data<119>	G15	I/O	**data<120>**	G13	I/O
data<121>	F15	I/O	**data<122>**	F14	I/O
data<123>	E15	I/O	**data<124>**	D15	I/O
data<125>	C14	I/O	**data<126>**	B16	I/O
data<127>	A14	I/O	**data_bus_req**	B03	Output
dc_ok	P01	Input	**devsel_l**	AB10	I/O
dimm_scl	P06	I/O	**dimm_sda**	P04	I/O
dqm	G11	Output	**dram_addr<0>**	D10	Output
dram_addr<1>	F09	Output	**dram_addr<2>**	D09	Output
dram_addr<3>	G09	Output	**dram_addr<4>**	L09	Output
dram_addr<5>	B07	Output	**dram_addr<6>**	D08	Output
dram_addr<7>	J09	Output	**dram_addr<8>**	B08	Output
dram_addr<9>	D06	Output	**dram_addr<10>**	K09	Output
dram_addr<11>	F08	Output	**dram_addr<12>**	B05	Output
dram_addr<13>	H09	Output	**dram_cke**	C10	Output
dram_clk<0>	F12	Output	**dram_clk<1>**	F13	Output
dram_clk<2>	D13	Output	**dram_clk<3>**	H11	Output
dram_clk<4>	B15	Output	**dram_clk<5>**	K11	Output
dram_clk<6>	L11	Output	**dram_clk<7>**	D14	Output
dram_clk<8>	B12	Output	**dram_clk<9>**	J11	Output

Table 3–3 Signal List (Alphanumeric) *(Sheet 7 of 8)*

Name	Pin	Type	Name	Pin	Type
dram_clk<10>	D12	Output	**dram_clk<11>**	B13	Output
dram_clk<12>	F11	Output	**dram_clk_in**	K03	Input
drive_tag_ctl_l	B09	I/O	**ecchi<0>**	P16	I/O
ecchi<1>	P12	I/O	**ecchi<2>**	N19	I/O
ecchi<3>	N17	I/O	**ecchi<4>**	N13	I/O
ecchi<5>	N11	I/O	**ecchi<6>**	M19	I/O
ecchi<7>	K19	I/O	**ecclo<0>**	Y17	I/O
ecclo<1>	T19	I/O	**ecclo<2>**	T18	I/O
ecclo<3>	T15	I/O	**ecclo<4>**	R19	I/O
ecclo<5>	P17	I/O	**ecclo<6>**	P18	I/O
ecclo<7>	P19	I/O	**fan_high**	N09	Output
fan_on	M08	Output	**fill**	C11	Output
fill_error	C09	Output	**fill_id**	A09	Output
flash_ce_l	A15	Output	**flash_we_l**	E13	Output
frame_l	AE11	I/O	**gnt_l**	AD13	Input
gp_sr_out	W05	Output	**halt_irq**	P02	I/O
idle_bc	D05	Output	**int_clk**	V04	Output
int_sr_in	P03	Input	**int_sr_load_l**	T03	Output
int4_valid<0>	A07	Input	**int4_valid<1>**	C07	Input
int4_valid<2>	A05	Input	**int4_valid<3>**	E07	Input
irdy_l	AC10	I/O	**irq<0>**	T05	I/O
irq<1>	R01	I/O	**irq<2>**	P08	I/O
irq<3>	N05	I/O	**mchk_irq**	Y03	I/O
mem_cs_l	A04	Input	**par**	AD11	I/O
par64	AA11	I/O	**pci_cbe_l<0>**	AC11	I/O
pci_cbe_l<1>	V10	I/O	**pci_cbe_l<2>**	Y10	I/O

Table 3–3 Signal List (Alphanumeric) *(Sheet 8 of 8)*

Name	Pin	Type	Name	Pin	Type
pci_cbe_l<3>	AC09	I/O	**pci_cbe_l<4>**	AE09	I/O
pci_cbe_l<5>	AE08	I/O	**pci_cbe_l<6>**	AE10	I/O
pci_cbe_l<7>	P10	I/O	**perr_l**	AB11	I/O
pll_avdd	T01	Input	**pll_fixed_vdd**	N01	Input
pll_lock	C15	Output	**pwr_fail_irq**	T02	I/O
ras_l	D11	Output	**req_l**	AC13	Output
req64_l	R11	I/O	**rst_l**	AE15	Output
serr_l	AA13	Input	**spare1**	AB12	N
sram_addr<4>	C06	Output	**sram_addr<5>**	B04	Output
sram_clk_en<0>	A06	Output	**sram_clk_en<1>**	G07	Output
sram_clk_in	AE13	Input	**sram_fill_clk<0>**	M05	Output
sram_fill_clk<1>	H03	Output	**sram_fill_clk<2>**	H01	Output
stop_l	Y11	I/O	**sys_clk**	K05	Input
sys_reset_l	K01	Output	**tag_dirty**	E09	Output
test_di1	V01	Input	**test_di2**	V03	Input
test_mode<0>	M01	Input	**test_out**	AC15	Output
test_ri	M03	Input	**trdy_l**	AE12	I/O
victim_pending	C05	Input	**we_l**	A08	Output

3.3 Signal Descriptions

Table 3–4 describes the 21174 signals in alphanumeric order.

Table 3–4 Signal Descriptions (Alphanumeric) *(Sheet 1 of 7)*

Pin	Type	Description
ack64_l	I/O	When this pin is low, it indicates that the target that has responded can transfer data using 64 bits. This pin has a weak pull-up.
ad<31:0>	I/O	PCI bus address, lower 32 bits.
ad<63:32>	I/O	PCI bus address, upper 32 bits, used only in 64-bit configuration.

Table 3–4 Signal Descriptions (Alphanumeric) *(Sheet 2 of 7)*

Pin	Type	Description
addr_bus_req	Output	The 21174 chip asserts this signal line to request use of the address/ command bus (**addr_h<39:4>** and **cmd<3:0>**). It is asserted one cycle before the 21174 transmits on the bus.
addr_cmd_par	I/O	This signal is used to send and receive odd parity for the address/command bus (**addr_h<39:4>** and **cmd<3:0>**). When communicating with 21164, the 21174 sends and receives odd parity. When communicating with flash ROM, this signal is driven by the 21174 chip to avoid a floating state.
addr_h<39:4>	I/O	These signal lines transfer addresses between the 21174 and the 21164. In addition, the 21174 uses some of these signal lines to communicate with the flash ROM (see Table 2–2).
addr_res<1:0>	Input	The 21164 provides cache status information on these signal lines during cache probes. The encoded information is described here: **addr_res<1:0>** — Description 00 — NOP 01 — NOACK — data not found or clean 10 — ACK/Scache — data from Scache 11 — ACK/Bcache — data from L3 cache
alt_clk	Input	Clock reference is provided on this signal line when the 21164 is powered down.
bank01_l	Output	Reserved.
bank23_l	Output	Reserved.
bank45_l	Output	Reserved.
bank67_l	Output	Reserved.
bypassn	Input	Used for PLL manufacturing test.
cache_isolate<2:0>	Output	These pins are asserted to isolate the cache from the rest of the system bus.
cack	Output	This signal is driven by the 21174 chip to acknowledge receipt of a command from the 21164. If the 21174 chip is not ready to accept the command, it will not assert **cack**.
cas_l	Output	This signal is column address select to the DRAM banks.
clk<6:0>	Output	PCI clocks out. These clock signals are driven by the 21174 chip.
clk_in	Input	PCI clock in. One of the **clk** signals is fed back to this pin.

Signal Descriptions

Table 3–4 Signal Descriptions (Alphanumeric) *(Sheet 3 of 7)*

Pin	Type	Description
cmd<3:0>	I/O	The 21174 sends and receives commands to/from the 21164 on these signal lines. The 21164 commands to the 21174 follow: **cmd<3:0>** — Command 0000 — Idle 0001 — Lock 0010 — Fetch (acknowledged but ignored) 0011 — Fetch_m (acknowledged but ignored) 0100 — Memory barrier 0101 — Set dirty (acknowledged but ignored) 0110 — Write block 0111 — Write block lock 1000 — Read miss 0 1001 — Read miss 1 1010 — Read miss mod 0 1011 — Read miss mod 1 1100 — L3 cache victim 1101 — Unused (treated as idle) 1110 — Read miss mod, STC 0 1111 — Read miss mod, STC 1 The 21174 sends the following four commands to the 21164. The remaining command codes are unused. **cmd<3:0>** — Command 0000 — Idle 0001 — Flush 0010 — Inval 0100 — Read
cpu_pwr_en	Output	When this signal is deasserted, the 21164 should be powered down. This signal is not used on systems that do not support independent power-down of the 21164.
cpu_reset_l	Output	The 21174 can demand a 21164 reset on this signal line.
cs0a_l	Output	Selects DIMM pair 0 bank A.
cs0b_l	Output	Selects DIMM pair 0 bank B.
cs1a_l	Output	Selects DIMM pair 1 bank A.
cs1b_l	Output	Selects DIMM pair 1 bank B.
cs2a_l	Output	Selects DIMM pair 2 bank A.
cs2b_l	Output	Selects DIMM pair 2 bank B.

Table 3–4 Signal Descriptions (Alphanumeric) *(Sheet 4 of 7)*

Pin	Type	Description
dack	Output	The 21174 uses this signal to indicate that data will be driven to the 21164 or accepted from the 21164 during the next cycle.
data<127:0>	I/O	These signals carry data between the 21174, 21164, and DRAMs.
data_bus_req	Output	The 21174 chip asserts this signal to request use of the data bus. The signal is asserted one cycle before the 21174 chip drives the bus.
dc_ok	Input	When **dc_ok** is asserted, the 21174 will reset itself and assert **rst_l**. The 21174 will reset various register bits and reset all sequencers to their idle states.
devsel_l	I/O	This pin has a weak pull-up.
dimm_scl	I/O	Clock to serial presence detect on DIMMs.
dimm_sda	I/O	Serial presence detect data pin from DIMMs.
dqm	Output	This signal is asserted high to DRAM banks during reset as some DRAMs require **dqm** to be high during initialization.
dram_addr<13:0>	Output	Address to DRAMs.
dram_cke	Output	Enables the clock to the DRAMs.
dram_clk<12:0>	Output	Direct drive clocks to the DRAMs.
dram_clk_in	Input	This signal is the feedback used to calibrate the **dram_clock<12:0>** clock signals. It is driven from **dram_clock<12>** through an appropriate length of etch and possibly a dummy load.
drive_tag_ctl_l	Output	This signal enables tag control drivers during fill.
ecchi<7:0>	I/O	High quadword ECC to 21164 and DRAMs.
ecclo<7:0>	I/O	Low quadword ECC to 21164 and DRAMs.
fan_high	Output	When high, selects maximum speed for the 21164 fan.
fan_on	Output	When high, this signal enables power to the 21164 fan.
fill	Output	This signal is asserted by the 21174 to the 21164 to indicate that fill data is to be written into the cache.
fill_error	Output	This signal is asserted by the 21174 chip when a bad address is presented by the 21164 or when other errors occur.
fill_id	Output	The 21174 drives this signal when returning data to the 21164. 1 – Data is associated with fill buffer 1. 0 – Data is associated with fill buffer 0.

Signal Descriptions

Table 3–4 Signal Descriptions (Alphanumeric) *(Sheet 5 of 7)*

Pin	Type	Description
flash_ce_l	Output	Chip enable to flash ROM.
flash_we_l	Output	Write enable to flash ROM.
frame_l	I/O	This pin has a weak pull-up.
gnt_l	Input	Indicates that the arbiter has granted the bus to a master.
gp_sr_out	Output	This signal carries the serial data to drive the general-purpose outputs. It is typically connected to the D input of the first 74HC595. If no general-purpose outputs are needed, this signal may be left unconnected.
halt_irq	I/O	Halt interrupt. Operation is similar to **irq<3:0>**.
idle_bc	Output	The 21174 asserts this signal before a fill. The 21164 will respond by releasing the L3 cache for the fill.
int_clk	Output	This signal is the shift clock for the external interrupt shift register and general-purpose output shift register. This signal can be connected directly to the clock pins of the 74HC165 and 74HC595 shift registers.
int_sr_in	Input	This signal carries the serial input data from the external interrupt shift register. It is typically connected to the Q7 output from the last 74HC165 in the chain. This signal is also used as the **testclk** when using the manufacturing test features.
int_sr_load_l	Output	When this signal is asserted low, the external interrupt shift register should be parallel loaded with interrupt data and general-purpose input data, and the general-purpose output data should be copied from the external output shift register into the external holding register. This signal is typically connected to the LD input of 74HC165 input shift registers and to the LCLK input of the 74HC595 output shift registers.
int4_valid<3:0>	Input	These signals are driven by the 21164 to indicate which longwords contain valid data within a given read command or write data cycle. The 21174 chip uses this information to determine which portion of the data is to be read or written to or from the PCI. These signals are also used as the **plltest<3:0>** pins for manufacturing test of the onchip PLL.
irdy_l	I/O	This pin has a weak pull-up.

Table 3–4 Signal Descriptions (Alphanumeric) *(Sheet 6 of 7)*

Pin	Type	Description
irq<3:0>	I/O	These pins are driven by the 21174 chip to send interrupts to the 21164. These pins are also driven during 21164 reset to provide **sys_clk** divider information to the 21164. The state of these pins is sensed when **dc_ok** is asserted and **sys_reset_l** is deasserted and this information is loaded into the interrupt configuration register. The register contents provide the default values to be driven during 21164 reset. See Section 5.9.6.
mchk_irq	I/O	Machine check interrupt. Operation is similar to **irq<3:0>**.
mem_cs_l	Input	This pin carries a programmable address decode signal to a host CPU bridge. The CPU bridge can use this signal to forward a PCI cycle to main memory behind the bridge.
par	I/O	Even parity across **ad<31:0>** and **pci_cbe_l<3:0>.**
par64	I/O	Even parity across **ad<63:32>** and **pci_cbe_l<7:4>**. This pin has a weak pull-up.
pci_cbe_l<3:0>	I/O	PCI command and byte enable.
pci_cbe_l<7:4>	I/O	PCI command and byte enable used only in 64-bit configuration.
perr_l	I/O	This pin has a weak pull-up.
pll_avdd	Input	PLL analog ground.
pll_fixed_vdd	Input	PLL reference ground.
pll_lock	Output	Indicates that the internal PLL has acquired lock. Used during debug and test.
pwr_fail_irq	I/O	Power fail interrupt. Operation is similar to **irq<3:0>**.
ras_l	Output	Row address select to DRAM banks.
req_l	Output	Indicates that a bus master is requesting the bus.
req64_l	I/O	Indicates that a bus master is requesting the bus and can transfer data using 64 bits. This pin has a weak pull-up.
rst_l	Output	PCI bus reset. This signal is used to bring controllers on the PCI bus into a known state. This does not imply that devices attached to a controller have been initialized.
serr_l	Input	This pin has a weak pull-up.

Table 3–4 Signal Descriptions (Alphanumeric) *(Sheet 7 of 7)*

Pin	Type	Description
sram_addr<5:4>	Output	L3 cache DRAM address bits for use during fills. These outputs are switched on **dram_clk**, and provide additional timing margin for the low address bits during cache fills. A pass-transistor multiplexer is used on the cache module to select these address bits during fills.
sram_clk_en<1:0>	Output	This signal controls an external multiplexer (typically QS3257) that selects either the **sram_clk<2:0>** signals, or the **st_clk** signal from the 21164 to drive the L3 Bcache SRAM clock inputs. This signal is asserted when the SRAMs are to be used for non-21164 cycles, that is cycles under control of the 21174 chip. Two copies of this signal are provided.
sram_clk_in	Input	Copy of DRAM clock.
sram_fill_clk<2:0>	Output	This signal clocks the cache SRAMs during L3 Bcache fill and write-back transactions. Three copies of this signal are provided so that it can be fanned out to all cache SRAMs without buffering.
stop_l	I/O	This pin has a weak pull-up.
sys_clk	Input	System clock from 21164. This clock is typically driven from **sys_clk_out2** from 21164 and delayed through module etch to arrive at 21174 at the same time as **sys_clk_out1** is asserted at the 21164.
sys_reset_l	Output	System reset.
tag_dirty	Output	Tag dirty condition bit.
test_di1	Input	Manufacturing test signal.
test_di2	Input	Manufacturing test signal.
test_mode<0>	Input	These bits control the manufacturing test features.
test_out	Output	Manufacturing test output signal.
test_ri	Input	Manufacturing test input signal.
trdy_l	I/O	This pin has a weak pull-up.
victim_pending	Input	Victim available for write.
we_l	Output	Write-enable to DRAMs.

4 Register Definitions

This chapter defines the 21174 registers.

Note: All addresses in this chapter are hexadecimal values unless otherwise noted.

4.1 Register Types

All register addresses are on naturally aligned 64-byte address space boundaries. Table 4–1 lists the categories of 21174 registers.

Table 4–1 21174 Register Categories

Category	Primary User
PCI control registers	Software
Scatter-gather address translation registers	Hardware and software
Error reporting registers	Software and firmware diagnostics
Hardware configuration registers	Firmware and diagnostic
Diagnostic registers	—

Software can read most of the control and status registers (CSRs). Some of the diagnostic registers are reserved for hardware debug and should not be accessed by software. These registers should only be manipulated in a well-controlled environment (such as during the power-up sequence of operations).

4.2 Register Addresses

The CSRs and flash ROM address range is 87.4000.0000 to 87.FFFF.FFFF.

Table 4–2 lists the beginning address of the hardware-specific register groups and the address region for flash ROM.

Table 4–2 Hardware-Specific Register Address Map

Start Address	Selected Region
87.4000.0000	21174 general control, diagnostic, performance monitor, and error log registers
87.5000.0000	21174 memory controller registers
87.6000.0000	21174 PCI window control registers and scatter-gather translation registers
87.7000.0000	Reserved
87.8000.0000	Miscellaneous registers
87.A000.0000	Interrupt control registers
87.C000.0000 to 87.FFFF.FFFF	Flash ROM read and write space — for programming

4.3 General Registers

Table 4–3 lists the 21174 general CSRs.

Table 4–3 General 21174 CSRs (Base = 87.4000.0000)

Name	Mnemonic	Offset	Block
Revision control register	PYXIS_REV	0080	CSR
PCI latency register	PCI_LAT	00C0	CSR
Control register	PYXIS_CTRL	0100	CSR
Control register 1	PYXIS_CTRL1	0140	CSR
Flash control register	FLASH_CTRL	0200	CSR
Hardware address extension register (memory)	HAE_MEM	0400	CSR
Hardware address extension register (I/O)	HAE_IO	0440	CSR
Configuration type register	CFG	0480	CSR

Table 4–4 lists the diagnostic registers.

Table 4–4 Diagnostic Registers (Base = 87.4000.0000)

Name	Mnemonic	Offset	Block
Diagnostic control register	PYXIS_DIAG	2000	CSR
Diagnostic check register	DIAG_CHECK	3000	CSR

Table 4–5 lists the performance monitor registers.

Table 4–5 Performance Monitor Registers (Base = 87.4000.0000)

Name	Mnemonic	Offset	Block
Performance monitor register	PERF_MONITOR	4000	CSR
Performance monitor control register	PERF_CONTROL	4040	CSR

Table 4–6 lists the error registers.

Table 4–6 Error Registers (Base = 87.4000.0000)

Name	Mnemonic	Offset	Block
Error register	PYXIS_ERR	8200	CSR
Status register	PYXIS_STAT	8240	CSR
Error mask register	ERR_MASK	8280	CSR
Syndrome register	PYXIS_SYN	8300	CSR
Error data register	PYXIS_ERR_DATA	8308	CSR
Memory error address register	MEAR	8400	MCTL
Memory error status register	MESR	8440	MCTL
PCI error register 0	PCI_ERR0	8800	PCI
PCI error register 1	PCI_ERR1	8840	PCI
PCI error register 2	PCI_ERR2	8880	PCI

4.4 Memory Controller Registers

Table 4–7 lists the memory controller registers.

Table 4–7 Memory Controller Registers *(Sheet 1 of 2)*

(Base Address = 87.5000.0000)

Name	Mnemonic	Offset	Block
Memory control register	MCR	0000	MCTL
Memory clock mask register	MCMR	0040	MCTL
Global timing register	GTR	0200	MCTL
Refresh timing register	RTR	0300	MCTL
Row history policy mask register	RHPR	0400	MCTL
Memory control debug register 1	MDR1	0500	MCTL
Memory control debug register 2	MDR2	0540	MCTL
Bank base address register 0	BBAR0	0600	MCTL
Bank base address register 1	BBAR1	0640	MCTL
Bank base address register 2	BBAR2	0680	MCTL
Bank base address register 3	BBAR3	06C0	MCTL
Bank base address register 4	BBAR4	0700	MCTL
Bank base address register 5	BBAR5	0740	MCTL
Bank base address register 6	BBAR6	0780	MCTL
Bank base address register 7	BBAR7	07C0	MCTL
Bank configuration register 0	BCR0	0800	MCTL
Bank configuration register 1	BCR1	0840	MCTL
Bank configuration register 2	BCR2	0880	MCTL
Bank configuration register 3	BCR3	08C0	MCTL
Bank configuration register 4	BCR4	0900	MCTL
Bank configuration register 5	BCR5	0940	MCTL
Bank configuration register 6	BCR6	0980	MCTL
Bank configuration register 7	BCR7	09C0	MCTL

Table 4–7 Memory Controller Registers *(Sheet 2 of 2)*

(Base Address = 87.5000.0000)

Name	Mnemonic	Offset	Block
Bank timing register 0	BTR0	0A00	MCTL
Bank timing register 1	BTR1	0A40	MCTL
Bank timing register 2	BTR2	0A80	MCTL
Bank timing register 3	BTR3	0AC0	MCTL
Bank timing register 4	BTR4	0B00	MCTL
Bank timing register 5	BTR5	0B40	MCTL
Bank timing register 6	BTR6	0B80	MCTL
Bank timing register 7	BTR7	0BC0	MCTL
Cache valid map register	CVM	0C00	MCTL

4.5 PCI Window Control Registers

Table 4–8 lists the PCI window control registers.

Table 4–8 PCI Window Control Registers *(Sheet 1 of 2)*

(Base Address = 87.6000.0000)

Name	Mnemonic	Offset	Block
Scatter-gather translation buffer invalidate register	TBIA	0100	PA
Window base 0 register	W0_BASE	0400	PA
Window mask 0 register	W0_MASK	0440	PA
Translated base 0 register	T0_BASE	0480	PA
Window base 1 register	W1_BASE	0500	PA
Window mask l register	W1_MASK	0540	PA
Translated base l register	Tl_BASE	0580	PA
Window base 2 register	W2_BASE	0600	PA
Window mask 2 register	W2_MASK	0640	PA
Translated base 2 register	T2_BASE	0680	PA
Window base 3 register	W3_BASE	0700	PA

Table 4–8 PCI Window Control Registers *(Sheet 2 of 2)*

(Base Address = 87.6000.0000)

Name	Mnemonic	Offset	Block
Window mask 3 register	W3_MASK	0740	PA
Translated base 3 register	T3_BASE	0780	PA
Window DAC base register	W_DAC	07C0	PA

4.6 Scatter-Gather Address Translation Registers

Table 4–9 lists the address translation registers.

Note: See Table 4–8 for information about the scatter-gather translation buffer invalidate register.

Table 4–9 Address Translation Registers *(Sheet 1 of 2)*

(Base Address = 87.6000.0000)

Name	Mnemonic	Offset	Block
Lockable translation buffer tag0 register	LTB_TAG0	0800	PA
Lockable translation buffer tagl register	LTB_TAG1	0840	PA
Lockable translation buffer tag2 register	LTB_TAG2	0880	PA
Lockable translation buffer tag3 register	LTB_TAG3	08C0	PA
Translation buffer tag4 register	TB_TAG4	0900	PA
Translation buffer tag5 register	TB_TAG5	0940	PA
Translation buffer tag6 register	TB_TAG6	0980	PA
Translation buffer tag7 register	TB_TAG7	09C0	PA
Translation buffer 0 page0 register	TB0_PAGE0	1000	PA
Translation buffer 0 pagel register	TB0_PAGE1	1040	PA
Translation buffer 0 page2 register	TB0_PAGE2	1080	PA
Translation buffer 0 page3 register	TB0_PAGE3	10C0	PA
Translation buffer 1 page0 register	TB1_PAGE0	1100	PA
Translation buffer 1 pagel register	TB1_PAGEl	1140	PA

Table 4–9 Address Translation Registers *(Sheet 2 of 2)*

(Base Address = 87.6000.0000)

Name	Mnemonic	Offset	Block
Translation buffer 1 page2 register	TB1_PAGE2	1180	PA
Translation buffer 1 page3 register	TB1_PAGE3	11C0	PA
Translation buffer 2 page0 register	TB2_PAGE0	1200	PA
Translation buffer 2 pagel register	TB2_PAGE1	1240	PA
Translation buffer 2 page2 register	TB2_PAGE2	1280	PA
Translation buffer 2 page3 register	TB2_PAGE3	12C0	PA
Translation buffer 3 page0 register	TB3_PAGE0	1300	PA
Translation buffer 3 page1 register	TB3_PAGE1	1340	PA
Translation buffer 3 page2 register	TB3_PAGE2	1380	PA
Translation buffer 3 page3 register	TB3_PAGE3	13C0	PA
Translation buffer 4 page0 register	TB4_PAGE0	1400	PA
Translation buffer 4 pagel register	TB4_PAGE1	1440	PA
Translation buffer 4 page2 register	TB4_PAGE2	1480	PA
Translation buffer 4 page3 register	TB4_PAGE3	14C0	PA
Translation buffer 5 page0 register	TB5_PAGE0	1500	PA
Translation buffer 5 pagel register	TB5_PAGE1	1540	PA
Translation buffer 5 page2 register	TB5_PAGE2	1580	PA
Translation buffer 5 page3 register	TB5_PAGE3	15C0	PA
Translation buffer 6 page0 register	TB6_PAGE0	1600	PA
Translation buffer 6 pagel register	TB6_PAGE1	1640	PA
Translation buffer 6 page2 register	TB6_PAGE2	1680	PA
Translation buffer 6 page3 register	TB6_PAGE3	16C0	PA
Translation buffer 7 page0 register	TB7_PAGE0	1700	PA
Translation buffer 7 pagel register	TB7_PAGE1	1740	PA
Translation buffer 7 page2 register	TB7_PAGE2	1780	PA
Translation buffer 7 page3 register	TB7_PAGE3	17C0	PA

4.7 Miscellaneous Registers

Table 4–10 lists the 21174 miscellaneous registers.

Table 4–10 Miscellaneous Registers (Base Address = 87.8000.0000)

Name	Mnemonic	Offset	Block
Clock control register	CCR	0000	CSR or CLK
Reserved	—	0040 to 01C0	—
Clock status register	CLK_STAT	0100	CLK
Reserved	—	0240 to 08C0	—
Reset register	RESET	0900	CSR
Reserved	—	0940 to FFFF	—

4.8 Interrupt Control Registers

The interrupt control registers control the external hardware interrupts to the 21174. Table 4–11 defines the registers and the addresses associated with them.

Table 4–11 Interrupt Control Registers (Base Address = 87.A000.0000)

Name	Mnemonic	Offset	Block
Interrupt request register	INT_REQ	0000	IRQ
Interrupt mask register	INT_MASK	0040	IRQ
Interrupt high/low select register	INT_HILO	00C0	IRQ
Interrupt routine select register	INT_ROUTE	0140	IRQ
General-purpose output register	GPO	0180	IRQ
Interrupt configuration register	INT_CNFG	01C0	IRQ
Real-time counter register	RT_COUNT	0200	IRQ
Interrupt time register	INT_TIME	0240	IRQ
Reserved	—	0280	—
I^2C control register	IIC_CTRL	02C0	IRQ

4.9 Flash ROM Address Space

The flash ROM is mapped to three regions of memory. Access to the first two regions is RO. The first two regions provide the software necessary to initialize the system and transfer execution to the next level of software. When power is turned on, address ranges 0 to 00.00FF.FFFF and 0F.FC00.0000 to 0F.FFFF.FFFF are enabled.

After the system has been initialized, these two address ranges are disabled. Byte mode is then enabled in the 21164 and 21174. Byte mode is the only way to access the flash ROM in address range 87.C000.0000 to 87.FFFF.FFFF. 21164 byte instructions LDBU and STB must be used to access this region. Any other instruction will produce UNDEFINED results with the possibility of damaging the flash ROM.

5 Register Descriptions

This chapter describes the 21174 registers in detail. It defines the fields, the access type, and the default register condition. The figures in this chapter show reserved register bit fields in gray.

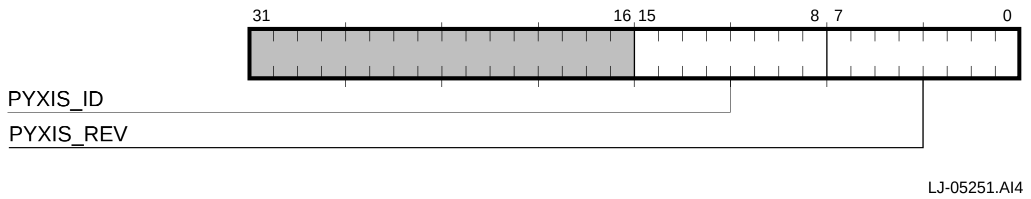
5.1 Registers – General Description

This section describes the functionality of the revision control register, the PCI latency register, the control register, the control register 1, the flash control register, the hardware address extension registers (HAE_MEM and HAE_IO), and the configuration type register.

5.1.1 Revision Control Register (PYXIS_REV)

The revision control register specifies the revision of the 21174. The revision control register access is RO to address 87.4000.0080. The register is shown in Figure 5–1.

Figure 5–1 Revision Control Register



The PYXIS_ID field can be used by software to dynamically determine if the device is a 21174 or other type of device that has similar functionality. Software can then change the behavior of the system based on this information.

The PYXIS_REV field can be used to determine the level of functionality within the device. Later revisions of the device are guaranteed to be backward compatible. When determining compatibility, software should always do unsigned comparisons and always compare for a value greater than or equal to a specific revision.

Table 5–1 describes the 21174 revision control register fields.

Table 5–1 Revision Control Register Fields

Name	Extent	Access	Init	Description
PYXIS_REV	<7:0>	RO	Device revision	This field specifies the revision of the 21174.
PYXIS_ID	<15:8>	RO	1	Identifies the device as 21174.
Reserved	<31:16>	RO	0	—

5.1.2 PCI Latency Register (PCI_LAT)

The PCI latency register access is RW to address 87.4000.00C0. Figure 5–2 shows the PCI latency register.

Figure 5–2 PCI Latency Register

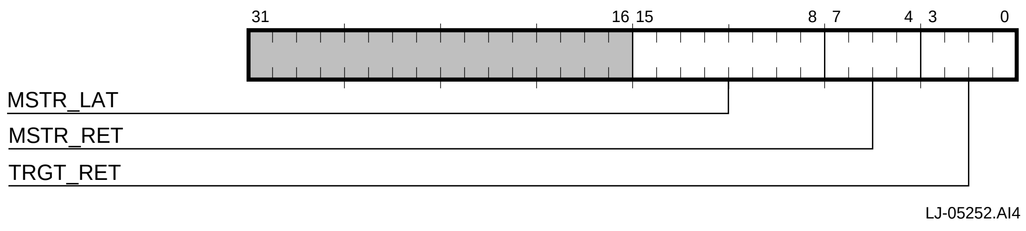


Table 5–2 describes the PCI latency register fields.

Table 5–2 PCI Latency Register Fields

Name	Extent	Access	Init	Description
TRGT_RET	<3:0>	RW	0	PCI target retry. This field specifies the number of cycles that the 21174 will wait after it has found a resource busy until it stops. Tune this value for best performance. Value — Cycles 0000 — 0 0001 — 1 .... — — 1110 — 14 1111 — Indefinite
MSTR_RET	<7:4>	RW	0	PCI master retry count. This field specifies the PCI master retry count in multiples of four PCI clock cycles. This is the number of cycles that the 21174 will wait after it has stopped until it retries the operation. The recommended value is 0. Value — Cycles 0000 — 2 0001 — 6 .... — — 1110 — 58 1111 — 2-66 (random)
MSTR_LAT	<15:8>	RW	0	PCI master latency timeout value expressed in PCI clock cycles.
Reserved	<31:16>	RO	0	—

5.1.3 Control Register (PYXIS_CTRL)

The control register access is RW to address 87.4000.0100. Figure 5–3 shows the control register.

Figure 5–3 Control Register

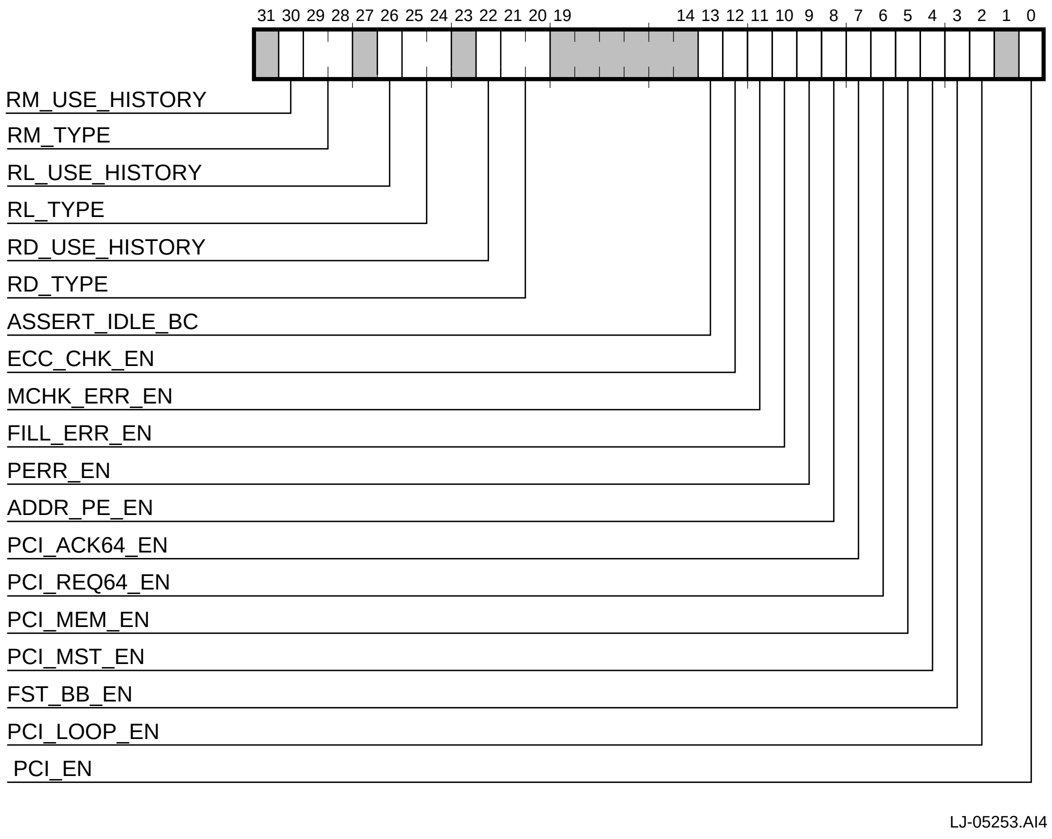


Table 5–3 describes the control register fields.

Table 5–3 Control Register Fields *(Sheet 1 of 3)*

Name	Extent	Access	Init	Description
PCI_EN	<0>	RW	0	0 – The 21174 asserts reset to the PCI. 1 – The 21174 does not assert reset to the PCI.
Reserved	<1>	RO	0	—

Table 5–3 Control Register Fields *(Sheet 2 of 3)*

Name	Extent	Access	Init	Description
PCI_LOOP_EN	<2>	RW	0	0 – 21174 will not respond as a target when it is the master. 1 – 21174 will respond as a target when it is the master.
FST_BB_EN	<3>	RW	0	0 – 21174 will not initiate fast back-to-back PCI transactions. 1– 21174 will initiate fast back-to-back PCI transactions.
PCI_MST_EN	<4>	RW	0	0 – 21174 will not initiate PCI transactions. 1 – 21174 will initiate PCI transactions.
PCI_MEM_EN	<5>	RW	0	0 – 21174 will not respond to PCI transactions. 1 – 21174 will respond to PCI transactions.
PCI_REQ64_EN	<6>	RW	0	0 – 21174 will not request 64-bit PCI data transactions. 1 – 21174 will request 64-bit PCI data transactions.
PCI_ACK64_EN	<7>	RW	0	0 – 21174 will not accept 64-bit PCI data transactions. 1 – 21174 will accept 64-bit PCI data transactions.
ADDR_PE_EN	<8>	RW	0	0 – 21174 will not check PCI address parity errors. 1 – 21174 will check PCI address parity errors.
PERR_EN	<9>	RW	0	0 – 21174 will not check PCI data parity errors. 1 – 21174 will check PCI data parity errors.
FILL_ERR_EN	<10>	RW	0	0 – 21174 will not assert **fill_error**. l – 21174 will assert **fill_error**, if an error occurs during a 21164 read miss.
MCHK_ERR_EN	<11>	RW	0	0 – 21174 will not assert the **mchk_irq** pin. 1 – 21174 will assert the **mchk_irq** pin to report system machine check conditions.
ECC_CHK_EN	<12>	RW	0	0 – 21174 will not check the IOD bus data. 1 – 21174 will check the IOD bus data.
ASSERT_IDLE_BC	<13>	RW	0	0 – 21174 will not assert the **idle_bc** pin while waiting for PCI read data. 1 – Not allowed.

Table 5–3 Control Register Fields *(Sheet 3 of 3)*

Name	Extent	Access	Init	Description
Reserved	<19:14>	RO	—	—
RD_TYPE[1]	<21:20>	RW	0	This field controls the prefetch algorithm used for PCI memory read command. See Table 5–4.
RD_USE_HISTORY	<22>	RW	0	When set, causes any translation buffer miss to use the prefetch algorithm selected by the RD_TYPE field of this register. A translation buffer hit uses the length of the preceding DMA as the prefetch length.
Reserved	<23>	RO	0	—
RL_TYPE[1]	<25:24>	RW	0	This field controls the prefetch algorithm used for PCI memory read line command. See Table 5–4.
RL_USE_HISTORY	<26>	RW	0	When set, causes any translation buffer miss to use the prefetch algorithm in RL_TYPE. A translation buffer hit uses the length of the preceding DMA as the prefetch length.
Reserved	<27>	RO	0	—
RM_TYPE[1]	<29:28>	RW	0	This field controls the prefetch algorithm used for PCI memory read multiple command. See Table 5-4.
RM_USE_HISTORY	<30>	RW	0	When set, causes any translation buffer miss to use the prefetch algorithm in RM_TYPE. A translation buffer hit uses the length of the preceding DMA as the prefetch length.
Reserved	<31>	RO	0	—

[1] This bit should be set to 0 for 21174 pass 1 devices.

Table 5–4 defines the default PCI READ prefetch algorithm.[1]

Table 5–4 Default PCI READ Prefetch Algorithm

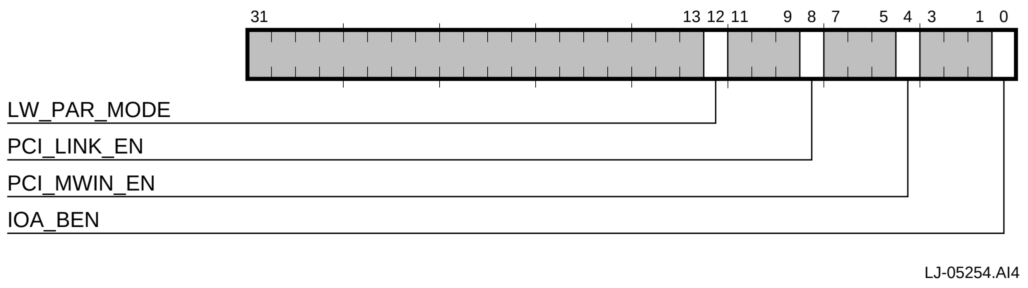
R*x*_TYPE	Description
00	No prefetch.
01	Fetch 2 cache lines. The operation will not cross an 8-KB boundary.
10	Fetch 4 cache lines. The operation will not cross an 8-KB boundary.
11	Fetch 8 cache lines. The operation will not cross an 8-KB boundary.

5.1.4 Control Register 1 (PYXIS_CTRL1)

Control register 1 contains miscellaneous bits.

The control register 1 access is RW to address 87.4000.0140. Figure 5–4 shows the control register 1 format.

Figure 5–4 Control Register 1



[1] This algorithm is used when the Rx_USE_HISTORY bit is not set or when the access misses in the transaction buffer.

Table 5–5 describes the control register 1 fields.

Table 5–5 Control Register 1 Fields

Name	Extent	Access	Init	Description
IOA_BEN	<0>	RW	0	Byte support enable. 1 – The address range 88.0000.0000 through EB.FFFF.FFFF is enabled for byte, word, longword, and quadword PCI addressing. 0 – Byte and word operations are disabled, and accessing the above mentioned addresses results in an error or undefined results. This is a new architecture feature.
Reserved	<3:1>	RO	0	—
PCI_MWIN_EN	<4>	RW	0	Monster window enable. 1 – Gives full access to main memory. The monster window can only be accessed in DAC mode when **ad<40>** equals 1. **addr_h<33:0>** equals **ad<33:0>**.
Reserved	<7:5>	RO	0	—
PCI_LINK_EN	<8>	RW	0	I/O write chaining enable.
Reserved	<11:9>	RW	0	—
LW_PAR_MODE	<12>	RW	0	1 – 21164 longword parity mode is selected.
Reserved	<31:13>	RW	0	—

5.1.5 Flash Control Register (FLASH_CTRL)

The flash control register access is RW to address 87.4000.0200. Figure 5–5 shows the register.

The flash control register controls access and basic timing of the flash ROM. The register controls the write pulse width, the read/write access time, and the ability of the flash ROM to map at address 0 for startup conditions.

Figure 5–5 Flash Control Register

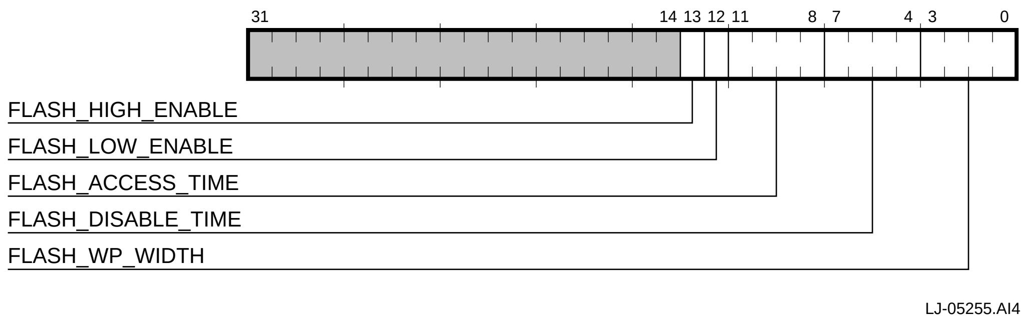


Table 5–6 describes the flash control register fields.

Table 5–6 Flash Control Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
FLASH_WP_WIDTH	<3:0>	RW	0F	Flash ROM write pulse width is defined (see Section 5.1.5.1).
FLASH_DISABLE_TIME	<7:4>	RW	07	Controls the number of cycles after **flash_ce_l** is deasserted before the 21174 deasserts **addr_bus_req** allowing the processor to use the bus.
FLASH_ACCESS_TIME	<11:8>	RW	0F	Flash access time as defined by the formula in Section 5.1.5.1.
FLASH_LOW_ENABLE	<12>	RW	1	1 – The flash ROM is mapped at address 0. This enables the device to be used in place of a serial ROM which would normally contain the system initialization and startup code. Initialize this bit to 1 on power-up so that code can be executed from the flash ROM. This bit should be disabled as soon after power-up as possible.

Table 5–6 Flash Control Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
FLASH_HIGH_ENABLE	<13>	RW	1	1 – The address range is F.FC00.0000 through F.FFFF.FFFF. This address range is in cacheable memory space and may contain program code. If all of the address bits are not connected, then the flash ROM may be shadowed at each flash ROM increment. This is not the address space for programming the device.
Reserved	<31:14>	RO	0	—

5.1.5.1 Calculating Flash ROM Access Time

Flash ROM write pulse width is determined in part by the system cycle time. The default value for FLASH_WP_WIDTH is $0F_{16}$ with write transactions enabled. The calculation of flash ROM write pulse width is performed as follows:

- Flash write pulse width (nominal) = (1 + FLASH_WP_WIDTH) $\times$ cycle time.
  For example, if cycle time is 15 ns and the value in FLASH_WP_WIDTH is $0C_{16}$, then the flash write pulse width would be (1+12) $\times$ 15 ns = 195 ns.

Flash ROM disable time is also determined in part by using the system cycle time. The default FLASH_DISABLE_TIME value is 07_{16} with write transactions enabled.

- Flash disable time = (1 + FLASH_DISABLE_TIME) $\times$ cycle time.
  For example, if cycle time is 15 ns and the value in FLASH_DISABLE is $0C_{16}$, then the flash disable time would be (1+12) $\times$ 15 ns = 195 ns.

The flash access time is also determined in part by the system cycle time. The default value is $0F_{16}$.

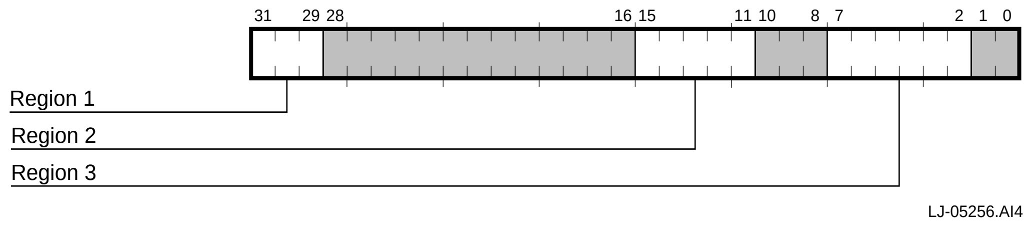
- Flash access time = (1 + FLASH_ACCESS_TIME) $\times$ cycle time – (T_{pd}+T_{setup}).
  For example, if cycle time is 15 ns and the value in FLASH_ACCESS_TIME is $0E_{16,}$ then the flash access time would be (1+14) $\times$ 15 ns – 5 ns = 255 ns.

Note: T_{pd} is the 21174 address bus clock-to-out delay and T_{setup} is the 21174 address bus setup time.

5.1.6 Hardware Address Extension Register (HAE_MEM)

The hardware address extension register access is RW to address 87.4000.0400. Figure 5–6 shows the register.

Figure 5–6 Hardware Address Extension Register (HAE_MEM)



The hardware address extension register (HAE_MEM) is used to extend a PCI sparse-space memory address up to the full 32-bit PCI address. In sparse address mode, the 21164 address provides the low-order PCI address bits, while the HAE_MEM provides the high-order bits. The high-order PCI address bits <31:26> are obtained from either the hardware extension register or the 21164 address depending on sparse-space regions, as shown in Table 5–8. See Chapter 6 for more details. Initializing HAE_MEM to 0000.2028_{16} will make all 3 regions contiguous starting at PCI address 0.

Table 5–7 shows the hardware address extension register fields.

Table 5–7 Hardware Address Extension Register (HAE_MEM) Fields

Name	Extent	Access	Init
Region 1	<31:29>	RW	0
Reserved	<28:16>	RO	0
Region 2	<15:11>	RW	0
Reserved	<10:8>	RO	0
Region 3	<7:2>	RW	0
Reserved	<1:0>	RO	0

Table 5–8 shows the PCI address mapping controlled by the hardware address extension register.

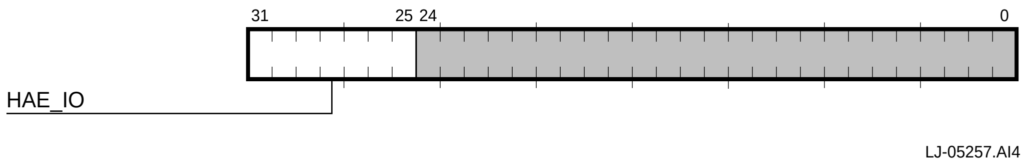
Table 5–8 PCI Address Mapping

21164 Address	Region	PCI Address					
		31	30	29	28	27	26
80.0000.0000 to 83.FFFF.FFFF	1	HAE_MEM <31>	HAE_MEM <30>	HAE_MEM <29>	CPU<33>	CPU<32>	CPU<31>
84.0000.0000 to 84.FFFF.FFFF	2	HAE_MEM <15>	HAE_MEM <14>	HAE_MEM <13>	HAE_MEM <12>	HAE_MEM <11>	CPU<31>
85.0000.0000 to 85.FFFF.FFFF	3	HAE_MEM <7>	HAE_MEM <6>	HAE_MEM <5>	HAE_MEM <4>	HAE_MEM <3>	HAE_MEM <2>

5.1.7 Hardware Address Extension Register (HAE_IO)

The hardware address extension register (HAE_IO) access is RW to address 87.4000.0440. Figure 5–7 shows the register.

Figure 5–7 Hardware Address Extension Register (HAE_IO)



The hardware address extension register (HAE_IO) is used to extend a PCI sparse-space I/O address up to the full 32-bit PCI address. In sparse address mode, the 21164 address provides the PCI addresses up to **ad<24>** and HAE_IO provides **ad<31:25>**.

When power is turned on, this register is set to zero. In this case, sparse I/O region A and region B both map to the lower 32MB of sparse I/O space. Setting HAE_IO to 200.0000_{16} will make region A and region B consecutive in the lower 64MB of PCI I/O space.

Table 5–9 describes the hardware address extension register fields.

Table 5–9 Hardware Address Extension Register (HAE_IO) Fields

Name	Extent	Access	Init	Description
Reserved	<24:0>	RO	0	—
HAE_IO	<31:25>	RW	0	—

5.1.8 Configuration Type Register (CFG)

The configuration type register access is RW to address 87.4000.0480. Figure 5–8 shows the register.

Figure 5–8 Configuration Type Register

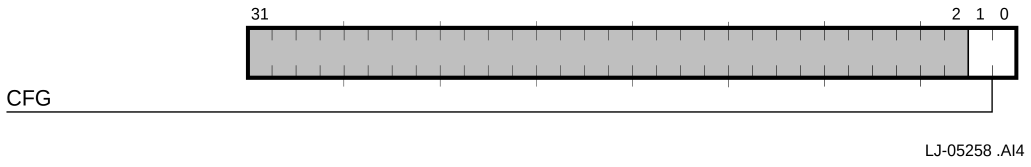


Table 5–10 describes the configuration type register fields.

Table 5–10 Configuration Type Register Fields

Name	Extent	Access	Init	Description
CFG	<1:0>	RW	0	The CFG field is used as the low two address bits during an access to PCI configuration space. __<1:0>__ __Meaning__ 00 Type 0 configuration cycle 01 Type 1 configuration cycle 10 Reserved 11 Reserved
Reserved	<31:2>	RO	0	—

5.2 Diagnostic Register Descriptions

This section describes the functionality of the diagnostic control register and the diagnostic check register.

5.2.1 Diagnostic Control Register (PYXIS_DIAG)

The diagnostic control register allows errors to be forced and tested. The register access is RW to address 87.4000.2000. Figure 5–9 shows the register.

Figure 5–9 Diagnostic Control Register

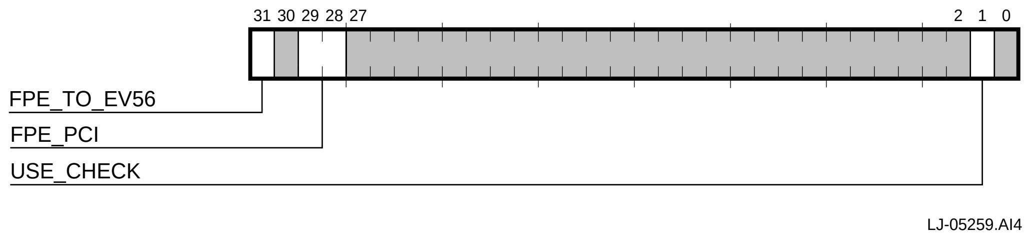


Table 5–11 describes the diagnostic control register fields.

Table 5–11 Diagnostic Control Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
Reserved	<0>	RW	0	—
USE_CHECK	<1>	RW	0	When set, DMA write cycles and PCI I/O read cycles use the value in the DIAG_CHECK register for ECC sent on the IOD bus.
Reserved	<27:2>	RO	0	00 – Normal parity is output to the PCI. 01 – Bad parity is forced onto the low 32 bits of the PCI during data cycles. 10 – Bad parity is forced onto the high 32 bits of the PCI during data cycles. 11 – Bad parity is forced onto the high and low 32 bits to the PCI during address and data cycles.
FPE_PCI	<29:28>	RW	0	00 – Normal parity is output to the PCI.
Reserved	<30>	RO	0	—

Table 5–11 Diagnostic Control Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
FPE_TO_EV56	<31>	RW	0	When FPE_CPU_EV56 is set, a parity error is forced on the 21164 address/cmd bus when the 21174 is the bus master.

5.2.2 Diagnostic Check Register (DIAG_CHECK)

The diagnostic check register is used to verify the 21174 error paths. This register is used for diagnostic DMA transactions that write a known ECC pattern into memory. It also provides the ECC pattern on any PCI I/O read operation. This register provides the ECC that gets written to memory or appended to the I/O read operation if the USE_CHECK bit is set in the PYXIS_DIAG register.

The register access is RW to address 87.4000.3000. Figure 5–10 shows the register.

Figure 5–10 Diagnostic Check Register

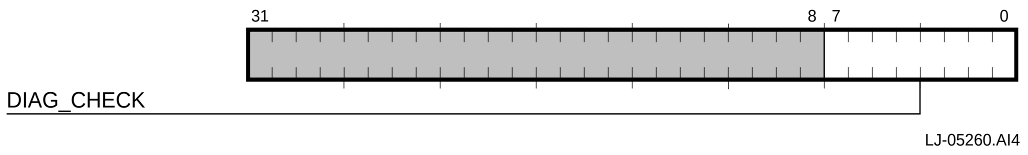


Table 5–12 describes the diagnostic check register fields.

Table 5–12 Diagnostic Check Register Fields

Name	Extent	Access	Init	Description
DIAG_CHECK	<7:0>	RW	X	For diagnostic DMA write transactions and PCI I/O read transactions, the DIAG_CHECK register provides the quadword ECC.
Reserved	<31:8>	RO	0	—

5.3 Performance Monitor Register Descriptions

This section describes the functionality of the performance monitor register and the performance monitor control register.

Performance Monitor Register Descriptions

5.3.1 Performance Monitor Register (PERF_MONITOR)

The 21174 performance monitor register contains two 16-bit counters that can be programmed to count a variety of events. The counters are set up using the PERF_CONTROL register. Each counter can be programmed to count events such as 21164 read transaction misses received by the 21174 or DMA write transactions. The PERF_MONITOR register can also be configured as a single 32-bit counter (by telling the high_count field to count the low_count field overflow).

The performance monitor register access is RO to address 87.4000.4000. Figure 5–11 shows the performance monitor register.

Figure 5–11 Performance Monitor Register

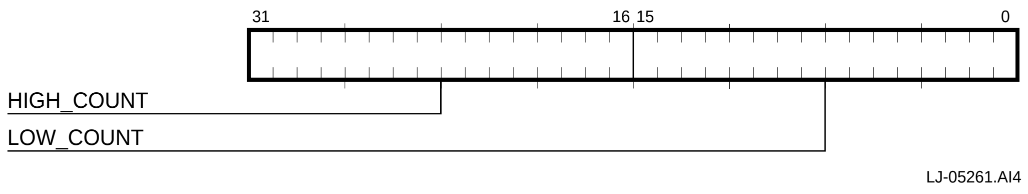


Table 5–13 describes the performance monitor register fields.

Table 5–13 Performance Monitor Register Fields

Name	Extent	Access	Init	Description
LOW_COUNT	<15:0>	RO	0	This is the value of the low counter.
HIGH_COUNT	<31:16>	RO	0	This is the value of the high counter.

5.3.2 Performance Monitor Control Register (PERF_CONTROL)

The performance monitor control register access is RW to address 87.4000.4040. Figure 5–12 shows the register.

Figure 5–12 Performance Monitor Control Register

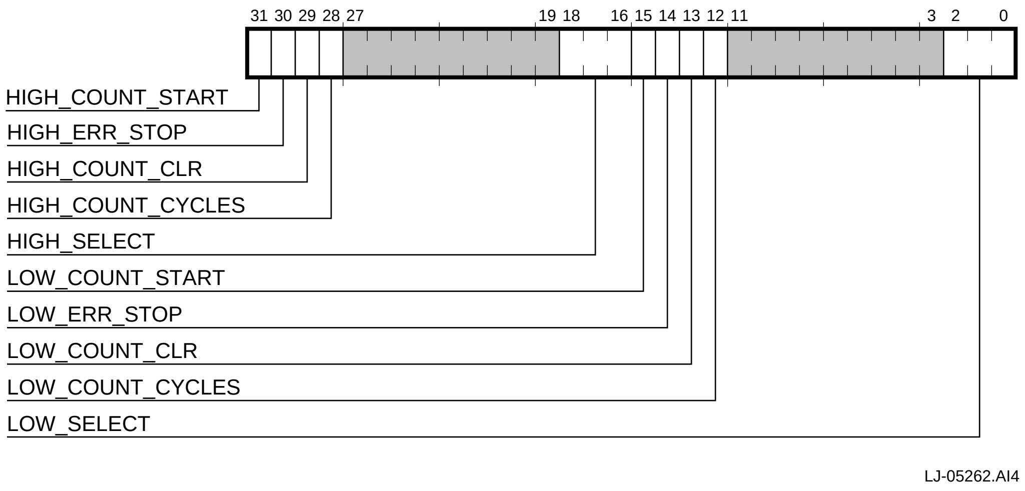


Table 5–14 describes the performance monitor control register fields.

Table 5–14 Performance Monitor Control Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
LOW_SELECT	<2:0>	RW	0	Enables certain debug features (see Table 5–15).
Reserved	<11:3>	RO	0	—
LOW_COUNT_CYCLES	<12>	RW	0	0 – The number of low to high transitions are counted. 1 – The number of cycles that the low_select event is asserted is counted.
LOW_COUNT_CLR	<13>	WO	0	Write a 1 to clear the low counter.
LOW_ERR_STOP	<14>	RW	0	If the 21174 detects an error and this bit is set, then stop counting.
LOW_COUNT_START	<15>	RW	0	0 – Don’t count; keep current values. 1 – Start counting.
HIGH_SELECT	<18:16>	RW	0	Enables certain debug features. See Table 5–15.
Reserved	<27:19>	RO	0	—

Table 5–14 Performance Monitor Control Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
HIGH_COUNT_CYCLES	<28>	RW	0	0 – The number of low to high transitions are counted. 1 – The number of cycles that the high_select event is asserted is counted.
HIGH_COUNT_CLR	<29>	WO	0	Write a 1 to clear the high counter.
HIGH_ERR_STOP	<30>	RW	0	Stop counting if the 21174 detects an error and this bit is set.
HIGH_COUNT_START	<31>	RW	0	0 – Don’t count; keep current values. 1 – Start counting.

Table 5–15 shows the performance monitor register low/high select field codes.

Table 5–15 PERF_MONITOR Register Low/High Select Field Codes

LOW_SELECT <2:0> and HIGH_SELECT <18:16>	Description
000	MCTL_DEBUG_OUT[0]
001	MCTL_DEBUG_OUT[1]
010	MCTL_DEBUG_OUT[2]
011	MCTL_DEBUG_OUT[3]
100	PA/PCI_DEBUG_OUT[0]
101	PA/PCI_DEBUG_OUT[1]
110	PA/PCI_DEBUG_OUT[2]
111	PA/PCI_DEBUG_OUT[3] for LOW_SELECT and make the counter a 32-bit counter on HIGH_SELECT.

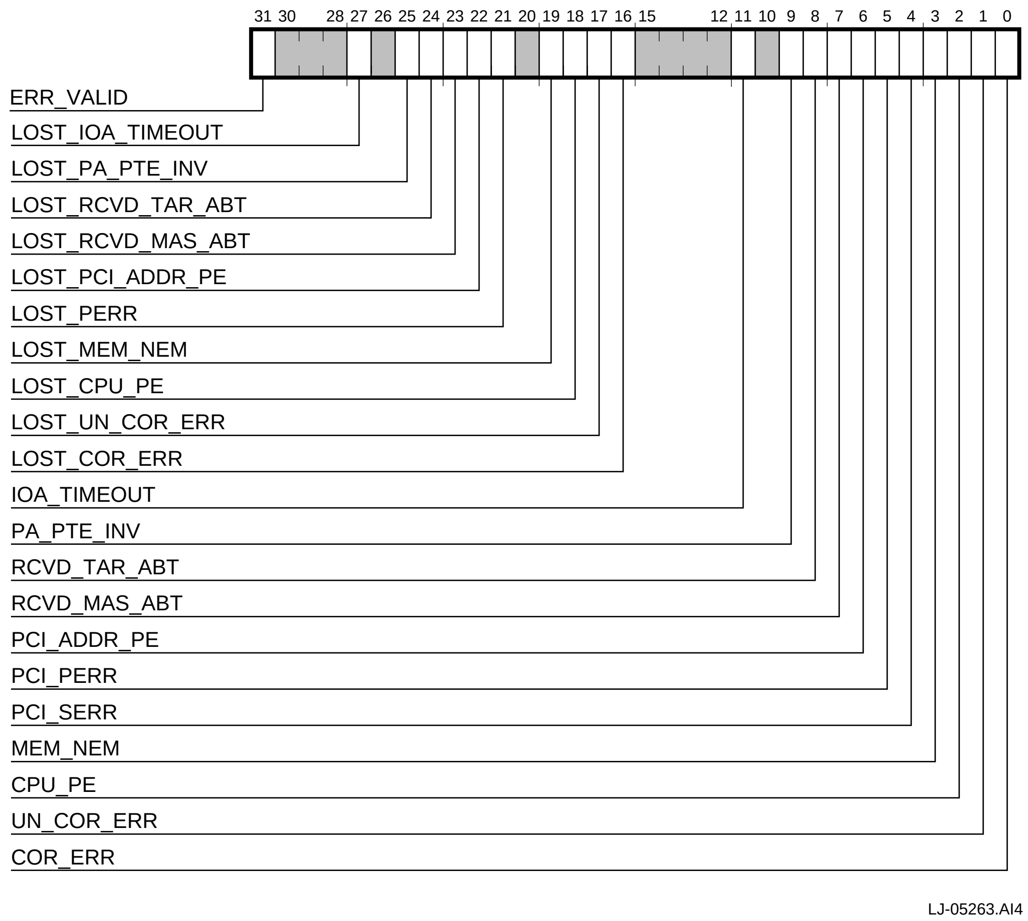
5.4 Error Register Descriptions

This section details the functionality of the error register, status register, error mask register, syndrome register, error data register, memory error address register, memory error status register, PCI error register 0, PCI error register 1, and the PCI error register 2.

5.4.1 Error Register (PYXIS_ERR)

The error register access is RW1C to address 87.4000.8200. Figure 5–13 shows the register.

Figure 5–13 Error Register



Error Register Descriptions

Table 5–16 describes the 21174 error register fields.

Table 5–16 Error Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
COR_ERR	<0>	RW1C	0	Correctable (single bit) ECC error detected. This error cannot occur for a 21164-to-memory read/write transaction. (21164-to-memory read transaction ECC errors are detected by the 21164. 21164-to-memory write transactions are not checked.) This error is applicable to a DMA, scatter-gather TLB miss, or an I/O write transaction from the 21164.
UN_COR_ERR	<l>	RW1C	0	Uncorrectable ECC error detected. This error cannot occur for a 21164-to-memory read/write transaction. (21164-to-memory read ECC errors are detected by the 21164. 21164-to-memory write transactions are not checked.) This error is applicable to a DMA, a scatter-gather TLB miss, or an I/O write from the 21164.
CPU_PE	<2>	RW1C	0	21164 bus parity error detected.
MEM_NEM	<3>	RW1C	0	Access to nonexistent memory detected.
PCI_SERR	<4>	RW1C	0	PCI bus SERR detected.
PCI_PERR	<5>	RW1C	0	PCI bus data parity error detected.
PCI_ADDR_PE	<6>	RW1C	0	PCI bus address parity error detected.
RCVD_MAS_ABT	<7>	RW1C	0	PCI master state machine generated master abort.
RCVD_TAR_ABT	<8>	RW1C	0	PCI master state machine received target abort.
PA_PTE_INV	<9>	RW1C	0	Invalid page table entry on scatter-gather transaction.
Reserved	<10>	RO	0	—
IOA_TIMEOUT	<11>	RW1C	0	I/O timeout occurred. I/O read/write transaction failed to get executed in 1 second.
Reserved	<15:12>	RO	0	—

Table 5–16 Error Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
LOST_COR_ERR	<16>	RO	0	A correctable ECC error was detected. The PYXIS_ERR register was locked.
LOST_UN_COR_ERR	<17>	RO	0	While PYXIS_ERR register was locked, an uncorrectable ECC error was detected.
LOST_CPU_PE	<18>	RO	0	While PYXIS_ERR register was locked, a 21164 parity error was detected.
LOST_MEM_NEM	<19>	RO	0	While PYXIS_ERR register was locked, an access to nonexistent memory was detected.
Reserved	<20>	RO	0	—
LOST_PERR	<21>	RO	0	While locked, a PCI data parity error was detected.
LOST_PC1_ADDR_PE	<22>	RO	0	While the PYXIS_ERR register was locked, a PCI address parity error was detected.
LOST_RCVD_MAS_ABT	<23>	RO	0	While the PYXIS_ERR register was locked, the PCI master state machine generated a master abort.
LOST_RCVD_TAR_ABT	<24>	RO	0	While the PYXIS_ERR register was locked, the PCI master state machine received a target abort.
LOST_PA_PTE_INV	<25>	RO	0	While the PYXIS_ERR register was locked, an invalid page table entry on scatter-gather access occurred.
Reserved	<26>	RO	0	—
LOST_IOA_TIMEOUT	<27>	RO	0	While the PYXIS_ERR register was locked, an I/O timeout occurred. An I/O read/write failed to get executed in 1 second.
Reserved	<30:28>	RO	0	—
ERR_VALID	<31>	RO	0	An error has been detected and the 21174 error registers are all locked.

5.4.2 Status Register (PYXIS_STAT)

The PYXIS_STAT register contains information about the state of the 21174 at the time an error occurred. This register, along with the error registers, can be used in isolating the error condition and determining a proper recovery action.

The status register access is RO to address 87.4000.8240. Figure 5–14 shows the register.

Figure 5–14 Status Register

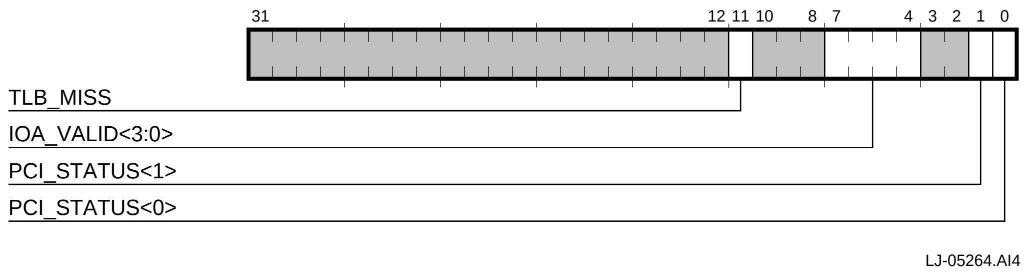


Table 5–17 describes the status register fields.

Table 5–17 Status Register Fields

Name	Extent	Access	Init	Description
PCI_STATUS<0>	<0>	RO	0	1 — The PCI target state machine is active.
PCI_STATUS<1>	<1>	RO	0	1 — The PCI master state machine is active.
Reserved	<3:2>	RO	0	—
IOA_VALID<3:0>	<7:4>	RO	0	Valid bits for the I/O command/address queue.
Reserved	<10:8>	RO	0	—
TLB_MISS	<11>	RO	0	1 — A TLB refill was in progress when this miss error occurred.
Reserved	<31:12>	RO	0	—

5.4.3 Error Mask Register (ERR_MASK)

Use the error mask register to disable the logging and reporting of errors. When power is turned on, error logging is in the default state — disabled with ERR_MASK = 0.

- 0 disables the logging/reporting of an error.
- 1 enables logging/reporting of an error.

The error mask register access is RW to address 87.4000.8280. Figure 5–15 shows the register.

Figure 5–15 Error Mask Register

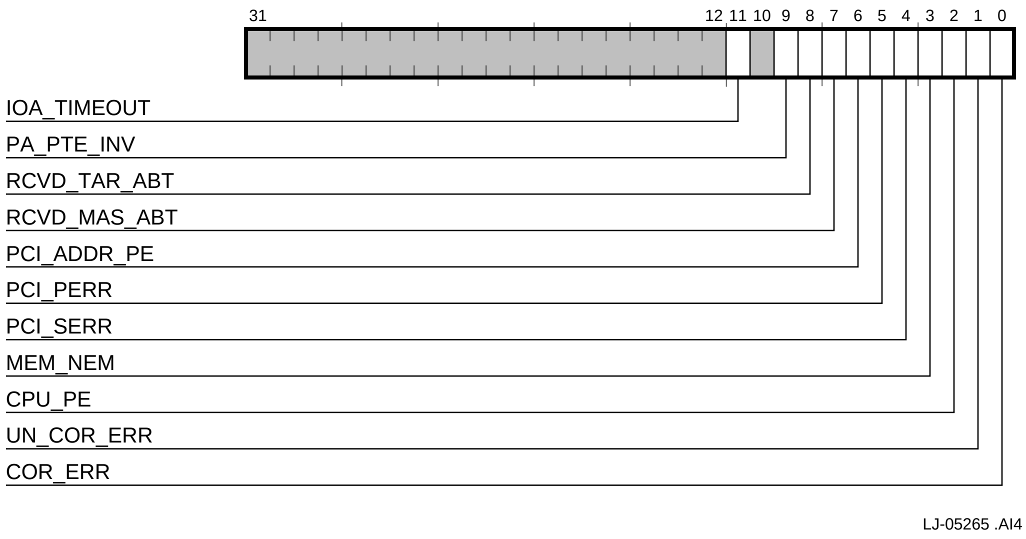


Table 5–18 describes the error mask register fields.

Table 5–18 Error Mask Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
COR_ERR	<0>	RW	0	Disable/enable error logging/reporting for correctable ECC errors.
UN_COR_ERR	<1>	RW	0	Disable/enable error logging/reporting for uncorrectable ECC errors.
CPU_PE	<2>	RW	0	Disable/enable error logging/reporting for 21164 parity errors.

Table 5–18 Error Mask Register Fields *(Sheet 2 of 2)*

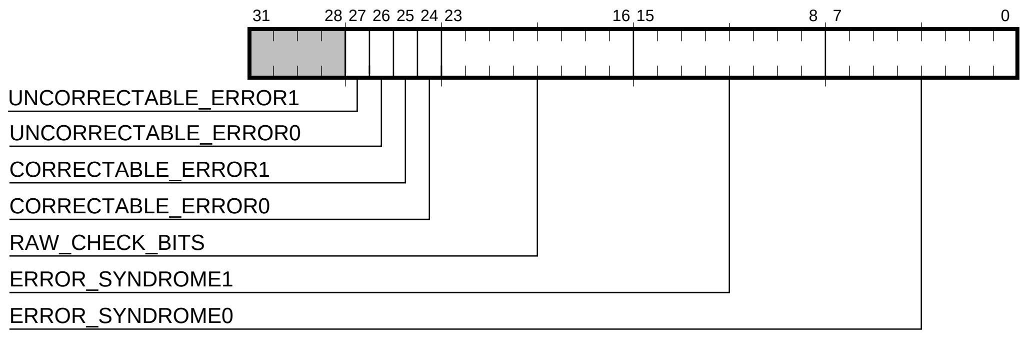
Name	Extent	Access	Init	Description
MEM_NEM	<3>	RW	0	Disable/enable error logging/reporting for nonexistent memory access errors.
PCI_SERR	<4>	RW	0	Disable/enable error logging/reporting for PCI SERR errors.
PCI_PERR	<5>	RW	0	Disable/enable error logging/reporting for PCI data parity errors.
PCI_ADDR_PE	<6>	RW	0	Disable/enable error logging/reporting for PCI address parity errors.
RCVD_MAS_ABT	<7>	RW	0	Disable/enable error logging/reporting for PCI master abort errors.
RCVD_TAR_ABT	<8>	RW	0	Disable/enable error logging/reporting for PCI target abort errors.
PA_PTE_INV	<9>	RW	0	Disable/enable error logging/reporting for invalid PTE errors.
Reserved	<10>	RO	0	—
IOA_TIMEOUT	<11>	RW	0	Disable/enable error timeout errors.
Reserved	<31:12>	RO	0	—

5.4.4 Syndrome Register (PYXIS_SYN)

The syndrome register has two 8-bit fields that contain the error syndrome bits. The error syndrome data is captured after an error condition has occurred and is held until a 0 is written to COR_ERR and UN_COR_ERR in the 21174 error register. The PALcode must save the contents of this register prior to clearing the error conditions. The state of this register is UNDEFINED except when an error has been detected.

The syndrome register access is RO to address 87.4000.8300. Figure 5–16 shows the register.

Figure 5–16 Syndrome Register



LJ-05266 .AI4

Table 5–19 describes the syndrome register fields.

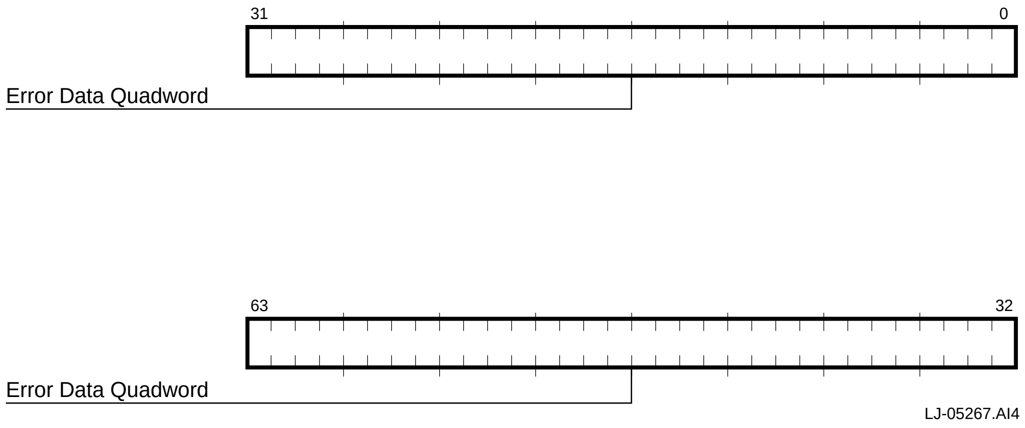
Table 5–19 Syndrome Register Fields

Name	Extent	Access	Init	Description
ERROR_SYNDROME0	<7:0>	RO	X	ECC syndrome bits for data bits <63:0>.
ERROR_SYNDROME1	<15:8>	RO	X	ECC syndrome bits for data bits <127:64>.
RAW_CHECK_BITS	<23:16>	RO	X	Raw check bits from ECC error. The actual ECC bits stored in the ECC field of the memory.
CORRECTABLE_ ERROR0	<24>	RO	X	A correctable error was detected in quadword 0.
CORRECTABLE_ ERROR1	<25>	RO	X	A correctable error was detected in quadword 1.
UNCORRECTABLE_ ERROR0	<26>	RO	X	An uncorrectable error was detected in quadword 0.
UNCORRECTABLE_ ERROR1	<27>	RO	X	An uncorrectable error was detected in quadword 1.
Reserved	<31:28>	RO	0	—

5.4.5 Error Data Register (PYXIS_ERR_DATA)

The error data register access is RO to address 87.4000.8308. Figure 5–17 shows the register.

Figure 5–17 Error Data Register



The error data register contains the data present when the ECC error was detected.

Note: The error data register is implemented to shadow at all internal CSR locations, by always driving the high 64 bits of the data path when CSR are selected. If the 21164 reads a CSR-region quadword with odd offset, it will always get the 21174 error data register.

5.4.6 Memory Error Address Register (MEAR)

The low-order address bits of the memory port address bus are locked into this register upon a 21174 detected error. Clearing all the error bits in the PYXIS_ERR register unlocks this register. When the register is not locked, the contents of this register are not defined.

The memory error address register access is RO to address 87.4000.8400. Figure 5–18 shows the register.

Figure 5–18 Memory Error Address Register

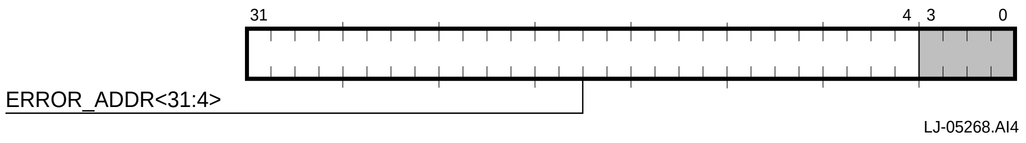


Table 5–20 describes the memory error address register fields.

Table 5–20 Memory Error Address Register Fields

Name	Extent	Access	Init	Description
Reserved	<3:0>	RO	0	—
ERROR_ADDR<31:4>	<31:4>	RO	X	Contains the current address in the memory port when the 21174 detects an error.

5.4.7 Memory Error Status Register (MESR)

The command, the memory sequencer state, the data cycle at the time of an error, and the remaining address field are locked into the MESR register upon a 21174 error. The error bits access is write one to clear. Clearing all error bits in the PYXIS_ERR register unlocks this register. When the register is not locked, the contents of this register are UNDEFINED.

The contents of the MESR bits 2 through 7 are UNPREDICTABLE on the nonexistent memory trap if the nonexistent memory is cacheable memory space. The contents of bits 2 through 7 are valid only if the address is to a noncacheable address. Ignore bits 2 through 7 if bit 12 or 13 in the MESR is not set.

The 21174 memory error status register access is RW to address 87.4000.8440. Figure 5–19 shows the register.

Error Register Descriptions

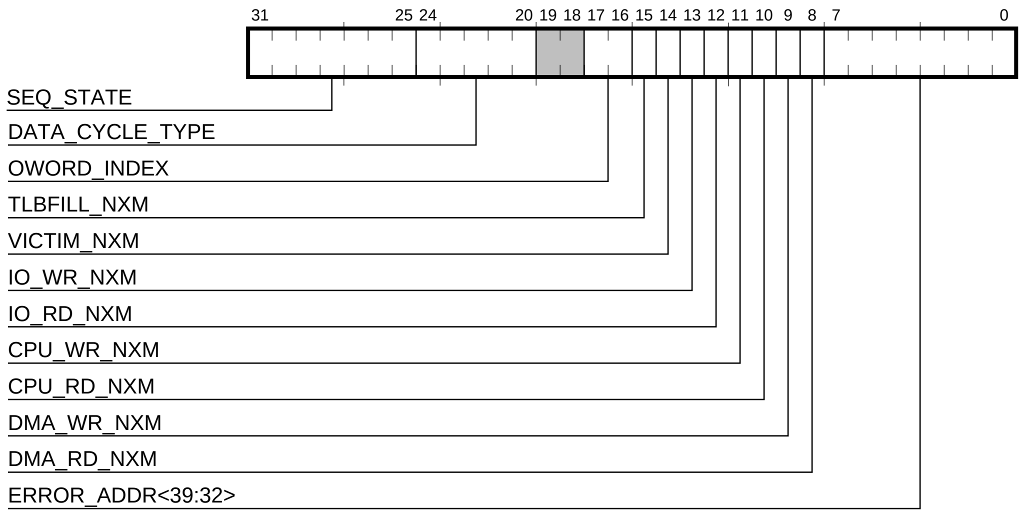


Figure 5–19 Memory Error Status Register

LJ-05269.AI4

Table 5–21 describes the memory error status register fields.

Table 5–21 Memory Error Status Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
ERROR_ADDR <39:32>	<7:0>	RO	X	Contains address bits <39:32> of the address in the memory port when the 21174 detects an error. Bits <39:34> are UNPREDICTABLE on memory errors —– only bits <33:32> are valid for memory errors.
DMA_RD_NXM	<8>	RO	X	Nonexistent memory trap during a DMA read transaction.
DMA_WR_NXM	<9>	RO	X	Nonexistent memory trap during a DMA write transaction.
CPU_RD_NXM	<10>	RO	X	Nonexistent memory trap during a 21164 read transaction.
CPU_WR_NXM	<11>	RO	X	Nonexistent memory trap during a 21164 write transaction.

Table 5–21 Memory Error Status Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
IO_RD_NXM	<12>	RO	X	Nonexistent memory trap during an I/O read transaction.
IO_WR_NXM	<13>	RO	X	Nonexistent memory trap during an I/O write transaction.
VICTIM_NXM	<14>	RO	X	Nonexistent memory trap during a Bcache victim operation.
TLBFILL_NXM	<15>	RO	X	Nonexistent memory trap during a scatter-gather translation buffer fill transaction.
OWORD_INDEX	<17:16>	RO	X	This field indicates the error data cycle within a memory access in which the data error was discovered. There are normally four data cycles. OWORD_INDEX = 0 is the first data cycle corresponding to the error address captured in the ERROR_ADDR register. The actual low-order bits of the error location are (ERROR_ADDR<5:4> + OWORD_INDEX) MOD 4.
Reserved	<19:18>	RO	0	—
DATA_CYCLE_ TYPE	<24:20>	RO	X	Contains the type of data cycle in progress when an ECC error occurred. See Table 5–22 for the definitions of the values in this field.
SEQ_STATE	<31:25>	RW	X	The memory sequencer-state when the nonexistent memory error occurred. Table 5–23 has the definitions for this field.

Table 5–22 contains the DATA_CYCLE_TYPE codes.

Table 5–22 DATA_CYCLE_TYPE Codes *(Sheet 1 of 2)*

DATA_CYCLE_TYPE	Description
00	IDLE
01	CPU_READ
02	CPU_READ_VICTIM
03	CPU_WRITE
04	IO_READ

Table 5–22 DATA_CYCLE_TYPE Codes *(Sheet 2 of 2)*

DATA_CYCLE_TYPE	Description
05	FLASH_BYTE_READ
06	PCI_READ
07	IO_WRITE
08	FLASH_BYTE_WRITE
09	DMA_READ
0A	DMA_READ_SCACHE
0B	DMA_READ_BCACHE
0C	DMA_READ_VICTIM
0D	DMA_WRITE
0E	DMA_MEM_MERGE
0F	DMA_SCACHE_MERGE
10	DMA_BCACHE_MERG
11	DMA_VICTIM_MERGE
12	FLASH_READ
13	VICTIM_WRITE
14	DUMMY_READ
15	VICTIM_EJECT

Table 5–23 contains the SEQ_STATE field codes.

Table 5–23 SEQ_STATE Codes *(Sheet 1 of 3)*

SEQ_STATE	Value	Description
IDLE	00	Command dispatch
WAIT	01	Wait until data transfer is idle
WAIT1	02	Wait one cycle
DMA_RD_START	03	Select DMA read address
DMA_RD_PROBE	04	Assert **ras_l**
DMA_RD_SCACHE_DATA	05	Read dirty data from Scache

Table 5–23 SEQ_STATE Codes *(Sheet 2 of 3)*

SEQ_STATE	Value	Description
DMA_RD_BCACHE_DATA	06	Read dirty data from Bcache
DMA_RD_CACHE_DATA	07	Read dirty data from Bcache or Scache
DMA_RD_RAS	08	Continue to assert **ras_l** after cache miss
DMA_RD_COL	09	Wait for column access
DMA_RD_VICTIM	0A	Wait for memory data to pass by
DMA_RD_NXM	0B	Assert error state for nonexistent memory
DMA_WR_START	0C	Select DMA write address
DMA_WR_WHOLE_RAS	0D	Assert **ras_l** for whole cache line write transaction
DMA_WR_WHOLE_DATA	0E	Watch data pass by
DMA_WR_PROBE	0F	Wait for probe result
DMA_WR_SCACHE_COPY	10	Read dirty data from Scache
DMA_WR_BCACHE_COPY	11	Read dirty data from Bcache or Scache
DMA_WR_CACHE_COPY	12	Read dirty data from Bcache or Scache
DMA_WR_RAS	13	Continue to assert **ras_l** after miss
DMA_WR_PQ_RD_RAS	14	Assert **ras_l** for partial octawords read
DMA_WR_PQ_RD_COL	15	Wait for column access
DMA_WR_PQ_RD_VICTIM	16	Wait for memory data to pass by
DMA_WR_NXM	17	Assert error state for nonexistent memory
DMA_WR_WHOLE_RASF	36	Assert **ras_l** for whole cache line write transaction. A Bcache flush is pending.
DMA_WR_WHOLE_FLUSH	37	Watch the data pass by
CPU_EJECT	18	Eject victim and assert **ras_l** for fill
CPU_RD_START	19	Assert **ras_l** for fill
CPU_RD_COL	1A	Wait for column access
CPU_RD_VICTIM	1B	Let DRAM data pass by, then read victim
CPU_RD_NXM	1C	Assert error state for nonexistent memory
CPU_WR_START	1D	Assert **ras_l** for Scache victim (no Bcache)

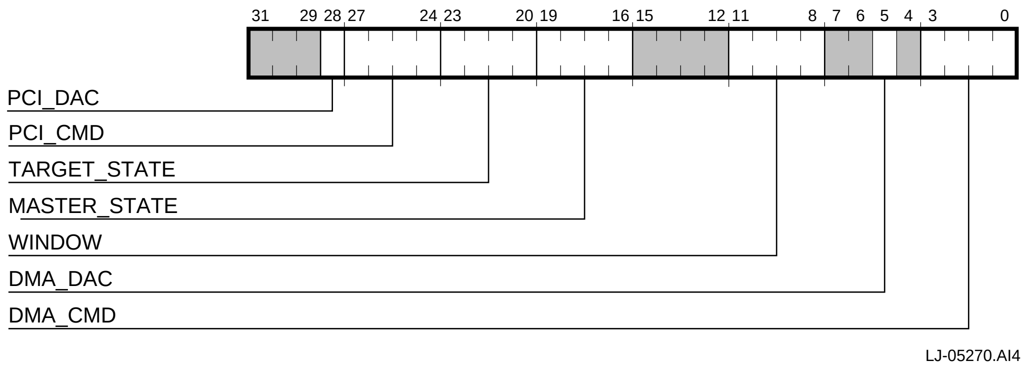
Table 5–23 SEQ_STATE Codes *(Sheet 3 of 3)*

SEQ_STATE	Value	Description
CPU_WR_NXM	1E	Assert error state for nonexistent memory
VICTIM_START	1F	Assert **ras_l** signal for Bcache victim in victim buffer
VICTIM_NXM	20	Assert error state for nonexistent memory
REFRESH_PRECHARGE	21	Deactivate all rows for refresh
REFRESH_COMMAND	22	Assert refresh for all banks
MODE_PRECHARGE	23	Deactivate all rows for mode cycle
MODE_COMMAND	24	Assert mode cycle for all banks and join refresh flow
CPU_IO_RD_ADDR	25	Send I/O read address to select a target
CPU_IO_RD_WAIT	26	Wait for return of read data (64-bit maximum)
CPU_IO_RD_START	27	Start read data transfer
CPU_FLASH_RD_WAIT	28	Wait for flash ROM byte-read transaction to complete
UNREACHABLE_STATE	29	State not reachable
CPU_PCI_RD_WAIT	2A	Wait for **idle_bc** signal
CPU_PCI_RD_START	2B	Delay for data cycle
CPU_IO_WR_ADDR	2C	Send I/O write address to select a target
CPU_IO_WR_NXM	2D	Error state for nonexistent I/O address
CPU_FLASH_WR_WAIT	2E	Wait for flash ROM byte-write transaction to complete
CPU_FLASH_START	2F	Start a fill from flash ROM
CPU_FLASH_COL	30	Start a fill from flash ROM
CPU_FLASH_DATA	31	Wait for flash ROM control to deliver all data
CPU_DUMMY_START	32	Start a fill from the dummy region
CPU_DUMMY_COL	33	Start the dummy data transfer
NO_BRAINER	34	Issue CACK and ignore
BAD_CPU_CMD	35	Assert machine check

5.4.8 PCI Error Register 0 (PCI_ERR0)

The PCI error register 0 access is RO to address 87.4000.8800. Figure 5–20 shows the register.

Figure 5–20 PCI Error Register 0



The PCI error register 0 is used by the 21174 to log information pertaining to the state of the PCI interface when an error condition is detected by 21174. The register is locked, as are all 21174 error registers, when the 21174 detects an error. The register is unlocked when the PYXIS_ERR register is cleared. When the register is not locked, the contents are UNPREDICTABLE.

The data in the WINDOW, DMA_DAC, and DMA_CMD fields is associated with the address stored in the PCI_ERR1 register. This group and PCI_ERR1 hold information related to the following errors that are associated with the memory while the 21174 is handling a DMA:

- Correctable ECC error (PYXIS_ERR<0>)
- Uncorrectable ECC error (PYXIS_ERR<l>)
- Access to nonexistent memory (PYXIS_ERR<3>)
- Invalid page table entry (PYXIS_ERR<9>)

Error Register Descriptions

The data in the PCI_DAC, PCI_CMD, TARGET_STATE, and MASTER_STATE fields is associated with the address stored in the PCI_ERR2 register. This group and the PCI_ERR2 register hold information related to the following error conditions that are associated with the PCI bus:

- PCI data parity error (PYXIS_ERR<5>)
- PCI address parity error (PYXIS_ERR<6>)
- PCI master abort (PYXIS_ERR<7>)
- PCI target abort (PYXIS_ERR<8>)
- IOA timeout (PYXIS_ERR<ll>)

The LOCK_STATE field is general information about the current state of 21174. It is not specifically associated with either the PCI_ERR1 or PCI_ERR2 fields.

Table 5–24 describes the PCI error register 0 fields.

Table 5–24 PCI Error Register 0 Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
DMA_CMD	<3:0>	RO	X	The PCI command of the current DMA.
Reserved	<4>	RO	0	—
DMA_DAC	<5>	RO	X	If set, then the current DMA is a dual-address cycle (DAC) command.
Reserved	<7:6>	RO	X	—
WINDOW	<11:8>	RO	X	Indicates which window (if any) was selected by the PCI address. 0000 No window active 0001 Window 0 hit 0010 Window 1 hit 0100 Window 2 hit 1000 Window 3 hit
Reserved	<15:12>	RO	X	—

Table 5–24 PCI Error Register 0 Fields *(Sheet 2 of 2)*

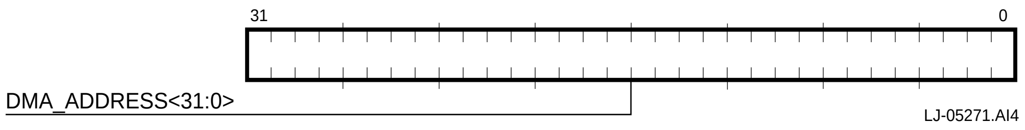
Name	Extent	Access	Init	Description
MASTER_STATE	<19:16>	RO	0	0 — Idle 1: Drive Bus 2 — Address Step Cycle 3 — Address Cycle 4 — Data Cycle 5 — Last Read Data Cycle 6 — Last Write Data Cycle 7 — Read Stop Cycle 8 — Write Stop Cycle 9 — Read Turnaround Cycle A — Write Turnaround Cycle B — Reserved C — Reserved D — Reserved E — Reserved F — Unknown State
TARGET_STATE	<23:20>	RO	0	0 — Idle 1 — Busy 2 — Read Data Cycle 3 — Write Data Cycle 4 — Read Stop Cycle 5 — Write Stop Cycle 6 — Read Turnaround Cycle 7 — Write Turnaround Cycle 8 — Read Delay Cycle 9 — Write Delay Cycle
PCI_CMD	<27:24>	RO	X	The current PCI command.
PCI_DAC	<28>	RO	X	If set, then the current PCI command is a dual-address cycle (DAC) command.
Reserved	<31:29>	RO	0	—

5.4.9 PCI Error Register 1 (PCI_ERR1)

The PCI error register 1 access is RO to address 87.4000.8840. Figure 5–21 shows the register.

Error Register Descriptions

Figure 5–21 PCI Error Register 1



The PCI error register 1 is used by the 21174 to log **ad<31:0>** for the current DMA associated with an error condition logged in the PCI_ERR0 register. This register is locked whenever the 21174 detects an error. This register always captures **ad<31:0>**, even for a DMA DAC cycle. DMA address **ad<39:32>** can be obtained from the W_DAC register. DMA address **ad<63:40>** had to be zero for the 21174 to hit on the DAC cycle. This register is unlocked when the error bits in the PYXIS_ERR register are all cleared. The contents of this register are UNPREDICTABLE when not locked.

The PCI_ERR1 register and some fields in PCI_ERR0 (WINDOW, DMA_DAC, and DMA_CMD) hold information related to the following errors (associated with the memory) that occurred while the 21174 is handling a DMA:

- Correctable ECC error (PYXIS_ERR<0>)
- Uncorrectable ECC error (PYXIS_ERR<1>)
- Access to nonexistent memory (PYXIS_ERR<3>)
- Invalid page table entry (PYXIS_ERR<5>)

Table 5–25 describes the PCI error register 1 fields.

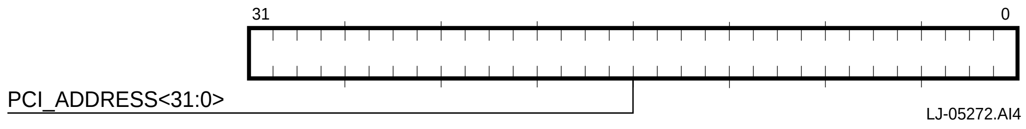
Table 5–25 PCI Error Register 1 Fields

Name	Extent	Access	Init	Description
DMA_ADDRESS<31:0>	<31:0>	RO	X	Contains the DMA address <31:0>

5.4.10 PCI Error Register 2 (PCI_ERR2)

The PCI error register 2 access is RO to address 87.4000.8880. Figure 5–22 shows the register.

Figure 5–22 PCI Error Register 2



The PCI_ERR2 register is used by the 21174 to log the **ad<31:0>** associated with an error condition logged in the PCI_ERR0 register. This register is locked whenever the 21174 detects an error. This register always captures **ad<31:0>**, even for a DMA DAC cycle. DMA PCI address **ad<31:0>** can be obtained from the W_DAC register. PCI address **ad<63:40>** had to be read for the 21174 to hit on the DAC cycle. The register is unlocked when the error bits in the PYXIS_ERR register have all been cleared. Contents of this register are UNPREDICTABLE when not locked.

The PCI_ERR2 register and some fields in PCI_ERR0 (PCI_DAC, PCI_CMD, TARGET_STATE, and MASTER STATE) hold information related to the following error conditions associated with the PCI bus:

- PCI data parity error (PYXIS_ERR<5>)
- PCI address parity error (PYXIS_ERR<6>)
- Master abort (PYXIS_ERR<7>)
- PCI target abort (PYXIS_ERR<8>)
- IOA timeout (PYXIS_ERR<11>)

Table 5–26 describes the PCI error register 2 fields.

Table 5–26 PCI Error Register 2 Fields

Name	Extent	Access	Init	Description
PCI_ADDRESS<31:0>	<31:0>	RO	X	Contains the PCI address

5.5 Memory Controller Register Descriptions

This section describes the functionality of the memory control register, the memory clock mask register, the global timing register, the refresh timing register, the row history policy mask register, the memory control debug register 1, the memory control debug register 2, the base address registers, the bank configuration registers, the bank timing registers, and the cache valid map register.

5.5.1 Memory Control Register (MCR)

The memory control register contains all of the functions needed to set up and configure the base control functions of the memory subsystem.

The memory control register access is RW to address 87.5000.0000. Figure 5–23 shows the register.

Figure 5–23 Memory Control Register

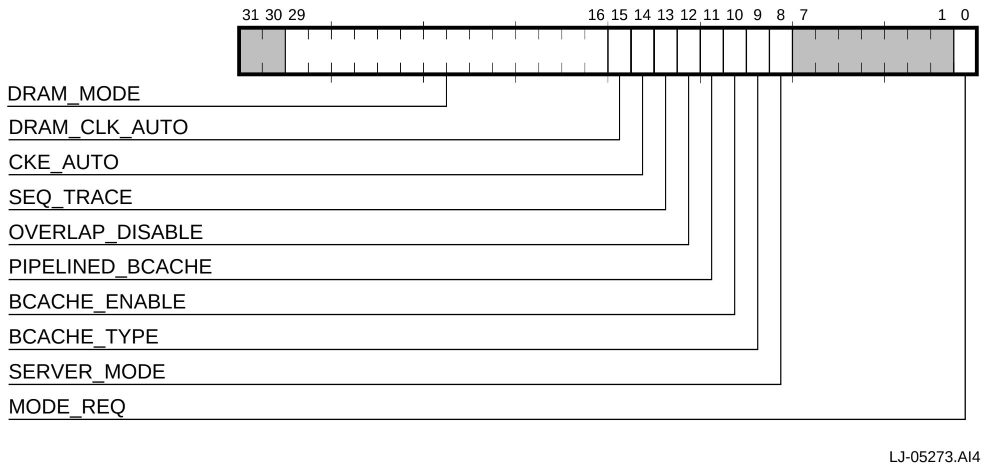


Table 5–27 describes the memory control register fields.

Table 5–27 Memory Control Register Fields *(Sheet 1 of 3)*

Name	Extent	Access	Init	Description
MODE_REQ	<0>	RW	0	Causes the 21174 to send the mode register set command to the memory DIMMs.
Reserved	<7:1>	RO	0	—
SERVER_MODE	<8>	RO	X	Shows the configuration of the system. 0 – workstation
BCACHE_TYPE	<9>	RO	X	Indicates the type of a Bcache. 0 – Nonpipelined Bcache 1 – Pipelined Bcache

Table 5–27 Memory Control Register Fields *(Sheet 2 of 3)*

Name	Extent	Access	Init	Description
BCACHE_ENABLE	<10>	RW	0	1 – Bcache-related functions in the memory controller, such as asserting **idle_bc,** probe type for certain DMA functions, and asserting **dbus_req** to acquire control of the data bus are enabled. 0 – It is necessary to operate BCACHE_ENABLE off when the Bcache is not installed or not enabled on the 21164.
PIPELINED_BCACHE	<11>	RW	0	1 – This causes the 21174 to use the **sys_clk** edge one cycle after DACK to capture read data rather than using **sram_clk_in**.
OVERLAP_DISABLE	<12>	RW	0	1 – This bit causes the memory controller to operate in a very conservative mode. New memory transactions will not be started until the data cycles of the previous transactions have completed. This feature provides a potential workaround for a large class of potential problems that might evade discovery in simulation, such as problems that might occur because of obscure interactions between transactions that overlap or occur in close proximity. Enabling this feature may reduce the maximum attainable memory bandwidth by as much as 30%, and that will result in about 3% to 5% performance loss on typical benchmarks such as Spec95 or SysmarkNT.
SEQ_TRACE	<13>	RW	0	This bit enables the output of the memory sequencer out to the DRAM address lines. Intended for debug.
CKE_AUTO[1]	<14>	RW	0	1 – Causes the automatic deassertion of **dram_cke** when there is no memory controller activity for 8 cycles.

Table 5–27 Memory Control Register Fields *(Sheet 3 of 3)*

Name	Extent	Access	Init	Description
DRAM_CLK_AUTO[2]	<15>	RW	0	1 – Causes automatic suppression of **dram_clk_in** when there is no memory controller activity for 8 cycles.
DRAM_MODE	<29:16>	RW	0	This field is subdivided into three fields that are forwarded directly to the memory DIMMs. Refer to the individual DIMM specification for details. Table 5–28 defines the fields associated with typical DIMMs. When the memory is not in use, this field drives the address lines. To conserve power, set this field to all 1s after initial setup.
Reserved	<31:30>	RO	0	—

[1] CKE_AUTO and DRAM_CLK_AUTO are used by power management routines to reduce operating power. There is no performance penalty associated with the use of these features.

Table 5–28 contains the DRAM_MODE fields.

Table 5–28 DRAM_MODE Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
BURST_LENGTH	<18:16>	WO	X	This value is dependent on the WRAP_TYPE field. The valid values are: Value — WT=0 — WT=1 000 — 1 — 1 001 — 2 — 2 010 — 4 — 4 011 — 8 — 8 100 — Reserved — Reserved 101 — Reserved — Reserved 110 — Reserved — Reserved 111 — Full Page — Reserved
WRAP_TYPE	<19>	WO	X	Tune this value for best performance. Value — Description 0 — Sequential 1 — Interleave

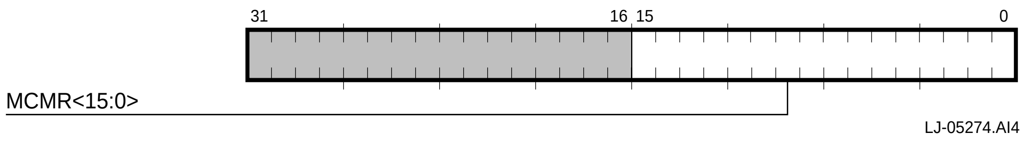
Table 5–28 DRAM_MODE Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
LATENCY_MODE	<22:20>	WO	X	This field controls memory latency mode. Value — Latency 000 — Reserved 001 — 1 010 — 2 011 — 3 100 – 111 — Reserved

5.5.2 Memory Clock Mask Register (MCMR)

The memory clock mask register access is RW to address 87.5000.0040. Figure 5–24 shows the register.

Figure 5–24 Memory Clock Mask Register



This register controls the **dram_clk<12:0>** and **sram_fill_clk<2:0>** pins. The **dram_clk<12>** pin should not be turned off because this signal line is connected to **dram_clk_in** and controls the operation of the clock delay circuit. When a register bit is set to a 0, the corresponding output pin is driven low. Any clock signal line that is not used should be turned off by setting the corresponding register bit to 0, limiting power dissipation and lowering the EMI.

Table 5–29 describes the memory clock mask register fields.

Table 5–29 Memory Clock Mask Register Fields

Name	Extent	Access	Init	Description
MCMR<15:0>	<15:0>	RW	FFFF	This field enables/disables the **dram_clk<12:0>** and **sram_fill_clk<2:0>** signal pins. See Table 5–30.
Reserved	<31:16>	RO	0	—

Table 5–30 describes the memory clock mask register fields.

Table 5–30 MCMR Bit Definitions

Name	Extent	Access	Init	Clock Pin
MCMR<0>	0	RW	1	**dram_clk<0>**
MCMR<1>	1	RW	1	**dram_clk<1>**
MCMR<2>	2	RW	1	**dram_clk<2>**
MCMR<3>	3	RW	1	**dram_clk<3>**
MCMR<4>	4	RW	1	**dram_clk<4>**
MCMR<5>	5	RW	1	**dram_clk<5>**
MCMR<6>	6	RW	1	**dram_clk<6>**
MCMR<7>	7	RW	1	**dram_clk<7>**
MCMR<8>	8	RW	1	**dram_clk<8>**
MCMR<9>	9	RW	1	**dram_clk<9>**
MCMR<10>	10	RW	1	**dram_clk<10>**
MCMR<11>	11	RW	1	**dram_clk<11>**
MCMR<12>	12	RW	1	**dram_clk<12>**
MCMR<13>	13	RW	1	**sram_fill_clk<0>**
MCMR<14>	14	RW	1	**sram_fill_clk<1>**
MCMR<15>	15	RW	1	**sram_fill_clk<2>**

5.5.3 Global Timing Register (GTR)

The global timing register contains parameters that are common to all memory transactions, including those to and from Bcache. Each parameter counts **dram_clk<12:0>** cycles. All pins on the memory interface are referenced to **dram_clk<12:0>** rising.

The global timing register access is RW to address 87.5000.0200. Figure 5–25 shows the register.

Figure 5–25 Global Timing Register

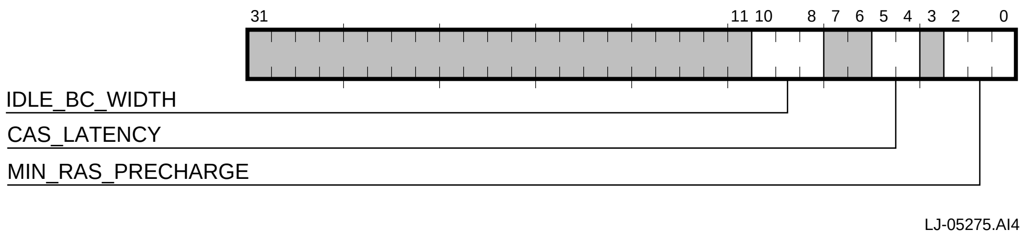


Table 5–31 describes the global timing register fields.

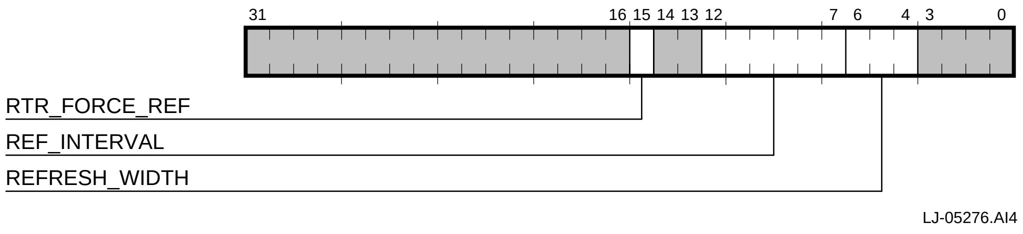
Table 5–31 Global Timing Register Fields

Name	Extent	Access	Init	Description
MIN_RAS_ PRECHARGE	<2:0>	RW	4	The minimum precharge width for the DRAMs when switch from one row to another.
Reserved	<3>	RO	0	—
CAS_LATENCY	<5:4>	RW	3	This field defines the **cas_l** latency of the SDRAMs used in the system. This field must be programmed to 3.
Reserved	<7:6>	RO	0	—
IDLE_BC_WIDTH	<10:8>	RW	0	The number of **sys_clk** cycles that the 21174 will wait before performing any Bcache transactions.
Reserved	<31:11>	RO	0	—

5.5.4 Refresh Timing Register (RTR)

The refresh timing register access is RW to address 87.5000.0300. Figure 5–26 shows the register.

Figure 5–26 Refresh Timing Register



The refresh timing register contains refresh timing information used to simultaneously refresh all banks using the **cas**-before-**ras** refresh method. These parameters should be programmed to the most conservative value across all banks.

The observed refresh interval may be greater than the value programmed in the REF_INTERVAL field by the number of **dram_clk<12:0>** cycles required to perform a read or write transaction, plus a **ras_l** precharge interval. The programmer must account for this behavior when writing to RTR[REF_INTERVAL].

All the timing parameters are in multiples of **dram_clk<12:0>** cycles. The parameters have a minimum value that is added to the programmed value. The programmer must subtract this value from the desired value before writing it to the register.

Table 5–32 describes the refresh timing register fields.

Table 5–32 Refresh Timing Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
Reserved	<3:0>	RO	0	—
REFRESH_ WIDTH	<6:4>	RW	6	The number of cycles after the refresh command is issued before any other command is attempted. This value corresponds to the **ras_l** active time (minimum) parameter in the vendor SDRAM specification.

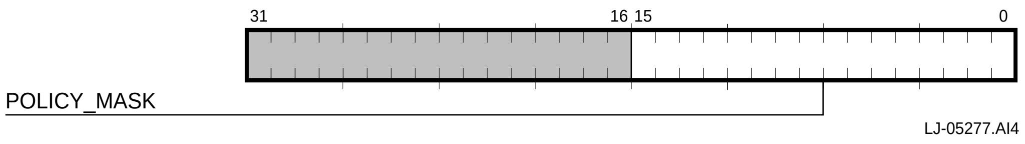
Table 5–32 Refresh Timing Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
REF_INTERVAL	<12:7>	RW	5	Refresh interval. The value of this field is multiplied by 64 to generate the number of **dram_clk<12:0>** cycles between refresh requests. A programmed value of zero is illegal.
Reserved	<14:13>	RO	0	Always zero fill.
RTR_FORCE_ REF	<15>	RW	0	Force refresh. Writing a 1 to this bit causes a single memory refresh and resets the internal refresh interval counter. The other timings in this register should not be changed while setting this bit.
Reserved	<31:16>	RO	0	—

5.5.5 Row History Policy Mask Register (RHPR)

The row history policy mask register access is RW to address 87.5000.0400. Figure 5–27 shows the register.

Figure 5–27 Row History Policy Mask Register



The state of the history buffer determines whether a row is deactivated. The history buffer remembers (for each of the last four requests to a subbank) whether the new row was the same as the old row. The 4-bit history is used as an index into the row history policy mask register (RHPR) to determine whether to deactivate the row at the end of the transaction. If the RHPR is set to all 1s, then the row is always left active. If the RHPR is set to all 0s, then the row is always deactivated at the end of the transaction.

To activate a row for an individual bank, the **ras_l** signal is asserted, along with the chip select pin to one memory bank. For refresh, the chip select signals are asserted to all memory banks simultaneously.

Table 5–33 describes the row history policy mask register fields.

Table 5–33 Row History Policy Mask Register Fields

Name	Extent	Access	Init	Description
POLICY_MASK	<15:0>	RW	E8809	Policy mask value
Reserved	<31:16>	RO	0	—

5.5.6 Memory Control Debug Register 1 (MDR1)

The memory control debug register 1 controls a debug multiplexer that drives signals on BANK01, BANK23, BANK45, and BANK67 pins when MDR1_EN is asserted. The debug signals also go to the performance monitor logic where they can be selected as inputs to the two event counters in the PERF_MON register.

The memory control debug register 1 access is RW to address 87.5000.0500. Figure 5–28 shows the register.

Figure 5–28 Memory Control Debug Register 1

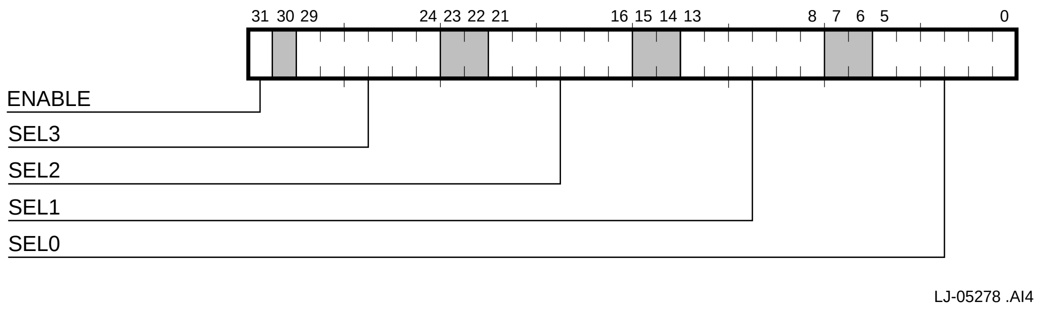


Table 5–34 describes the memory control debug register 1 fields.

Table 5–34 Memory Control Debug Register 1 Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
SEL0	<5:0>	RW	0	Select signals for output to BANK01 and MCTL_DEBUG_OUT[0]
Reserved	<7:6>	RO	0	--
SEL1	<13:8>	RW	0	Select signals for output to BANK23 and MCTL_DEBUG_OUT[1]
Reserved	<15:14>	RO	0	--

Table 5–34 Memory Control Debug Register 1 Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
SEL2	<21:16>	RW	0	Select signals for output to BANK45 and MCTL_DEBUG_OUT[2]
Reserved	<23:22>	RO	0	--
SEL3	<29:24>	RW	0	Select signals for output to BANK67 and MCTL_DEBUG_OUT[3]
Reserved	<30>	RO	0	--
ENABLE	<31>	RW	0	Enable the debug information onto BANK01, BANK34, BANK45, and BANK67

5.5.7 Memory Control Debug Register 2 (MDR2)

The memory control debug register 2 controls a debug multiplexer that drives signals on CBE[4], CBE[5], CBE[6], and CBE[7] pins when MDR2_EN is asserted. The debug signals also go to the PERF_MON register logic where they can be selected as inputs to the two event counters.

The memory control debug register 2 access is RW to address 87.5000.0540. Figure 5–29 shows the register.

Figure 5–29 Memory Control Debug Register 2

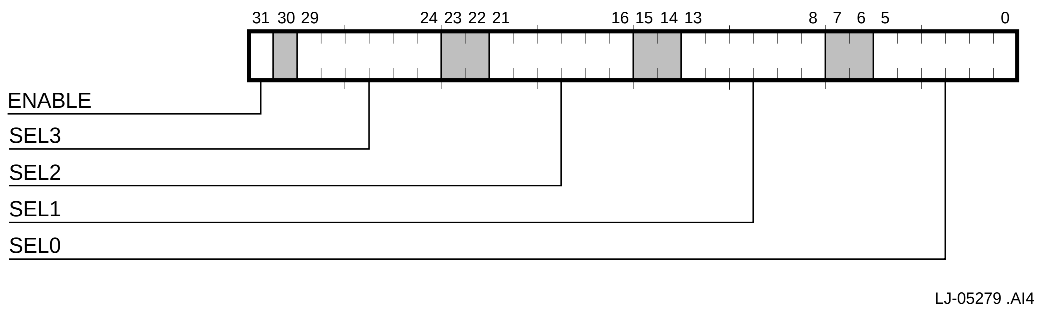


Table 5–35 describes the memory control debug register 2 fields.

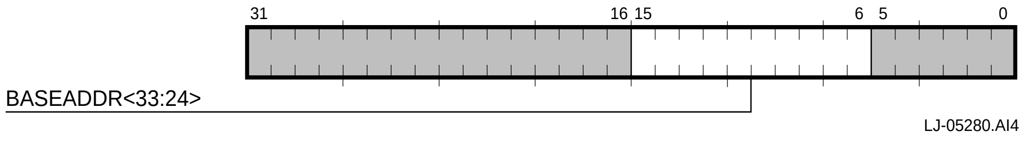
Table 5–35 Memory Control Debug Register 2 Fields

Name	Extent	Access	Init	Description
SEL0	<5:0>	RW	0	Select signals for output to CBE[4] and DEBUG_OUT[0]
Reserved	<7:6>	RO	0	—
SEL1	<13:8>	RW	0	Select signals for output to CBE[5] and DEBUG_OUT[1]
Reserved	<15:14>	RO	0	—
SEL2	<21:16>	RW	0	Select signals for output to CBE[6] and DEBUG_OUT[2]
Reserved	<23:22>	RO	0	—
SEL3	<29:24>	RW	0	Select signals for output to CBE[7] and DEBUG_OUT[3]
Reserved	<30>	RO	0	—
ENABLE	<31>	RW	0	Enable the debug information onto CBE[4], CBE[5], CBE[6], and CBE[7]

5.5.8 Base Address Registers (BBAR0–BBAR7)

The base address registers access is RW to addresses in the range 87.5000.0600 to 87.5000.07C0. Figure 5–30 shows the register.

Figure 5–30 Base Address Register



Each memory bank has a corresponding base address register. The bits in this register are compared with the incoming system address to determine the bank being addressed. The contents of this register are validated by setting the valid bit in the configuration register of that bank.

The base address of each bank must begin on a naturally aligned boundary. (For a bank with 2*n* addresses, the *n* least significant bits must be zero.)

Note: Software could require contiguous memory. Because banks must be naturally aligned, the programmer should ensure that the largest bank is placed at the lowest base address, the next largest bank is placed at a base address following the end of the largest bank, and so on, to create contiguous memory.

Table 5–36 describes the base address register fields.

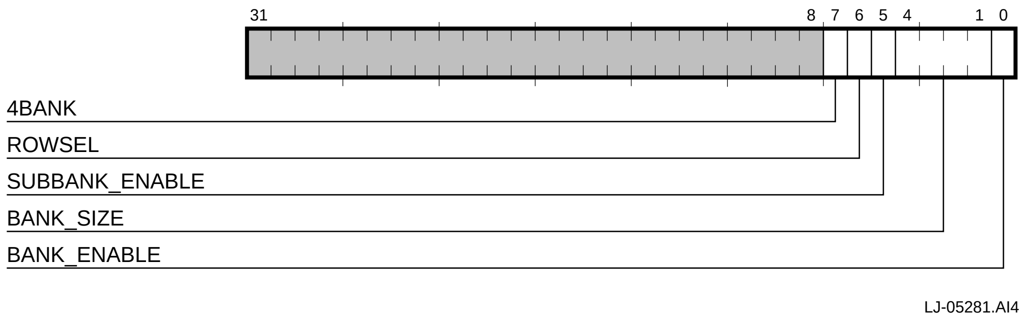
Table 5–36 Base Address Register Fields

Name	Extent	Access	Init	Description
Reserved	<5:0>	RO	0	—
BASEADDR<33:24>	<15:6>	RW	0	Starting memory address for the bank
Reserved	<31:16>	RO	0	—

5.5.9 Bank Configuration Registers (BCR0–BCR7)

The bank configuration registers access is RW to addresses in the range 87.5000.0800 to 87.5000.09C0. Figure 5–31 shows the register.

Figure 5–31 Bank Configuration Register



Each memory bank has a corresponding configuration register. This register contains mode bits and memory address generation bits, as well as bank decoding. Banks 0 through 7 have the same limits on bank size and type of DRAMs used. The format of the configuration register is the same for banks 0 through 7. Bank 8 is the VRAM bank. It supports different minimum DRAM sizes and configurations: therefore, its configuration register is different.

With the exception of the valid bit, this register is not initialized.

Table 5–37 describes the bank configuration register fields.

Table 5–37 Bank Configuration Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
BANK_ENABLE	<0>	RW	0	Bank enabled. 1 – All timing and configuration parameters for bank are valid, and access to banks allowed. If cleared, access to bank is not allowed.
BANK_SIZE<3:0>	<4:1>	RW	X	Bank size in MB. Indicates the size of the bank in order to determine which bits are used to compare the bank base address with the physical address (PA) and to generate the subset. Corresponds to the total size of the bank, including subbanks, if present.

Size<3:0>	Compared	Subset	Set Size
0000	—	—	Reserved
0001	PA<33:29>	PA<28>	512MB
0010	PA<33:28>	PA<27	256MB
0011	PA<33:27>	PA<26>	128MB
0100	PA<33:26>	PA<25>	64MB
0101	PA<33:25>	PA<24>	32MB
0110	PA<33:24>	PA<23>	16MB
0111	PA<33:23>	PA<22>	8MB
1xxx	—	—	Reserved

Name	Extent	Access	Init	Description
SUBBANK_ ENABLE	<5>	RW	0	Enable subbanks. 1 – Subbanks are enabled and determined according to the BANK_SIZE<3:0> field. 0 – Subbanks are disabled, and the **ras_l** pins will be asserted only during refreshes.
ROWSEL	<6>	RW	X	Row address selection. Indicates the number of valid row bits expected at the DRAMs. Used along with memory width information to generate row or column addresses. 0 – Indicates 12 bits of row address (16Mb DRAM) 1 – Indicates 14 bits of row address (64Mb DRAMs)

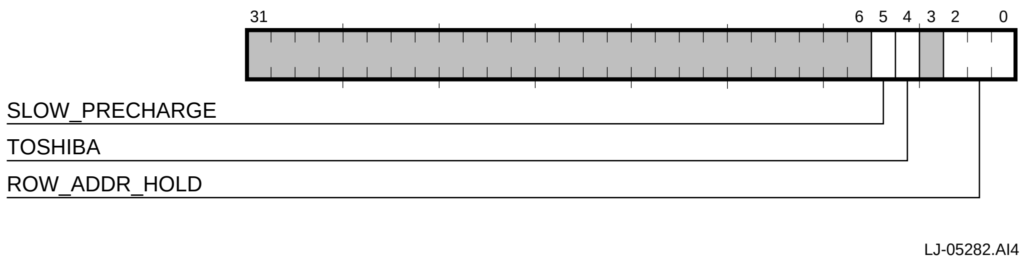
Table 5–37 Bank Configuration Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
4BANK	<7>	RW	0	1 – Four bank operation is enabled for this bank. It typically has 64MB DIMMs.
Reserved	<31:8>	RO	X	This field should always be written to zero.

5.5.10 Bank Timing Registers (BTR0–BTR7)

The bank timing registers access is RW to addresses in the range 87.5000.0A00 to 87.5000.0BC0. Figure 5–32 shows the register.

Figure 5–32 Bank Timing Register



The bank timing registers enable specific setup of memory modules. The register allows mixing of memory DIMMs. Table 5–38 describes the bank timing register fields.

Table 5–38 Bank Timing Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
ROW_ADDR_HOLD	<2:0>	RW	0	Contains the minimum number of **sys_clk** cycles that **ras_l** will be asserted before **cas_l** is asserted.
Reserved	<3>	RO	0	—
TOSHIBA	<4>	RW	0	Toshiba SDRAMs do not permit **cas_l** to be reasserted for several odd cycles after **cas_l** has been deasserted. This bit is provided for those devices. Operation of this function is implemented by forcing autoprecharge.

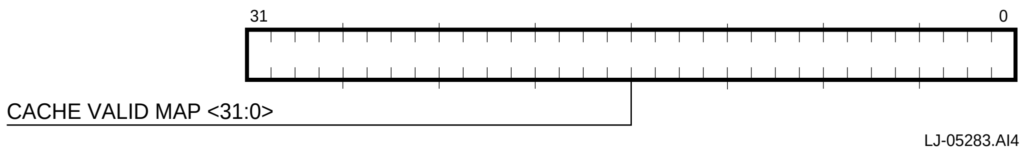
Table 5–38 Bank Timing Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
SLOW_PRECHARGE	<5>	RW	0	1 – Precharge operations are delayed for one cycle more than needed for the next **cas_l** cycle. Some vendor SDRAMs (such as NEC) require this when operating with a **cas_l** latency of 3.
Reserved	<31:6>	RO	0	—

5.5.11 Cache Valid Map Register (CVM)

The cache valid map register access is RW1C to address 87.5000.0C00. Figure 5–33 shows the register.

Figure 5–33 Cache Valid Map Register



The cache valid map register can be used by a flusher to divide the cache flush operation into smaller parts and continue the flush after clock and other short interruptions. The flusher flushes a section, and clears the CVM bits corresponding to the sections flushed. Later, the flusher can check the CVM register to find any areas of cache that have been reloaded since the flush and flush them again.

The primary use of this register is during power management. There is the possibility that cache may be large enough that it would make flushing the entire cache take longer than a single interval timer cycle. This register provides the power-management code the means to break the cache flushing sequence into parts.

Table 5–39 describes the mapping of the register. The table contains the base address for the particular bit position and is 32KB in length. The CVM register will support cache sizes from 0MB to 1MB.

The register can be used for caches larger than 1MB as each bit is aliased. The bit field then represents the offset within each 1MB bank.

Table 5–39 shows the cache valid map register fields.

Table 5–39 Cache Valid Map Register Fields

Bit Position	CVM	Bit Position	CVM
0	00000000	16	00080000
1	00008000	17	00088000
2	00010000	18	00090000
3	00018000	19	00098000
4	00020000	20	000A0000
5	00028000	21	000A8000
6	00030000	22	000B0000
7	00038000	23	000B8000
8	00040000	24	000C0000
9	00048000	25	000C8000
10	00050000	26	000D0000
11	00058000	27	000D8000
12	00060000	28	000E0000
13	00068000	29	000E8000
14	00070000	30	000F0000
15	00078000	31	000F8000

5.6 PCI Window Control Register Descriptions

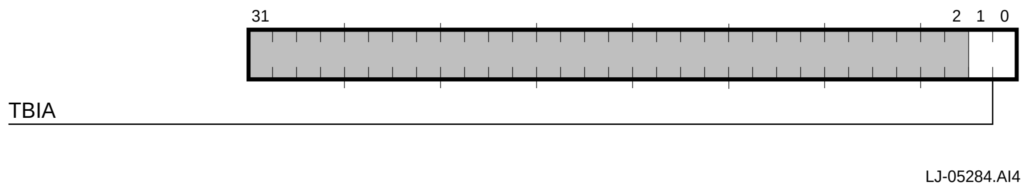
This section describes the functionality of the scatter-gather translation buffer invalidate register (TBIA), the window base registers, the window mask registers, the translated base registers, and the window DAC base register.

5.6.1 Scatter-Gather Translation Buffer Invalidate Register (TBIA)

The scatter-gather translation buffer invalidate register access is WO to address 87.6000.0100. Figure 5–34 shows the register.

PCI Window Control Register Descriptions

Figure 5–34 Scatter-Gather Translation Buffer Invalidate Register



A write to the TBIA register will result in the specified group of scatter-gather TLB tags to be marked invalid and unlocked.

Table 5–40 describes the scatter-gather translation buffer invalidate register fields.

Table 5–40 Scatter-Gather Translation Buffer Invalidate Register Fields

Name	Extent	Access	Init	Description
TBIA	<1:0>	WO	0	A write to this register invalidates the scatter-gather translation buffers. TBIA<1:0> Meaning 00 No operation. 01 Invalidate and unlock the TLB tags that are currently locked. 10 Invalidate the TLB tag that is currently unlocked. 11 Invalidate and unlock all of the TLB tag entries.[1]
Reserved	<31:2>	RO	0	—

[1] The 21174 may hang with TBIA=3. Consult Section 5.6.1.1 for the solutions to this problem.

5.6.1.1 Preventing 21174 Hang when TBIA=3

The following four techniques will prevent the 21174 from hanging when TBIA=3.

1. Allocate a dedicated PCI window and do four or eight PCI writes (or reads) to this window to four or eight separate PCI pages. This method will use up all available TLB entries and flush out any state translations.
2. Use direct mapped DMA.
3. Use a fixed table that statistically maps all of physical memory.
4. Allocate the TLB entries yourself using the lock bit.

The advantages and disadvantages of these solutions are discussed below.

Solution #1 has a performance impact. This impact can be limited by performing the invalidation sequence only when the scatter-gather table “wraps.” The scatter-gather table “wraps” when entries that might cause a hit on a stale TLB begin to be re-used.

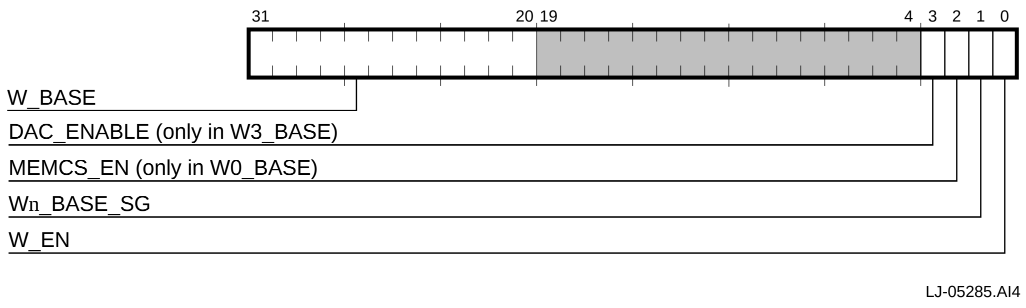
Solution #3 is easy and general. However, it requires 0.1% of the physical memory dedicated to the map. On a 128-MB system, 128 Kb of memory would be required for mapping. Unlike solution #2, this method allows the PCI addresses to be offset from the memory addresses.

Solution #4 is too restrictive and difficult to implement because of its implied limit of eight active DMA pages.

5.6.2 Window Base Registers (W*n*_BASE, *n*=0–3)

The window base register access is RW to addresses 87.6000.0400, 87.6000.0500, 87.6000.0600, and 87.6000.0700. Figure 5–35 shows the register.

Figure 5–35 Window Base Register



The window base register provides the base address for a particular target window. There are four window base registers: W0_BASE, W1_BASE, W2_BASE, and W3_BASE. The W*n*_BASE registers should not be modified unless software ensures that the no PCI traffic is targeted for the window being modified.

Table 5–41 describes the window base register fields.

Table 5–41 Window Base Register Fields

Name	Extent	Access	Init	Description
W_EN	<0>	RW	X	0 – The PCI target window is disabled and will not be used to respond to PCI initiated transfers. 1 – The PCI target window is enabled and will be used to respond to PCI initiated transfers that hit in the address range of the target window.
W*n*_BASE_SG	<1>	RW	X	0 – The PCI target window uses direct mapping to translate a PCI address to a 21164 address (see Table 5–42). 1 – The PCI target window uses scatter-gather mapping to translate a PCI address to a physical memory address (see Table 5–43).
MEMCS_EN (only in **W0_BASE)**	<2>	RW	X	When the MEMCS_EN bit is set, then the MEMCS signal from the PCI ISA or PCI EISA bridge is ANDed with the normal window hit.
DAC_ENABLE (only in **W3_BASE)**	<3>	RW	X	1 – The W_DAC register is compared against PCI address<39:32> for a PCI DAC cycle. If this compare hits, and the 32-bit portion of the PCI address hits, then a DAC cycle hit occurs.
Reserved	<19:4>	RO	0	—
W_BASE	<31:20>	RW	X	W_BASE specifies the PCI base address of the PCI target window and is used to determine a hit in the window. See MEMCS_EN and DAC_ENABLE also.

5.6.2.1 Determining a Hit in the Target Window

The incoming **ad<31:20>** is compared with each of the four translated base registers (T*n*_BASE). The associated W*n*_MASK register determines which bits are involved in the comparison.

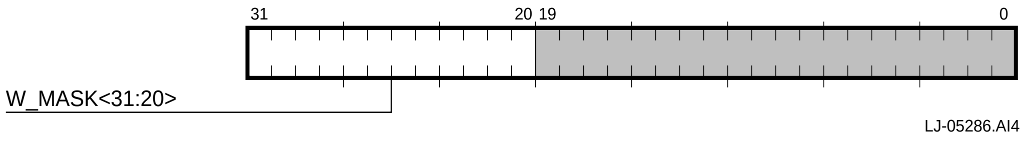
The target window is hit when a masked address matches a valid translated base register (T*n*_BASE). If the W0_BASE[MEMCS_EN] is set, then the hit is further qualified by the state of the **mem_cs_l** input signal — this is used if peripheral component compatibility holes are required in the 21174 (see Section 6.16.1).

When the DAC_ENABLE bit is set in the W3_BASE register, the W_DAC base register is used to compare **ad<39:32>** of a DAC cycle.

5.6.3 Window Mask Registers (W*n*_MASK, *n*=0–3)

The window mask register access is RW to addresses 87.6000.0440, 87.6000.0540, 87.6000.0640, and 87.6000.0740. Figure 5–36 shows the register.

Figure 5–36 Window Mask Register



The window mask register provides a mask corresponding to **ad<31:20>**. The size of each window can be programmed to be from 1MB to 4GB in powers of two by masking bits of the incoming PCI address via the window mask register as shown in Table 5–42.

There are four window mask registers: W0_MASK, W1_MASK, W2_MASK, and W3_MASK. The W*n*_MASK registers should not be modified unless software ensures that no PCI traffic is targeted for the window being modified.

5.6.3.1 Determining a Hit in the Target Window

The incoming PCI address **ad<31:20>** is compared with each of the four W*n*_BASE registers — the associated W*n*_MASK register determines which bits are involved in the comparison. The target window is hit when a masked address matches a valid W*n*_BASE register.

PCI Window Control Register Descriptions

Table 5–42 describes the window mask register fields.

Table 5–42 Window Mask Register Fields

Name	Extent	Access	Init	Description
Reserved	<19:0>	RO	0	—
W_MASK<31:20>	<31:20>	RW	X	This field specifies the size of the PCI target window (see Table 5–44) and it is also used to mask out address bits not used when determining a PCI target window hit.

Table 5–43 shows the W_MASK<31:20> field.

Table 5–43 W_MASK<31:20> Field

W_MASK<31:20>	Size of Window	W_MASK<31:20>	Size of Window
0000 0000 0000	1MB	0000 0111 1111	128MB
0000 0000 0001	2MB	0000 1111 1111	256MB
0000 0000 0011	4MB	0001 1111 1111	512MB
0000 0000 0111	8MB	0011 1111 1111	1GB
0000 0000 1111	16MB	0111 1111 1111	2GB
0000 0001 1111	32MB	1111 1111 1111	4GB
0000 0011 1111	64MB	Otherwise	Not supported

Table 5–44 shows PCI address translation with scatter-gather mapping disabled.

Table 5–44 PCI Address Translation — Scatter-Gather Mapping Disabled *(Sheet 1 of 2)*

W_MASK<31:20>	Translated Address <33:0>	Unused Translated Base Register Bits[1]
0000 0000 0000	T*n*_BASE<33:20> : **ad<19:0>**	T*n*_BASE<19:10>
0000 0000 0001	T*n*_BASE<33:21> : **ad<20:0>**	T*n*_BASE<20:10>
0000 0000 0011	T*n*_BASE<33:22> : **ad<21:0>**	T*n*_BASE<21:10>
0000 0000 0111	T*n*_BASE<33:23> : **ad<22:0>**	T*n*_BASE<22:10>
0000 0000 1111	T*n*_BASE<33:24> : **ad<23:0>**	T*n*_BASE<23:10>
0000 0001 1111	T*n*_BASE<33:25> : **ad<24:0>**	T*n*_BASE<24:10>

Table 5–44 PCI Address Translation — Scatter-Gather Mapping Disabled *(Sheet 2 of 2)*

W_MASK<31:20>	Translated Address <33:0>	Unused Translated Base Register Bits[1]
0000 0011 1111	T*n*_BASE<33:26> : **ad<25:0>**	T*n*_BASE<25:10>
0000 0111 1111	T*n*_BASE<33:27> : **ad<26:0>**	T*n*_BASE<26:10>
0000 1111 1111	T*n*_BASE<33:28> : **ad<27:0>**	T*n*_BASE<27:10>
0001 1111 1111	T*n*_BASE<33:29> : **ad<28:0>**	T*n*_BASE<28:10>
0011 1111 1111	T*n*_BASE<33:30> : **ad<29:0>**	T*n*_BASE<29:10>
0111 1111 1111	T*n*_BASE<33:31> : **ad<30:0>**	T*n*_BASE<30:10>
1111 1111 1111	T*n*_BASE<33:32> : **ad<31:0>**	T*n*_BASE<31:10>

[1] Unused translation base register bits must be zero for correct operation.

Table 5–45 shows PCI address translation with scatter-gather mapping enabled.

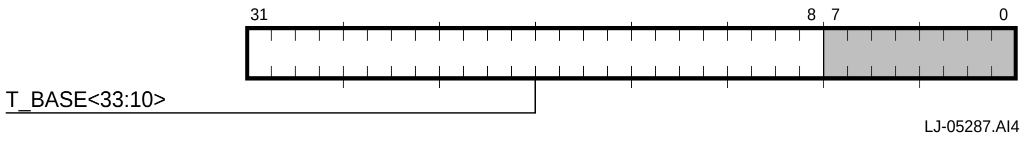
Table 5–45 PCI Address Translation — Scatter-Gather Mapping Enabled

W_MASK<31:20>	SG Map Table Size	Scatter-Gather Map Address<33:0> (Used to Index SG Table in Memory)
0000 0000 0000	1KB	T*n*_BASE<33:10> : **ad<19:13>**:000
0000 0000 0001	2KB	T*n*_BASE<33:11> : **ad<20:13>**:000
0000 0000 0011	4KB	T*n*_BASE<33:12> : **ad<21:13>**:000
0000 0000 0111	8KB	T*n*_BASE<33:13> : **ad<22:13>**:000
0000 0000 1111	16KB	T*n*_BASE<33:14> : **ad<23:13>**:000
0000 0001 1111	32KB	T*n*_BASE<33:15> : **ad<24:13>**:000
0000 0011 1111	64KB	T*n*_BASE<33:16> : **ad<25:13>**:000
0000 0111 1111	128KB	T*n*_BASE<33:17> : **ad<26:13>**:000
0000 1111 1111	256KB	T*n*_BASE<33:18> : **ad<27:13>**:000
0001 1111 1111	512KB	T*n*_BASE<33:19> : **ad<28:13>**:000
0011 1111 1111	1MB	T*n*_BASE<33:20> : **ad<29:13>**:000
0111 1111 1111	2MB	T*n*_BASE<33:21> : **ad<30:13>**:000
1111 1111 1111	4MB	T*n*_BASE<33:22> : **ad<31:13>**:000

5.6.4 Translated Base Registers (T*n*_BASE, *n*=0–3)

The translated base register access is RW to addresses 87.6000.0480, 87.6000.0580, 87.6000.0680, and 87.6000.0780. Figure 5–37 shows the register.

Figure 5–37 Translated Base Register



The translated base register is used to map PCI addresses into memory. There are four translated base registers: T0_BASE, T1_BASE, T2_BASE, and T3_BASE, one for each window. If W*n*_BASE[**W*n*_BASE_SG]** is clear, the translated base register provides the base physical address of this window. If W*n*_BASE[**W*n*_BASE_SG]** is set, then the translated base register provides the base address of the scatter-gather map for this window. The T*n*_BASE registers should not be modified unless software ensures that the no PCI traffic is targeted for the window being modified.

Table 5–46 describes the translated base register fields.

Table 5–46 Translated Base Registers Fields

Name	Extent	Access	Init	Description
Reserved	<7:0>	RO	0	—
T_BASE<33:10>	<31:8>	RW	X	If scatter-gather mapping is disabled, T*n*_BASE<33:10> specifies the base 21164 address of the translated PCI address for the PCI target window (see Table 5–43). If scatter-gather mapping is enabled, T*n*_BASE<33:10> specifies the base 21164 address for the scatter-gather map table for the PCI target window (see Table 5–44).

The field W*n*_MASK<31:20> sets the size of the PCI target window and the number of 8-KB pages that fall into the window. Every 8-KB page requires one 8-byte scatter-gather map entry.

Table 5–43 shows the relationship of W*n*_MASK to the size of the scatter-gather map in memory. The number of entries required can be calculated as follows:

$$\frac{\text{Size of window (in bytes)}}{\text{8KB}} = \text{Number of entries required}$$

The size of the scatter-gather table can be calculated as follows:

$$\text{Number of entries} \times \text{8KB} = \text{Size of the scatter-gather table}$$

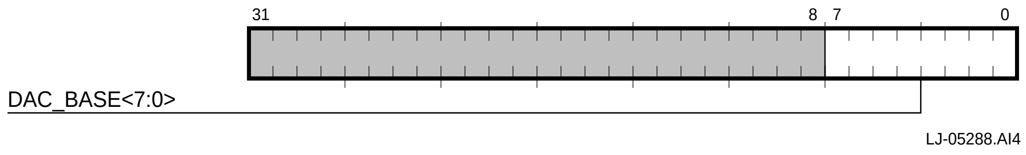
Concatenate the appropriate T*n*_ BASE and PCI address bits (based on the size of the scatter-gather map) to generate a quadword address to index into the table. The PCI address forms the index into the table while the T*n*_BASE forms the naturally aligned base of the table.

For example, for a mask of 0000 0000 0000, there are 128 entries in the scatter-gather table and the table size is 1KB. Entries are quadwords, so the lower three bits of the address (<2:0>) are always zero. Now, mask off PCI bits <31:20> (because of the W*n*_MASK). Then use **ad<19:13>** (7 bits, 2 to the power 7 = 128 entries in the table) as the table index. Use the T*n*_BASE<33:10> to get the other bits of the 34-bit address.

5.6.5 Window DAC Base Register (W_DAC)

The window DAC base register access is RW to address 87.6000.07C0. Figure 5–38 shows the register.

Figure 5–38 Window DAC Base Register



The window DAC base register provides the <7:0> address bits for comparison against **ad<39:32>** during a DAC cycle. The **ad<63:40>** has to be zero for a PCI window hit. The window DAC base register is used in conjunction with the W*n*_BASE register. For more details, see Chapter 6.

The window DAC base register is only applicable to window 3 and only if enabled by W3_BASE[DAC_ENABLE].

The target window is hit when the following is satisfied:

- The incoming **ad<31:20>** matches one of the four window base registers; the W*n*_MASK register determines which bits are involved in the comparison.
- **ad<63:40>** is zero.
- **ad<39:32>** match W_DAC[DAC_BASE].

Table 5–47 describes the window DAC base register fields.

Table 5–47 Window DAC Base Register Fields

Name	Extent	Access	Init	Description
DAC_BASE<7:0>	<7:0>	RW	X	DAC_BASE specifies bits <39:32> of the PCI base address used to determine a hit in the target window for the DAC cycle.
Reserved	<31:8>	RO	0	—

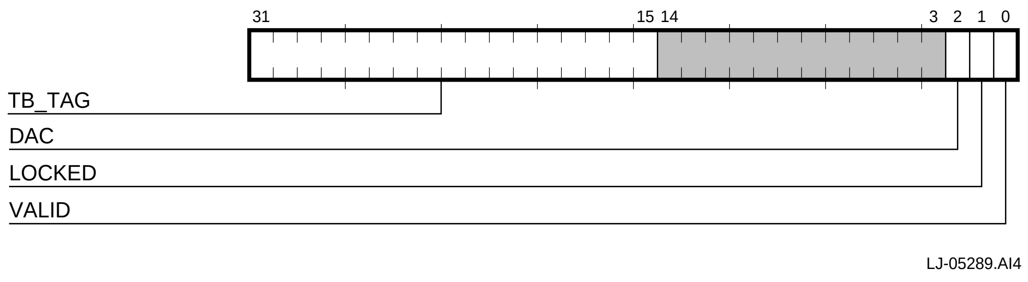
5.7 Scatter-Gather Address Translation Register Descriptions

This section describes the functionality of the lockable translation buffer tag registers, the translation buffer tag registers, and the translation buffer page registers.

5.7.1 Lockable Translation Buffer Tag Registers (LTB_TAG*n, n*=0–3)

The lockable translation buffer tag register access is RW to addresses 87.6000.0800, 87.6000.0840, 87.6000.0880, and 87.6000.08C0. Figure 5–39 shows the register.

Figure 5–39 Lockable Translation Buffer Tag Register



There are four lockable translation buffer tag registers. Software can write to these LTB_TAG*n* entries. Furthermore, they can be locked such that the hardware will not evict the entry on a scatter-gather table miss.

Note: Be careful when writing to this register. Writing to this register while a DMA operation is in progress will cause UNPREDICTABLE results. Write to this register only when certain that a DMA operation is not in progress, or always assert the LOCKED bit in the register.

Table 5–48 describes the lockable translation buffer tag register fields.

Table 5–48 Lockable Translation Buffer Tag Register Fields

Name	Extent	Access	Init	Description
VALID	<0>	RW	0	If VALID and PYXIS_CTRL[SG TLB_EN] are set, then this entry will be used for a translation.
LOCKED	<1>	RW	0	If LOCKED is set, the hardware will never evict this entry.
DAC	<2>	RW	0	If set, then this TAG entry corresponds to a 64-bit PCI address (DAC cycle); otherwise, it belongs to a 32-bit PCI address (SAC cycle).
Reserved	<14:3>	RO	0	—
TB_TAG	<31:15>	RW	X	TB_TAG<31:15> is the TAG for each translation buffer entry.

5.7.1.1 Determining a Hit in the Translation Buffer

After a PCI address hits one of the window registers with scatter-gather operation enabled, the incoming **ad<31:15>** is compared with each of the eight translation buffer tag registers. If there is a match, the corresponding translation buffer page register group is indexed by **ad<14:13>,** and if the page entry is valid there is a translation buffer hit.

5.7.1.2 Operation on a SG_TLB Miss

A scatter-gather TLB miss is handled by hardware using a round-robin algorithm. An entry is overwritten if it is not locked. The hardware will write all four PTEs on a miss.

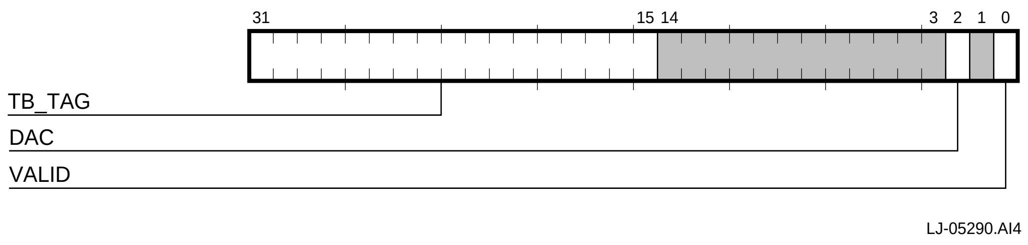
Note: Be careful when writing to this register. Writing to this register while a DMA is in progress causes UNPREDICTABLE results. Write to this register only when certain that a DMA operation is not in progress, or always assert the LOCKED bit in the register.

5.7.2 Translation Buffer Tag Registers (TB_TAG*n*, *n*=4–7)

There are four translation buffer tag registers that cannot be locked by software. Software can write to the TB_TAG entries, but they cannot be locked (and so, may be evicted by the hardware on a scatter-gather table miss).

The translation buffer tag registers access is RW to addresses 87.6000.0900, 87.6000.0940, 87.6000.0980, and 87.6000.09C0. Figure 5–40 shows the register.

Figure 5–40 Translation Buffer Tag Register



Note: Be careful when writing to this register. Writing to this register while a DMA operation is in progress will cause UNPREDICTABLE results. Write to this register only when certain that a DMA operation is not in progress.

Table 5–49 describes the translation buffer tag register fields.

Table 5–49 Translation Buffer Tag Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
VALID	<0>	RW	0	If PYXIS_CTRL[SG_TLB_EN] and VALID are set, this entry will be used for address translation.
Reserved	<1>	RO	0	—

Table 5–49 Translation Buffer Tag Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
DAC	<2>	RW	0	1 – This tag entry corresponds to a 64-bit PCI address (DAC cycle); otherwise, it belongs to a 32-bit PCI address (SAC cycle).
Reserved	<14:3>	RO	0	—
TB_TAG	<31:15>	RW	X	TB_TAG<31:15> is the TAG for each translation buffer entry.

5.7.2.1 Determining a Hit in the Translation Buffer

The incoming **ad<31:15>** is compared with each of the eight translation buffer tag registers. If there is a match, the corresponding translation buffer page register group is indexed by **ad<14:13>**, and if it is valid then there is a translation buffer hit.

5.7.2.2 Operation on a SG_TLB Miss

A scatter-gather TLB miss is handled by hardware using a round-robin algorithm. An entry is overwritten if it is not locked. The hardware will write all four PTEs on a miss.

Note: Writing to this register while a DMA transaction is in progress will cause UNPREDICTABLE results.

5.7.3 Translation Buffer Page Registers (TB*m*_PAGE*n*, *m*=0–7, *n*=0–3)

There are 32 translation buffer page registers, a group of four for each of the eight translation buffer entries. The TB*m*_PAGE*n* registers are automatically updated on a TLB miss (a group of four at a time) by the 21174 hardware.

The translation buffer page registers access is RW to addresses 87.6000.1000 through 87.6000.17C0. Figure 5–41 shows the register.

Figure 5–41 Translation Buffer Page Register

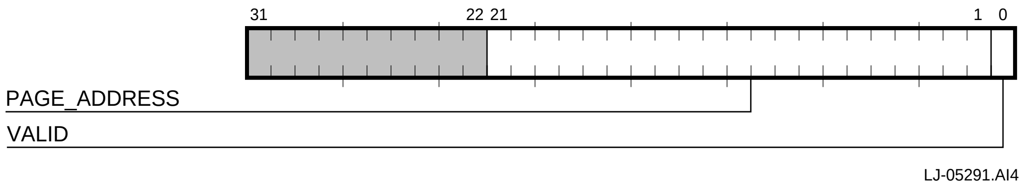


Table 5–50 describes the translation buffer page register fields.

Table 5–50 Translation Buffer Page Register (TB*m*_PAGE*n*) Fields

Name	Extent	Access	Init	Description
VALID	<0>	RW	X	The entry is valid when this bit is set to a one.
PAGE_ADDRESS	<21:1>	RW	X	The PAGE_ADDRESS<21:1> forms physical address<33:13>. ad_H<12:0> forms physical address<12:0>.
Reserved	<31:22>	RO	0	—

5.7.3.1 Determining a Hit in the Translation Buffer

The incoming **ad<31:15>** are compared with each of the eight translation buffer tag registers. If there is a match, the corresponding translation buffer page register group is indexed by **ad<14:13>,** and if it is valid, then there is a translation buffer hit.

If the address bits do not match the tag, or the page entry is invalid, then a TLB miss occurs. If the PTE fetched by the hardware TLB-miss handler is still invalid, then a DMA write transaction error bit is set in MESR1, causing an interrupt to be asserted.

5.8 Miscellaneous Register Descriptions

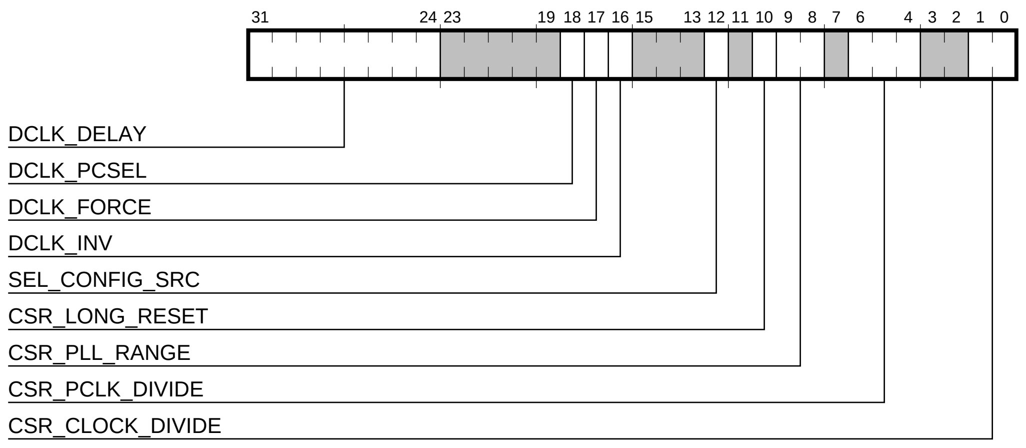
This section describes the functionality of the clock control register, the clock status register, and the reset register.

5.8.1 Clock Control Register (CCR)

The clock control register determines how the 21174 clock and the DRAM clock will be presented to the system on the next warm reset. A warm reset is executed when the RESET register is written appropriately. The values are maintained across a warm reset.

The clock control register access is RW to address 87.8000.0000. Figure 5–42 shows the register.

Figure 5–42 Clock Control Register



LJ-05292.AI4

Table 5–51 describes the clock control register fields.

Table 5–51 Clock Control Register Fields *(Sheet 1 of 3)*

Name	Extent	Access	Init	Description
CSR_CLOCK_DIVIDE	<1:0>	RW	1	This value will be used for CLK_DIVIDE on the next warm reset if CCR[SEL_CONFIG_SRC] = 1.
Reserved	<3:2>	RO	0	—
CSR_PCLK_DIVIDE	<6:4>	RW	3	This value will be used for PCLK_DIVIDE on the next warm reset if CCR[SEL_CONFIG_SRC] = 1.
Reserved	<7>	RO	0	—

Table 5–51 Clock Control Register Fields *(Sheet 2 of 3)*

Name	Extent	Access	Init	Description
CSR_PLL_RANGE	<9:8>	RW	2	This value will be used for PLL_RANGE on the next warm reset if CCR[SEL_CONFIG_SRC] = 1.
CSR_LONG_RESET	<10>	RW	1	This value will be used for LONG_RESET on the next warm reset if CCR[SEL_CONFIG_SRC] = 1.
Reserved	<11>	RO	0	—
SEL_CONFIG_SRC	<12>	RW	0	This bit selects the clock configuration source at the next warm reset (driven by software through the RESET register). 0 – The clock power-up configuration is taken from the CLK_STAT register (default). 1 – The clock configuration is taken from this register.
Reserved	<15:13>	RO	0	—
DCLK_INV	<16>	RW	0	1 – Inverts the internal DRAM _CLK. It does not invert the external DRAM clock driven to the DIMMs. This effectively changes the range of the programmable delay elements. This bit is established by the power-up software and should be clear for normal **sys_clk** divide ratios.
DCLK_FORCE	<17>	RW	1	1 – Forces the internal DRAM_CLK to be DCLK_DELAY. This bit should be cleared by the power-up software at least 2,048 **sys_clk** cycles before accessing DRAM. It should be left cleared.
DCLK_PCSEL	<18>	RW	0	Selects the best (TBD) phase comparator for the DRAM clock feedback and the auto aligning delay circuitry.
Reserved	<23:19>	RO	0	—

Table 5–51 Clock Control Register Fields *(Sheet 3 of 3)*

Name	Extent	Access	Init	Description
DCLK_DELAY	<31:24>	RW	18	Represents the delay value added to the internal DRAM_CLK if DRAM_FORCE is asserted. This value drives the delay count chain.

5.8.2 Clock Status Register (CLK_STAT)

The clock status register shows the current state of the clock generator.

The clock status register access is RO to address 87.8000.0100. Figure 5–43 shows the register.

Figure 5–43 Clock Status Register

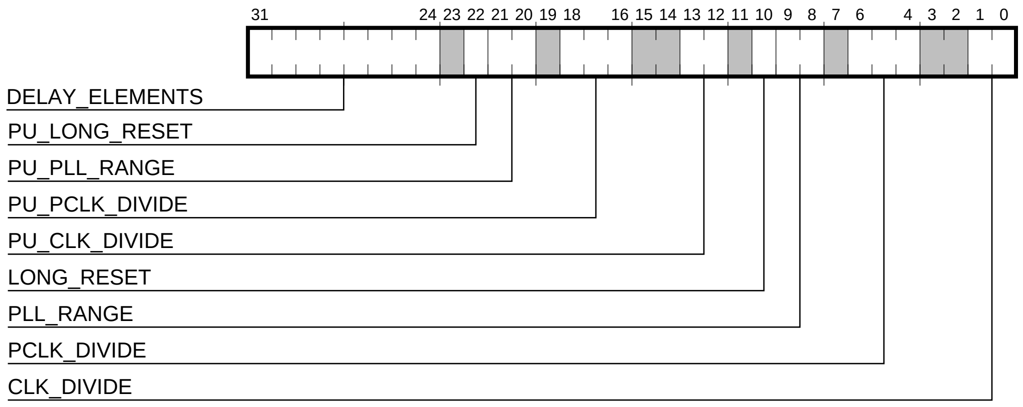


Table 5–52 describes the clock status register fields.

Table 5–52 Clock Status Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
CLK_DIVIDE	<1:0>	RO	X	The current running value of the programmable divisor in the PLL feedback path (the internal chip PLL multiplication factor) +1. For example, a value of 01 indicates a divide by two. The 21174 CLK always runs at the **sys_clk** ratio determined by the 21164.
Reserved	<3:2>	RO	0	—
PCLK_DIVIDE	<6:4>	RO	X	The CURRENT running value of the programmable PCLK divisor from the PLL, +1. Thus, the PCI interface is running at: $\frac{\text{sys\_clk} \times (\text{CLK\_DIVIDE} + 1)}{(\text{PCLK\_DIVIDE} + 1)\text{MHz}}$
Reserved	<7>	RO	0	—
PLL_RANGE	<9:8>	RO	X	The current range bits to the PLL for the appropriate CLK divide ratio and **sys_clk** frequency.
LONG_RESET	<10>	RO	X	Controls the current reset assertion length after **dc_ok** is asserted. 0 – Selects a short reset (about 15 ms). 1 – Selects a long reset (about 240 ms).
Reserved	<11>	RO	0	—
PU_CLK_DIVIDE	<13:12>	RO	X	The value of PU_CLK_DIVIDE is read off the **addr_h<31:30>** pins — with pull-up/pull-down resistors at cold power-up.
Reserved	<15:14>	RO	0	—
PU_PCLK_DIVIDE	<18:16>	RO	X	The value of PU_PCLK_DIVIDE is read off the **addr_h<34:32>** pins — with pull-up/pull-down resistors at cold power-up.
Reserved	<19>	RO	0	—

Table 5–52 Clock Status Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
PU_PLL_RANGE	<21:20>	RO	X	The value of PU_PLL_RANGE is read off the **addr_h<38:37>** pins — with pull-up/ pull-down resistors at cold power-up.
PU_LONG_RESET	<22>	RO	X	The value of PU_LONG_RESET is read off the **addr_h<35>** pin — with pull-up/ pull-down resistors at cold power-up. 1 (pull-up) – Indicates long reset. 0 (pull-down) – Indicates a short reset.
Reserved	<23>	RO	0	—
DELAY_ELEMENTS	<31:24>	RO	X	The number of delay elements currently used in the DRAM clock generation, possible range is 00_{16} – $7F_{16}$.

5.8.3 Reset Register (RESET)

This 21174 register is used by software to reset the system. Writing the value $0000DEAD_{16}$ will cause a complete system reset.

The reset register access is WO to address 87.8000.0900. Figure 5–44 shows the reset register.

Figure 5–44 Reset Register

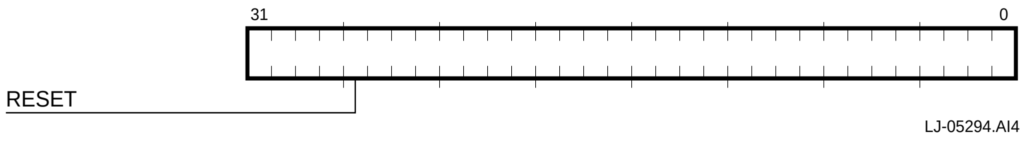


Table 5–53 describes the reset register fields.

Table 5–53 Reset Register Fields

Name	Extent	Access	Init	Description
RESET	<31:0>	WO	X	Writing $0000DEAD_{16}$ to this register will force a system reset.

5.9 Interrupt Control Registers Descriptions

This section describes the functionality of the interrupt request register, the interrupt mask register, the interrupt high/low select register, the interrupt routine select register, the general-purpose output register, the interrupt configuration register, the real-time counter register, the interrupt time register, and the I^{2}C control register.

5.9.1 Interrupt Request Register (INT_REQ)

This register is used to read the interrupt request lines from the main interrupt logic. If a bit is set, then it signifies that an interrupt is active.

The interrupt request register access is RW1C to address 87.A000.0000. Figure 5–45 shows the register.

Figure 5–45 Interrupt Request Register

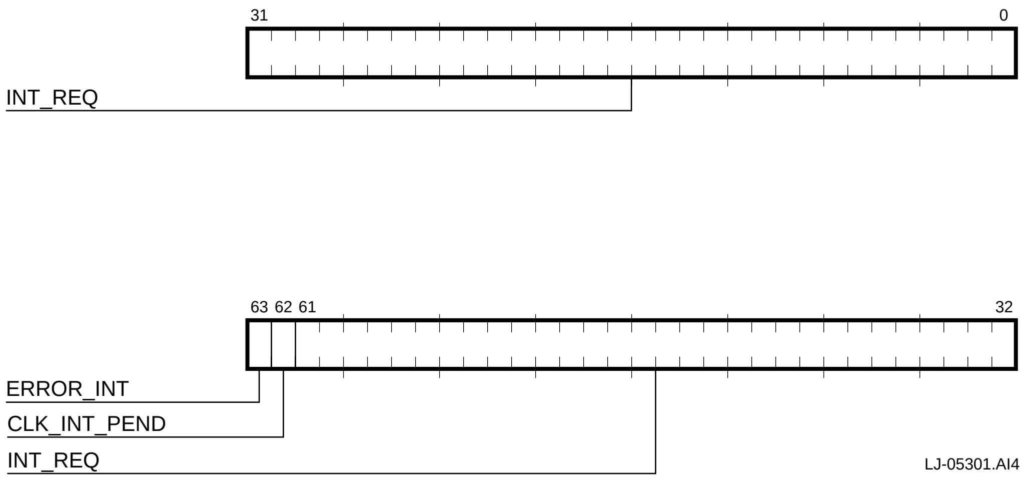


Table 5–54 describes the interrupt request register fields.

Table 5–54 Interrupt Request Register Fields

Name	Extent	Access	Init	Description
INT_REQ	<61:0>	RW1C	X	An interrupt is asserted when a bit is set to one. Each bit indicates a single interrupt request line.
CLK_INT_PEND	<62>	RW1C	0	Real-time count interrupt pending.
ERROR_INT	<63>	RO	X	Machine check error detected. This is the logical OR of all of the sources of the machine check error interrupts.

5.9.2 Interrupt Mask Register (INT_MASK)

The interrupt mask register is used to access the interrupt mask register, which is physically located in the main interrupt logic. The main interrupt logic has 64 inputs that can all be individually masked.

The interrupt mask register access is RW to address 87.A000.0040. Figure 5–46 shows the register.

Figure 5–46 Interrupt Mask Register

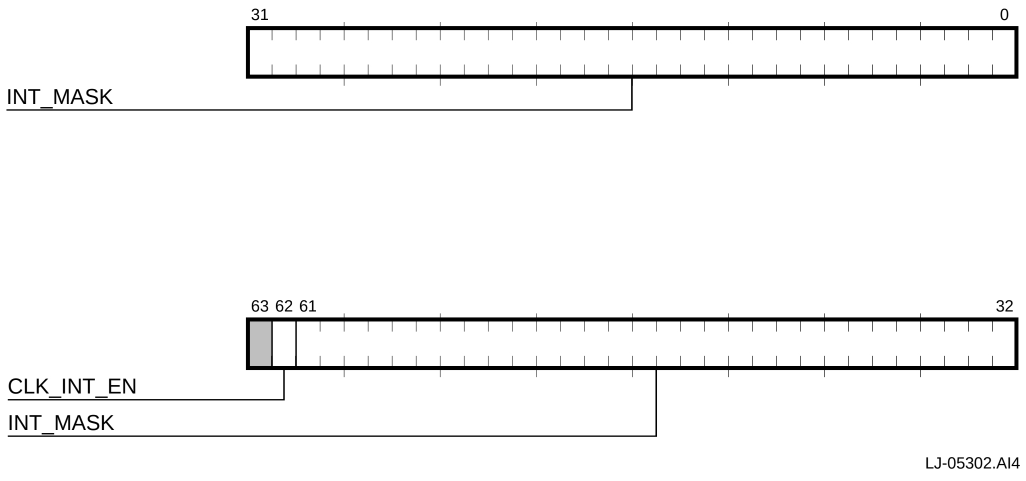


Table 5–55 describes the interrupt mask register fields.

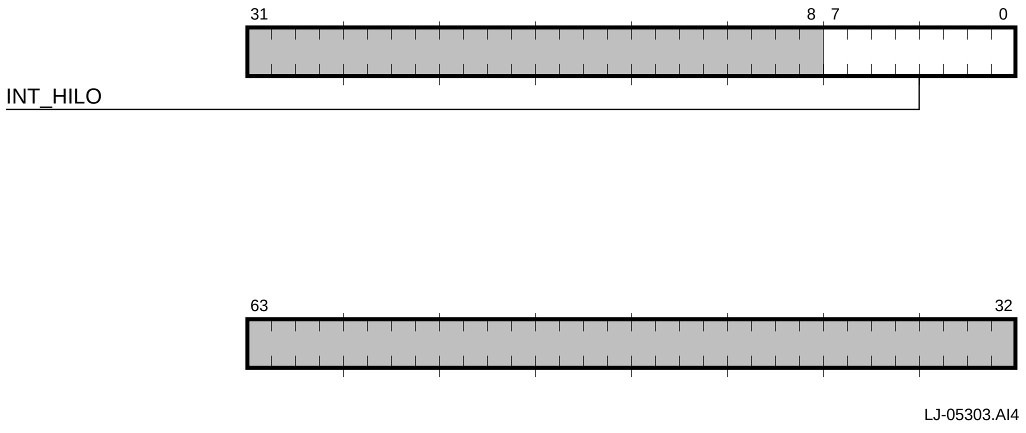
Table 5–55 Interrupt Mask Register Fields

Name	Extent	Access	Init	Description
INT_MASK	<61:0>	RW	0	1 – Interrupt IRQ is enabled for this bit. 0 – Interrupts are disabled for this bit.
CLK_INT_EN	<62>	RW	0	Enable the real-time counter interrupt.
Reserved	<63>	RO	0	—

5.9.3 Interrupt High/Low Select Register (INT_HILO)

The interrupt high/low select register access is RW to address 87.A000.00C0. Figure 5–47 shows the register.

Figure 5–47 Interrupt High/Low Select Register



This register is used to control the main interrupt logic. A set bit signifies that the associated **irq** signal line is active high. Otherwise, it is active low.

Table 5–56 describes the interrupt high/low select register fields.

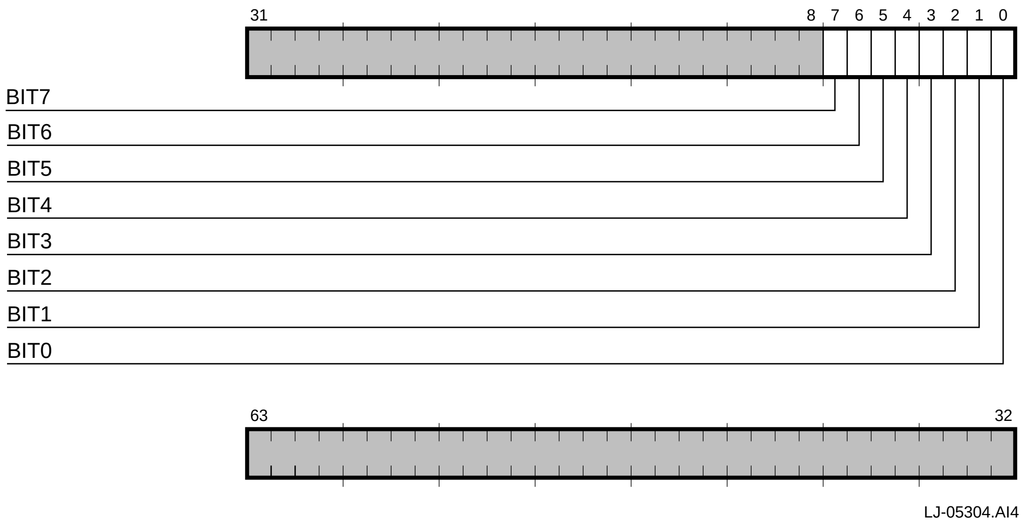
Table 5–56 Interrupt High/Low Select Register Fields

Name	Extent	Access	Init	Description
INT_HILO	<7:0>	RW	0	0 – Active low interrupt (PCI type) 1 – Active high interrupt (such as PCI-EISA bridge interrupt)
Reserved	<63:8>	RO	0	—

5.9.4 Interrupt Routine Select Register (INT_ROUTE)

The interrupt routine select register access is RW to address 87.A000.0140. Figure 5–48 shows the register.

Figure 5–48 Interrupt Routine Select Register



This register is used to control the main interrupt logic. A set bit signifies that the **irq** line is routed to the specified source. Otherwise, the interrupt is routed to **irq<1>**.

Table 5–57 describes the interrupt routine select register fields. The table defines the actual source and the interrupt to which it is routed.

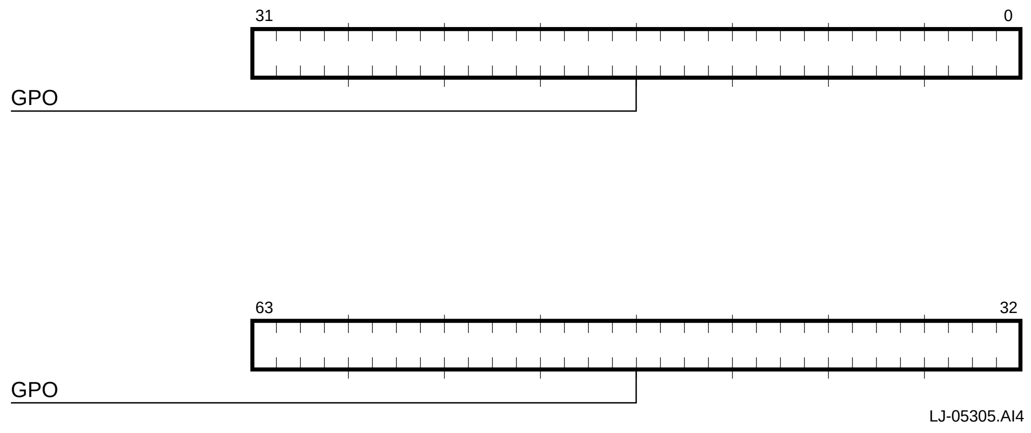
Table 5–57 Interrupt Routine Select Register Fields

Name	Extent	Access	Init	Description
BIT0	<0>	RW	0	1 – The request is routed to **mchk_irq**. 0 – The request is routed to **irq<1>**.
BIT1	<1>	RW	0	1 – The request is routed to **mchk_irq**. 0 – The request is routed to **irq<1>**.
BIT2	<2>	RW	0	1 – The request is routed to **hlt_irq**. 0 – The request is routed to **irq<1>**.
BIT3	<3>	RW	0	1 – The request is routed to **hlt_irq**. 0 – The request is routed to **irq<1>**.
BIT4	<4>	RW	0	1 – The request is routed to **irq<0>**. 0 – The request is routed to **irq<1>**.
BIT5	<5>	RW	0	1 – The request is routed to **irq<0>.** 0 – The request is routed to **irq<1>**.
BIT6	<6>	RW	0	1 – The request is routed to **irq<2>.** 0 – The request is routed to **irq<1>**.
BIT7	<7>	RW	0	1 – The request is routed to **irq<3>**. 0 – The request is routed to **irq<1>**.
Reserved	<63:8>	RO	0	—

5.9.5 General-Purpose Output Register (GPO)

The general-purpose output register access is WO to address 87.A000.0180. Figure 5–49 shows the register.

Figure 5–49 General-Purpose Output Register



This register is a general-purpose output register. The values in this register can be used for a special purpose. The data is converted to a bit stream and shifted out of the 21174. It is up to the hardware designer to provide the proper external hardware to support this register.

Table 5–58 describes the general-purpose output register fields.

Table 5–58 General-Purpose Output Register Fields

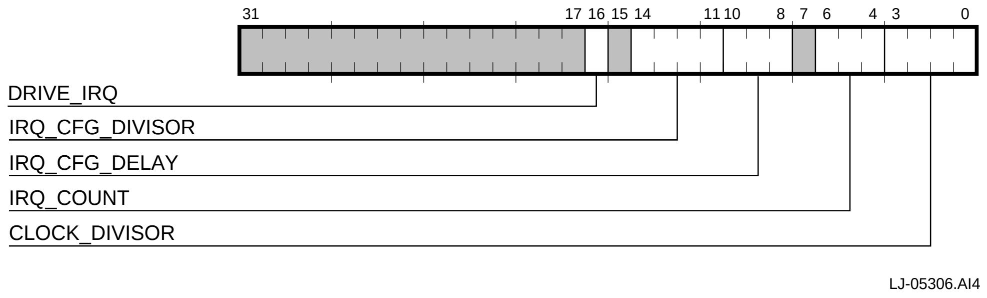
Name	Extent	Access	Init	Description
GPO	<63:0>	WO	X	General-purpose output

5.9.6 Interrupt Configuration Register (INT_CNFG)

The interrupt configuration register access is RW to address 87.A000.01C0. Figure 5–50 shows the register.

Interrupt Control Registers Descriptions

Figure 5–50 Interrupt Configuration Register



This register is used to determine the behavior of the interrupt request register. The three fields determine the state of the IRQ lines, the number of IRQ, and the clock divisor value.

Table 5–59 describes the interrupt configuration register fields.

Table 5–59 Interrupt Configuration Register Fields *(Sheet 1 of 2)*

Name	Extent	Access	Init	Description
CLOCK_DIVISOR	<3:0>	RW	0	This value + 1 represents the clock divisor value for the external shift register. The clock presented to the external logic is the 21174 clock divided by this value. A value of zero disables this operation.
IRQ_COUNT	<6:4>	RW	3	This value + 1 is the size of the external shift register in multiples of 8.
Reserved	<7>	RO	0	—
IRQ_CFG_DELAY	<10:8>	RW	X	This field shows the state of the IRQ pins (going to the 21164). The initial value is present when **dc_ok** is asserted and **sys_reset_l** is deasserted. The initial value controls the delay between **sys_clk_out1_h** and **sys_clk_out2_h,** as shown in Table 5–60. After the delay value has been obtained from this field, the field can be changed to alter the speed of the 21164. This field should only be changed by the startup code as the system reset cycle must be initiated in order for the change to take effect.

Table 5–59 Interrupt Configuration Register Fields *(Sheet 2 of 2)*

Name	Extent	Access	Init	Description
IRQ_CFG_DIVISOR	<14:11>	RW	X	This field shows the state of the IRQ pins (going to the 21164). The initial value is present when **dc_ok** is asserted and **sys_reset_l** is deasserted. The initial value controls the system clock divider ratio as shown in Table 5–61. After the system clock divider ratio has been obtained from this field, the field can be changed to alter the speed of the 21164. This field should only be changed by the startup code as the system reset cycle must be initiated in order for the change to take effect.
Reserved	<15>	RO	0	—
DRIVE_IRQ	<16>	RW	0	Forces the 21174 to drive the IRQ lines to the 21164, so that the next time reset is asserted the value on the IRQ lines will be taken from the value written to IRQ_CFG bits located in this register.
Reserved	<31:17>	RO	0	—

Table 5–60 lists the contents of the IRQ_CFG_DELAY field of the interrupt configuration register.

Table 5–60 Clock Delay Values

Bit 10 (HALT_IRQ)	Bit 9 (MCHK_IRQ)	Bit 8 (PWR_FAIL_IRQ)	Delay Cycles
0	0	0	0
0	0	1	2
0	1	0	4
0	1	1	6
1	0	0	1
1	0	1	3
1	1	0	5
1	1	1	7

Table 5–61 lists the contents of the IRQ_CFG_DIVISOR field of the interrupt configuration register.

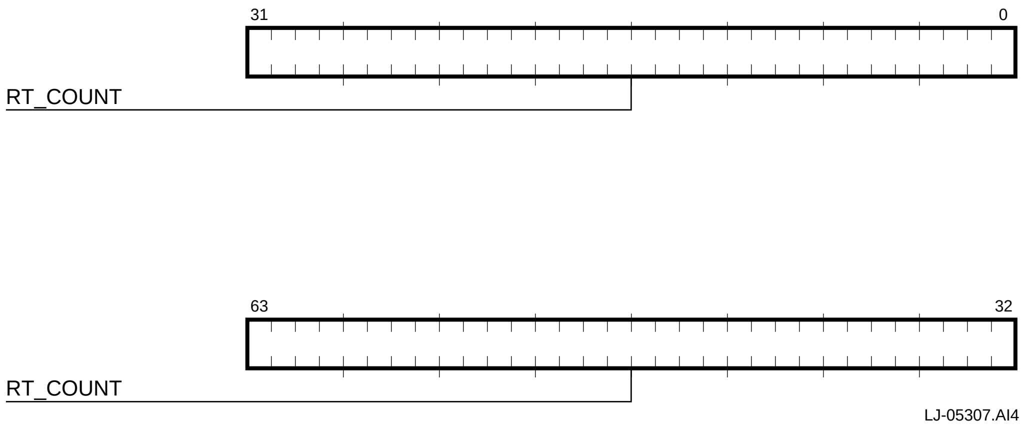
Table 5–61 Clock Divisor Values

Bit 14 (IRQ[3])	Bit 13 (IRQ[2])	Bit 12 (IRQ[1])	Bit 11 (IRQ[0])	System Clock Divisor
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

5.9.7 Real-Time Counter Register (RT_COUNT)

The real-time counter register access is RW to address 87.A000.0200. Figure 5–51 shows the register.

Figure 5–51 Real-Time Counter Register



This register contains a free-running clock that is incremented once for every **sys_clk** cycle. This register is initialized when power is turned on to 0 and can be written with any value.

Table 5–62 describes the real-time counter register fields.

Table 5–62 Real-Time Counter Register Fields

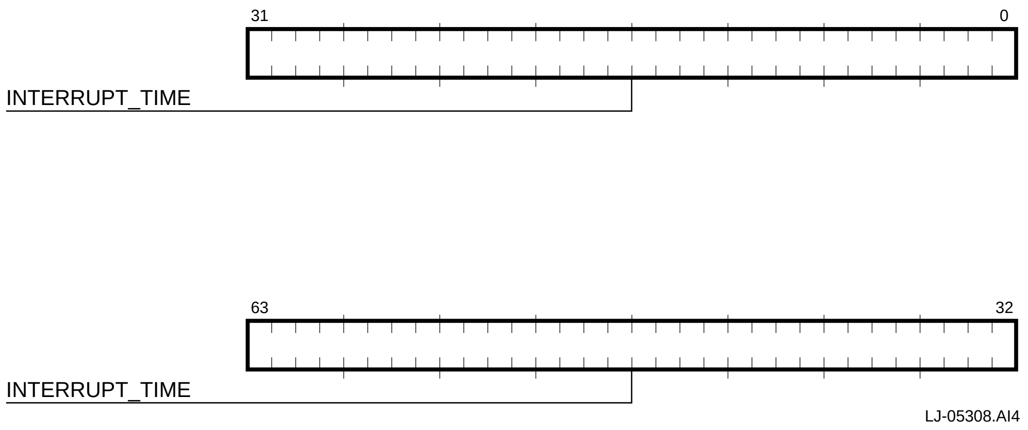
Name	Extent	Access	Init	Description
RT_COUNT	<63:0>	RW	0	Current clock value

5.9.8 Interrupt Time Register (INT_TIME)

The interrupt time register access is RW to address 87.A000.0240. Figure 5–52 shows the register.

Interrupt Control Registers Descriptions

Figure 5–52 Interrupt Time Register



The interrupt time register determines the cycle count at which an interrupt to the 21164 will be generated. When the real-time counter register matches this value, a clock interrupt to the 21164 will be generated if the interrupt is enabled by the timer control register.

Table 5–63 describes the interrupt time register fields.

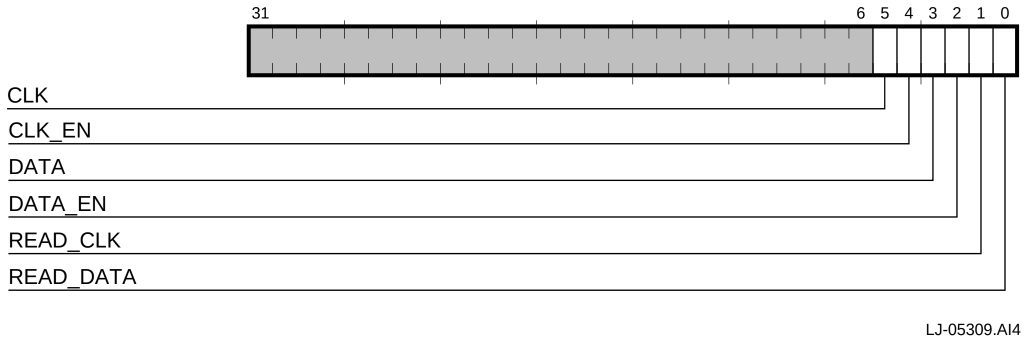
Table 5–63 Interrupt Time Register Fields

Name	Extent	Access	Init	Description
INTERRUPT_TIME	<63:0>	RW	0	Value at which a clock interrupt will be generated

5.9.9 I^{2}C Control Register (IIC_CTRL)

The I^{2}C control register access is RW to address 87.A000.02C0. Figure 5–53 shows the register.

Figure 5–53 I^{2}C Control Register



The I^{2}C control register is used to access the I^{2}C interface on the memory modules. With careful programming, the data can be obtained from the DIMMs to configure the memory system.

Table 5–64 describes the I^{2}C control register fields.

Table 5–64 I^{2}C Control Register Fields

Name	Extent	Access	Init	Description
READ_DATA	<0>	RO	0	Current state of the read data pin.
READ_CLK	<1>	RO	0	Current state of clock pin.
DATA_EN	<2>	WO	0	1 – Enable the data out to the pins. This causes the value in this register's DATA field to be driven onto the **dimm_sda** signal line.
DATA	<3>	WO	0	Data to be driven on the **dimm_sda** signal line.
CLK_EN	<4>	WO	0	1 – Enable the clock out to the pins. This causes the value in this register's CLK field to be driven onto the **dimm_scl** signal line.
CLK	<5>	WO	0	Clock data.
Reserved	<31:6>	—	—	—

6 System Address Space

This chapter describes the mapping of 21164 40-bit physical addresses to memory and I/O space addresses. It also describes the translation of a 21164-initiated address (**addr_h<39:4>**) into a PCI address (**ad<63:0>**) and the translation of a PCI-initiated address into a physical memory address.

PCI addressing topics include dense and sparse address space and scatter-gather address translation for DMA operations.

6.1 Address Map

The system address mapping operates with byte/word transactions enabled or disabled. Byte/word operation is controlled by PYXIS_CTRL1<0> (IOA_BEN). Table 6–1 shows system address mapping operations when IOA_BEN equals 0 (byte/word operation disabled).

Table 6–1 Physical Address Map (Byte/Word Mode Disabled) *(Sheet 1 of 2)*

21164 Address[1]	Size (GB)	Selection
00.000.0000 – 01.FFFF.FFFF	8.00	Main memory
E.0000.0000 – E.FFFF.FFFF	4.00	Dummy memory region
80.0000.0000 – 83.FFFF.FFFF	16.00	PCI sparse memory region 0, 512MB
84.0000.0000 – 84.FFFF.FFFF	4.00	PCI sparse memory region 1, 128MB
85.0000.0000 – 85.7FFF.FFFF	2.00	PCI sparse memory region 2, 64MB
85.8000.0000 – 85.BFFF.FFFF	1.00	PCI sparse I/O space region A, 32MB
85.C000.0000 – 85.FFFF.FFFF	1.00	PCI sparse I/O space region B, 32MB
86.0000.0000 – 86.FFFF.FFFF	4.00	PCI dense memory
87.0000.0000 – 87.1FFF.FFFF	0.50	PCI sparse configuration space

Table 6–1 Physical Address Map (Byte/Word Mode Disabled) *(Sheet 2 of 2)*

21164 Address[1]	Size (GB)	Selection
87.2000.0000 – 87.3FFF.FFFF	0.50	PCI special/interrupt acknowledge
87.4000.0000 – 87.4FFF.FFFF	0.25	21174 main CSRs
87.5000.0000 – 87.5FFF.FFFF	0.25	21174 memory control CSRs
87.6000.0000 – 87.6FFF.FFFF	0.25	21174 PCI address translation
87.7000.0000 – 87.7FFF.FFFF	0.25	Reserved
87.8000.0000 – 87.8FFF.FFFF	0.25	21174 miscellaneous CSRs
87.9000.0000 – 87.9FFF.FFFF	0.25	21174 power management CSRs
87.A000.0000 – 87.AFFF.FFFF	0.25	21174 interrupt control CSRs
87.B000.0000 – 87.FFFF.FFFF	1.25	Reserved

[1] All addresses in the range of 80.0000.0000 and 8F.FFFF.FFFF are aliased. Address bits 36 through 38 are ignored in the address.

Table 6–2 shows system address mapping operations when IOA_BEN equals 1 (byte/word operation enabled).

Table 6–2 Physical Address Map (Byte/Word Mode Enabled) *(Sheet 1 of 2)*

21164 Address	Size (GB)	Selection
00.000.0000 – 01.FFFF.FFFF	8.00	Main memory
E.0000.0000 – E.FFFF.FFFF	4.00	Dummy memory region
80.0000.0000 – 83.FFFF.FFFF	16.00	PCI sparse memory region 0, 512MB
84.0000.0000 – 84.FFFF.FFFF	4.00	PCI sparse memory region 1, 128MB
85.0000.0000 – 85.7FFF.FFFF	2.00	PCI sparse memory region 2, 64MB
85.8000.0000 – 85.BFFF.FFFF	1.00	PCI sparse I/O space region A, 32MB
85.C000.0000 – 85.FFFF.FFFF	1.00	PCI sparse I/O space region B, 32MB
86.0000.0000 – 86.FFFF.FFFF	4.00	PCI dense memory
87.0000.0000 – 87.1FFF.FFFF	0.50	PCI sparse configuration space
87.2000.0000 – 87.3FFF.FFFF	0.50	PCI special/interrupt acknowledge
87.4000.0000 – 87.4FFF.FFFF	0.25	21174 main CSRs
87.5000.0000 – 87.5FFF.FFFF	0.25	21174 memory control CSRs

Table 6–2 Physical Address Map (Byte/Word Mode Enabled) *(Sheet 2 of 2)*

21164 Address	Size (GB)	Selection
87.6000.0000 – 87.6FFF.FFFF	0.25	21174 PCI address translation
87.7000.0000 – 87.7FFF.FFFF	0.25	Reserved
87.8000.0000 – 87.8FFF.FFFF	0.25	21174 miscellaneous CSRs
87.9000.0000 – 87.9FFF.FFFF	0.25	21174 power management CSRs
87.A000.0000 – 87.AFFF.FFFF	0.25	21174 interrupt control CSRs
87.B000.0000 – 87.BFFF.FFFF	0.25	Reserved
88.0000.0000 – 88.FFFF.FFFF	4.00	PCI memory space INT8
98.0000.0000 – 98.FFFF.FFFF[1]	4.00	PCI memory space INT4
A8.0000.0000 – A8.FFFF.FFFF[1]	4.00	PCI memory space INT2
B8.0000.0000 – B8.FFFF.FFFF[1]	4.00	PCI memory space INT1
89.0000.0000 – 89.FFFF.FFFF	4.00	PCI I/O space INT8
99.0000.0000 – 99.FFFF.FFFF[1]	4.00	PCI I/O space INT4
A9.0000.0000 – A9.FFFF.FFFF[1]	4.00	PCI I/O space INT2
B9.0000.0000 – B9.FFFF.FFFF[1]	4.00	PCI I/O space INT1
8A.0000.0000 – 8A.FFFF.FFFF	4.00	PCI configuration space, type 0, INT8
9A.0000.0000 – 9A.FFFF.FFFF[1]	4.00	PCI configuration space, type 0, INT4
AA.0000.0000 – AA.FFFF.FFFF[1]	4.00	PCI configuration space, type 0, INT2
BA.0000.0000 – BA.FFFF.FFFF[1]	4.00	PCI configuration space, type 0, INT1
8B.0000.0000 – 8B.FFFF.FFFF	4.00	PCI configuration space, type 1, INT8
9B.0000.0000 – 9B.FFFF.FFFF[1]	4.00	PCI configuration space, type 1, INT4
AB.0000.0000 – AB.FFFF.FFFF[1]	4.00	PCI configuration space, type 1, INT2
BB.0000.0000 – BB.FFFF.FFFF[1]	4.00	PCI configuration space, type 1, INT1
C7.C000.0000 – C7.FFFF.FFFF[2]	1.00	Flash ROM read/write space

[1] Address bits 37 and 38 are generated by the 21164 and not by software. These address bits are used by the 21164 to indicate to external hardware that this transaction is a byte, word, longword, or quadword operation.

[2] Read/write transactions to flash ROM must be done with byte transactions to address range 87.C000.0000 through 87.FFFF.FFFF. All other transaction types will produce UNDEFINED results.

Address Map

The 21164 address space is divided into two regions using physical address <39>:

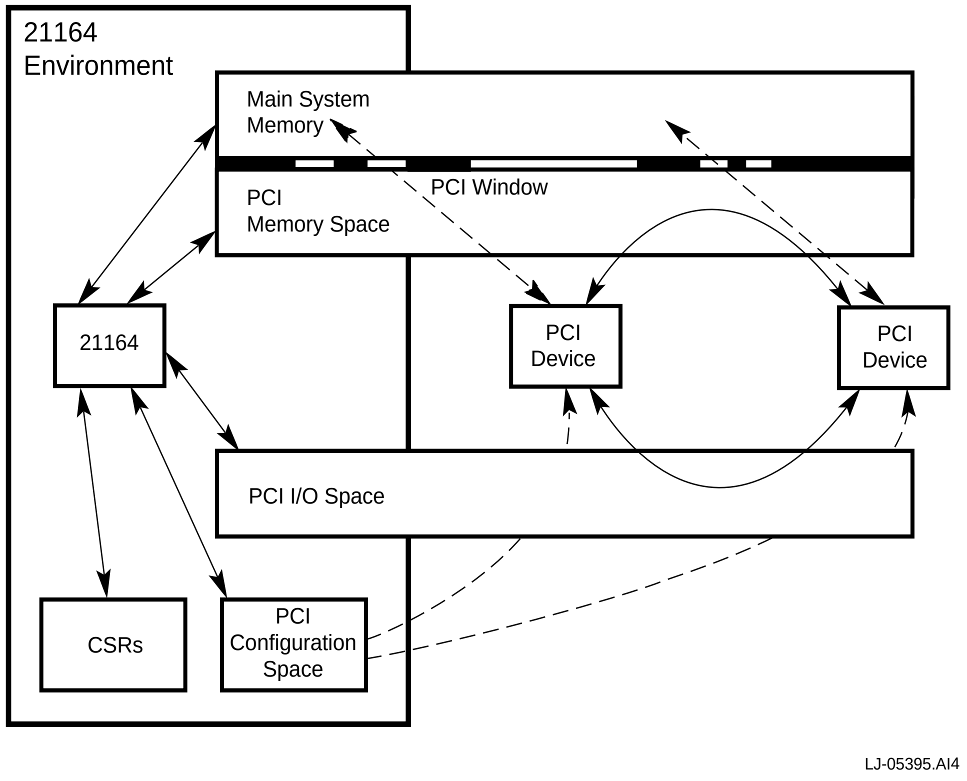
- 0 – 21164 access is to the cached memory space.
- 1 – 21164 access is to noncached space. This noncached space is used to access memory-mapped I/O devices. Mailboxes are not supported.

The noncached space contains the CSRs, noncached memory space (for diagnostics), and the PCI address space. The PCI defines three physical address spaces: a 64-bit PCI memory space, a 4GB PCI I/O space, and a 256 byte-per-device PCI configuration space. In addition to these three address spaces on the PCI, the 21164's noncached space is also used to generate PCI interrupt acknowledge and special cycles.

The 21164 has visibility to the complete address space. It can access the cached memory region, the CSR region, the PCI memory region, the PCI I/O region, and the configuration regions (see Figure 6–1).

The PCI devices have a restricted view of the address space. They can access any PCI device through the PCI memory space or the PCI I/O space; but they have no access to the PCI configuration space. The system restricts access to the system memory (for DMA operations) to the use of five programmable windows in the PCI memory space (see Figure 6–1).

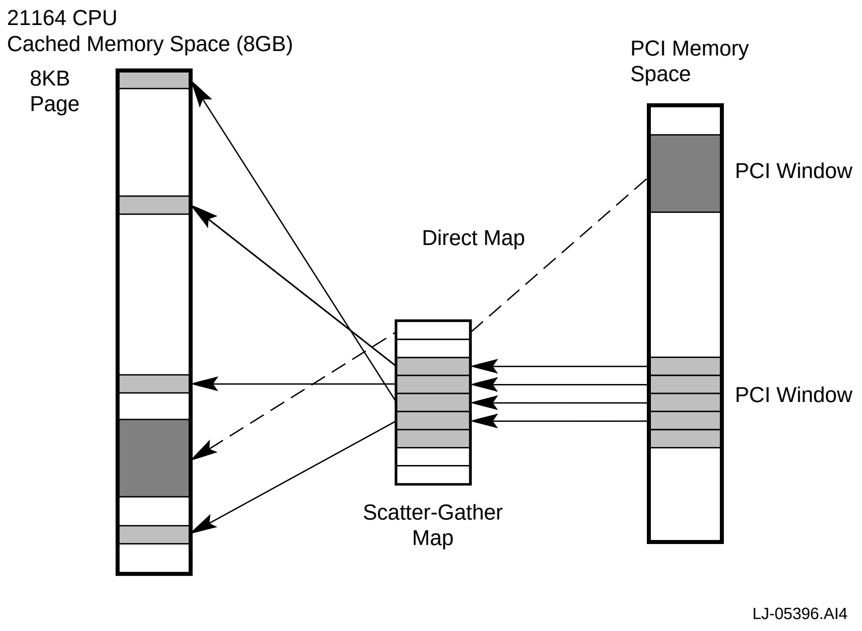
Figure 6–1 Address Space Overview



DMA access to the system memory is achieved using windows in one of the following three ways:

- Directly, using the “Monster Window” with dual-address cycles (DAC), where **ad<33:0>** equals **addr_h<33:0>**.
- Directly-mapped, by concatenating an offset to a portion of the PCI address.
- Virtually, through a scatter-gather translation map. The scatter-gather map allows any 8KB page of PCI memory address region to be redirected to any 8KB cached memory page, as shown in Figure 6–2.

Figure 6–2 Memory Remapping



6.2 PCI Address Space

The system generates 32-bit PCI addresses but accepts both 64-bit address (DAC[1]) cycles and 32-bit PCI address (SAC[2]) cycles. Accessing main memory is as follows:

- Window 4, the “Monster Window,” provides full access to main memory. It is accessed by DAC only with **ad<40>** equal to 1. Memory address **addr_h<33:0>** equals PCI address **ad<33:0>**.
- Window 3 can be either DAC or SAC, but not both. If DAC, **ad<63:40>** must be zero, **ad<39:32>** must match the DAC register, and **ad<31:0>** must hit in window 3.
- Windows 0, 1, and 2 are SAC-only.

[1] Dual-address cycle (PCI 64-bit address transfer) requires that address bits <63:32> contain a nonzero value.

[2] Single-address cycle (PCI 32-bit address transfer) requires that address bits <63:32> contain a value of zero.

6.3 21164 Address Space

Figure 6–3 shows an overview of the 21164 address space. Figure 6–4 shows how the 21164 address map translates to the PCI address space and how PCI devices access the 21164 memory space using DMA transactions. The PCI memory space is double mapped via dense and sparse space.

The 21164 I/O address map has the following characteristics:

- Provides 4GB of dense[3] address space to completely map the 32-bit PCI memory space.
- Provides abundant PCI sparse[3] memory address space because sparse-space regions have byte granularity and is the safest memory space to use (that is, no prefetching). Furthermore, the larger the space the less likely software will need to dynamically relocate the sparse-space segments. The main problem with sparse space is that it wastes 21164 address space (for example, 16GB of 21164 address space maps to 512MB of PCI sparse space).

  The system provides three PCI sparse-space memory regions, allowing 704MB of total sparse-space memory. The three regions are relocatable using the HAE_MEM CSR. The simplest configuration allows for 704MB of contiguous memory space.

  – 512MB region, which may be located in any naturally aligned 512MB segment of the PCI memory space. Software programmers may find this region sufficient for their needs and can ignore the remaining two regions.

  – 128MB regions, which may be located on any naturally aligned 128MB segment of the PCI memory space.

  – 64MB region, which may be located on any naturally aligned 64MB segment of the PCI memory space.
- Limits the PCI I/O space to sparse space. Although the PCI I/O space can handle 4GB, most PCI devices will not exceed 64KB for the foreseeable future. The system provides 64MB of sparse I/O space because address decoding is faster.
- Provides two PCI I/O sparse-space regions: region A, which is 32MB and is fixed in PCI segment 0–32MB; and region B, which is also 32MB, but is relocatable using the HAE_IO register.

[3] Dense and sparse space address space are described later in this chapter.

21164 Address Space

Figure 6–3 21164 Address Space Configuration

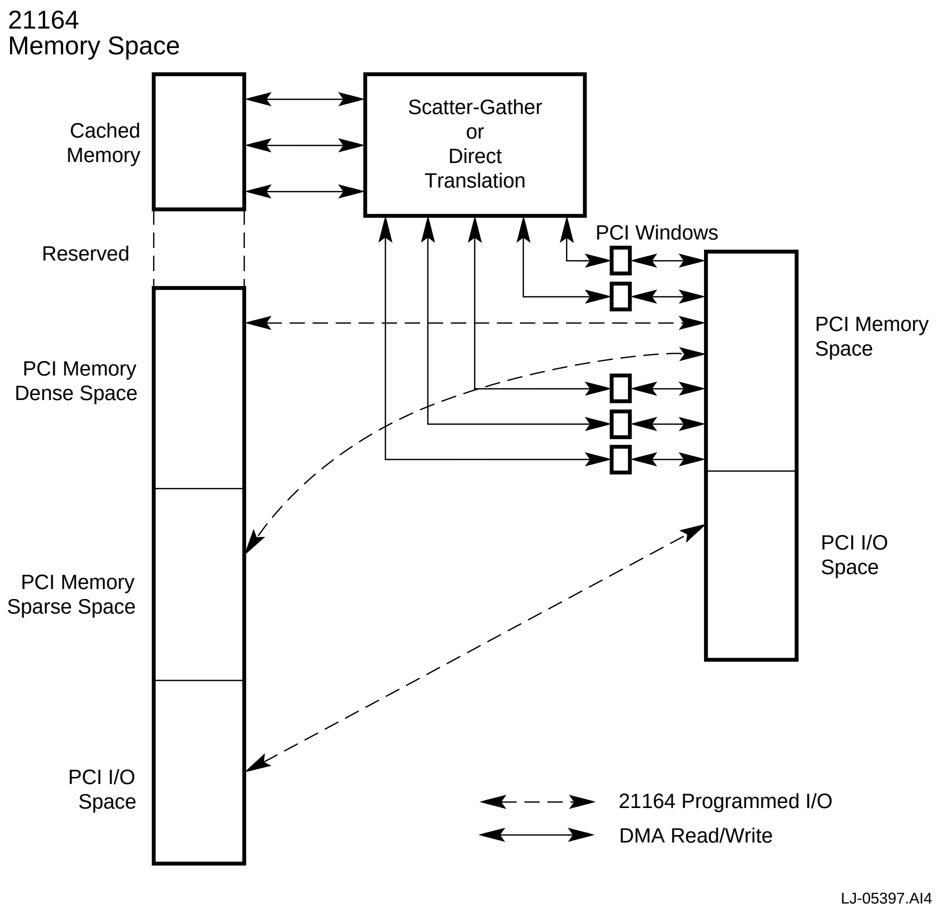
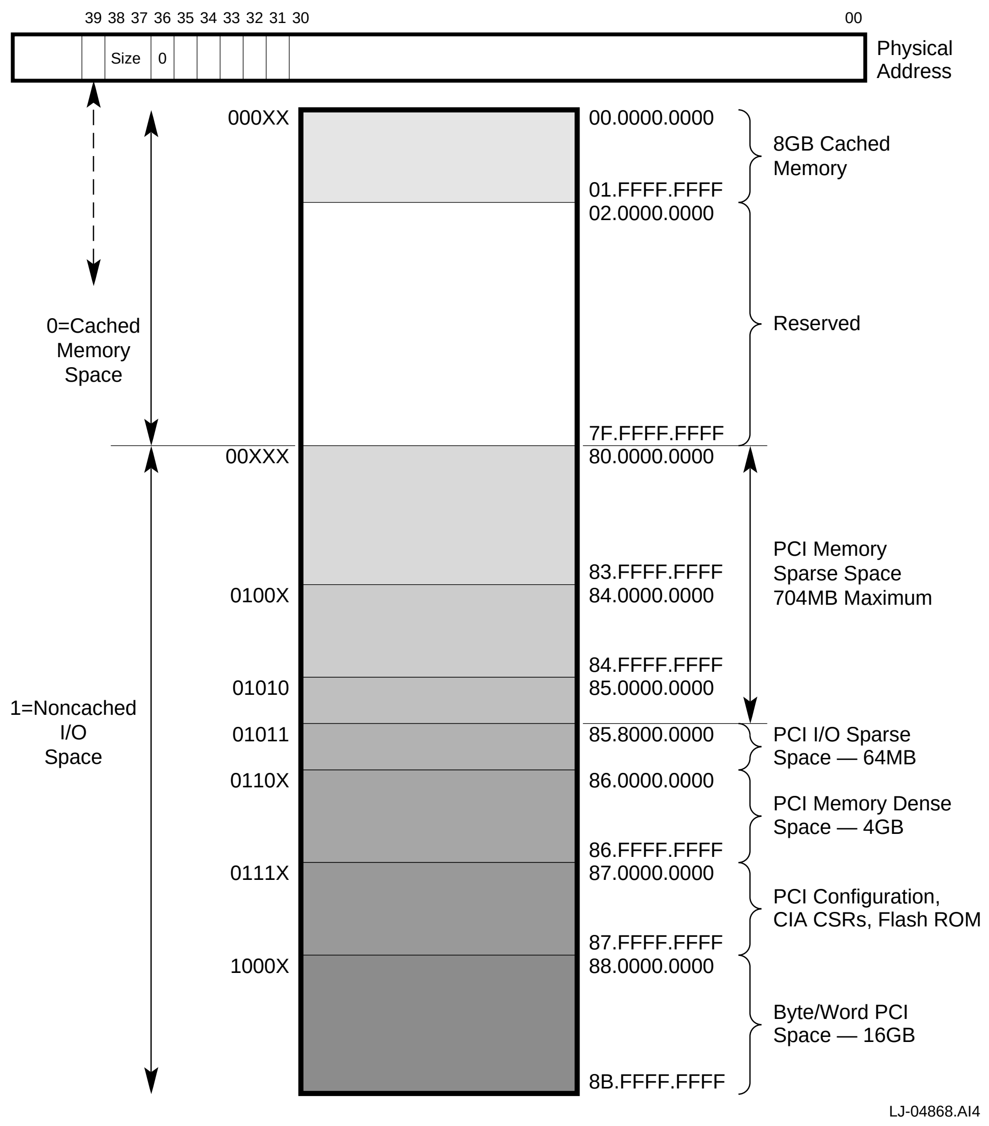


Figure 6–4 21164 and DMA Read and Write Transactions



6.3.1 System Address Map

Figure 6–5 shows the following system address regions:

- Main memory address space contains 8GB. All transactions contain 64 bytes, are cache-block aligned, and are placed in cache by the 21164. Both Istream and Dstream transactions access this address space.
- PCI sparse-space memory region 1 contains 512MB. Noncached 21164 read/write transactions are allowed, including byte, word, tribyte, longword (LW), and quadword (QW) types. There is no read prefetching.
- PCI sparse-space memory region 2 contains 128MB.
- PCI sparse-space memory region 3 contains 64MB.
- PCI I/O sparse-space memory region A contains 32MB and is not relocatable.
- PCI I/O sparse-space memory region B contains 32MB and is relocatable by way of the HAE_IO register.
- PCI dense memory space contains 4GB for 21164 noncached 21164 transactions. It is used for devices with access granularity greater or equal to a LW. Read prefetching is allowed, and thus read transactions can have no side effects.
- The PCI configuration space is used for noncached 21164 access. Sparse-space read/write transactions are allowed, including byte, word, tribyte, LW, and QW types. Prefetching of read data is not allowed.

  Figure 6–6 shows a detailed view of PCI configuration space that includes 21174 CSRs. The 21174 CSR address space is chosen for hardware convenience.

Figure 6–5 System Address Map

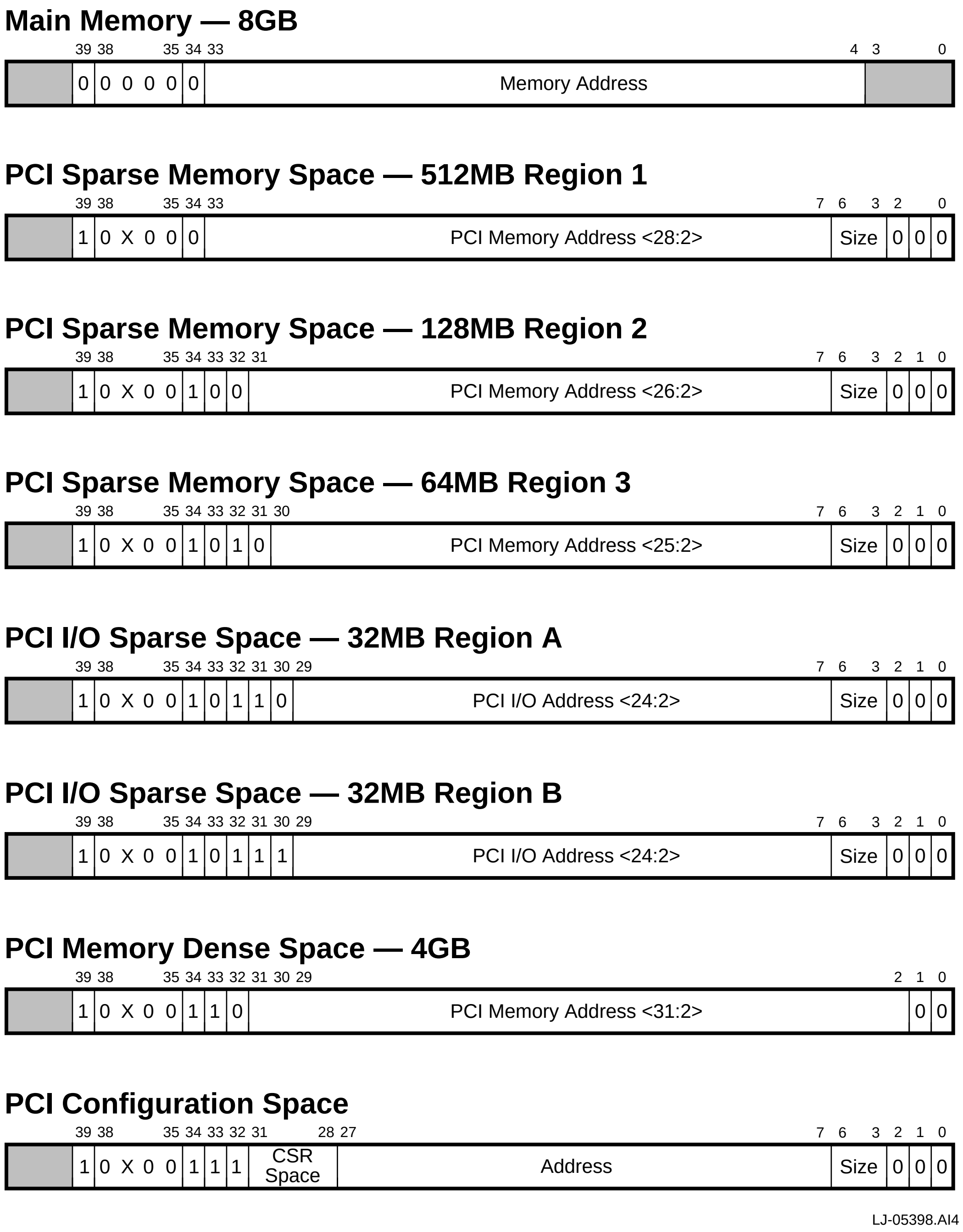
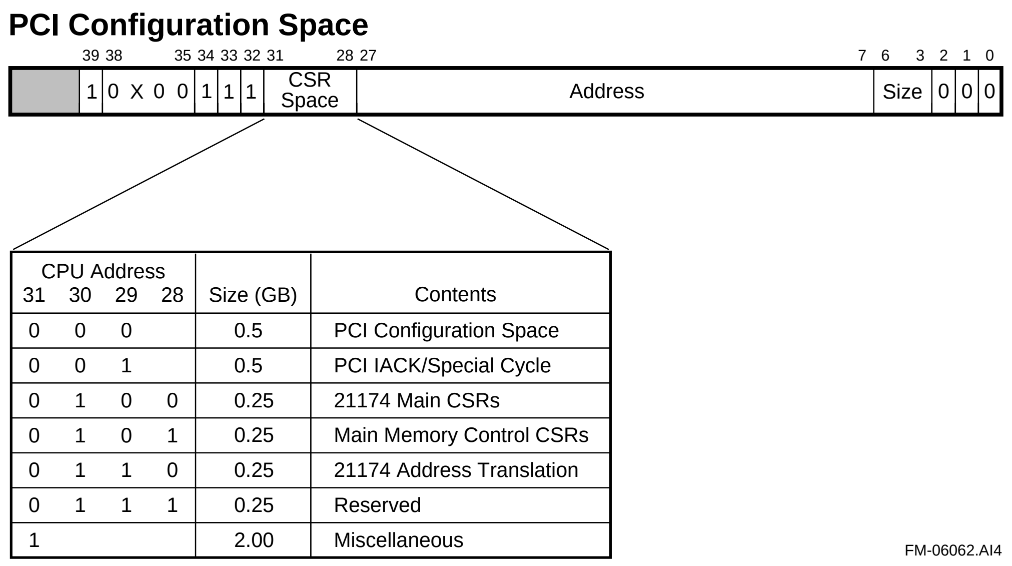

Main Memory — 8GB
39 38 35 34 33 4 3 0
0 0 0 0 0 0 Memory Address
PCI Sparse Memory Space — 512MB Region 1
39 38 35 34 33 7 6 3 2 0
1 0 X 0 0 0 PCI Memory Address <28:2> Size 0 0 0
PCI Sparse Memory Space — 128MB Region 2
39 38 35 34 33 32 31 7 6 3 2 1 0
1 0 X 0 0 1 0 0 PCI Memory Address <26:2> Size 0 0 0
PCI Sparse Memory Space — 64MB Region 3
39 38 35 34 33 32 31 30 7 6 3 2 1 0
1 0 X 0 0 1 0 1 0 PCI Memory Address <25:2> Size 0 0 0
PCI I/O Sparse Space — 32MB Region A
39 38 35 34 33 32 31 30 29 7 6 3 2 1 0
1 0 X 0 0 1 0 1 1 0 PCI I/O Address <24:2> Size 0 0 0
PCI I/O Sparse Space — 32MB Region B
39 38 35 34 33 32 31 30 29 7 6 3 2 1 0
1 0 X 0 0 1 0 1 1 1 PCI I/O Address <24:2> Size 0 0 0
PCI Memory Dense Space — 4GB
39 38 35 34 33 32 31 30 29 2 1 0
1 0 X 0 0 1 1 0 PCI Memory Address <31:2> 0 0
PCI Configuration Space
39 38 35 34 33 32 31 28 27 7 6 3 2 1 0
1 0 X 0 0 1 1 1 CSR Space Address Size 0 0 0
LJ-05398.AI4

Figure 6–6 21174 CSR Space

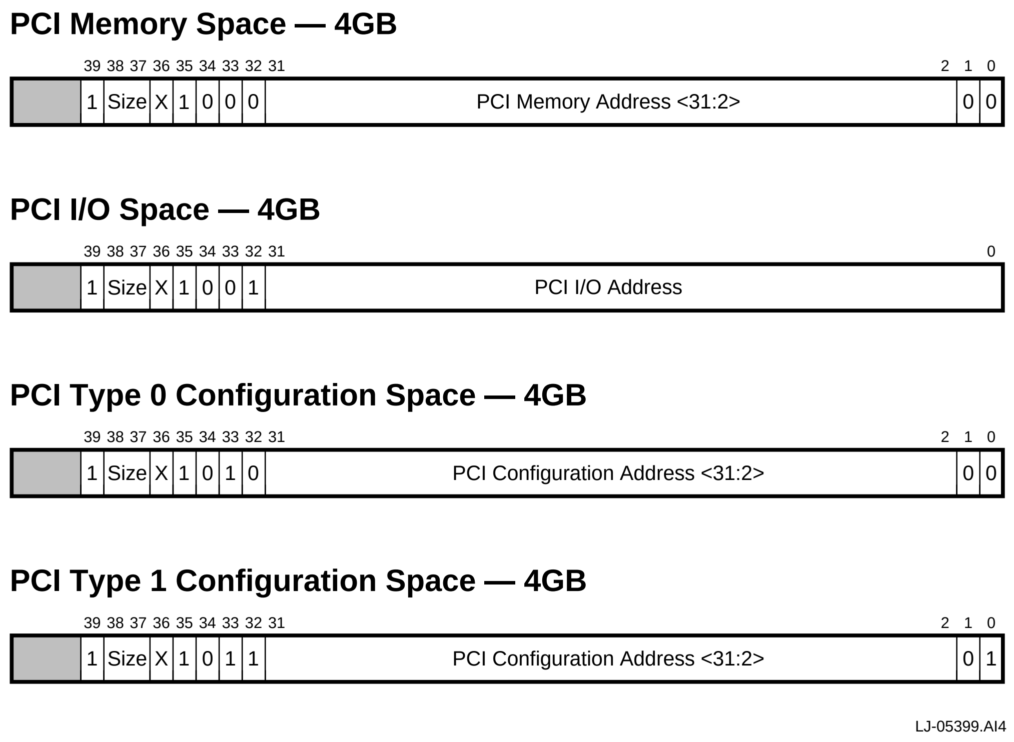


CPU Address 31	30	29	28	Size (GB)	Contents
0	0	0		0.5	PCI Configuration Space
0	0	1		0.5	PCI IACK/Special Cycle
0	1	0	0	0.25	21174 Main CSRs
0	1	0	1	0.25	Main Memory Control CSRs
0	1	1	0	0.25	21174 Address Translation
0	1	1	1	0.25	Reserved
1				2.00	Miscellaneous

6.4 21164 Byte/Word PCI Space

The 21164 supports byte/word instructions that allow software to perform byte granularity transactions to and from I/O space without using sparse address space. This space is divided into four regions: memory, I/O, configuration – type 0, and configuration – type 1, as shown in Figure 6–7.

Figure 6–7 Byte/Word PCI Space


PCI Memory Space — 4GB
39 38 37 36 35 34 33 32 31 2 1 0
1 Size X 1 0 0 0 PCI Memory Address <31:2> 0 0
PCI I/O Space — 4GB
39 38 37 36 35 34 33 32 31 0
1 Size X 1 0 0 1 PCI I/O Address
PCI Type 0 Configuration Space — 4GB
39 38 37 36 35 34 33 32 31 2 1 0
1 Size X 1 0 1 0 PCI Configuration Address <31:2> 0 0
PCI Type 1 Configuration Space — 4GB
39 38 37 36 35 34 33 32 31 2 1 0
1 Size X 1 0 1 1 PCI Configuration Address <31:2> 0 1
LJ-05399.AI4

Operations are the same for the four regions. The 21164 will issue a single byte/word read or write transaction for PCI byte and word instructions. The 21164 will not pack longword load instructions. The 21164 can pack up to eight longword store instructions for a single 32-byte block into one transaction. Up to four quadword instructions can also be packed to the same 32-byte block. Byte/word support is enabled when 21164 IPR register ICSR<17> equals 1 and when 21174 CSR register PYXIS_CTRL1<0> also equals 1.

21164 Byte/Word PCI Space

Table 6–3 shows noncached 21164 addresses when byte/word support is enabled.

Table 6–3 21164 Byte/Word Addressing

	addr_h	int4_valid			
Instruction	**<38:37>**	**<3>**	**<2>**	**<1>**	**<0>**
LDQ	00	INT8	—	—	—
LDL	01	**addr_h<3:2>**	—	Undefined	—
LDWU	10	**addr_h<3:1>**	—	—	Undefined
LDBU	11	**addr_h<3:0>**	—	—	—
STQ	00	INT4 Mask	—	—	—
STL	01	INT4 Mask	—	—	—
STW	10	**addr_h<3:1>**	—	—	Undefined
STB	11	**addr_h<3:0>**	—	—	—

6.4.1 21164 Size Field

Table 6–4 shows the calculation of the 21164 size field.

Table 6–4 21164 Byte/Word Translation Values

Size<38:37>	Data Size
00	INT8 (Quadword — 8 bytes, 64 bits)
01	INT4 (Longword — 4 bytes, 32 bits)
10	INT2 (Word — 2 bytes, 16 bits)
11	INT1 (Byte — 1 byte, 8 bits)

The following transactions use single data transfers on the PCI:

- INT1 and INT2 read and write transactions
- INT4 read transactions

The following transactions have multiple data transfers on the PCI:

- INT4 write transactions
- INT8 read and write transactions

6.5 Cacheable Memory Space

Cacheable memory space is located in the range 00.0000.0000 to 01.FFFF.FFFF. The 21174 recognizes the first 8GB to be in cacheable memory space. The block size is fixed at 64 bytes. Read and flush commands to the 21164 caches occur for DMA traffic.

6.6 PCI Dense Memory Space

PCI dense memory address space is located in the range 86.0000.0000 to 86.FFFF.FFFF. This address space is typically used for memory-like data buffers such as a video frame buffer or a nonvolatile RAM (NVRAM). Dense space does not allow byte or word access, but has the following advantages over sparse space:

- Contiguous locations — Some software, such as the default graphics routines of the Windows NT operating system, requires memory-like transactions. These routines cannot use sparse-space addresses, because they require transactions on the PCI bus to be at adjacent 21164 addresses, instead of being widely separated as in sparse space. As a result, if the user-mode driver manipulates its frame buffer in sparse space, it cannot hand over the buffer to the common Windows NT operating system graphics code.
- Higher bus bandwidth — PCI bus burst transfers are not usable in sparse space except for a 2-longword burst for quadword write transactions. Dense space is defined to allow both burst read and write transactions.
- Efficient read/write buffering — In sparse space, separate transactions use separate read or write buffer entries. Dense space allows separate transactions to be collapsed in read and write buffers (as the 21164 does).
- Few memory barriers (MBs) — In general, sparse-space transactions are separated by MB instructions to avoid read/write buffer collapsing. Dense-space transactions only require barriers when explicit ordering is required by the software.

Dense space is provided for the 21164 to access PCI memory space, not for access to PCI I/O space. Dense space has the following characteristics:

- It holds a one-to-one mapping between 21164 addresses and PCI addresses. A longword address from the 21164 will map to a longword on the PCI with no shifting of the address field. Hence, the term dense space. Sparse space, on the other hand, maps a large piece of 21164 memory space (32 bytes) to a small piece (such as a byte) on the PCI.

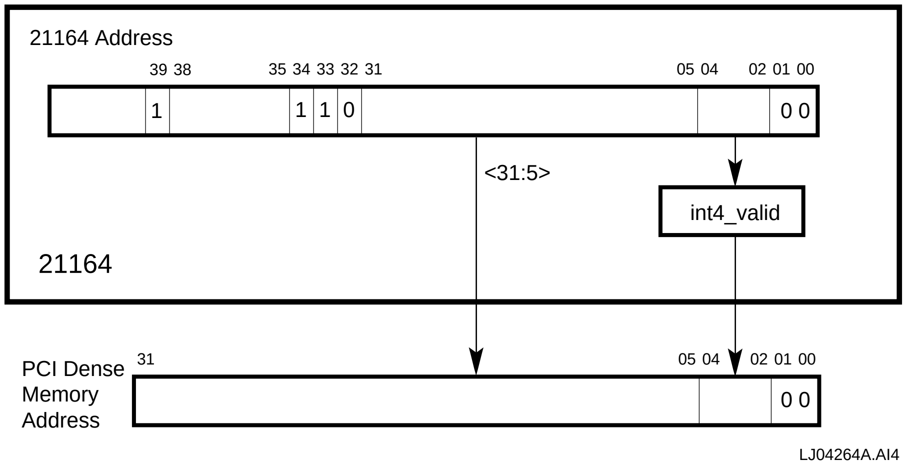
PCI Dense Memory Space

- The concept of dense space (and sparse space) is applicable only to a 21164-generated address. There is no such thing as dense space (or sparse space) for a PCI generated address.
- Byte or word transactions are not possible in dense space. The minimum access granularity is a longword on write transactions and a quadword on read transactions. The maximum transfer length is 32 bytes (performed as a burst of eight longwords on the PCI). Any combination of longwords may be valid on write transactions. Valid longwords surrounding an invalid longword(s) (called a hole) are required to be handled correctly by all PCI devices. The 21174 will allow such holes to be issued.
- Read transactions will always be performed as a burst of two or more longwords on the PCI because the minimum granularity is a quadword. The 21164 can request a longword but the 21174 will always fetch a quadword, thus prefetching a second longword. Therefore, this space cannot be used for devices that have read side effects. Although a longword may be prefetched, the prefetch buffer is not treated as a cache and so coherency is not an issue. A quadword read transaction is not atomic on the PCI; that is, the target device is at liberty to force a retry after the first longword of data is sent, and then to allow another PCI device to take control of the PCI bus[4].
- The 21164 merges noncached reads of up to 32 bytes maximum. The largest dense-space read transaction is 32 bytes from the PCI bus.
- Write transactions to dense space are buffered in the 21164 chip. The 21174 supports a burst length of 8 on the PCI, corresponding to 32 bytes of data. Also, the 21174 provides four 32-byte write buffers to maximize I/O write transaction performance. These four buffers are strictly ordered. Write transactions are sent out on the bus in the order that they were received from the 21164. Avoid write buffer merging and use memory barrier (MB) and write memory barrier (WMB) instructions carefully.

[4] The 21174 does not drive the PCI lock signal and this cannot ensure atomicity. This is true of all current Alpha microprocessors.

Figure 6–8 shows dense-space address generation.

Figure 6–8 Dense-Space Address Generation



The following list describes address generation in dense space:

- **addr_h<31:5>** value is sent directly out on **ad<31:5>**.
- **addr_h<4:2>** is not sent out by the 21164 and instead is inferred from the **int4_valid<3:0>**.
- **ad<4:3>** is a copy of **addr_h<4:3>**.
- **ad<2>** differs for read and write transactions as follows:
  - For a read transaction, **ad<2>** is zero (that is, the minimum read transaction resolution in noncached space is a quadword).
  - For a write transaction, **ad<2>** equals **addr_h<2>**.

6.7 PCI Sparse Memory Space

The system provides three regions of contiguous 21164 address space that maps to PCI sparse memory space. The total 21164 range is from 80.0000.0000 to 85.7FFF.FFFF.

6.7.1 Hardware Extension Register (HAE_MEM)

In sparse space, **addr_h<7:3>** are used to encode byte enable bits, size bits and the low-order PCI address, **ad<2:0>**. This means that there are now five fewer address bits available to generate the PCI physical address.

The system provides three sparse-space PCI memory regions and allows all three sparse-space regions to be relocated by way of bits in the HAE_MEM register. This provides software with great flexibility.

6.7.2 Memory Access Rules and Operation

The Alpha instruction set can express only aligned longword and quadword data references. The PCI bus requires the ability to express byte, word, tribyte, longword (double word), and quadword references. Intel processors are capable of generating unaligned references, so the 21174 should be able to emulate the resulting PCI transactions to ensure compatibility with PCI devices designed for Intel systems.

The size of the data transfer (byte, word, tribyte, longword, or quadword) and the byte enables are encoded in the 21164 address. The 21164 signals **addr_h<6:3>** are used for this purpose, leaving the remaining **addr_h<31:7>** signals to generate a PCI longword address <26:3>[5]. This loss of address bits has resulted in a 21164 22GB sparse 32-bit address space that maps to only 704MB of address space on the PCI.

The rules for accessing sparse space are as follows:

- Sparse space supports all the byte encodings that may be generated in an Intel system to ensure compatibility with PCI devices/drivers. The results of some references are not explicitly defined. These are the missing entries in Table 6–6 (that is, word size with address<6:5> = 11). The hardware will complete the reference, but the reference is not required to produce any particular result, nor will the system report an error.
- Software must use longword load or store instructions (LDVSTL) to perform a reference of longword length or less on the PCI bus. The bytes to be transferred must be positioned within the longword in the correct byte lanes as indicated by the PCI byte enable bits. The hardware does not shift bytes within the longword. Quadword load and store instructions must be used only to perform quadword transfers. Use of STQ/LDQ instructions for any other references will produce UNPREDICTABLE results.

[5] Quadword encoding is provided by way of 21164 address bits <6:3>. In this case, 21164 address bit <7> is treated as zero by the hardware.

- Hardware does not perform read-ahead (prefetch) transactions in sparse space because read-ahead transactions may have detrimental side effects.
- Programmers are required to insert memory barrier (MB) instructions between sparse-space transactions to prevent collapsing in the 21164 write buffer. However, this is not always necessary. For example, consecutive sparse-space addresses will be separated by 32 bytes (and will not be collapsed by the 21164).
- Programmers are required to insert MB instructions if the sparse-space address ordering/coherency to a dense-space address is to be maintained.
- Table 6–6 shows encoding of the 21164 address for sparse-space read transactions to PCI space. An important point to note is that signals **addr_h<33:5>** are directly available from the 21164 pins. On read transactions, the 21164 sends out **addr_h<2:0>** indirectly on the **int4_valid** pins. Signals **addr_h<2:0>** are required to be zero. Transactions with **addr_h<2:0>** not equal to zero will produce UNPREDICTABLE results.
- Table 6–5 shows the relation between **int4_valid<3:0>** and **addr_h<4:3>** for a sparse-space write transaction. Unlisted **int4_valid** patterns will produce UNPREDICTABLE results (that is, as a result of collapsing in the 21164 write buffer; or by issuing a STQ instruction when a STL instruction is required).

Table 6–5 Int4_valid and 21164 Address Relationship

EV5 Data Cycle	Int4_valid<3:0>[1]	Address<4:3>
First	00 01	0 0
	00 10	0 0
	01 00	0 1
	10 00	0 1
Second	00 01	1 0
	00 10	1 0
	01 00	1 1
	10 00	1 1
	11 00 (STQ)[2]	1 1

[1] All other **int4_valid** patterns result in UNPREDICTABLE results.
[2] Only one valid STQ case is allowed.

PCI Sparse Memory Space

Table 6–6 defines the low-order PCI sparse memory address bits. Signals **addr_h<7:3>** are used to generate the length of the PCI transaction in bytes, the byte enable bits, and **ad<2:0>**. The 21164 signals **addr_h<30:8>** correspond to the quadword PCI address and are sent out on **ad<25:3>**.

Table 6–6 PCI Memory Sparse-Space Read/Write Encodings

Size		Byte Offset addr_h <6:5>	21164 Instruction Allowed	ad<2:0>	PCI Byte Enable[1]	Data-In Register Byte Lanes 63.....32 31.......0
	addr_h<4:3>					
		00		A<7>[2],00[3]	1110	OOOX
		01		A<7>,00	1101	OOXO
Byte	00	10	LDL,STL	A<7>,00	1011	OXOO
		11		A<7>,00	0111	XOOO
		00		A<7>,00	1100	OOXX
Word[4]	01	01	LDL,STL	A<7>,00	1001	OXXO
		10		A<7>,00	0011	XXOO
		00		A<7>,00	1000	OXXX
Tribyte	10	01	LDL,STL	A<7>,00	0001	XXXO
Longword	11	00	LDL,STL	A<7>,00	0000	XXXX
Quadword	11	11	LDQ,STQ	000	0000	XXXX XXXX

[1] Byte enable set to 0 indicates that byte lane carries meaningful data.
[2] A<7> = **addr_h<7>.**
[3] In PCI sparse memory space, **ad<1:0>** is always zero.
[4] Missing entries (for example, word size with 21164 address = 11) enjoy UNPREDICTABLE results.

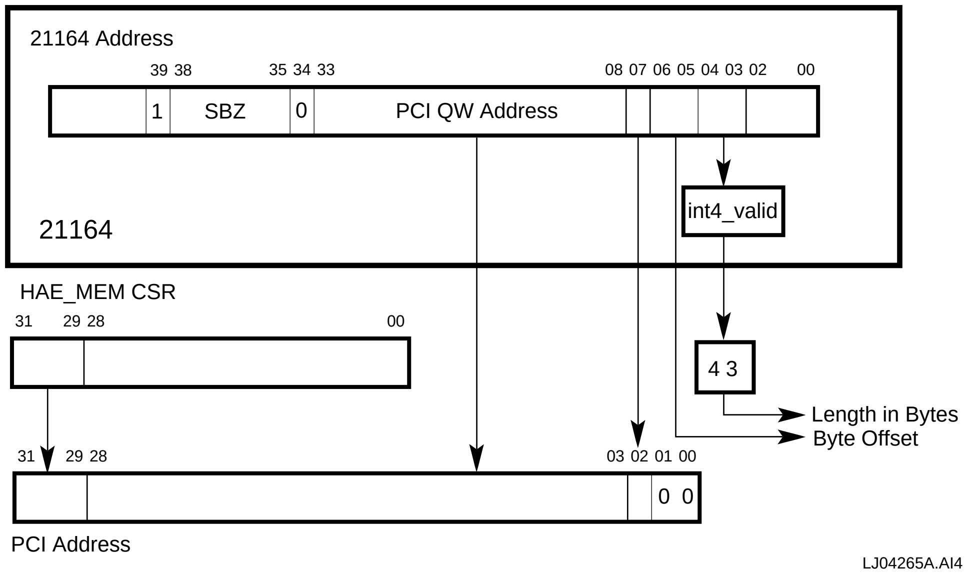
The high-order **ad<31:26>** are obtained from either the hardware extension register (HAE_MEM) or the 21164 address depending on sparse-space regions, as shown in Table 6–7. For more information about the 21174 HAE_MEM CSR, see Section 5.1.6.

Table 6–7 PCI Address Mapping

21164 Address	Region	ad					
		<31>	<30>	<29>	<28>	<27>	<26>
80.0000.0000 to 83.FFFF.FFFF	1	HAE_MEM <31>	HAE_MEM <30>	HAE_MEM <29>	CPU<33>	CPU<32>	CPU<31>
84.0000.0000 to 84.FFFF.FFFF	2	HAE_MEM <15>	HAE_MEM <14>	HAE_MEM <13>	HAE_MEM <12>	HAE_MEM <11>	CPU<31>
85.0000.0000 to 85.FFFF.FFFF	3	HAE_MEM <7>	HAE_MEM <6>	HAE_MEM <5>	HAE_MEM <4>	HAE_MEM <3>	HAE_MEM <2>

Figure 6–9 shows the mapping for region 1.

Figure 6–9 PCI Memory Sparse-Space Address Generation – Region 1



PCI Sparse Memory Space

Figure 6–10 shows the mapping for region 2.

Figure 6–10 PCI Memory Sparse-Space Address Generation – Region 2

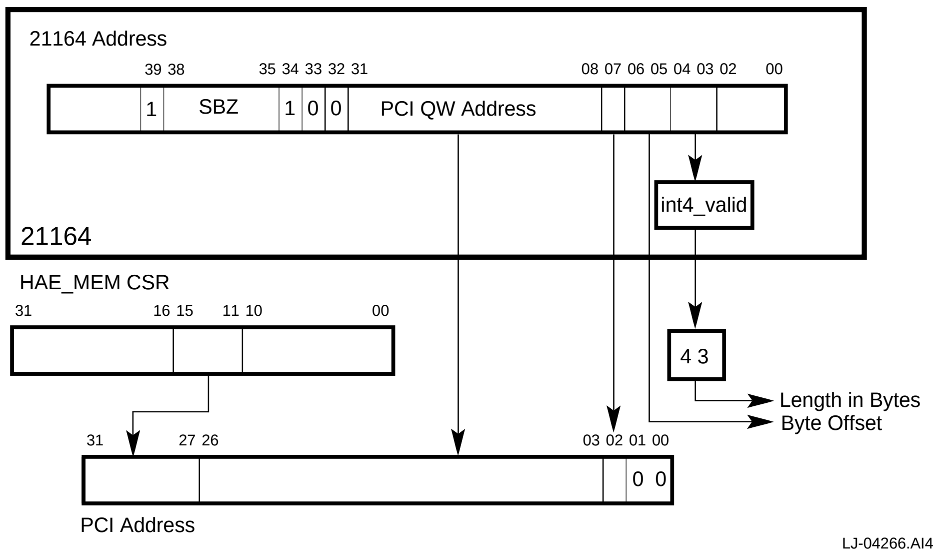
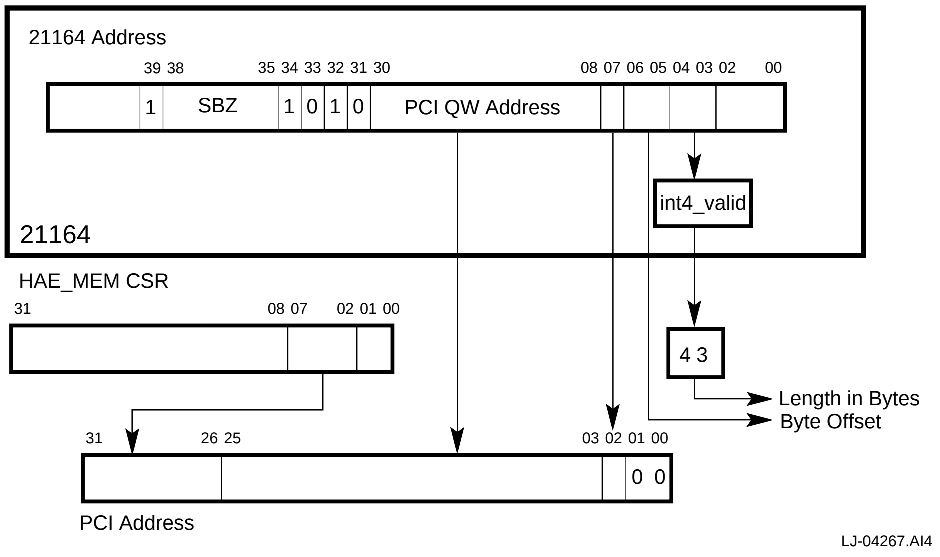


Figure 6–11 shows the mapping for region 3.

Figure 6–11 PCI Memory Sparse-Space Address Generation – Region 3



6.8 PCI Sparse I/O Space

The PCI sparse I/O space is divided into two regions — region A and region B. Region A addresses the lower 32MB of PCI I/O space and is never relocated. This region will be used to address the (E)ISA devices. Region B is used to address a further 32MB of PCI I/O space and is relocatable using the HAE_IO register.

6.8.1 Hardware Extension Register (HAE_IO)

In sparse space, the 21164 address bits <7:3> are used to encode byte enable bits, size bits, and the low-order **ad<2:0>**. This means that there are now five fewer address bits available to generate the PCI physical address.

The system provides two PCI sparse I/O space regions and allows one region to be relocated by way of bits in the HAE_IO register.

6.8.2 PCI Sparse I/O Space Access Operation

The PCI sparse I/O space is located in the range 85.8000.0000 to 85.FFFF.FFFF. This space has characteristics similar to the PCI sparse memory space. This 2GB 21164 address segment maps to two 32MB regions of PCI I/O address space. A read or write transaction to this space causes a PCI I/O read or write command. The high-order PCI address bits are handled as follows:

- Region A: This region has **addr_h<34:30>** = 10110 and addresses the lower 32MB of PCI sparse I/O space. Signals **ad<31:25>** are asserted at zero by the hardware (see Figure 6–12). Region A is used to address (E)ISA address space (the EISA 64KB I/O space cannot be relocated). Figure 6–12 shows PCI sparse I/O space address translation in Region A.
- Region B: This region has **addr_h<34:30>** = 10111 and addresses a relocatable 32MB of PCI sparse I/O space. This 32MB segment is relocated by assigning **ad<31:25>** to equal HAE_IO<31:25>. Figure 6–13 shows PCI sparse I/O space address translation in Region B.

The remainder of the PCI I/O address is formed in the same way for both regions:

- **ad<24:3>** are derived from **addr_h<29:8>**.
- **ad<2:0>** are defined in Table 6–8.

PCI Sparse I/O Space

Table 6–8 contains the PCI sparse I/O space read/write encodings.

Table 6–8 PCI Sparse I/O Space Read/Write Encodings

Size		Byte Offset	21164			Data-In Register
	addr_h<4:3>	addr_h <6:5>	Instruction Allowed	ad<2:0>	PCI Byte Enable[1]	Byte Lanes 63.....32 31.......0
		00		A<7>[2],00	1110	OOOX
		01		A<7>,00	1101	OOXO
Byte	00	10	LDL,STL	A<7>,00	1011	OXOO
		11		A<7>,00	0111	XOOO
		00		A<7>,00	1100	OOXX
Word[3]	01	01	LDL,STL	A<7>,00	1001	OXXO
		10		A<7>,00	0011	XXOO
		00		A<7>,00	1000	OXXX
Tribyte	10	01	LDL,STL	A<7>,00	0001	XXXO
Longword	11	00	LDL,STL	A<7>,00	0000	XXXX
Quadword	11	11	LDQ,STQ	000	0000	XXXX XXXX

[1] Byte enable set to 0 indicates that byte lane carries meaningful data.
[2] A<7> = **addr_h<7>.**
[3] Missing entries (for example, word size with 21164 address = 11) enjoy UNPREDICTABLE results.

Figure 6–12 PCI Sparse I/O Space Address Translation (Region A, Lower 32MB)

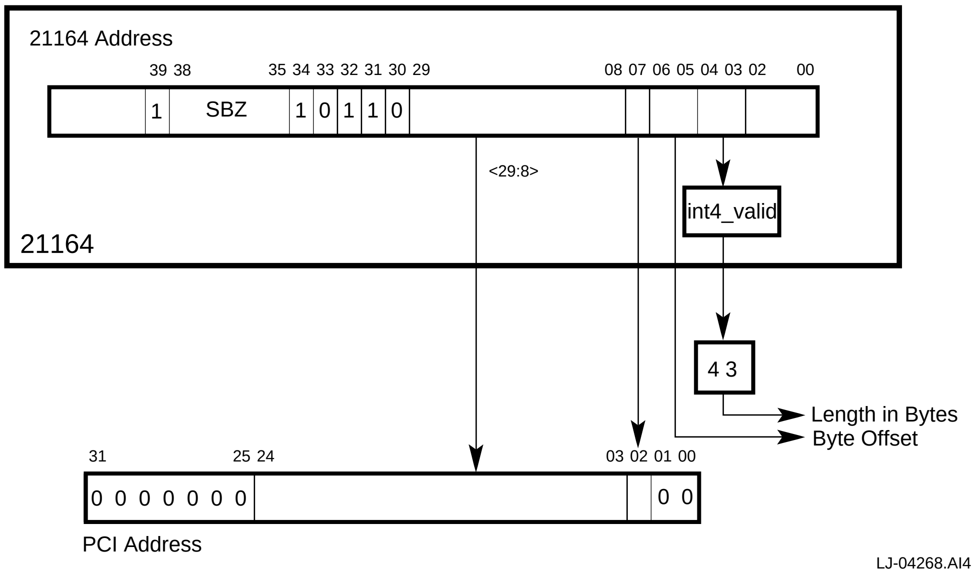
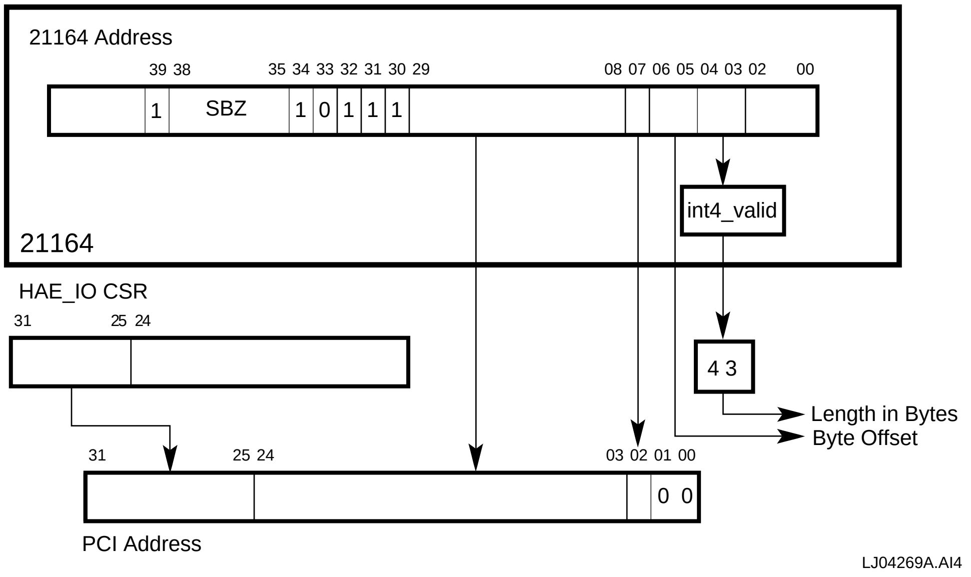


Figure 6–13 PCI Sparse I/O Space Address Translation (Region B, Higher Area)



6.9 PCI Configuration Space

The PCI configuration space is located in the range 87.0000.0000 to 87.1FFF.FFFF. Software is advised to clear PYXIS_CTRL<FILL_ERR_EN> when probing for PCI devices by way of configuration space read transactions. This will prevent the 21174 from generating an ECC error if no device responds to the configuration cycle (and random data is picked up on the PCI bus).

A read or write transaction to this space causes a configuration read or write cycle on the PCI. There are two classes of targets that are selected, based on the value of the CFG register.

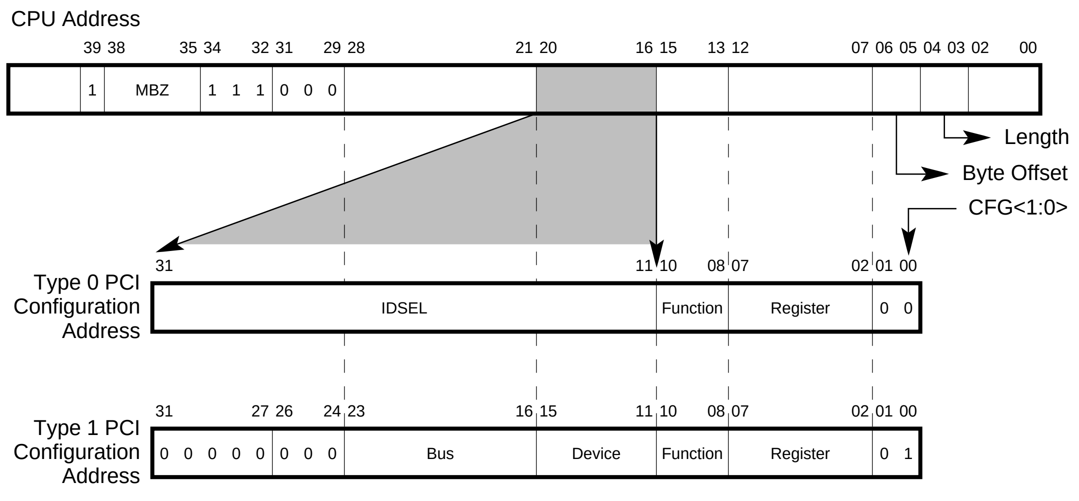
- Type 0 — These are targets on the primary 64-bit PCI bus. These targets are selected by making CFG<1:0> = 0.
- Type 1 — These are targets on the secondary 32-bit PCI bus (that is, behind a PCI-to-PCI bridge). These targets are selected by making CFG<1:0> = 1.

Note: CFG<1:0> = 10 or 11 are reserved (by the PCI specification).

Software must program the CFG register before running a configuration cycle. Sparse address decoding is used. Signals **addr_h<6:3>** are used to generate both the length of the PCI transaction in bytes and the byte enable bits. Signals **ad<1:0>** are obtained from CFG<1:0>. Signals **addr_h<28:7>** correspond to **ad<23:2>** and provide the configuration command information (such as which device to select). The high-order **ad<31:24>** are always zero.

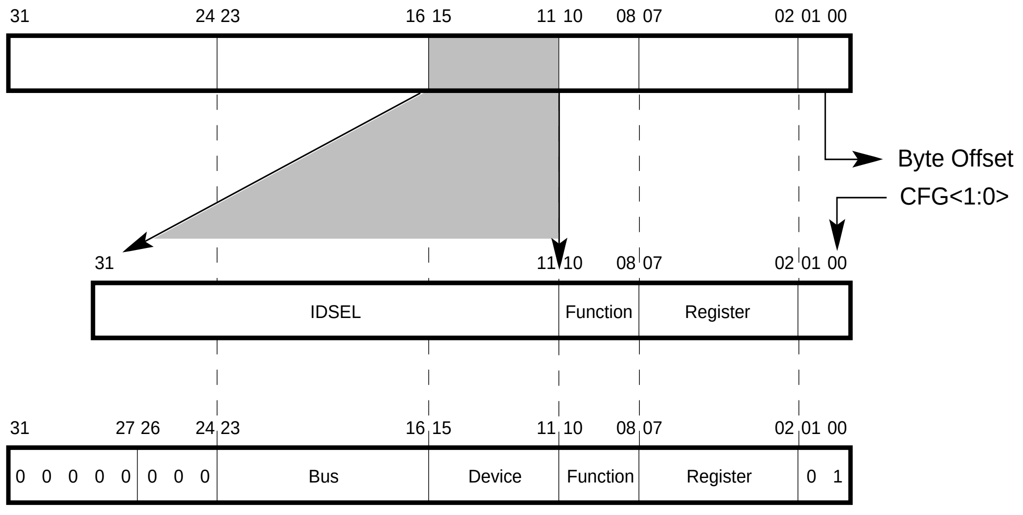
Figure 6–14 depicts PCI configuration space (sparse). Figure 6–15 shows PCI configuration space (dense).

Figure 6–14 PCI Configuration Space Definition (Sparse)



LJ04270A.AI4

Figure 6–15 PCI Configuration Space Definition (Dense)



LJ-05400.AI4

PCI Configuration Space

Peripherals are selected during a PCI configuration cycle if the following three conditions are met:

1. Their IDSEL pin is asserted.
2. The PCI bus command indicates a configuration read or write.
3. Address bits <1:0> are 00.

Address bits <7:2> select a Dword (longword) register in the peripheral’s 256-byte configuration address space. Transactions can use byte masks.

Peripherals that integrate multiple functional units (for example, SCSI and Ethernet) can provide configuration space for each function. Address bits <10:8> can be decoded by the peripheral to select one of eight functional units.

Signals **ad<31:11>** are available to generate the IDSEL bits (note that IDSEL bits behind a PCI-to-PCI bridge are determined from the device field encoding of a type 1 access). The IDSEL pin of each device is connected to a unique PCI address bit from **ad<31:11>**. The binary value of **addr_h<20:16>** is used to select which **ad<31:11>** is asserted, as shown in Table 6–9.

Table 6–9 CPU Address to IDSEL Conversion

CPU Address <20:16>	ad<31:11> – IDSEL
00000	0000 0000 0000 0000 0000 1
00001	0000 0000 0000 0000 0001 0
00010	0000 0000 0000 0000 0010 0
00011	0000 0000 0000 0000 0100 0
.....	....
.....	....
10011	0100 0000 0000 0000 0000 0
10100	1000 0000 0000 0000 0000 0
10101	0000 0000 0000 0000 0000 0
.....	...(No device selected)
.....	—
11111	0000 0000 0000 0000 0000 0

Note: If a quadword access is specified for the configuration cycle, then the least significant bit of the register number field (such as **ad<2>**) must be zero. Quadword transactions must access quadword aligned registers.

If the PCI cycle is a configuration read or write cycle but the **ad<1:0>** are 01 (that is, a type 1 transfer), then a device on a hierarchical bus is being selected via a PCI-to-PCI bridge. This cycle is accepted by the PCI-to-PCI bridge for propagation to its secondary PCI bus. During this cycle, <23:16> selects a unique bus number, and address <15:8> selects a device on that bus (typically decoded by the PCI-to-PCI bridge to generate the secondary PCI address pattern for IDSEL). In addition, address <7:2> selects a Dword (longword) in the device's configuration space.

Table 6–10 contains the PCI configuration space read/write encodings.

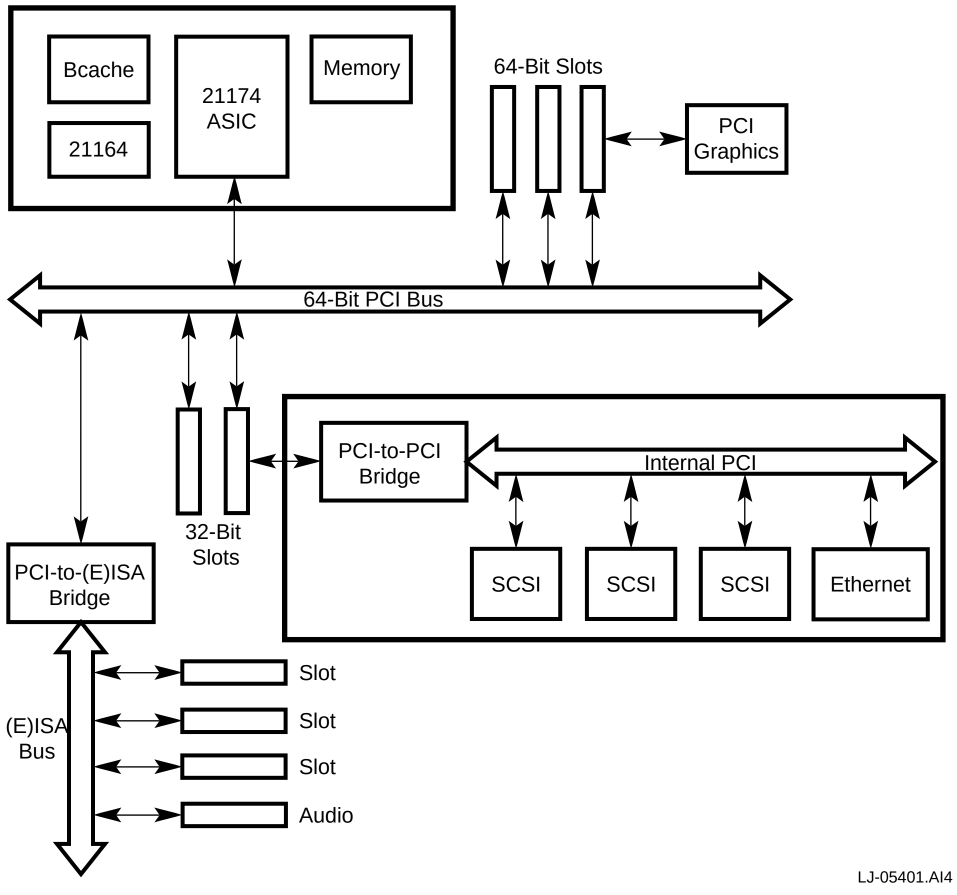
Table 6–10 PCI Configuration Space Read/Write Encodings

Size		Byte Offset addr_h	21164 Instruction		PCI Byte	Data-In Register Byte Lanes
	addr_h<4:3>	<6:5>	Allowed	ad<2:0>	Enable[1]	63.....32 31.......0
		00		A<7>[2],00	1110	OOOX
		01		A<7>,00	1101	OOXO
Byte	00	10	LDL,STL	A<7>,00	1011	OXOO
		11		A<7>,00	0111	XOOO
		00		A<7>,00	1100	OOXX
Word[3]	01	01	LDL,STL	A<7>,00	1001	OXXO
		10		A<7>,00	0011	XXOO
		00		A<7>,00	1000	OXXX
Tribyte	10	01	LDL,STL	A<7>,00	0001	XXXO
Longword	11	00	LDL,STL	A<7>,00	0000	XXXX
Quadword	11	11	LDQ,STQ	000	0000	XXXX XXXX

[1] Byte enable set to 0 indicates that byte lane carries meaningful data.
[2] A<7> = **addr_h<7>.**
[3] Missing entries (for example, word size with **addr_h<6:5>** = 11) generate UNPREDICTABLE results.

Each PCI-to-PCI bridge can be configured via PCI configuration cycles on its primary PCI interface. Configuration parameters in the PCI-to-PCI bridge will identify the bus number for its secondary PCI interface and a range of bus numbers that may exist hierarchically behind it. If the bus number of the configuration cycle matches the bus number of the bridge chip's secondary PCI interface, it will accept the configuration cycle, decode it, and generate a PCI configuration cycle with **ad<1:0>** = 00 on its secondary PCI interface. If the bus number is within the range of bus numbers that may exist hierarchically behind its secondary PCI interface, the bridge chip passes the PCI configuration cycle on unmodified (**ad<1:0>** = 01). It will be accepted by a bridge further downstream. Figure 6–16 shows a typical PCI hierarchy. This is only one example of how the 21174 can be used in a system design.

Figure 6–16 PCI Bus Hierarchy



6.10 PCI Special/Interrupt Cycles

PCI special/interrupt cycles are located in the range 87.2000.0000 to 87.3FFF.FFFF.

The Special cycle command provides a simple message broadcasting mechanism on the PCI. The Intel processor uses this cycle to broadcast processor status; but in general it may be used for logical sideband signaling between PCI agents. The special cycle contains no explicit destination address, but is broadcast to all agents. Each receiving agent must determine if the message contained in the data field is applicable to it.

A write access in the range 87.2000.0000 to 87.3FFF.FFFF causes a special cycle on the PCI. The 21164's write data will be passed unmodified to the PCI. Software must write the data in longword 0 of the hexword with the following fields:

- Bytes 0 and 1 contain the encoded message.
- Bytes 2 and 3 are message dependent (optional) data fields.

A read of the same address range will result in an Interrupt Acknowledge cycle on the PCI and return the vector data provided by the PCI-EISA bridge to the 21164.

6.11 Hardware-Specific and Miscellaneous Register Space

These registers are located in the range 87.4000.0000 to 87.FFFF.FFFF.

Table 6–11 lists the address map for the hardware-specific registers.

Table 6–11 Hardware and Miscellaneous Address Map

CPU Address <39:28>	Selected Region
1000 0111 0100	General control, diagnostic, performance monitoring, and error logging registers
1000 0111 0101	Memory controller registers
1000 0111 0110	PCI window control registers and scatter-gather translation registers
1000 0111 0111	Reserved
1000 0111 1000	Miscellaneous registers
1000 0111 1010	Interrupt control registers
1000 0111 11xx	Flash ROM read/write space – for programming

The address space here is a hardware-specific variant of sparse-space encoding. For the CSRs, **addr_h<27:6>** specifies a longword address where **addr_h<5:0>** must be zero. All the 21174 registers are accessed with a LW granularity. For the flash ROM, **addr_h<30:6>** defines a byte address. The fetched byte is always returned in the first byte lane (bits <7:0>).

6.12 PCI to Physical Memory Address

Incoming PCI addresses (32-bit or 64-bit) have to be mapped to the 21164 cached memory space (8GB). The 21174 provides five programmable address windows that control access of PCI peripherals to system memory.

The mapping from the PCI address to the physical address can be direct, direct mapped (physical mapping with an address offset), or scatter-gather mapped (virtual mapping). These five address windows are referred to as the PCI target windows.

Window 4 maps directly, using the “Monster Window” with dual-address cycles (DAC), where **ad<33:0>** equals **addr_h<33:0>**.

The following three registers are associated with windows <3:0>:

- Window base (W_BASE) register
- Window mask (W_MASK) register
- Translated base (T_BASE) register

In addition, there is an extra register associated with window 3 only. This is the window DAC register and is used for PCI 64-bit addressing (that is, the DAC mode). The following text applies only to windows <3:0>.

The window mask register provides a mask corresponding to **ad<31:20>** of an incoming PCI address. The size of each window can be programmed to be from 1MB to 4GB in powers of two, by masking bits of the incoming PCI address using the window mask register, as shown in Table 6–12. (Note that the mask field pattern was chosen to speed up timing-critical logic circuits.)

Table 6–12 shows the PCI target window mask fields.

Table 6–12 PCI Target Window Mask Register Fields[1]

PCI_MASK<31:20>	Size of Window	Value of n
0000 0000 0000	1MB	20
0000 0000 0001	2MB	21
0000 0000 0011	4MB	22
0000 0000 0111	8MB	23
0000 0000 1111	16MB	24
0000 0001 1111	32MB	25
0000 0011 1111	64MB	26
0000 0111 1111	128MB	27
0000 1111 1111	256MB	28
0001 1111 1111	512MB	29
0011 1111 1111	1GB	30
0111 1111 1111	2GB	31
1111 1111 1111	4GB	32
Otherwise	UNPREDICTABLE	—

[1] Only the incoming **ad<31:n>** are compared with <31:*n*> of the window base register, as shown in Figure 6–18. If *n*=32, no comparison is performed.

Based on the value of the window mask register, the unmasked bits of the incoming PCI address are compared with the corresponding bits of each window's window base register. If one of the window base registers and the incoming PCI address match, then the PCI address has hit the PCI target window. Otherwise, the PCI address has missed the window. A window enable bit, W_EN, is provided in each window's window base register to allow windows to be independently enabled (W_EN = 1) or disabled (W_EN = 0).

If a hit occurs in any of the four windows that are enabled, then the 21174 will respond to the PCI cycle by asserting the signal **devsel**. The PCI target windows must be programmed so that their address ranges do not overlap; otherwise, the results are UNDEFINED.

PCI to Physical Memory Address

The window base address must be on a naturally aligned boundary address depending on the size of the window[6]. This rule is not particularly difficult to obey, because the address space of any PCI device can be located anywhere in the PCI's 4GB memory space, and this scheme is compatible with the PCI specification:

> A PCI device specifies the amount of memory space it requires via the Base registers in its configuration space. The Base Address registers are implemented so that the address space consumed by the device is a power of two in size, and is naturally aligned on the size of the space consumed.

A PCI device need not use all the address range it consumes (that is, the size of the PCI address window defined by the base address) and it does not need to respond to unused portions of the address space. The one exception to this is a PCI bridge that requires two additional registers (the base and limit address registers). These registers accurately specify the address space that the bridge device will respond to[7] and are programmed by the power-on self-test (POST) code. The 21174, as a PCI host-bridge device, does not have base and limit registers[8], but does respond to all the addresses defined by the window base register (that is, all addresses within a window).

Figure 6–17 shows how the DMA address ranges of a number of PCI devices are accepted by the PCI-window ranges. PCI devices are allowed to have multiple DMA address ranges, as shown for device 2. The example also shows that the window can be larger than the corresponding device's DMA address range, as shown for device 0. Device 1 and device 2 have address ranges that are accepted by one window. Each window determines whether direct mapping or scatter-gather mapping is used to access physical memory.

[6] For example, a 4MB window cannot begin at address 1MB. It must start at addresses 4MB, 8MB, 12MB, ...

[7] A PCI bridge device responds to all addresses in the range: base $\leq$ address $<$ limit.

[8] Host-bridge devices, because they are under system control, are free to violate the rules.

Figure 6–17 PCI DMA Addressing Example

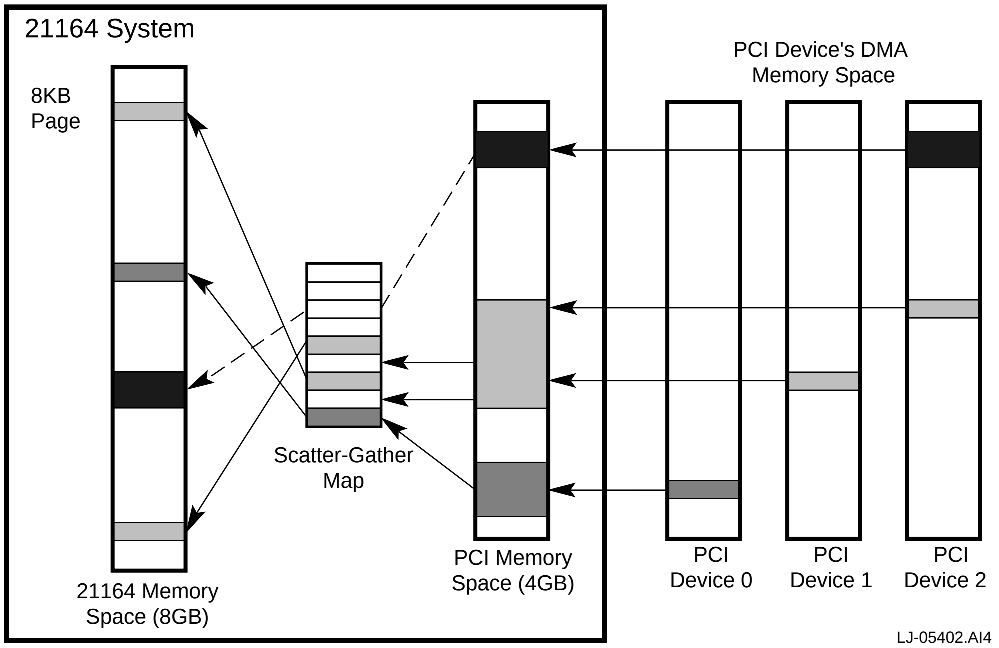


Figure 6–18 shows the PCI window logic. The comparison logic associated with **ad<63:32>** is only used for DAC[9] mode; and only if enabled by a bit in the window base register for window 3. This logic is only applicable to window 3. The remaining windows only recognize 32-bit PCI addresses (that is, SAC[10] cycles).

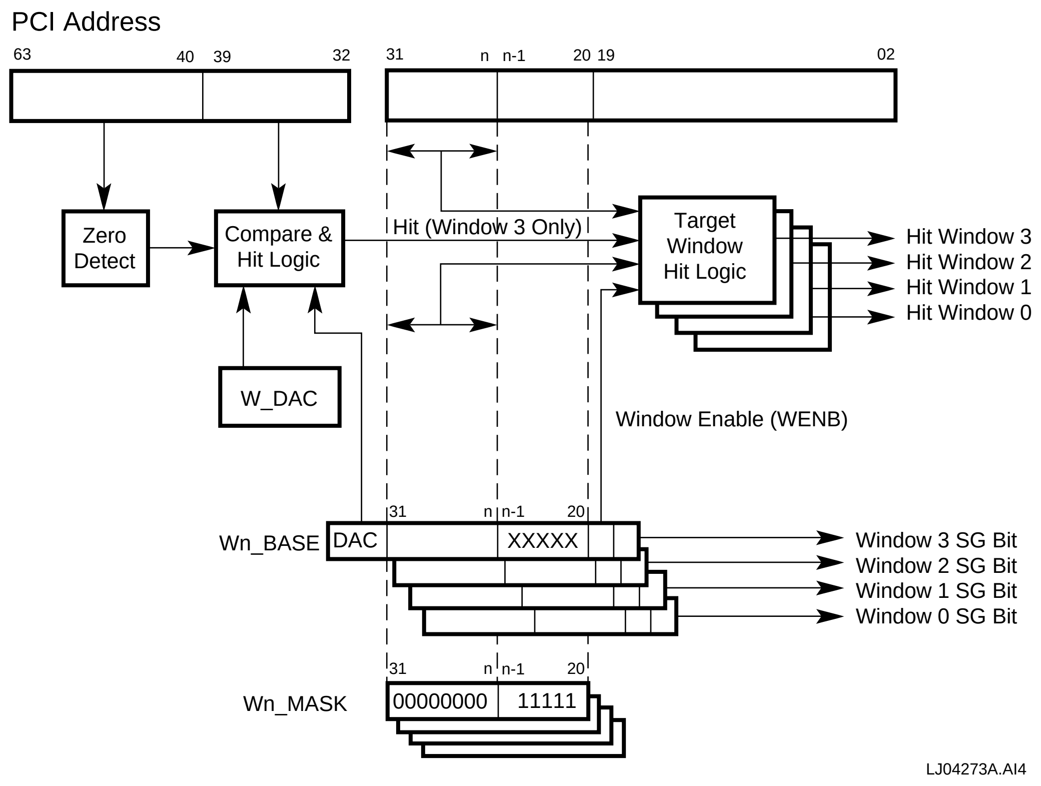
For a hit to occur in a DAC address, **ad<63:40>** must be zero, **ad<39:32>** must match the window DAC base register, and **ad<31:20>** must also have a compare hit. This scheme allows a naturally aligned, 1MB–4GB PCI window to be placed any-where in the first 1TB of a 64-bit PCI address. When an address match occurs with a PCI target window, the 21174 translates the 32-bit PCI address to **addr_h<33:0>**.

[9] Dual-address cycle (DAC) — only issued if <63:32> are nonzero for a 64-bit address.

[10] Single-address cycle (SAC) — all 32-bit addresses. A PCI device must use SAC if <63:32> equals 0.

PCI to Physical Memory Address

Figure 6–18 PCI Target Window Compare



6.13 Direct-Mapped Addressing

The target address is translated by direct mapping or scatter-gather mapping as determined by the Wx_BASE_SG (scatter-gather) bit of the window's PCI base register. If the Wx_BASE_SG bit is clear, the DMA address is direct mapped, and the translated address is generated by concatenating bits from the matching window's translated base register (T_BASE) with bits from the incoming PCI address. The bits involved in the concatenation are defined by the window mask register as shown in Table 6–13. The unused bits of the translated base register (also in Table 6–13) must be cleared (that is, the hardware performs an AND-OR operation to accomplish the concatenation). Because memory is located in the lower 8GB of the 21164 address space, the 21174 ensures (implicitly) that address bits <39:33> are always zero.

Because the translated base is simply concatenated to the PCI address, then the direct mapping is to a naturally aligned memory region. For example, a 4MB direct-mapped window will map to any 4MB region in main memory that falls on a 4MB boundary (for instance, it is not possible to map a 4MB region to the main memory region 1MB–5MB).

Table 6–13 lists direct-mapped PCI target address translations.

Table 6–13 Direct-Mapped PCI Target Address Translation *(Sheet 1 of 2)*

W_MASK<31:20>	Size of Window	Translated Address <32:2>
0000 0000 0000	1MB	Translated Base<33:20> : **ad<19:2>**
0000 0000 0001	2MB	Translated Base<33:21> : **ad<20:2>**
0000 0000 0011	4MB	Translated Base<33:22> : **ad<21:2>**
0000 0000 0111	8MB	Translated Base<33:23> : **ad<22:2>**
0000 0000 1111	16MB	Translated Base<33:24> : **ad<23:2>**
0000 0001 1111	32MB	Translated Base<33:25> : **ad<24:2>**
0000 0011 1111	64MB	Translated Base<33:26> : **ad<25:2>**
0000 0111 1111	128MB	Translated Base<33:27> : **ad<26:2>**
0000 1111 1111	256MB	Translated Base<33:28> : **ad<27:2>**
0001 1111 1111	512MB	Translated Base<33:29> : **ad<28:2>**
0011 1111 1111	1GB	Translated Base<33:30> : **ad<29:2>**

Table 6–13 Direct-Mapped PCI Target Address Translation *(Sheet 2 of 2)*

W_MASK<31:20>	Size of Window	Translated Address <32:2>
0111 1111 1111	2GB	Translated Base<33:31> : **ad<30:2>**
1111 1111 1111	4GB	Translated Base<33:32> : **ad<31:2>**
Otherwise	Not supported	—

6.14 Scatter-Gather Addressing

If the Wx_BASE_SG bit of the PCI base register is set, then the translated address is generated by a lookup table. This table is called a scatter-gather map. Figure 6–20 shows the scatter-gather addressing scheme — full details of this scheme are provided later in Section 6.15, but for now a quick description is provided. The incoming PCI address is compared to the PCI window addresses looking for a hit. The translated base register, associated with the PCI window that is hit, is used to specify the starting address of the scatter-gather map table in memory. Bits of the incoming PCI address are used as an offset from this starting address, to access the scatter-gather PTE. This PTE, in conjunction with the remaining, least-significant PCI address bits, forms the required memory address.

Each scatter-gather map entry maps an 8KB page of PCI address space into an 8KB page of the 21164 address space. This offers a number of advantages to software:

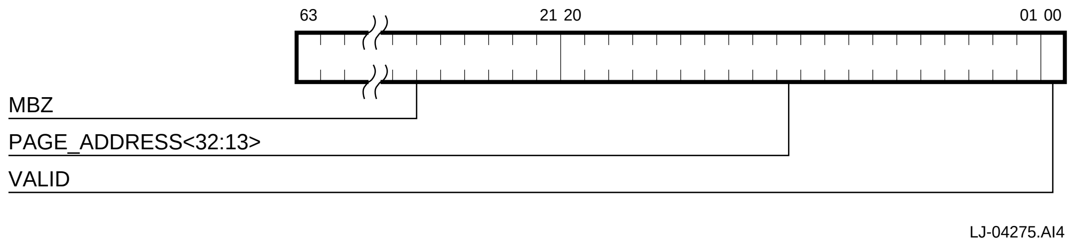
- Performance: ISA devices map to the lower 16MB of memory. The Windows NT operating system currently copies data from here to user space. The scatter-gather map eliminates the need for this copy operation.
- User I/O buffers might not be physically contiguous or contained within a page. With scatter-gather mapping, software does not have to manage the scattered nature of the user buffer by copying data.

In the personal computer (PC) world, scatter-gather mapping is not an address translation scheme but is used to signify a DMA transfer list. An element in this transfer list contains the DMA address and the number of data items to transfer. The DMA device fetches each item of the list until the list is empty. Many of the PCI devices (such as an EISA bridge) support this form of scatter-gather mapping.

Each scatter-gather map page table entry (PTE) is a quadword and has a valid bit in bit position 0, as shown in Figure 6–19. Address bit 13 is at bit position 1 of the map entry. Because the 21174 implements valid memory addresses up to 16GB, then bits <63:22> of the scatter-gather map entry must be programmed to 0. Bits <21:1> of the scatter-gather map entry are used to generate the physical page address. The physical page address is appended to **ad<12:5>** of the incoming PCI address to generate the memory address.

System implementations may support less than 16GB of physical addressing; however, any unused address bits must be forced to zero. Otherwise, behavior will be UNPREDICTABLE.

Figure 6–19 Scatter-Gather PTE Format



The size of the scatter-gather map table is determined by the size of the PCI target window as defined by the window mask register shown in Table 6–14. The number of entries in the table equals the window size divided by the page size (8KB). The size of the table is simply the number of entries multiplied by 8 bytes.

The scatter-gather map table address is obtained from the translated base register and the PCI address as shown in Table 6–14.

Table 6–14 Scatter-Gather Mapped PCI Target Address Translation *(Sheet 1 of 2)*

W_MASK<31:20>	Size of SG Map Table	Translated Address <32:2>
0000 0000 0000	1KB	Translated Base<33:10>[1] : **ad<19:13>**
0000 0000 0001	2KB	Translated Base<33:11> : **ad<20:13>**
0000 0000 0011	4KB	Translated Base<33:12> : **ad<21:13>**
0000 0000 0111	8KB	Translated Base<33:13> : **ad<22:13>**
0000 0000 1111	16KB	Translated Base<33:14> : **ad<23:13>**

Table 6–14 Scatter-Gather Mapped PCI Target Address Translation *(Sheet 2 of 2)*

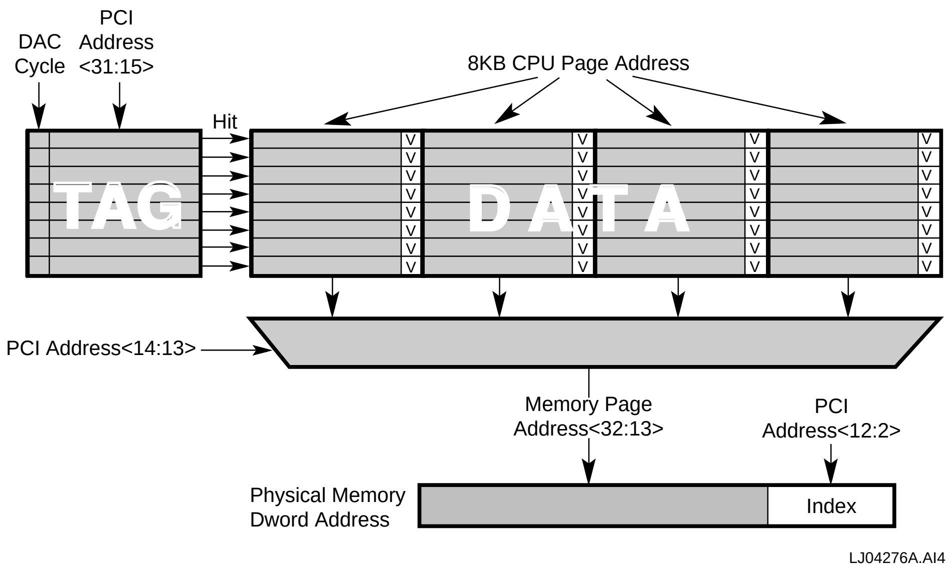
W_MASK<31:20>	Size of SG Map Table	Translated Address <32:2>
0000 0001 1111	32KB	Translated Base<33:15> : **ad<24:13>**
0000 0011 1111	64KB	Translated Base<33:16> : **ad<25:13>**
0000 0111 1111	128KB	Translated Base<33:17> : **ad<26:13>**
0000 1111 1111	256KB	Translated Base<33:18> : **ad<27:13>**
0001 1111 1111	512KB	Translated Base<33:19> : **ad<28:13>**
0011 1111 1111	1MB	Translated Base<33:20> : **ad<29:13>**
0111 1111 1111	2MB	Translated Base<33:21> : **ad<30:13>**
1111 1111 1111	4MB	Translated Base<33:22> : **ad<31:13>**

[1] Unused bits of the Translated Base Register must be zero for correct operation.

6.15 Scatter-Gather TLB

An eight-entry translation lookaside buffer (TLB) is provided in the 21174 for scatter-gather map entries. The TLB is a fully associative cache and holds the eight most-recent scatter-gather map lookup PTEs. Four of these entries can be locked to prevent their being displaced by the hardware TLB-miss handler. Each of the eight TLB entries holds a PCI address for the tag and four consecutive 8KB 21164 page addresses as the TLB data, as shown in Figure 6–20.

Figure 6–20 Scatter-Gather Associative TLB



Each time an incoming PCI address hits in a PCI target window that has scatter-gather translation enabled, **ad<31:15>** are compared with the 32KB PCI page address in the TLB tag. If a match is found, the required 21164 page address is one of the four items provided by the data of the matching TLB entry. PCI address **ad<14:13>** selects the correct 8KB 21164 page from the four pages fetched.

A TLB hit avoids having to look up the scatter-gather map PTEs in memory, resulting in improved system performance. If no match is found in the TLB, the scatter-gather map lookup is performed and four PTE entries are fetched and written over an existing entry in the TLB.

The TLB entry to be replaced is determined by a round-robin algorithm on the unlocked entries. Coherency of the TLB is maintained by software write transactions to the SG_TBIA (scatter-gather translation buffer invalidate all) register.

The tag portion contains a DAC flag to indicate that the PCI tag address <31:15> corresponds to a 64-bit DAC address. Only one bit is required instead of the high-order PCI address bits <39:32> because only one window is assigned to a DAC cycle, and the window-hit logic has already performed a comparison of the high-order bits with the PCI DAC base register. Figure 6–21 shows the entire translation from PCI address to physical address on a window that implements scatter-gather mapping. Both paths are indicated — the right side shows the path for a TLB hit, while the left side shows the path for a TLB miss. The scatter-gather TLB is shown in a slightly simplified, but functionally equivalent form.

6.15.1 Scatter-Gather TLB Hit Process

The process for a scatter-gather TLB hit is as follows:

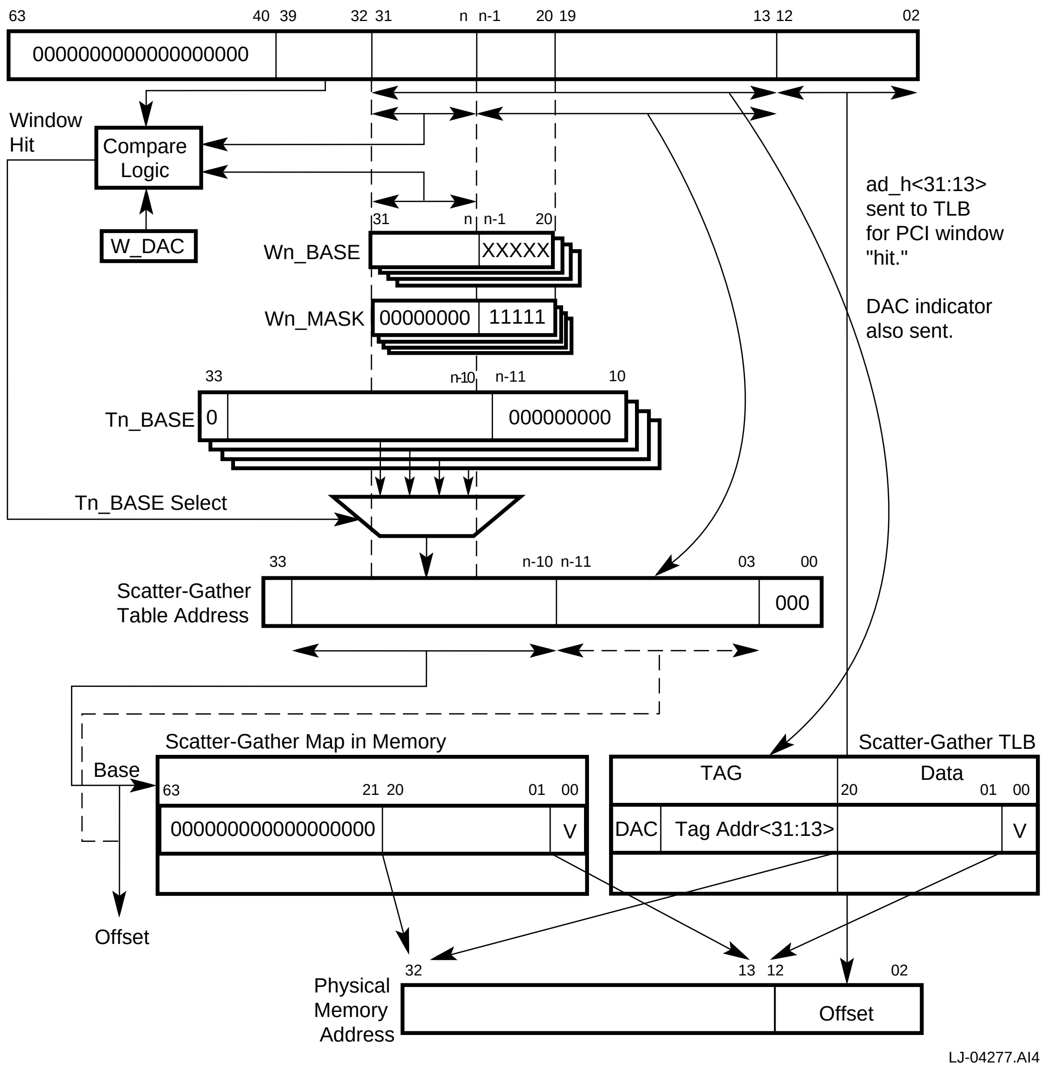
1. The window compare logic determines if the PCI address has hit in one of the four windows, and the PCI_BASE<SG> bit determines if the scatter-gather path should be taken. If window 3 has DAC-mode enabled, and the PCI cycle is a DAC cycle, then a further comparison is made between the high-order PCI bits and the PCI DAC BASE register.
2. PCI address **ad<31:13>** is sent to the TLB associative tag together with the DAC hit indication. If **ad<31:13>** and the DAC bits match in the TLB, then the corresponding 8KB 21164 page address is read out of the TLB. If this entry is valid, then a TLB hit has occurred and this page address is concatenated with **ad<12:2>** to form the physical memory address. If the data entry is invalid, or if the TAG compare failed, then a TLB miss occurs.

6.15.2 Scatter-Gather TLB Miss Process

The process for a scatter-gather TLB miss is as follows:

1. The relevant bits of the PCI address (as determined by the window mask register) are concatenated with the relevant translated base register bits to form the address used to access the scatter-gather map entry (PTE) from a table located in main memory.
2. Bits <20:1> of the map entry (PTE from memory) are used to generate the physical page address, which is appended to the page offset to generate the physical memory address. The TLB is also updated at this point, using a round-robin algorithm, with the four PTE entries that correspond to the 32KB PCI page address that first missed the TLB. The tag portion of the TLB is loaded with this PCI page address, and the DAC bit is set if this PCI cycle is a DAC cycle.
3. If the requested PTE is marked invalid (bit 0 is clear), then a TLB invalid entry exception is taken.

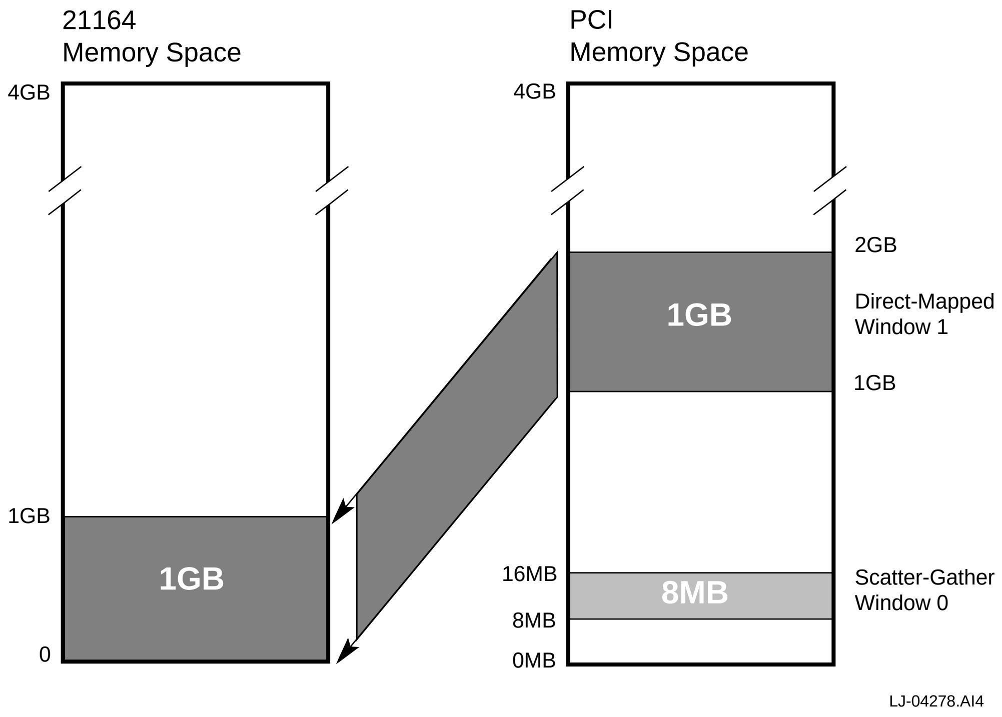
Figure 6–21 Scatter-Gather Map Translation



6.16 Suggested Use of a PCI Window

Figure 6–22 shows the PCI window assignment after power is turned on (configured by firmware), and Table 6–15 lists the details. PCI window 0 was chosen for the 8MB to 16MB EISA region because this window incorporates the **mem_cs_l** logic. PCI window 3 was not used as it incorporates the DAC cycle logic. PCI window 1 was chosen arbitrarily for the 1GB, direct-mapped region, and PCI window 2 is not assigned.

Figure 6–22 Default PCI Window Allocation



Suggested Use of a PCI Window

Table 6–15 lists the PCI window power-up configuration characteristics.

Table 6–15 PCI Window Power-Up Configuration

PCI Window	Assignment	Size	Comments
0	Scatter-gather	8MB	Not used by firmware; **mem_cs_l** disabled
1	Direct-mapped	1GB	Mapped to 0GB to 1GB of main memory
2	Disabled	—	—
3	Disabled	—	—

6.16.1 Peripheral Component Architecture Compatibility Addressing and Holes

The peripheral component architecture allows certain (E)ISA devices to respond to hardwired memory addresses. An example is a VGA graphics device that has its frame buffer located in memory address region A0000–BFFFF. Such devices “pepper” memory space with holes, which are collectively known as peripheral component compatibility holes.

The PCI-EISA bridge decodes PCI addresses and generates a signal, **mem_cs_l**, which takes into account the various PC compatibility holes.

6.16.2 Memory Chip Select Signal mem_cs_l

The PCI-EISA bridge can be made using the following two chips:

- Intel 82374EB EISA System Component (ESC)
- Intel 82375EB PCI-EISA Bridge (PCEB)

The PCI-EISA bridge provides address decode logic with considerable attributes (such as read only, write only, VGA frame buffer, memory holes, and BIOS shadowing) to help manage the EISA memory map and peripheral component compatibility holes.

This is known as main memory decoding in the PCI-EISA chip, and results in the generation of the memory chip select (**mem_cs_l**) signal. One exception is the VGA memory hole region that never asserts **mem_cs_l**. If enabled, the 21174 uses this signal with the W0_BASE register.

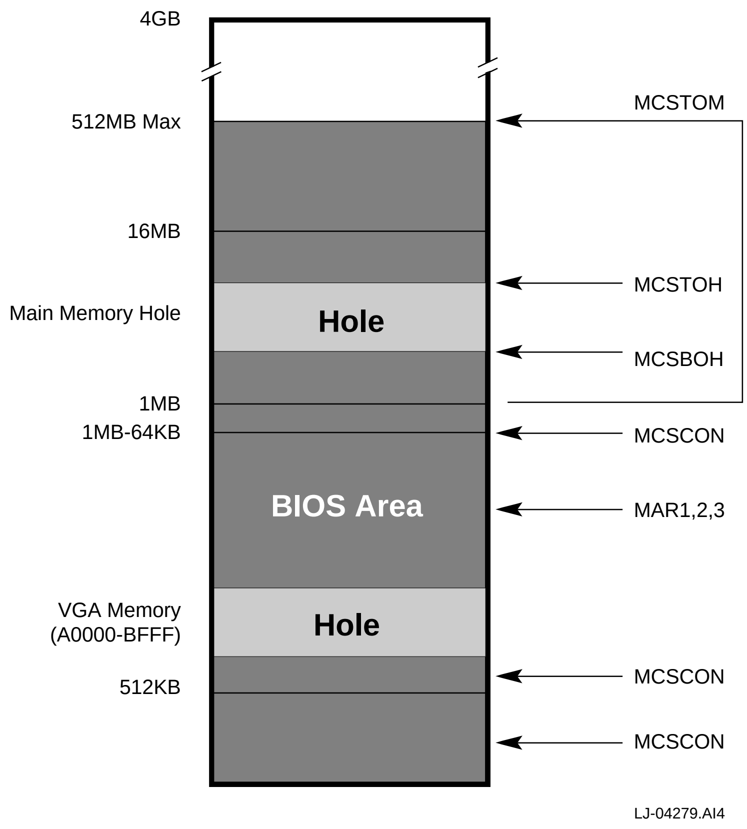
In Figure 6–23, the two main holes are shown lightly shaded, while the **mem_cs_l** range is darkly shaded.

Suggested Use of a PCI Window

This **mem_cs_l** range in Figure 6–23 is subdivided into several portions (such as the BIOS areas) that are individually enabled/disabled using CSRs as listed here:

- The MCSTOM (top of memory) register has a 2MB granularity and can be programmed to select the regions from lMB up to 512MB.
- The MCSTOH (top of hole) and MCSBOH (bottom of hole) registers define a memory hole region where **mem_cs_l** is not selected. The granularity of the hole is 64KB.
- The MARl,2,3 registers enable various BIOS regions.
- The MCSCON (control) register enables the **mem_cs_l** decode logic, and in addition selects a number of regions (0KB to 512KB).
- The VGA memory hole region never asserts **mem_cs_l**.

Figure 6–23 mem_cs_l Decode Area

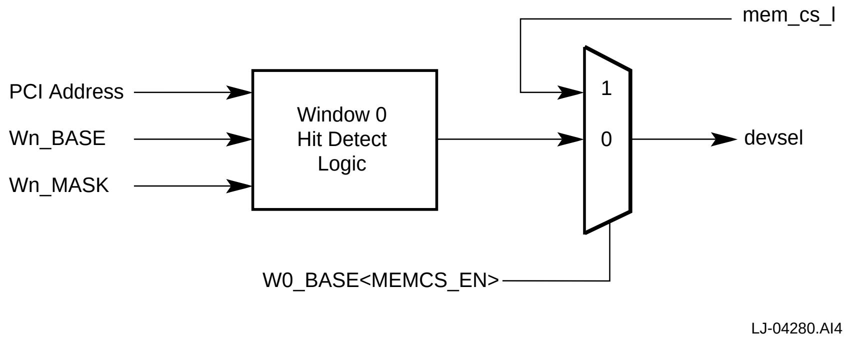


Note: For more detail, please refer to the *Intel 82378 System I/O Manual*.

Suggested Use of a PCI Window

As shown in Figure 6–24, PCI window 0 in the 21174 can be enabled to accept the **mem_cs_l** signal as the PCI memory decode signal. With this path enabled, the PCI window hit logic simply uses the **mem_cs_l** signal. For example, if **mem_cs_l** is asserted, then a PCI window 0 hit occurs and the **devsel** signal is asserted on the PCI.

Figure 6–24 mem_cs_l Logic



Consequently, the window address area must be large enough to encompass the **mem_cs_l** region programmed into the PCI-EISA bridge. The remaining window attributes are still applicable and/or required:

- The Wx_BASE_SG bit in the W0_BASE register determines if scatter-gather or direct-mapping is applicable.
- The W0_ MASK register size information must match the **mem_cs_l** size for the scatter-gather and direct-mapping algorithms to correctly use the translated base register.
- The **mem_cs_l** enable bit, W0_BASE<MEMCS_EN>, takes precedence over W0_BASE<W_EN>.

7 Electrical Specifications

This chapter specifies the 21174 dc specifications.

7.1 PCI Electrical Specification Conformance

The 21174 PCI pins conform to the basic set of PCI electrical specifications in the *PCI Local Bus Specification, Revision 2.1*. See that specification for a complete description of the PCI I/O protocol.

7.2 Absolute Maximum Ratings

Table 7–1 lists the absolute maximum electrical ratings for the 21174. These are stress ratings only; extended exposure to the maximum ratings may affect the reliability of the device.

Table 7–1 Absolute Maximum Electrical Ratings

Parameter	Minimum	Maximum
Supply voltage **Vcc**	3.15 V	3.45 V
Power dissipation	—	3.00 W

7.3 DC Specifications

The 21174 dc specifications with Vcc=3.3 V ± 5% are listed in Table 7–2.

Table 7–2 DC Specifications

Parameter	Description	Minimum	Maximum
Vil	Input level low	– 0.5 V	0.8 V
Vih	Input level high	2.0 V	Vcc+0.5 V
Vol	Output level low (at Iol)	—	0.5 V
Voh	Output level high (at Ioh)	2.4 V	—
Iol	Output low current	—	8 ma
Ioh	Output high current	—	8 ma
Iin	Input leakage current	—	10 µa
Icc	Supply current	—	900 ma

8 Mechanical and Thermal Specifications

This chapter includes drawings that detail the mechanical specifications of the 21174. This chapter also provides operating temperature recommendations and thermal design considerations. Drawings of the recommended heat sinks are included in this chapter.

8.1 Mechanical Specifications

The 21174 is contained in a 474-pin ball grid array (BGA). Figure 8–1 and Figure 8–2 show the physical dimensions of the 21174. All dimensions shown in Figure 8–1 and Figure 8–2 are in millimeters.

Mechanical Specifications

Figure 8–1 474-Pin BGA Package

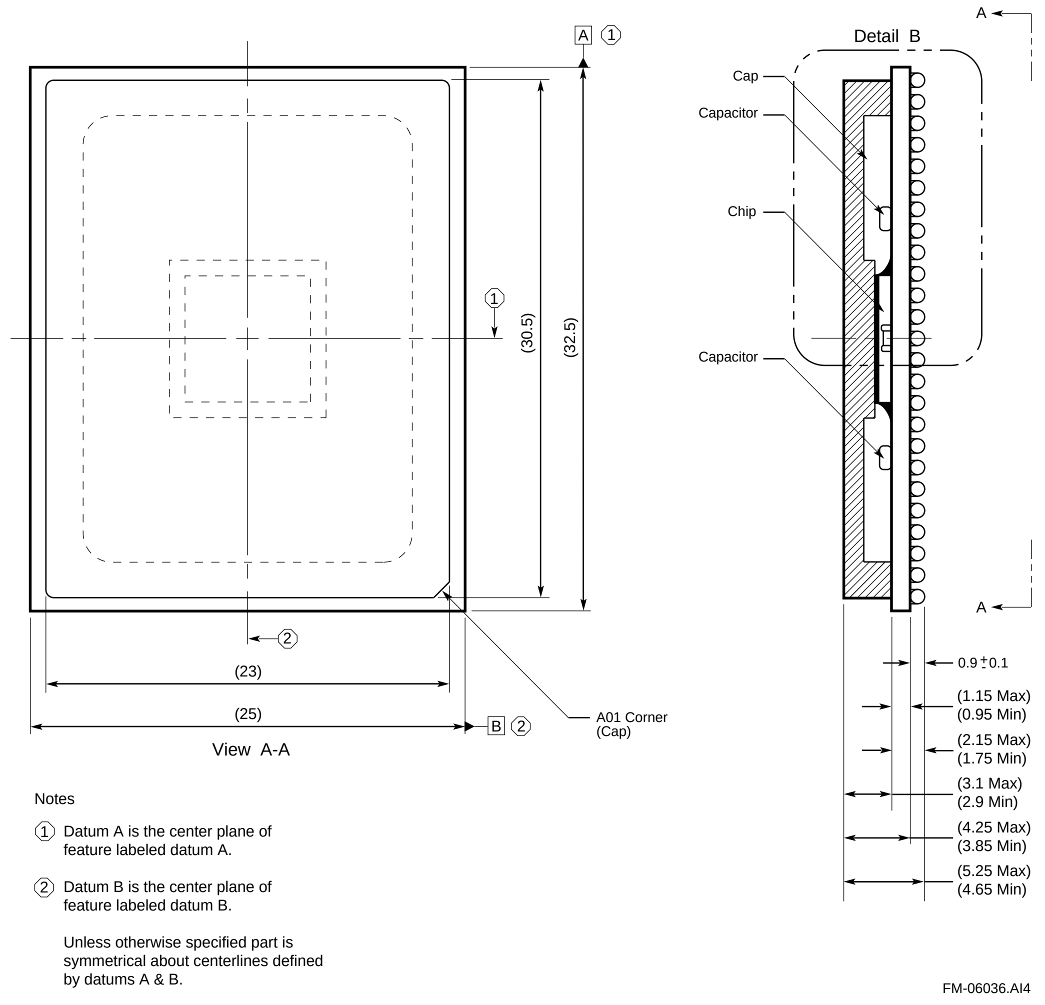
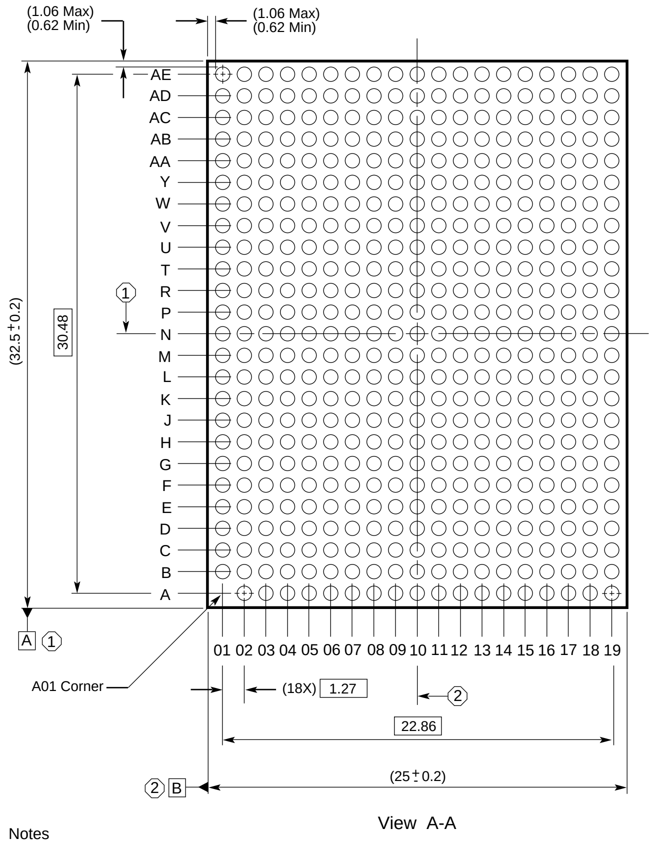
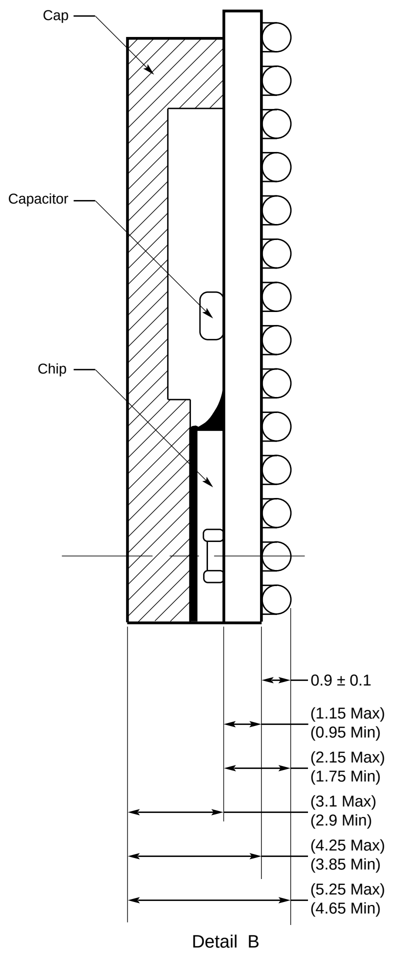


Figure 8–2 21174 Physical Specification





Notes

1. Datum A is the center plane of feature labeled datum A.
2. Datum B is the center plane of feature labeled datum B.

Unless otherwise specified part is symmetrical about centerlines defined by datums A & B.

FM-06037.AI4

8.2 Thermal Specifications

This section describes 21174 thermal management and thermal design recommendations.

8.2.1 Operating Temperature

For reliable operation, the 21174 is recommended to operate at a maximum device case temperature (T_C), measured at the center of the package, of 80^oC.

The following section offers specific thermal design recommendations.

8.2.2 Thermal Design Recommendations

Depending on the system environment, a heat sink may be required for adequate cooling. In the case of low air flow (less than 200 lfpm), a heat sink is required. Table 8–1 shows three recommended thermal management configurations for the 21174.

Table 8–1 Thermal Management Configurations for the 21174

Cooling Options	Airflow Requirement	Maximum Ambient Temperature	Estimated Case Temperatures (T_c)	Maximum Allowed T_c
Air flow (No heat sink)	Minimum 200 lfpm	40^oC	67^oC	80^oC
Clip-on heat sink	Natural convection	40^oC	73^oC	80^oC
Heat sink with adhesive tape	Natural convection	40^oC	70^oC	80^oC

8.2.3 Heat Sinks

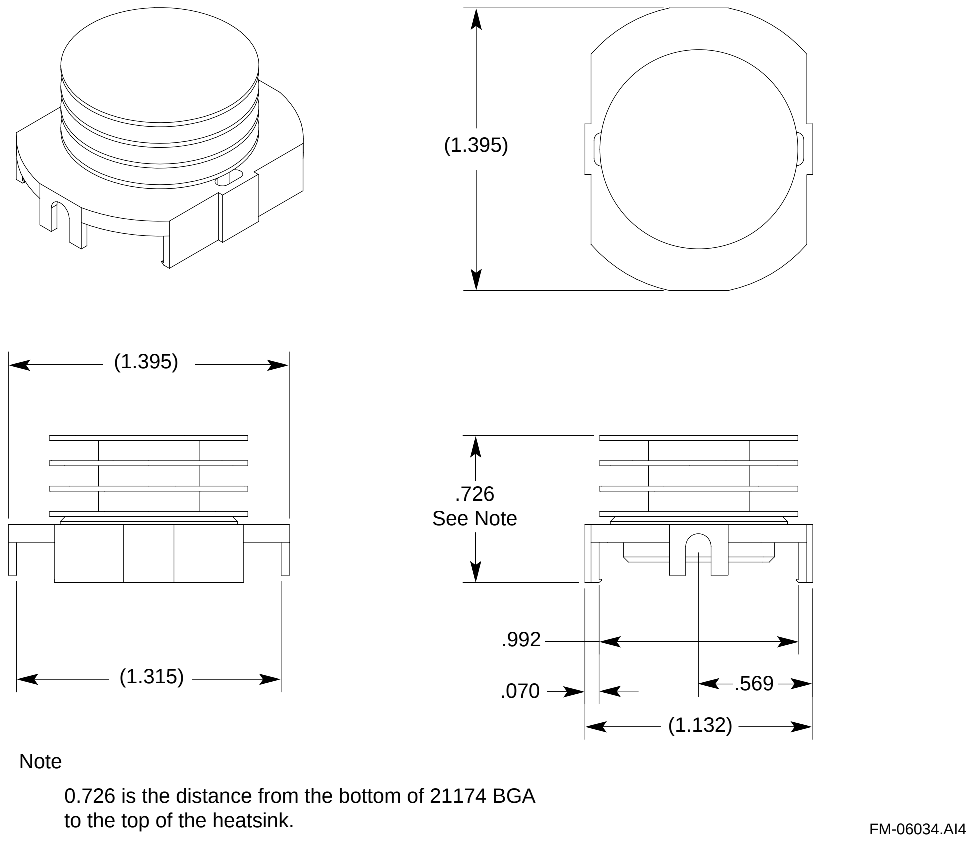
DIGITAL recommends that you qualify the heat sink and the heat sink attachment process to ensure that the configuration meets your requirements.

Heat sink vendors and physical specifications for the heat sinks used in Table 8–1 are detailed in Sections 8.2.3.1 and 8.2.3.2.

8.2.3.1 Clip-on Heat Sink Assembly

Figure 8–3 shows the clip-on heat sink assembly. All dimensions are in inches.

Figure 8–3 Clip-on Heat Sink Assembly


(1.395)
(1.395)
.726
See Note
.992
(1.315)
.070
.569
(1.132)
Note
0.726 is the distance from the bottom of 21174 BGA
to the top of the heatsink.
FM-06034.AI4

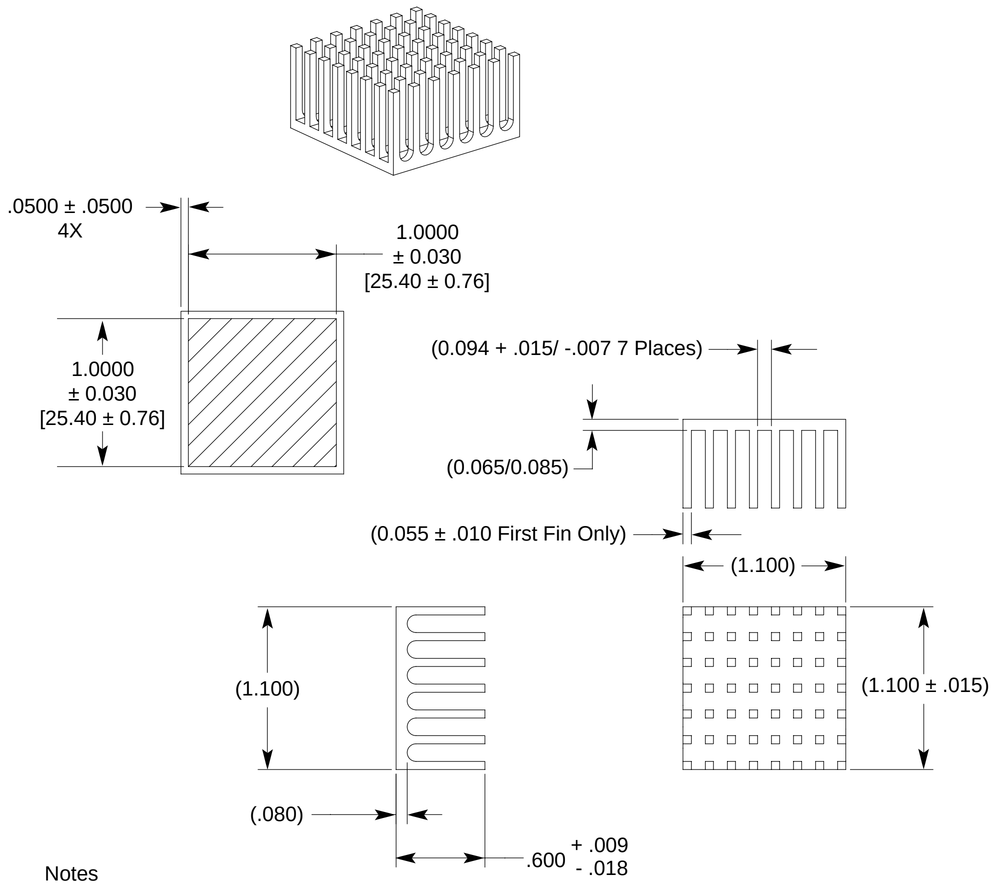
Clip-on heat sinks can be purchased from the following vendor:

Chip Cooler (Part Number: HTS149-1)
333 Strawberry Field Rd.
Warwick, RI 02886
1-800-227-0254

8.2.3.2 Tape Heat Sink Assembly

Figure 8–4 shows the tape heat sink assembly. All dimensions are in inches.

Figure 8–4 Tape Heat Sink Assembly



Notes

1. Material: Aluminum Alloy.
2. Finish: Clear Anodize.
3. Slot Spacing non-cumulative.

FM-06035.AI4

Heat sinks with adhesive tape can be purchased from the following vendor:

Wakefield Engineering (Part Number: 919452)
60 Audubon Rd.
Wakefield, MA 01880
617-245-5900

A
21174 DMA Page Boundary Solution

A.1 Read Page Problem

PCI DMA reads that attempt to cross 8K page boundaries cause data corruption problems. A fix has been implemented with an Altera 7032 and two Pericom PI5C3400 bus switches and a diode.

A.2 Recommended Solution

To solve this data corruption problem, use a 7 nsec 44-pin PLCC EPLD supplied by Altera (part number EPM7032LC44-7) and a Pericom PI5C3400 bus switch.

Contact DIGITAL Semiconductor's Customer Technology Center, (978) 568-7474, for the required programming files.

A.2.1 DMA Access Verilog Equations

This section describes the verilog model of the bug patch for the 21174 8K page crossing problem.

The code for the verilog file, patch13k.v, follows:

```
subdesign patch13k (
    FRAME_GRANT_CONNECT_L: output;
    STOP_CONNECT_L:        output;
    DEVSEL_CONNECT_L:      output;
    DEASSERT_STOP:         output;
    DEASSERT_DEVSEL:       output;
    DEASSERT_DEVSEL_OE_L:  output;

    TRDY_CONNECT_L:        output;
    DEVSEL_L:              input;   % not used %
    PYX_DEVSEL_L:          input;
    PYX_TRDY_L:            input;
```

Recommended Solution

```
        SYS_FRAME_L:           input;
        SYS_GRANT_L:           input;

        DDOL_IN:               input;   % connect externally to
                                          DEASSERT_DEVSEL_OE_L %
        PCI_ADDR[9..2]:        input;   % [9] not used %
        PCI_IRDY_L:            input;
        PCI_CBE2_L:            input;
        PCI_CBE0_L:            input;
        PCI_ACK64_L:           input;
        PCI_RESET_L:           input;
        PCI_CLK:               input;
        BANK01:                input;
        BANK23:                input;   % not used %
        BANK45:                input;
        BANK67:                input;
        )
    variable
        STATE:          machine of bits (
                                         FRAME_GRANT_CONNECT_L,
                                         DEVSEL_CONNECT_L,
                                         STOP_CONNECT_L,
                                         TRDY_CONNECT_L,
                                         DEASSERT_DEVSEL_XL,
                                         DEASSERT_DEVSEL_OE,
                                         DEASSERT_STOP_XL
        )
        with states (
        IDLE      = B"0000000"   ,  %wait for a transaction %
        WTFORRTRY = B"0001000"   ,     % right type of transaction, wait for mem
                                     request %
        WTFORMREQ = B"0000000"   ,     % right type of transaction, wait for mem
                                     request %
        TARGET    = B"0000000"   ,     % pyxis is target,process the transaction %
        EOPAGE    = B"1000010"   ,   % last cycle of page, do this, and then
                                    stop %
        RETRY     = B"1101110"   ,     % retry after prefetch, wait here until no
                                     DMA_RD %
        STOP      = B"1111111"   ,  % asserting stop %
        STOP2     = B"1110010"      % deasserting stop %
        );
        ADDR[8..2]:           dff;
        ACK64:                dff;
        VBUFF_HIT:            dff;
        PREFETCH:             dff;
        CMD_CYCLE_POSSIBLE:   dff;
        OLD_DMA_RD_REQ:       dff;
        OLD2_DMA_RD_REQ:      dff;
        OLD3_DMA_RD_REQ:      dff;
        OLD4_DMA_RD_REQ:      dff;
        ANY_DMA_RD_REQ:       dff;
        DEASSERT_DEVSEL_TRI:  tri;
        READ_CMD:             node;
        LAST_ADDR:            node;
        LAST_ADDR64:          node;
        PA_DMA_RD_REQ:        node;
        MCTL_DMA_RD_STROBE:   node;
        MCTL_SEL_VBUFF_DATA:  node;
    begin
        ADDR[8..2].clk   = PCI_CLK;
        ADDR[8..2].CLRN  = PCI_RESET_L;
        STATE.clk        = PCI_CLK;
        STATE.reset      = ! PCI_RESET_L;
        ACK64.clk        = PCI_CLK;
        ACK64.d          = ! PCI_ACK64_L;
        OLD_DMA_RD_REQ.clk     = PCI_CLK;
        OLD_DMA_RD_REQ.d       = PA_DMA_RD_REQ;
        OLD2_DMA_RD_REQ.clk    = PCI_CLK;
        OLD2_DMA_RD_REQ.d      = OLD_DMA_RD_REQ;
        OLD3_DMA_RD_REQ.clk    = PCI_CLK;
        OLD3_DMA_RD_REQ.d      = OLD2_DMA_RD_REQ;
        OLD4_DMA_RD_REQ.clk    = PCI_CLK;
        OLD4_DMA_RD_REQ.d      = OLD3_DMA_RD_REQ;
        ANY_DMA_RD_REQ.clk = PCI_CLK;
        ANY_DMA_RD_REQ   = (PA_DMA_RD_REQ #
        OLD_DMA_RD_REQ #
        OLD2_DMA_RD_REQ #
        OLD3_DMA_RD_REQ #
        OLD4_DMA_RD_REQ);
        VBUFF_HIT.clk    = PCI_CLK;
        VBUFF_HIT        = MCTL_DMA_RD_STROBE & MCTL_SEL_VBUFF_DATA;  % victim
                                                            buffer hit %
        PREFETCH.clk     = PCI_CLK;
        PREFETCH         =   (PYX_DEVSEL_L &
        !TRDY_CONNECT_L &
        ANY_DMA_RD_REQ)     % set when not DEVSEL & prefetch %
        #
        (PREFETCH & PYX_DEVSEL_L); % reset when DEVSEL happens %
        CMD_CYCLE_POSSIBLE.clk = PCI_CLK;
        CMD_CYCLE_POSSIBLE     = SYS_FRAME_L & (SYS_GRANT_L #
                                         FRAME_GRANT_CONNECT_L);
                                          % disallow pyxis commands! %
```

Recommended Solution

```
    PA_DMA_RD_REQ       = BANK45;   % program MDR1 to 80170000 %
    MCTL_DMA_RD_STROBE  = BANK01;   % program MDR1 to 80000027 %
    MCTL_SEL_VBUFF_DATA = BANK67;   % program MDR1 to AE000000 %
    % overall: MDR1 = AE170027 %
    DEASSERT_STOP   = !DEASSERT_STOP_XL;
    DEASSERT_DEVSEL_OE_L = !DEASSERT_DEVSEL_OE;
    DEASSERT_DEVSEL = DEASSERT_DEVSEL_TRI.out;
    DEASSERT_DEVSEL_TRI.oe = !DDOL_IN;
    DEASSERT_DEVSEL_TRI.in = !DEASSERT_DEVSEL_XL;
    if ((STATE == IDLE) & !SYS_FRAME_L) then
       ADDR[8..2] =  PCI_ADDR[8..2];
    end if;
    if (!PYX_TRDY_L & !PCI_IRDY_L & ACK64) then
       ADDR[8..2] =  ADDR[8..2] + B"0000010";
    end if;
    if (!PYX_TRDY_L & !PCI_IRDY_L & !ACK64) then
       ADDR[8..2] =  ADDR[8..2] + B"0000001";
    end if;
    if (!(STATE == IDLE) & PCI_IRDY_L) then
       ADDR[8..2] =  ADDR[8..2];
    end if;
    if (!(STATE == IDLE) & PYX_TRDY_L) then
       ADDR[8..2] =  ADDR[8..2];
    end if;
    READ_CMD  =  !SYS_FRAME_L & CMD_CYCLE_POSSIBLE &
      (PCI_CBE2_L == B"1") & (PCI_CBE0_L == B"0");  % RD, RL, or RM command %
    LAST_ADDR   =  (ADDR[8..3] == B"111111") &
      (ADDR2 # (STATE == TARGET));
    LAST_ADDR64 =  (ADDR[8..4] == B"11111") &
      (ADDR3 # (STATE == TARGET));
    table
    STATE,
    READ_CMD,
    SYS_FRAME_L,
    PCI_IRDY_L,
    PYX_TRDY_L,
    LAST_ADDR,
    LAST_ADDR64,
    PCI_ACK64_L,
    PREFETCH,
    ANY_DMA_RD_REQ,
    VBUFF_HIT
=>
```

```
STATE;
%-----------------------------------------------------------------%

IDLE,          0, x, x, x, x, x, x, x, x, x     =>  IDLE;
IDLE,          1, x, x, x, x, x, x, 0, x, x     =>  WTFORMREQ;  % DMA read
                                                             command %
IDLE,          1, x, x, x, x, x, x, 1, x, x     =>  WTFORRTRY;  % do a retry %
WTFORRTRY,     x, 0, x, x, x, x, x, x, 1, x     =>  RETRY;   % terminate this %
WTFORRTRY,     x, 1, 0, x, x, x, x, x, 1, x     =>  RETRY;   % terminate this %
WTFORRTRY,     x, 0, x, x, x, x, x, x, 0, x     =>  WTFORRTRY; % wait here for
                                                      PA_DMA_RD_REQ %
WTFORRTRY,     x, 1, 0, x, x, x, x, x, 0, x     =>  WTFORRTRY; % wait here for
                                                    PA_DMA_RD_REQ %
WTFORRTRY,     x, 1, 1, x, x, x, x, x, x, x     =>  IDLE;      % happens for
                                                 stop or peer-to-peer!%
WTFORMREQ,     x, 0, x, x, 1, x, 1, x, 1, x     =>  EOPAGE;   % wait for
                                                finaldata, then stop %
WTFORMREQ,     x, 0, x, x, x, 1, 0, x, 1, x     =>  EOPAGE;   % wait for final
                                                     data, then stop %
WTFORMREQ,     x, 0, x, x, 0, x, 1, x, 1, x     =>  TARGET;   % wait for data %
WTFORMREQ,     x, 0, x, x, x, 0, 0, x, 1, x     =>  TARGET;   % wait for data
                                                        (or victim) %
WTFORMREQ,     x, 0, x, x, x, x, x, x, 0, x     =>  WTFORMREQ; % wait here for
                                                      PA_DMA_RD_REQ %
WTFORMREQ,     x, 1, 0, x, x, x, x, x, 1, x     =>  TARGET;   % wait for data
                                                         or victim %
WTFORMREQ,     x, 1, 0, x, x, x, x, x, 0, x     =>  WTFORMREQ; % wait here for
                                                      PA_DMA_RD_REQ %
WTFORMREQ,     x, 1, 1, x, x, x, x, x, x, x     =>  IDLE;     % stop
                                                  (or peer-to-peer!) %
TARGET,        x, 0, x, x, x, x, x, x, x, 1     =>  RETRY;    % vbuff hit %
TARGET,        x, 1, 0, 1, x, x, x, x, x, 1     =>  RETRY;    % vbuff hit %
TARGET,        x, 1, 0, 0, x, x, x, x, x, 1     =>  IDLE;     % vbuff hit, but
                                                     doesn't matter %
TARGET,        x, 1, 1, x, x, x, x, x, x, x     =>  IDLE;     % end of
                                                        transaction %
TARGET,        x, 1, 0, 0, x, x, x, x, x, 0     =>  IDLE;     % end of
                                                        transaction %
TARGET,        x, 1, 0, 1, x, x, x, x, x, 0     =>  TARGET;   % wait for last
                                                               data %
```

Recommended Solution

```
TARGET,       x, 0, 1, x, x, x, x, x, x, 0     =>  TARGET;   % wait for irdy %
TARGET,       x, 0, 0, 0, 1, x, 1, x, x, 0     =>  EOPAGE;   % suppress
                                          frame, wait for last data %

TARGET,       x, 0, 0, 0, x, 1, 0, x, x, 0     =>  EOPAGE;   % suppress
                                          frame, wait for last data %
TARGET,       x, 0, 0, 1, x, x, x, x, x, 0     =>  TARGET;   % wait for trdy %
TARGET,       x, 0, 0, 0, 0, x, 1, x, x, 0     =>  TARGET;   % data cycle,
                                            wait for page crossing %
TARGET,       x, 0, 0, 0, x, 0, 0, x, x, 0     =>  TARGET;   % data cycle,
                                            wait for page crossing %
EOPAGE,       x, 0, x, x, x, x, x, x, x, 1     =>  RETRY;    % vbuff hit %
EOPAGE,       x, 1, 0, 1, x, x, x, x, x, 1     =>  RETRY;    % vbuff hit %
EOPAGE,       x, 1, 0, 0, x, x, x, x, x, 1     =>  IDLE;     % vbuff hit, but
                                                    doesn’t matter %
EOPAGE,       x, 1, 1, x, x, x, x, x, x, x     =>  IDLE;     % end of
                                                       transaction %
EOPAGE,       x, 1, 0, 0, x, x, x, x, x, 0     =>  IDLE;     % end of
                                                       transaction %
EOPAGE,       x, 1, 0, 1, x, x, x, x, x, 0     =>  EOPAGE;   % watch for
                                                            victim %
EOPAGE,       x, 0, 1, x, x, x, x, x, x, 0     =>  RETRY;       % note 1 %
EOPAGE,       x, 0, 0, 1, x, x, x, x, x, 0     =>  EOPAGE;       % note 2 %
EOPAGE,       x, 0, 0, 0, x, x, x, x, x, 0     =>  STOP;     % issue stop %
RETRY,        x, 1, 1, x, x, x, x, x, x, x     =>  IDLE;     % end of
                                                       transaction %
RETRY,        x, 0, 1, x, x, x, x, x, x, x     =>  RETRY;   % wait here for
                                                       IRDY %
RETRY,        x, x, 0, x, x, x, x, x, 1, x     =>  RETRY;   % wait for fetch
                                                       to stop  %
RETRY,        x, x, 0, x, x, x, x, x, 0, x     =>  STOP;     % do stop %
STOP,         x, 1, x, x, x, x, x, x, x, x     =>  STOP2;
STOP,         x, 0, x, x, x, x, x, x, x, x     =>  STOP;
STOP2,        x, x, x, x, x, x, x, x, x, x     =>  IDLE;
end table;
% note 1: IRDY was high, so 21174 will quit transaction with no frame or
irdy, goto RETRY to clean up %
% note 2: only happens if transaction started at last location in page %
end;
```

B
21174 DMA Lock Solution

B.1 DMA Lock Problem

The 21164 sometimes issues LOCK commands on the CMD bus. The 21174 treats the LOCK command as a no-op command and goes back to idle. This does not actually clear the LOCK command. Thus, the process repeats indefinitely, blocking DMA requests that may be waiting for service.

B.2 Recommended Solutions

The DMA lock issue is resolved by adding a quick switch, QS3253, between the 21164 and the 21174. Whenever CMD<3> is asserted low by the 21164, CMD<0> to 21174 is forced low by the quick switch. In all other instances, CMD<0> is connected normally.

CMD<3:0> (Before QS3253)	CMD<3:0> (After QS3253)
0001 (LOCK)	0000 (NOP)

This solution has been implemented and verified on the AlphaPC 164LX motherboard.

C
AlphaPC 164LX Layout Design Rules

C.1 Application Note

Currently DIGITAL Semiconductor has designed two Alpha motherboards based on the 21174 chipset. These boards both have four 168-pin unbuffered DIMM slots.

Based on design experience and results from our OEMs, DIGITAL Semiconductor is only in a position to support designs with a maximum of four DIMM slots. The following issues prevent DIGITAL Semiconductor from being able to support designs with more than four DIMM slots:

- Data bus loading.
- Transfers need to be completed in a 15-ns cycle (66 MHz).

Example C–1 and Example C–2 denote a four DIMM application and a greater than four DIMM application, respectively.

Example C–1 Four DIMM Slots (Two Banks of 128 bits) – Data Bus Loading

- CPU data bus:
  - 21164 or 21164PC
  - SRAM
  - Quickswitch
  - PCB trace loading
- Memory data bus:
  - Quickswitch
  - 21174
  - Two DIMM banks (up to four SDRAM loads)
  - PCB trace loading

Application Note

Example C–2 Greater than Four DIMM Slots – Data Bus Loading

- CPU data bus:
  - 21164 or 21164PC
  - SRAM
  - Quickswitch
  - PCB trace loading
- Memory data bus:
  - Quickswitch
  - 21174
  - Greater than two DIMM banks (greater than four SDRAM loads)
  - PCB trace loading

Example C–1 is a configuration that we have demonstrated in our designs operates at 66 MHz.

Example C–2 is a case that we, DIGITAL Semiconductor, have not built or verified. However, working with our OEMs, we have seen loading problems using greater than four DIMMs.

The main issue is the data bus loading when the SRAM needs to drive data back to the SDRAM. In this case, the Quickswitch is closed and the CPU data bus and the memory data bus become one. The additional trace loading and SDRAM loading in Example C–2 is enough to cause the transfers not to meet the timing requirements at 66 MHz (see Figure C–4).

Solution

It is critically important that you carefully lay out your 21164, 21174, Bcache, and DIMM components.

Based on the experience that we have gained designing and debugging our 21174 based systems, we have outlined some critical layout design rules to help you minimize potential debug problems. These layout design rules are described in Section C.2 through Section C.6. It is imperative that you closely follow these rules.

C.2 AlphaPC 164LX Layer Construction

The AlphaPC 164LX is an 8-layer board. The layer characteristics are listed in Table C–1.

Table C–1 Layer Construction

Layer	Dielectric Spacing	Layer Type	Routing
L1	= = = = = = = = = = = = = = 0.0045 +/- 0.001	Logic	Horizontal
L2	- - - - - - - - - - - - - - - - - - - 0.01 +/- 0.002	Ground	
L3	- - - - - - - - - - - - - - - - - - - 0.00 +/- 0.002	Logic	Vertical
L4	- - - - - - - - - - - - - - - - - - - 0.004 Minimum Reference	+2.5 V	
L5	- - - - - - - - - - - - - - - - - - - 0.00 +/- 0.002	+3.3 V	
L6	- - - - - - - - - - - - - - - - - - - 0.01 +/- 0.002	Logic	Horizontal
L7	- - - - - - - - - - - - - - - - - - - 0.0045 +/- 0.001	+5.0 V	
L8	= = = = = = = = = = = = = =	Logic	Vertical

The board is made using FR4 material. The target impedance for the board is 65 ohms +/- 10%. Propagation delay for the inner layers is 180 ps +/- 10% and outer layers is 150 ps +/- 10%.

Varying from this layer construction or board material may change some values given in this document. The designer should take this factor into consideration when laying out the board.

The numbers in parentheses, for example (15.1 inches), in this document are length measurements taken from DIGITAL's AlphaPC 164LX board design. DIGITAL Semiconductor strongly recommends that the designer follow these example values as closely as possible.

AlphaPC 164LX Layer Construction

If you need exact length or routing information on a particular signal within the AlphaPC 164LX board, refer to the latest AlphaPC 164LX Topography report and/or view the latest AlphaPC 164LX .WDS layout database using SALT.

Note: SALT is our layout database tool. It runs on an Alpha system running Windows NT with Exceed Xserver software installed. This is not a Digital Equipment Corporation supported product. This tool is to be used to assist in gaining a more graphical understanding of our layout.

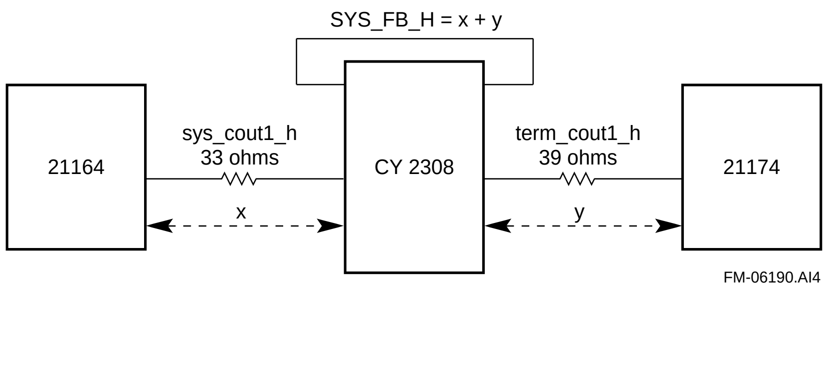
Table C–2 lists the system clock signals layout rules.

Table C–2 System Clock Signals Layout Rules

Pins	Rules
CLK_IN_H, CLK_IN_L	Must be routed on inner layers with 12 mil etch and 40 mil spacing. Do not route any signals on adjacent layers, within the clearance area outlined above. Must be balanced (0.354 inch).
OSC_H, OSC_L	Must be routed on inner layers with 12 mil etch and 40 mil spacing. Do not route any signals on adjacent layers, within the clearance area outlined above. Must be balanced (0.395 inch).
SYS_FB_H	Must be equal to length of **sys_cout1_h** plus the length of **term_cout1_h.**

Figure C–1 shows the layout for signal **sys_cout1_h**.

Figure C–1 sys_cout1_h Layout



C.3 Bcache Signal Layout Lengths

Table C–3 lists the Bcache signal layout lengths.

Table C–3 Bcache Signal Layout Lengths

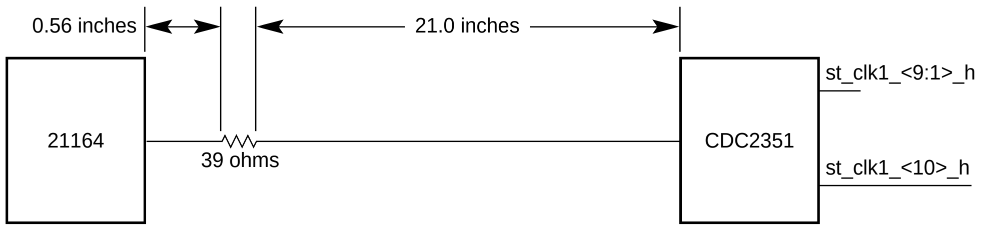
Pin	Length
Bcache **st_clk***[1]	10 mil spacing from itself and 15 mil from other signals. This signal must be routed on inner layers.
st_clk1_h	This signal must be delayed using 21.0[2] inches of etch.
st_clk1_<9:1>_h	These signals must be balanced (4.000 inches).
st_clk1_<10>_h	This signal etch must be 7.162 inches.

[1] The asterisk is a wildcard indicating all signal names beginning with the preceding letters or symbols.

[2] DIGITAL uses FR4 material to build boards. The value 21.0 inches may have to be changed if you use a different board material.

Figure C–2 shows the **st_clk1_h** signal layout.

Figure C–2 st_clk1_h Layout



C.4 21174 Clock Layout

The phase-locked loop (PLL) filter must be as thick as possible on outer layers. The designer should copy the AlphaPC 164LX board layout exactly. This data can be viewed on our layout database, PC164LX_xxx.WDS, using SALT (where xxx indicates the latest file version).

C.4.1 DRAM Clock Signal Layout Rules

Table C–4 lists the DRAM clock signal layout rules.

Table C–4 DRAM Clock Signal Layout Rules

Signal Pins	Rule
DRAM clocks	10 mil spacing.
dram_clk_<7:0>	Should be less than 1.0 inch.
dram_clk_<7:0>a	Must be balanced. Match the longest run (5.20 inches).
dram_clk_<7:0>b	Must be balanced and matched with **dram_clk_<7:0>a**.
dram_clk_fb	Must be matched with **dram_clk_<7:0>a** and **dram_clk_<7:0>b**.

C.4.2 PCI Clock Signal Layout Rules

Table C–5 lists the PCI clock signal layout rules.

Table C–5 PCI Clock Signal Layout Rules

Signal Pins	Rule
PCI clocks	10 mil spacing.
pci_clk_<6:0>	Should be less than 1.0 inch.
pciclk_slot<3:0>	The etch must be balanced (5.657 inches).
pciclk_ide, pciclk_usb, pciclk_sio, pciclk_arb, pci_fb	The etch must be 2.5 inches longer than **Pciclk_slot<3:0>** etch (8.157 inches).

C.5 Bcache Signal Layout Rules

Table C–6 lists the Bcache signal layout rules.

Table C–6 Bcache Signal Layout Rules

Signal Pins	Rule
un_index_h*[1]	Should be less than 1.0 inch.
index_h*	The signals must be routed as shown in Figure C–3. Lengths a, b, and c should be equal. Length a should not be longer than 1.2 inches.
data_ram_oe_h	Must be tree'ed similar to **index_h*** (10.612 inches, total length).
data_ram_we_h	Must be tree'ed similar to **index_h*** (10.091 inches, total length).
data_ram_we_h, data_ram_oe_h	These signals should be matched.
tag_data_h*	The longest length on the AlphaPC 164LX is 4.549 inches.
data_h*	Tree with CPU on one end, SRAMs in the middle, and quick switch on the other end (maximum length = 9.53 inches and minimum length = 4.495 inches). See Figure C–4.
mem_data_h*	Maximum length = 7.99 inches and minimum length = 4.23 inches.

[1] The asterisk is a wildcard indicating all signal names beginning with the preceding letters or symbols.

Figure C–3 Bcache Signal Layout

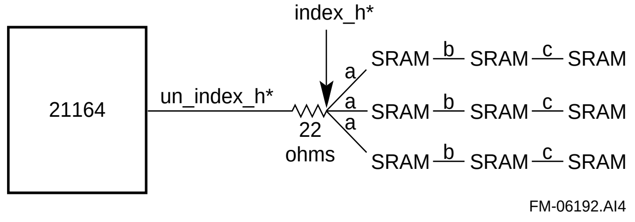
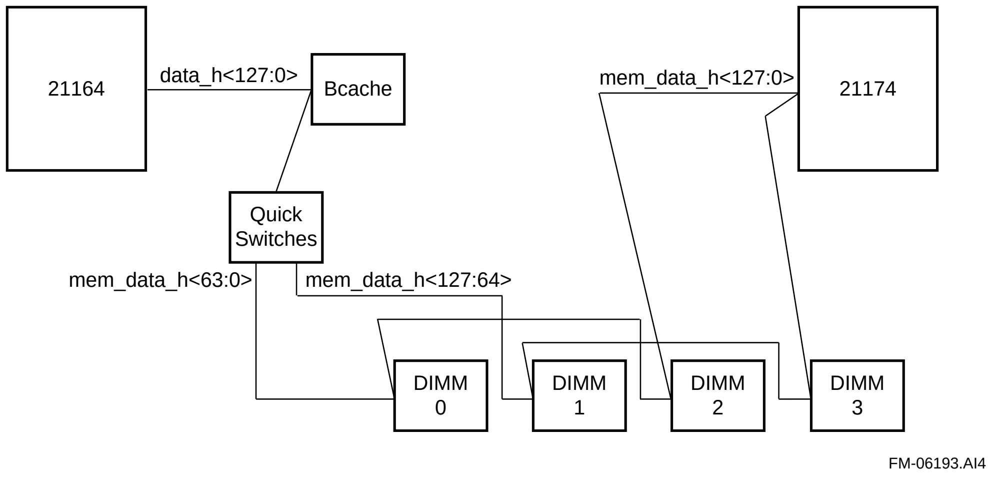


Figure C–4 Bcache Signal Layout



C.6 PCI General Layout

The following signal runs should not be longer than 15 inches.

PCI_*	CBE*	PRSNT*	*RDY_L	DEVSEL_L
LOCK_L	PERR_L	SERR_L	FRAME_L	STOP_L
SDONE	SBO	PAR	PAR64	ACK64*
REQ64_*				

D
Support, Products, and Documentation

If you need technical support, a *DIGITAL Semiconductor Product Catalog*, or help deciding which documentation best meets your needs, visit the DIGITAL Semiconductor World Wide Web Internet site:

http://www.digital.com/semiconductor

You can also call the DIGITAL Semiconductor Information Line or the DIGITAL Semiconductor Customer Technology Center. Please use the following information lines for support.

For documentation and general information:	
DIGITAL Semiconductor Information Line	
United States and Canada:	1–800–332–2717
Outside North America:	1–510–490–4753
Electronic mail address:	semiconductor@digital.com

For technical support:	
DIGITAL Semiconductor Customer Technology Center	
Phone (U.S. and international):	1–978–568–7474
Fax:	1–978–568–6698
Electronic mail address:	ctc@hlo.mts.dec.com

DIGITAL Semiconductor Products

Note: The following products and order numbers might have been revised. For the latest versions, contact your local distributor.

To order the AlphaPC 164LX motherboard, contact your local distributor. The following tables list some of the semiconductor products available from DIGITAL Semiconductor.

Chips	**Order Number**
DIGITAL Semiconductor 21174 Core Logic Chip	21174–AA
DIGITAL Semiconductor 21164 Alpha microprocessor (466 MHz)	21164–IB
DIGITAL Semiconductor 21164 Alpha microprocessor (533 MHz)	21164–P8
DIGITAL Semiconductor 21164 Alpha microprocessor (600 MHz)	21164–MB

Motherboard kits include the motherboard and motherboard user’s manual.

Motherboard Kits	**Order Number**
DIGITAL Semiconductor AlphaPC 164LX Motherboard Kit for Windows NT	21A04–C0
DIGITAL Semiconductor AlphaPC 164LX Motherboard Kit for DIGITAL UNIX	21A04–C1

Design kits include full documentation and schematics. They do not include related hardware.

Design Kits	**Order Number**
AlphaPC 164LX Motherboard Software Developer’s Kit (SDK) and Firmware Update	QR–21A04–12 (Available Fall, 1997)

DIGITAL Semiconductor Documentation

The following table lists some of the available DIGITAL Semiconductor documentation.

Title	Order Number
Alpha AXP Architecture Reference Manual[1]	EY–T132E–DP
Alpha Architecture Handbook[2]	EC–QD2KB–TE
DIGITAL Semiconductor 21164 Alpha Microprocessor Hardware Reference Manual	EC–QP99B–TE
DIGITAL Semiconductor 21164 Alpha Microprocessor Data Sheet	EC–QP98B–TE

[1] To purchase the *Alpha AXP Architecture Reference Manual*, contact your local distributor or call Butterworth-Heinemann (Digital Press) at 1-800-366-2665.
[2] This handbook provides information subsequent to the *Alpha AXP Architecture Reference Manual*.

Third–Party Documentation

You can order the following third-party documentation directly from the vendor.

Title	Vendor
PCI Local Bus Specification, Revision 2.1	PCI Special Interest Group
PCI Multimedia Design Guide, Revision 1.0	U.S. 1–800–433–5177
PCI System Design Guide	International 1–503–797–4207
PCI-to-PCI Bridge Architecture Specification, Revision 1.0	Fax 1–503–234–6762
PCI BIOS Specification, Revision 2.1	

Index