DECchip 21171 Core Logic Chipset

Technical Reference Manual

Order Number: EC–QE18B–TE

Revision/Update Information: Preliminary—This document supersedes the *DECchip 21171 Core Logic Chipset Technical Reference Manual* (EC–QE18A–TE).

Digital Equipment Corporation
Maynard, Massachusetts

September 1995

This document was prepared using VAX DOCUMENT Version 2.1.

Contents

3 DECchip 21171-CA Architecture Overview

4 DECchip 21171-CA Control and Status Registers

5 DECchip 21171-CA Power-Up and Initialization

6 DECchip 21171-CA Address Mapping

Part II DECchip 21171-BA (DSW) Information

7 DECchip 21171-BA Pin Descriptions

8 DECchip 21171-BA Architecture Overview

A Technical Support and Ordering Information

Index

Examples

Figures

Tables

Preface

Purpose and Audience

The document is a support and reference document for engineers using the Alpha 21164 microprocessor to design uniprocessor systems.

Organization

This document contains the following parts, chapters, and appendix:

- Chapter 1, DECchip 21171 Core Logic Chipset Overview, provides an overview of the DECchip 21171-AA core logic chipset features.

Part I—DECchip 21171-CA (CIA) Information

- Chapter 2, DECchip 21171-CA Pin Descriptions, provides control, I/O interface, and address chip (CIA) pin descriptions.
- Chapter 3, DECchip 21171-CA Architecture Overview, provides the CIA architecture overview.
- Chapter 4, DECchip 21171-CA Control and Status Registers, provides a programmer's reference for control and status registers (CSRs).
- Chapter 5, DECchip 21171-CA Power-Up and Initialization, provides information on the behavior of CIA during power-up and initialization.
- Chapter 6, DECchip 21171-CA Address Mapping, provides information on system address mapping.

Part II—DECchip 21171-BA (DSW) Information

- Chapter 7, DECchip 21171-BA Pin Descriptions, provides data switch chip (DSW) pin descriptions.
- Chapter 8, DECchip 21171-BA Architecture Overview, provides an overview of the DSW architecture.

- Appendix A, Technical and Ordering Information, provides information on how to obtain technical support and how to order products and documentation.

Document Conventions

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

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.

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.

Ranges

Ranges are specified by a pair of numbers separated by two periods (..) and are inclusive. For example, a range of integers 0..4 includes the integers 0, 1, 2, 3, and 4.

Extents

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

Must Be Zero

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

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. Non-zero values in SBZ fields produce UNPREDICTABLE results.

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.

Register Field Notation

Register figures and tabulated descriptions have a mnemonic that indicates the bit or field characteristic as described in Table 1.

Table 1 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 2.

Table 2 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.

Bit Notation

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

Warning

Warning provides information to prevent personal injury.

Caution

Caution indicates potential damage to equipment or data.

Note

Note provides general information that could be helpful.

Data Units

Table 3 defines the data unit terminology used throughout this manual.

Table 3 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.

Signal Name References

Signal names in text are printed in boldface lowercase. 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_h<1>**.

1

DECchip 21171 Core Logic Chipset Overview

The DECchip 21171 core logic chipset (21171) is used to support the Alpha 21164 microprocessor (21164) in high-performance uniprocessor systems. The chipset includes an interface to the 64-bit peripheral component interconnect (PCI) bus, and associated control and data paths for the 21164, memory, and backup cache (Bcache). Little discrete logic is needed on the module.

1.1 Features

The DECchip 21171 core logic chipset has the following features.

- Central processing unit—Supports Alpha 21164 microprocessor
- Components
  - One DECchip 21171-CA chip—The control, I/O interface, and address chip (CIA) has 383 pins and is in a plastic pin grid array (PPGA) package.
  - Four DECchip 21171-BA chips—The data switch chip (DSW) has 208 pins and is in a plastic quad flat pack (PQFP) package.
- Data paths
  - 64-bit, ECC-protected data path between CIA and DSW
  - 128-bit ECC-protected data path between the 21164 and DSW
  - 256-bit ECC-protected memory data path between DSW and memory
- Timing
  - Supports all 21164 clock frequencies
  - Supports system clock frequencies up to 33 MHz (PCI clock)
  - PCI clock, cache, and memory clock period are an integer multiple of the 21164 clock period

- Bcache
  - Third-level cache for the 21164
  - Cache size from 2MB to 64MB
  - Write-back, ECC-protected cache
- Memory
  - Supports up to 8GB of ECC-protected physical memory
  - Industry-standard memory array configurations
- High-performance PCI
  - 64-bit multiplexed address and data
  - 64-bit PCI address handling
  - Scatter-gather map support
  - No glue logic required to connect PCI-compliant devices

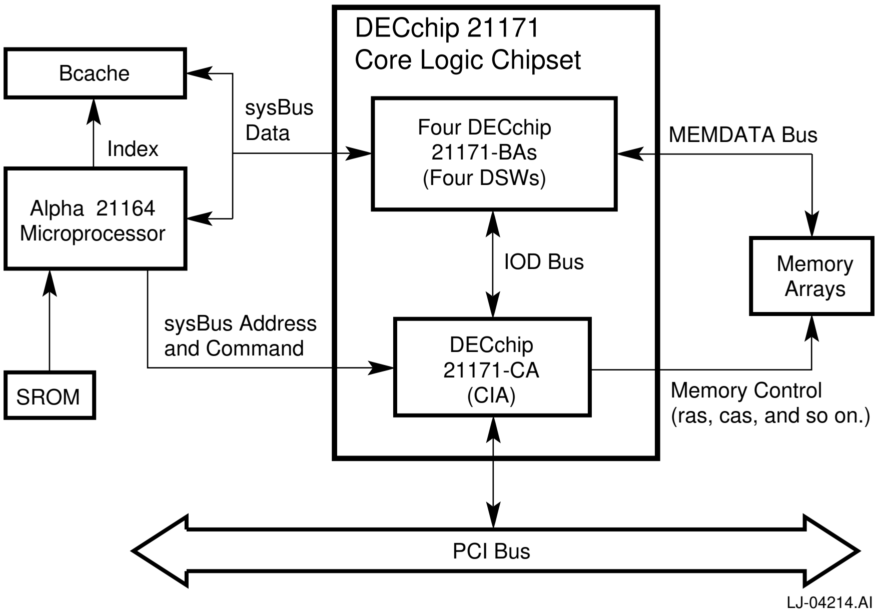
1.2 System Overview

To build a system, such as that shown in Figure 1–1, with the DECchip 21171 core logic chipset, the following additional components are required:

- Alpha 21164 microprocessor
- Bcache
- Memory arrays
- Serial ROM (SROM) (for 21164)
- Circuit to generate input clock for 21164
- Circuit to generate input clock for 21171
- Support logic
  - PCI interrupt controller
  - PCI arbiter
  - Reset logic
  - Console ROM logic and flash ROM

- PCI peripheral devices

Figure 1–1 DECchip 21171 Core Logic Chipset System Block Diagram



The MEMDATA bus provides a 256-bit, ECC-protected data path between the four data path slices (DSWs) and the memory arrays. Each bit slice (DSW) connects to 64 bits of the MEMDATA bus.

The input/output data (IOD) bus provides a 64-bit, ECC-protected data path between the CIA and DSW chips.

The 21164 is connected to the CIA and DSW by the sysBus. The sysBus contains address, data, and command signal lines. The sysBus is the 21164 interface bus with several additional signals to handle direct memory access (DMA) transactions through the CIA and DSW.

Systems built with the DECchip 21171 core logic chipset have the capability to support up to 8GB of memory. The chipset supports the following industry standard dynamic RAMs (DRAM) in the memory arrays:

1MB ∗ 36	2MB ∗ 36
4MB ∗ 36	8MB ∗ 36

1.2.1 DECchip 21171 Core Logic Chipset Microarchitecture

As shown in Figure 1–1, the DECchip 21171 core logic chipset consists of the CIA and four DSW application-specific integrated circuits (ASICs).

- DECchip 21171-CA (CIA), 383-pin PPGA: Provides control functions to main memory, and a bridge to the 64-bit PCI bus. In addition, the CIA provides all control functions for the DSW and part of the I/O data path.
- DECchip 21171-BA (DSW), 208-pin PQFP: A bit slice ASIC (four required), provides the data path between the 21164, main memory, Bcache, and the CIA, and part of the I/O data path.

1.3 CIA Microarchitecture

The CIA provides control functions to main memory and the PCI interface. As shown in Figure 1–2, the CIA is divided into the following major functional logic blocks.

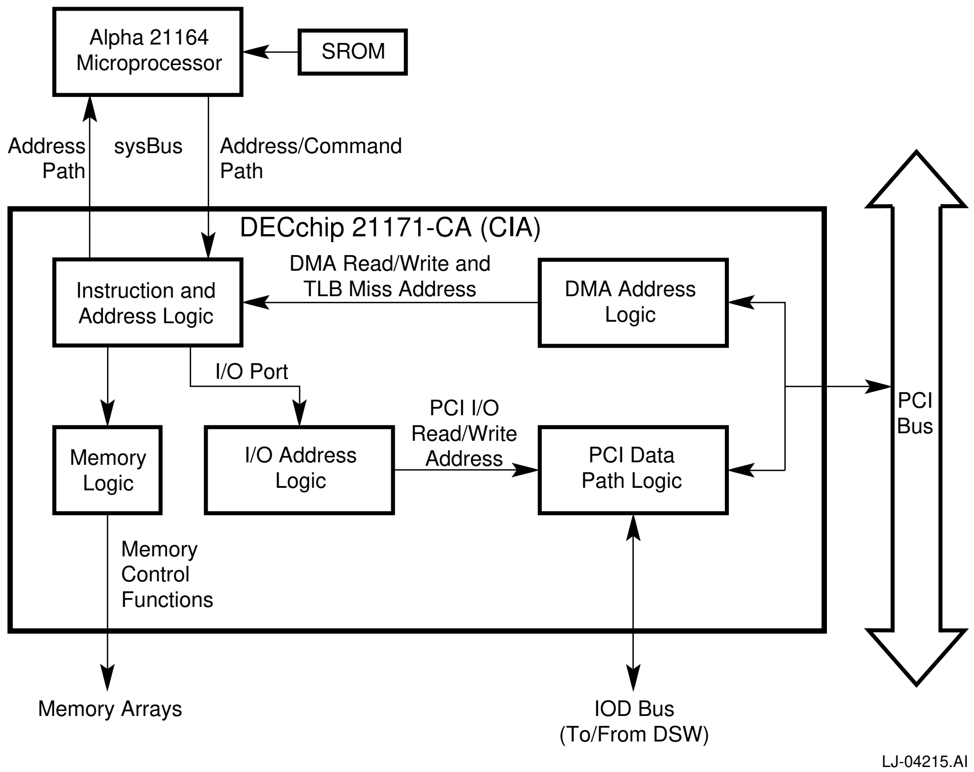
The instruction and address logic accepts commands from the 21164 and directs instructions to either the memory port or the I/O port. A 3-entry 21164 instruction queue provides buffering in the event that the memory and I/O ports are busy. A flush address register is also provided to allow DMA read and write operations to interrogate a supported Bcache for the most recent data.

The memory logic provides the row and column addresses for the supported memory banks and all memory control functions (row address select [**ras**], column address select [**cas**], write-enable, and so on). In addition, the memory logic provides control signals to the DSW to initiate data transactions to and from memory.

The I/O address logic handles I/O read and write addresses. The decode logic extracts the PCI address from the dense or sparse space 21164 address encoding, generates PCI I/O read or write addresses, and passes them to the PCI data path logic. The I/O logic can also increment the current address each data cycle. The incremented address provides a pointer to the next data item to be transferred to or from the I/O read/write buffers.

A 3-entry I/O address queue is provided together with a single-entry current address register. These components allow four I/O write operations to be outstanding. The DSW chip provides four corresponding I/O write data buffers.

Figure 1–2 CIA Block Diagram



In addition, a bypass path is provided for the special case of a dense space I/O write transaction with all 32 data bytes valid.

The PCI data path logic provides the interface to the 64-bit PCI, and contains a portion of the data path control logic. Its major functions are to provide:

- ECC code generation on data transactions to the DSW
- ECC code check and correction on data transactions from the DSW
- I/O read/write addresses to the PCI
- Buffers for all I/O and DMA read and write data transactions

The DMA address logic converts PCI addresses to 21164 memory address space. Two address conversion methods are provided:

- A direct path where a base offset is concatenated with the PCI address
- A scatter-gather map, which maps any 8KB PCI page to any 8KB memory space page

The scatter-gather map translation lookaside buffer (TLB) has eight entries, with each entry containing four consecutive page table entries (PTEs).

1.4 DSW Microarchitecture

The DSW provides bidirectional data paths between the 21164, the memory arrays, and the CIA, as shown in Figure 1–3. Four DSW chips, using a bit slice design, provide the complete data path.

The majority of the DSW logic is comprised of data buffers and multiplexers. The CIA, using two encoded control fields, directs data flow to and from the DSW.

The victim buffer has a 64-byte single-entry buffer to contain victim data during a 21164 memory read transaction that generates a victim. Victim data is saved in the buffer, allowing the read transaction to fetch read data from memory and send it to the 21164. Later the victim data is written to memory.

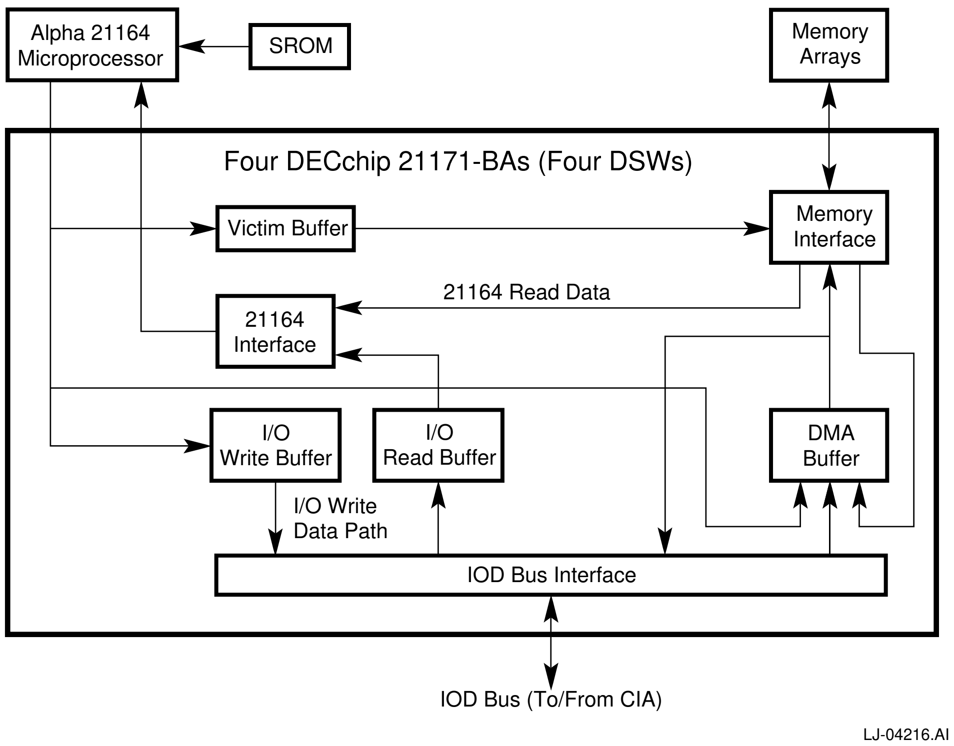
The memory interface contains the data-out register and the data-in register. The data-out register buffers either 21164 victim data to memory, or DMA write data to memory. The data-in register accepts 21164 memory read data. The DSW clocks the register on optimum 15-ns clocks to minimize memory latency.

The I/O write buffer consists of four 32-byte buffers. The four buffers allow the chipset to sustain maximum bandwidth on a large copy operation from memory, through the 21164, to I/O space.

The 32-byte I/O read buffer is provided to assemble the 128-bit data required by the 21164 from the 64-bit data transfers from the IOD bus (CIA-DSW interface).

The DMA read/write buffers consist of two identical buffers (DMA buffer 0 and DMA buffer 1). Each buffer consists of three, 64-byte, single-entry data buffers used to hold data from memory, Bcache, and the PCI. When a buffer is written, its data valid bits are asserted, along with data, to the buffer. The valid bits are used to select appropriate read/write data.

Figure 1–3 DSW Block Diagram



1.5 System Clocks

The 21164 internal clock will be divided by a programmable value to create an output clock, **sys_clk_out1_h**, which is used as the system clock. The divisor must produce a clock between 25 MHz and 33 MHz.

The system designer must supply a circuit to maintain stability of the system clock and distribute it to PCI devices.

The system clock can be used as a source for a circuit that generates a clock for (E)ISA devices.

1.6 DECchip 21171 Core Logic Chipset Characteristics

Table 1–1 and Table 1–2 list the characteristics of the CIA and DSW chips.

Table 1–1 DECchip 21171-CA (CIA) Chip Characteristics

Characteristic	Specification
Power supply	**Vss** 0.0 V, **Vdd** 5 V ±5%
Operating temperature	Tj maximum = 70°C (154°F)
Storage temperature range	–55°C (–67°F) to 125°C (257°F)
Power dissipation @**Vdd** = 5.25 V, Freq = 33 MHz	2.75 W maximum

Table 1–2 DECchip 21171-BA (DSW) Chip Characteristics

Characteristic	Specification
Power supply	**Vss** 0.0 V, **Vdd** 5 V ±5%
Operating temperature	Tj maximum = 100°C (212°F)
Storage temperature range	–55°C (–67°F) to 125°C (257°F)
Power dissipation @**Vdd** = 5.25 V, Freq = 33 MHz	1.0 W maximum

Part I

DECchip 21171-CA (CIA) Information

Part I contains five chapters that provide information about the DECchip 21171-CA (CIA) chip. The following table lists the topics described in each chapter:

Chapter	Description
2	Pin signals
3	Architecture
4	Control and status registers
5	Power-up and initialization
6	System address mapping

2
DECchip 21171-CA Pin Descriptions

This chapter provides a description of the DECchip 21171-CA pin signals.

2.1 DECchip 21171-CA Pin List

Table 2–1 lists the pin signals grouped by function. The information in the Type column identifies a signal as input (I), output (O), bidirectional (B), or power (P).

Table 2–1 DECchip 21171-CA Pin List

Signal Name	Quantity	Type	Signal Name	Quantity	Type
			sysBus Signals (55 Total)		
addr_cmd_par_h	1	B	**addr_h<39,34:4>**	32	B
addr_res_h<1:0>	2	I	**cmd_h<3:0>**	4	B
addr_bus_req_h	1	B	**cack_h**	1	O
dack_h	1	O	**error_h**[1]	1	O
fill_h	1	O	**fill_id_h**	1	B
fill_error_h	1	O	**idle_bc_h**	1	O
int4_valid_h<3:0>	4	I	**int_h**[1]	1	O
tag_dirty_h	1	O	**tag_ctl_par_h**	1	O
victim_pending	1	I	—	—	—
			IOD Bus Signals (88 Total)		
cmc_h<8:0>	9	O	**ioc_h<6:0>**	7	O
iod_h<63:0>	64	B	**iod_e_h<7:0>**	8	B

[1]See Section 2.2.1.

(continued on next page)

Table 2–1 (Cont.) DECchip 21171-CA Pin List

Signal Name	Quantity	Type	Signal Name	Quantity	Type
		Memory Control Signals (42 Total)			
mem_ack_l	1	O	**mem_addr_h<12:0>**	13	O
set_sel_h<15:0>	16	O	**ras_h<3:0>**	4	O
cas_h<3:0>	4	O	**mem_we_l<1:0>**	2	O
mem_en_h	1	O	**mem_req_l**	1	I
		PCI Local Bus Signals (88 Total)			
ad_h<63:0>	64	B	**cbe_l<7:0>**	8	B
par_h	1	B	**par64_h**	1	B
req64_l	1	B	**ack64_l**	1	B
frame_l	1	B	**irdy_l**	1	B
devsel_l	1	B	**trdy_l**	1	B
stop_l	1	B	**perr_l**	1	B
serr_l	1	B	**req_l**	1	O
rst_l	1	O	**gnt_l**	1	I
mem_cs_l	1	I	**lock_l**	1	I
		Phase-Locked Loop Signals (4 Total)			
pll_agnd_h	1	O	**pll_clk_h**	1	I
pll_lp1_h	1	I	**pll_vss_h**	1	I
pll_test_h<3:0>	4	I	—	—	—
Miscellaneous Power Signals			**(4 Total)**		
aux_vdd_h	1	P	**aux_vss_h**	1	P
pll_vdd_h	1	P	**pll_lp2_h**	1	P
		Miscellaneous Signals (8 Total)			
count_out_h	1	O	**debug_h<1:0>**	2	I
scan_in_h	1	I	**sys_rst_l**	1	I
test_mode_h<1:0>	2	I	**test_or_scan_out_h**	1	O

(continued on next page)

Table 2–1 (Cont.) DECchip 21171-CA Pin List

Signal Name	Quantity	Type	Signal Name	Quantity	Type
		Reserved Signals (2 Total)			
gru_ack_h	1	I	**gru_sel_h**	1	O

Pin Totals

Total input pins:	21
Total output pins:	72
Total bidirectional pins:	196
	—-
Total signal pins:	289

Total signal pins:	289
Total PWR5 pins:	36
Total GND pins:	51
Total miscellaneous power pins:	4
Total spare pins:	1
Total reserved pins:	2
	—-
Total pins:	383

2.2 DECchip 21171-CA Signal Descriptions

This section provides a description of the pin function. Signal descriptions are grouped by function and correspond to the pin list (see Table 2–1).

Note

The Alpha 21164 microprocessor uses the rising edge of **sys_clk_out1_h** (clk1R) as a reference when generating and sampling the system interface signals.

2.2.1 sysBus Signals

The *Alpha 21164 Microprocessor Hardware Reference Manual* defines and describes sysBus signals in detail, except for **int_h** and **error_h**. Those two signals are described here.

int_h

The **int_h** pin is connected to 21164 pin **irq<0>**. The CIA asserts **int_h** to notify the 21164 that the CIA has detected a corrected ECC error.

error_h

The **error_h** pin is connected to 21164 pin **irq<3>**. The CIA asserts this signal line to notify the 21164 that the CIA has detected an error other than a corrected ECC error.

2.2.2 PCI Local Bus Signals

The PCI local bus signals

ack64_l

Acknowledge 64-bit transfer—Asserted by the device that has decoded its address as the target of the current access, indicating that the target device is willing to transfer 64-bit data.

ad_h<63:0>

The 64-bit address and 64 bits of data are multiplexed on these signal lines. The upper 32 bits (<63:32>) are reserved except for those transfers where **req_64_l** is asserted.

cbe_l<7:0>

Bus command and byte enable are multiplexed on the same signal lines. During an address phase (when using the DAC command and when **req_64_l** is aserted) the bus command is transferred on **cbe_l<7:0>**. During a data phase these lines carry the byte enable bits, indicating which bytes have meaningful data.

devsel_l

When asserted, indicates that the asserting device has decoded its address as the target of the current access. As an input, **devsel_l** indicates whether any device on the bus has been selected.

frame_l

The cycle frame signal is asserted by the current master to indicate the beginning and duration of an access. When **frame_l** is deasserted, the final phase of the access has started.

gnt_l

Assertion of **gnt_l** indicates that access to the bus has been granted. This is a point-to-point signal. Every master has its own grant line.

irdy_l

Assertion of **irdy_l** (initiator ready) indicates that the bus master is ready to complete the current phase of the transaction. Signal **irdy_l** is used in conjunction with **trdy_l**. A data phase is completed on any clock where both signals are sampled asserted.

lock_l

Assertion of **lock_l** indicates an atomic operation that may require multiple transactions to complete. When **lock_l** is asserted, nonexclusive transactions may proceed to an address that is not currently locked.

mem_cs_l

The *PCI Local Bus Specification* does not preclude product-specific function /performance enhancements by way of sideband signals. A sideband signal is loosely defined as any signal that is not part of the PCI specification and connects two or more PCI nodes and has meaning only to these nodes. Sideband signals lines must not violate PCI local bus specifications.

Signal **mem_cs_l** is a sideband signal line. It is used by the PCI-to-EISA bridge to help manage the EISA memory map and peripheral component architecture (PCA) compatibility holes (see Section 6.3.4.2).

par_h

The **par_h** signal is used to create even parity across **ad_h<31:0>** and **cbe_l<3:0>**. Signal **par_h** is stable and valid one clock cycle after the address phase. Signal **par_h** remains valid until one clock cycle after the completion of the current data phase. The master asserts **par_h** for address and write data phases; the target asserts **par_h** for read data phases.

par64_h

The **par64_h** signal is used to create even parity across **ad_h<63:32>** and **cbe_l<7:4>**. Signal **par64_h** is stable and valid one clock cycle after the address phase. Signal **par64_h** remains valid until one clock cycle after the completion of the current data phase. The master asserts **par64_h** for address and write data phases; the target asserts **par64_h** for read data phases.

perr_l

The **perr_l** signal (parity error) is used to report data parity errors that occur during all PCI transactions except a special cycle. Signal **perr_l** is a tristate signal line that must be asserted by the data-receiving node two cycles after data with bad parity is detected. Signal **perr_l** must be asserted for a minimum of one clock cycle for each phase during which bad data parity is detected. Signal **perr_l**, like all sustained tristate signals, must be asserted high for at least one cycle before being returned from asserted to the tristate condition.

req_l

Assertion of the **req_l** signal line (request) indicates to the arbiter that this node wants to use the PCI bus. Every master node has its own **req_l** pin.

req64_l

The current bus master asserts **req64_l** (request 64-bit transfer), indicating that it wants to transfer data by using 64 bits. Signal **req64_l** has the same timing as **frame_l**. Signal **req64_l** is asserted low by the system reset signal. Nodes connected to the 64-bit PCI bus extension will see the line asserted low. Those nodes not connected to the 64-bit PCI bus extension will see the signal asserted high by a pull-up device.

rst_l

The **rst_l** signal line (reset) is used to bring PCI-specific registers, sequencers, and signals to a consistent state. When **rst_l** is asserted, all PCI output pins must be driven to their benign state, usually tristated. Although the signal is asynchronous, deassertion must be a clean, bounce-free edge.

serr_l

The **serr_l** pin (system error) is used to report address parity errors, data parity errors on the special cycles command, or any other system error where the result will be catastrophic. If a node does not want a nonexistent memory (NMI) error to be generated, a different reporting mechanism is required. **serr_l** is an open-drain signal line and is asserted for a minimum of one clock cycle by the node reporting the error.

stop_l

The **stop_l** pin indicates that the current target is requesting that the master stop the current transaction.

trdy_l

When asserted, the **trdy_l** (target ready) pin indicates the target node's ability to complete the current phase of the transaction. Signal **trdy_l** is used in conjunction with **irdy_l**. A data phase is completed on any clock assertion where both **trdy_l** and **irdy_l** are sampled asserted. During a read transaction, **trdy_l** indicates that valid data is present on **ad<31:0>**. During a write transaction, **trdy_l** indicates that the target is ready to accept data. Wait cycles are inserted until both **trdy_l** and **irdy_l** are asserted together.

2.2.3 IOD Bus Signals

This section provides a description of the IOD bus signals.

iod_h<63:0>

The **iod_h<63:0>** signals carry data between the CIA and DSW on the IOD bus.

iod_e_h<7:0>

The **iod_e_h<7:0>** signals carry ECC between the CIA and DSW on the IOD bus. All ECC generation and detection is performed in the 21164 and CIA.

ioc_h<6:0>

The **ioc_h<6:0>** signals allow the CIA to control data flow on the IOD bus. The data transfers are between memory and the PCI bus. CIA control is encoded as shown in Table 2–2.

Table 2–2 ioc_h<6:0> Control Functions

ioc_h <6:4>	ioc_h <3>	ioc_h <2>	ioc_h <1>	ioc_h <0>	Function
000	Buffer *n*	x	x	x	Clear valid bits in PCI buffer *n*.
001	x	x	x	x	Load the LW register (Reserved).
010	Buffer *n*	Adr m2	Adr m1	Adr m0	Write PCI buffer *n*, adr m.
011	Mask3	Mask2	Mask1	Mask0	Write IOR (QW mask).
100	Buffer *n*	Adr m2	Adr m1	Adr m0	Read DMA buffer *n*, adr m.
101	Buffer *n*1	Buffer *n*0	Adr m1	Adr m0	Read IOW buffer *n*, adr m.
110	Buffer *n*1 Buffer *n*	Buffer *n*0 Adr m2	Adr m1	Adr m0	Read buffer (IOW or DMA) low longword.
111	Mask3	Mask2	Mask1	Mask0	Start IOW (buffer mask).

cmc_h<8:0>

The **cmc_h<8:0>** signal lines control flow of data between the 21164 and the memory arrays. They also start and stop loading of the free-running buffers (victim, flush, IOW buffers) in the DSW. In Table 2–3, A5 and A4 are address bits. The Var field encodes the transaction being performed as shown in Table 2–4, Table 2–5, and Table 2–6.

Table 2–3 cmc_h<8:0> Control Functions

cmc_h <8:5>	cmc_h <4>	cmc_h <3>	cmc_h <2>	cmc_h <1>	cmc_h <0>	Function
0000	Var 4[1]	Var 3	Var 2	Var 1	Var 0	Start/stop the buffer.
0010	x	A4[2]	Delay	Var 1	Var 0	Move fill data from memory to 21164.
0011	x	A4	x	Var 1	Var 0	Move IOR buffer data to 21164.
0100	A5[2]	A4	Delay	Var 1	Var 0	Move memory data to memory buffer 0.
0101	A5	A4	Delay	Var 1	Var 0	Move memory data to memory buffer 1.
0110	A5	A4	Delay	Stop IOW*n*1	Stop IOW*n*0	Move memory data to memory buffer 0, stop IOW buffer *n*.
0111	A5	A4	Delay	Stop IOW*n*1	Stop IOW*n*0	Move memory data to memory buffer 1, stop IOW buffer *n*.
1000	A5	A4	Var 2	Var 1	Var 0	Write victim buffer to memory.
1010	A5	A4	Var 2	Var 1	Var 0	Write DMA0 buffer to memory.
1011	A5	A4	Var 2	Var 1	Var 0	Write DMA1 buffer to memory.

[1]The variable field encodes the transaction being performed. See Table 2–4, Table 2–5, and Table 2–6.

[2]A5 and A4 are address bits.

Table 2–4 Var Encodings for CMC Command 0

Var<4:3>	Function
00	None.
01	Start victim.
10	Start flush buffer 0 and clear PCI 0 valid bits.
11	Start flush buffer 1 and clear PCI 0 valid bits.

Table 2–5 Var Encodings for CMC Commands 0, 8, A, and B

Var<2:0>	Function
000	None.
001	Stop victim.
010	Stop flush buffer 0.
011	Stop flush buffer 1.
100	Stop IOW 0.
101	Stop IOW 1.
110	Stop IOW 2.
111	Stop IOW 3.

Table 2–6 Var Encodings for CMC Commands 2, 3, 4, 5, 6, and 7

Var<1:0>	Function
00	None.
01	Stop victim.
10	Stop flush buffer 0.
11	Stop flush buffer 1.

2.2.4 Memory Control Signals

This section provides a description of the memory control signals.

cas_h<3:0>

The **cas_h<3:0>** signal lines select the column address in the memory arrays.

mem_ack_l

The **mem_ack_l** PCI sideband signal line is asserted by the CIA when it has cleared the transaction path in preparation to receive a transaction from the PCI-to-EISA bridge. See **mem_req_l**.

mem_addr_h<12:0>

The **mem_addr_h<12:0>** signal lines carry the memory array address.

mem_en_h

The **mem_en_h** signal line enables operation of the memory arrays.

mem_req_l

The **mem_req_l** signal line is asserted by the PCI-to-EISA bridge to notify the CIA that it has a transaction pending. The CIA completes all transactions in progress and then asserts **mem_ack_l** to the PCI-to-EISA bridge. The PCI-to-EISA bridge then starts the transaction. See **mem_ack_l**.

The PCI-to-EISA bridge set can be made using these two chips:

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

The PCEB allows a certain time period for a PCI transaction to complete. The **mem_req_l** and **mem_ack_l** PCI sideband signal lines are used to ensure that that the CIA performs transactions from the PCI-to-EISA bridge immediately.

mem_we_l<1:0>

The **mem_we_l<1:0>** signal lines carry the write-enable signal to the memory arrays.

ras_h<3:0>

The **ras_h<3:0>** signal lines select the row address in the memory arrays.

set_sel_h<15:0>

The **set_sel_h<15:0>** signal lines are used to select a memory array.

2.2.5 Phase-Locked Loop Signals

This section describes the phase-locked loop signals.

pll_agnd_h

Signal **pll_agnd_h** is the analog ground input to the PLL and is connected to the system board ground.

pll_clk_h

The **pll_clk_h** signal is the system clock (or reference signal) input to the phase-locked loop (PLL) circuit. The **pll_clk_h** signal line is the source for CIA timing. It supplies a clock with the same frequency as that supplied to the PCI.

pll_lp1_h

Signal **pll_lp1_h** is an output of the PLL (phase detector output).

pll_test_h<3:0>

Signals **pll_test_h<3:0>** select which internal chip information is presented at output pins **debug_h<1:0>**. These signal lines are reserved to Digital.

pll_vss_h

Signal **pll_vss_h** is the ground input input to the PLL. It is connected to the system board ground.

2.2.6 Miscellaneous Power Signals

This section describes the miscellaneous power signals.

aux_vdd_h

Signal **aux_vdd_h** is modified (filtered) within the CIA chip before going to the PLL section of the CIA. Signal **aux_vdd_h** should be connected to **vdd_h** on the module.

aux_vss_h

Signal **aux_vss_h** is modified (filtered) within the CIA chip before going to the PLL section of the CIA. Signal **aux_vss_h** should be connected to **vss_h** on the module.

pll_vdd_h

Signal **pll_vdd_h** is the power input to the PLL. It is connected to the system board power.

pll_lp2_h

Signal **pll_lp2_h** is an input to the PLL, driving a VCO input.

2.2.7 Miscellaneous Signals

This section provides a description of the miscellaneous signals.

count_out_h

Signal **count_out_h** is used during PLL tests.

debug_h<1:0>

Signals **debug_h<1:0>** present internal CIA information selected by **pll_test_h<3:0>**. These signal lines are reserved to Digital.

scan_in_h

Signal **scan_in_h** is the enable pin for the scan chain.

sys_rst_l

The **sys_rst_l** signal line is the system reset input signal to the CIA. When **sys_rst_l** is asserted, the CIA resets its internal state.

test_mode_h<1:0>

Signals **test_mode_h<1:0>** define the four operating modes of the CIA as listed in Table 2–7.

Table 2–7 DSW Operating Modes

test_mode_h<1:0>	DSW Operating Mode
00	Tristate mode
01	Normal mode
10	Scan mode
11	PLL test mode

test_or_scan_out_h

Signal **test_or_scan_out_h** is the TEST_OUT output that is the end of the parametric NAND tree.

2.2.8 Reserved Signals

This section provides a description of the reserved signals.

gru_ack_h

Signal **gru_ack_h** is reserved for use by Digital.

gru_sel_h

Signal **gru_sel_h** is reserved for use by Digital.

2.2.9 DECchip 21171-CA Pin Name List (Alphabetic)

Table 2–8 lists the DECchip 21171-CA (CIA) pins by name in alphabetical order. The following abbreviations are used in the Type column of the table:

- B = Bidirectional
- I = Input
- P = Power
- O = Output

Table 2–8 DECchip 21171-CA Pin Names (Alphabetic)

Pin Name	Pin	Type	Pin Name	Pin	Type
ack64_l	U5	B	—	—	—
ad_h<0>	Y18	B	**ad_h<1>**	V15	B
ad_h<2>	U14	B	**ad_h<3>**	W16	B
ad_h<4>	V14	B	**ad_h<5>**	W14	B
ad_h<6>	U13	B	**ad_h<7>**	Y15	B
ad_h<8>	V13	B	**ad_h<9>**	W12	B
ad_h<10>	AA14	B	**ad_h<11>**	Y13	B
ad_h<12>	AA13	B	**ad_h<13>**	AA12	B
ad_h<14>	AA11	B	**ad_h<15>**	AA10	B
ad_h<16>	V10	B	**ad_h<17>**	U10	B
ad_h<18>	AA8	B	**ad_h<19>**	W9	B
ad_h<20>	V9	B	**ad_h<21>**	U9	B
ad_h<22>	W7	B	**ad_h<23>**	W8	B
ad_h<24>	V8	B	**ad_h<25>**	Y5	B
ad_h<26>	AA4	B	**ad_h<27>**	W5	B
ad_h<28>	AA3	B	**ad_h<29>**	AA2	B
ad_h<30>	U6	B	**ad_h<31>**	Y3	B
ad_h<32>	W17	B	**ad_h<33>**	AA18	B
ad_h<34>	Y17	B	**ad_h<35>**	AA17	B
ad_h<36>	Y16	B	**ad_h<37>**	W15	B
ad_h<38>	AA16	B	**ad_h<39>**	AA15	B
ad_h<40>	Y14	B	**ad_h<41>**	W13	B
ad_h<42>	U12	B	**ad_h<43>**	V12	B
ad_h<44>	Y12	B	**ad_h<45>**	Y11	B
ad_h<46>	W10	B	**ad_h<47>**	Y10	B
ad_h<48>	AA9	B	**ad_h<49>**	Y9	B
ad_h<50>	Y8	B	**ad_h<51>**	AA7	B

(continued on next page)

Table 2–8 (Cont.) DECchip 21171-CA Pin Names (Alphabetic)

Pin Name	Pin	Type	Pin Name	Pin	Type
ad_h<52>	Y7	B	**ad_h<53>**	AA6	B
ad_h<54>	Y6	B	**ad_h<55>**	AA5	B
ad_h<56>	W6	B	**ad_h<57>**	U8	B
ad_h<58>	V7	B	**ad_h<59>**	Y4	B
ad_h<60>	U7	B	**ad_h<61>**	V6	B
ad_h<62>	T6	B	**ad_h<63>**	Y2	B
addr_h<4>	N18	B	**addr_h<5>**	R21	B
addr_h<6>	R20	B	**addr_h<7>**	T21	B
addr_h<8>	N17	B	**addr_h<9>**	R19	B
addr_h<10>	P19	B	**addr_h<11>**	T20	B
addr_h<12>	P18	B	**addr_h<13>**	U21	B
addr_h<14>	T19	B	**addr_h<15>**	U20	B
addr_h<16>	P17	B	**addr_h<17>**	V21	B
addr_h<18>	R18	B	**addr_h<19>**	U19	B
addr_h<20>	V20	B	**addr_h<21>**	R17	B
addr_h<22>	W21	B	**addr_h<23>**	T18	B
addr_h<24>	V19	B	**addr_h<25>**	W20	B
addr_h<26>	V26	B	**addr_h<27>**	T17	B
addr_h<28>	Y20	B	**addr_h<29>**	U18	B
addr_h<30>	U17	B	**addr_h<31>**	E16	B
addr_h<32>	B19	B	**addr_h<33>**	F15	B
addr_h<34>	C18	B	**addr_h<39>**	D17	B
addr_bus_req_h	D18	B	**addr_cmd_par_h**	E15	B
addr_res_h<0>	D10	I	**addr_res_h<1>**	B11	I
aux_vdd_h	C4	P	**aux_vss_h**	D5	P
cack_h	B21	O	—	—	—
cas_h<0>	G19	O	**cas_h<1>**	F21	O
cas_h<2>	H19	O	**cas_h<3>**	G20	O
cbe_l<0>	AA19	B	**cbe_l<1>**	AA20	B
cbe_l<2>	Y21	B	**cbe_l<3>**	W18	B
cbe_l<4>	W2	B	**cbe_l<5>**	T5	B
cbe_l<6>	V3	B	**cbe_l<7>**	U4	B

(continued on next page)

Table 2–8 (Cont.) DECchip 21171-CA Pin Names (Alphabetic)

Pin Name	Pin	Type	Pin Name	Pin	Type
cmc_h<0>	D14	O	**cmc_h<1>**	B16	O
cmc_h<2>	C15	O	**cmc_h<3>**	B15	O
cmc_h<4>	E13	O	**cmc_h<5>**	C14	O
cmc_h<6>	F12	O	**cmc_h<7>**	D13	O
cmc_h<8>	B14	O	—	—	—
cmd_h<0>	B18	B	**cmd_h<1>**	F14	B
cmd_h<2>	C17	B	**cmd_h<3>**	D16	B
count_out_h	R6	O	**dack_h**	F16	O
debug_h<0>	T3	I	**debug_h<1>**	V5	I
devsel_l	V17	B	**error_h**	P5	O
fill_error_h	M17	O	**fill_h**	C19	O
fill_id_h	P20	B	**frame_l**	V18	B
GND	A2	P	**GND**	A4	P
GND	A5	P	**GND**	A6	P
GND	A8	P	**GND**	A10	P
GND	A12	P	**GND**	A14	P
GND	A16	P	**GND**	A17	P
GND	A18	P	**GND**	A20	P
GND	A21	P	**GND**	B1	P
GND	C22	P	**GND**	D1	P
GND	D19	P	**GND**	E22	P
GND	F1	P	**GND**	F22	P
GND	G1	P	**GND**	G22	P
GND	J1	P	**GND**	J22	P
GND	K1	P	**GND**	M1	P
GND	M22	P	**GND**	N22	P
GND	P1	P	**GND**	R22	P
GND	T1	P	**GND**	U1	P
GND	U22	P	**GND**	V1	P
GND	W4	P	**GND**	W19	P
GND	W22	P	**GND**	Y1	P
GND	AA22	P	**GND**	AB2	P
GND	AB3	P	**GND**	AB5	P
GND	AB7	P	**GND**	AB9	P
GND	AB12	P	**GND**	AB13	P
GND	AB15	P	**GND**	AB17	P

(continued on next page)

Table 2–8 (Cont.) DECchip 21171-CA Pin Names (Alphabetic)

Pin Name	Pin	Type	Pin Name	Pin	Type
GND	AB18	P	**GND**	AB19	P
GND	AB21	P	—	—	—
gnt_l	U3	I	—	—	—
gru_ack_h (Reserved)	L6	I	**gru_sel_h** (Reserved)	L5	O
idle_bc_h	B20	O	**int_h**	L17	O
int4_valid_h<0>	E14	I	**int4_valid_h<1>**	C16	I
int4_valid_h<2>	F13	I	**int4_valid_h<3>**	D15	I
ioc_h<0>	E12	O	**ioc_h<1>**	C13	O
ioc_h<2>	D12	O	**ioc_h<3>**	B13	O
ioc_h<4>	C12	O	**ioc_h<5>**	B12	O
ioc_h<6>	C11	O	—	—	—
iod_h<0>	B10	B	**iod_h<1>**	C10	B
iod_h<2>	E10	B	**iod_h<3>**	B9	B
iod_h<4>	F10	B	**iod_h<5>**	C9	B
iod_h<6>	B8	B	**iod_h<7>**	C8	B
iod_h<8>	D9	B	**iod_h<9>**	B7	B
iod_h<10>	E9	B	**iod_h<11>**	C7	B
iod_h<12>	D8	B	**iod_h<13>**	B6	B
iod_h<14>	D7	B	**iod_h<15>**	C6	B
iod_h<16>	F9	B	**iod_h<17>**	B5	B
iod_h<18>	E8	B	**iod_h<19>**	D6	B
iod_h<20>	C5	B	**iod_h<21>**	E7	B
iod_h<22>	B4	B	**iod_h<23>**	F8	B
iod_h<24>	N4	B	**iod_h<25>**	P3	B
iod_h<26>	P4	B	**iod_h<27>**	N6	B
iod_h<28>	T2	B	**iod_h<29>**	N5	B
iod_h<30>	M4	B	**iod_h<31>**	F6	B
iod_h<32>	D3	B	**iod_h<33>**	E4	B
iod_h<34>	G5	B	**iod_h<35>**	D2	B
iod_h<36>	H6	B	**iod_h<37>**	E3	B
iod_h<38>	F4	B	**iod_h<39>**	H5	B
iod_h<40>	E2	B	**iod_h<41>**	H4	B
iod_h<42>	F3	B	**iod_h<43>**	G4	B
iod_h<44>	F2	B	**iod_h<45>**	J6	B
iod_h<46>	G3	B	**iod_h<47>**	J5	B

(continued on next page)

Table 2–8 (Cont.) DECchip 21171-CA Pin Names (Alphabetic)

Pin Name	Pin	Type	Pin Name	Pin	Type
iod_h<48>	G2	B	**iod_h<49>**	J4	B
iod_h<50>	H3	B	**iod_h<51>**	H2	B
iod_h<52>	J3	B	**iod_h<53>**	K6	B
iod_h<54>	J2	B	**iod_h<55>**	K5	B
iod_h<56>	K3	B	**iod_h<57>**	K2	B
iod_h<58>	K4	B	**iod_h<59>**	L2	B
iod_h<60>	L3	B	**iod_h<61>**	M2	B
iod_h<62>	M3	B	**iod_h<63>**	N2	B
iod_e_h<0>	F5	B	**iod_e_h<1>**	C3	B
iod_e_h<2>	G6	B	**iod_e_h<3>**	C2	B
iod_e_h<4>	M5	B	**iod_e_h<5>**	N3	B
iod_e_h<6>	M6	B	**iod_e_h<7>**	P2	B
irdy_l	V11	B	**lock_l**	U15	I
mem_ack_l	R4	O	—	—	—
mem_addr_h<0>	J20	O	**mem_addr_h<1>**	K17	O
mem_addr_h<2>	J21	O	**mem_addr_h<3>**	K18	O
mem_addr_h<4>	K20	O	**mem_addr_h<5>**	K21	O
mem_addr_h<6>	K19	O	**mem_addr_h<7>**	L21	O
mem_addr_h<8>	L20	O	**mem_addr_h<9>**	M21	O
mem_addr_h<10>	M20	O	**mem_addr_h<11>**	N21	O
mem_addr_h<12>	M19	O	—	—	—
mem_cs_l	U2	I	**mem_en_h**	H21	O
mem_req_l	T4	I	**mem_we_l<0>**	N20	O
mem_we_l<1>	M18	O	**par_h**	R5	B
par64_h	V2	B	**perr_l**	AA21	B
pll_agnd_h	E6	O	**pll_clk_h**	F11	I
pll_test_h<0>	E18	I	**pll_test_h<1>**	L4	I
pll_test_h<2>	R2	I	**pll_test_h<3>**	R3	I
pll_lp1_h	B3	I	**pll_lp2_h**	F7	P
pll_vdd_h	E5	P	**pll_vss_h**	D4	I
PWR5	A3	P	**PWR5**	A7	P
PWR5	A9	P	**PWR5**	A11	P
PWR5	A13	P	**PWR5**	A15	P
PWR5	A19	P	**PWR5**	A22	P

(continued on next page)

Table 2–8 (Cont.) DECchip 21171-CA Pin Names (Alphabetic)

Pin Name	Pin	Type	Pin Name	Pin	Type
PWR5	B2	P	**PWR5**	B22	P
PWR5	C1	P	**PWR5**	D22	P
PWR5	E1	P	**PWR5**	H1	P
PWR5	H22	P	**PWR5**	K22	P
PWR5	L1	P	**PWR5**	L22	P
PWR5	N1	P	**PWR5**	P22	P
PWR5	R1	P	**PWR5**	T22	P
PWR5	V22	P	**PWR5**	W1	P
PWR5	Y22	P	**PWR5**	AA1	P
PWR5	AB1	P	**PWR5**	AB4	P
PWR5	AB6	P	**PWR5**	AB8	P
PWR5	AB10	P	**PWR5**	AB11	P
PWR5	AB14	P	**PWR5**	AB16	P
PWR5	AB20	P	**PWR5**	AB22	P
ras_h<0>	J18	O	**ras_h<1>**	G21	O
ras_h<2>	J19	O	**ras_h<3>**	H20	O
req_l	P6	O	**req64_l**	V4	B
rst_l	U16	O	**scan_in_h**	L19	I
serr_l	W3	B	—	—	—
set_sel_h<0>	F17	O	**set_sel_h<1>**	G17	O
set_sel_h<2>	C20	O	**set_sel_h<3>**	F18	O
set_sel_h<4>	C21	O	**set_sel_h<5>**	D20	O
set_sel_h<6>	E19	O	**set_sel_h<7>**	H17	O
set_sel_h<8>	D21	O	**set_sel_h<9>**	G18	O
set_sel_h<10>	E20	O	**set_sel_h<11>**	F19	O
set_sel_h<12>	H18	O	**set_sel_h<13>**	E21	O
set_sel_h<14>	J17	O	**set_sel_h<15>**	F20	O
spare1	U11	I	**stop_l**	Y19	B
sys_rst_l	L18	I	**tag_ctl_par_h**	P21	O
tag_dirty_h	N19	O	**test_mode_h<0>**	E11	I
test_mode_h<1>	D11	I	**test_or_scan_out_h**	E17	O
trdy_l	W11	B	**victim_pending_h**	B17	I

2.2.10 DECchip 21171-CA Pin Assignment List (Alphanumeric)

Table 2–9 lists the DECchip 21171-CA pins in alphanumeric order. The following abbreviations are used in the Type column of the table:

- B = Bidirectional
- I = Input
- P = Power
- O = Output

Table 2–9 DECchip 21171-CA Pin Assignment (Alphanumeric)

Pin	Pin Name	Type	Pin	Pin Name	Type
—	—	—	A2	**GND**	P
A3	**PWR5**	P	A4	**GND**	P
A5	**GND**	P	A6	**GND**	P
A7	**PWR5**	P	A8	**GND**	P
A9	**PWR5**	P	A10	**GND**	P
A11	**PWR5**	P	A12	**GND**	P
A13	**PWR5**	P	A14	**GND**	P
A15	**PWR5**	P	A16	**GND**	P
A17	**GND**	P	A18	**GND**	P
A19	**PWR5**	P	A20	**GND**	P
A21	**GND**	P	A22	**PWR5**	P
B1	**GND**	P	B2	**PWR5**	P
B3	**pll_lp1_h**	I	B4	**iod_h<22>**	B
B5	**iod_h<17>**	B	B6	**iod_h<13>**	B
B7	**iod_h<9>**	B	B8	**iod_h<6>**	B
B9	**iod_h<3>**	B	B10	**iod_h<0>**	B
B11	**addr_res_h<1>**	I	B12	**ioc_h<5>**	O
B13	**ioc_h<3>**	O	B14	**cmc_h<8>**	O
B15	**cmc_h<3>**	O	B16	**cmc_h<1>**	O
B17	**victim_pending_h**	I	B18	**cmd_h<0>**	B
B19	**addr_h<32>**	B	B20	**idle_bc_h**	O
B21	**cack_h**	O	B22	**PWR5**	P
C1	**PWR5**	P	C2	**iod_e_h<3>**	B
C3	**iod_e_h<1>**	B	C4	**aux_vdd**	P
C5	**iod_h<20>**	B	C6	**iod_h<15>**	B
C7	**iod_h<11>**	B	C8	**iod_h<7>**	B

(continued on next page)

Table 2–9 (Cont.) DECchip 21171-CA Pin Assignment (Alphanumeric)

Pin	Pin Name	Type	Pin	Pin Name	Type
C9	**iod_h<5>**	B	C10	**iod_h<1>**	B
C11	**ioc_h<6>**	O	C12	**ioc_h<4>**	O
C13	**ioc_h<1>**	O	C14	**cmc_h<5>**	O
C15	**cmc_h<2>**	O	C16	**int4_valid_h<1>**	I
C17	**cmd_h<2>**	B	C18	**addr_h<34>**	B
C19	**fill_h**	O	C20	**set_sel_h<2>**	O
C21	**set_sel_h<4>**	O	C22	**GND**	P
D1	**GND**	P	D2	**iod_h<35>**	B
D3	**iod_h<32>**	B	D4	**pll_vss_h**	I
D5	**aux_vss**	P	D6	**iod_h<19>**	B
D7	**iod_h<14>**	B	D8	**iod_h<12>**	B
D9	**iod_h<8>**	B	D10	**addr_res_h<0>**	I
D11	**test_mode_h<1>**	I	D12	**ioc_h<2>**	O
D13	**cmc_h<7>**	O	D14	**cmc_h<0>**	O
D15	**int4_valid_h<3>**	I	D16	**cmd_h<3>**	B
D17	**addr_h<39>**	B	D18	**addr_bus_req**	B
D19	**GND**	P	D20	**set_sel_h<5>**	O
D21	**set_sel_h<8>**	O	D22	**PWR5**	P
E1	**PWR5**	P	E2	**iod_h<40>**	B
E3	**iod_h<37>**	B	E4	**iod_h<33>**	B
E5	**pll_vdd**	P	E6	**pll_agnd_h**	O
E7	**iod_h<21>**	B	E8	**iod_h<18>**	B
E9	**iod_h<10>**	B	E10	**iod_h<2>**	B
E11	**test_mode_h<0>**	I	E12	**ioc_h<0>**	O
E13	**cmc_h<4>**	O	E14	**int4_valid_h<0>**	I
E15	**addr_cmd_par**	B	E16	**addr_h<31>**	B
E17	**test_or_scan_out_h**	O	E18	**pll_test_h<0>**	I
E19	**set_sel_h<6>**	O	E20	**set_sel_h<10>**	O
E21	**set_sel_h<13>**	O	E22	**GND**	P
F1	**GND**	P	F2	**iod_h<44>**	B
F3	**iod_h<42>**	B	F4	**iod_h<38>**	B
F5	**iod_e_h<0>**	B	F6	**iod_h<31>**	B
F7	**pll_lp2_h**	P	F8	**iod_h<23>**	B
F9	**iod_h<16>**	B	F10	**iod_h<4>**	B
F11	**pll_clk_h**	I	F12	**cmc_h<6>**	O
F13	**int4_valid_h<2>**	I	F14	**cmd_h<1>**	B

(continued on next page)

Table 2–9 (Cont.) DECchip 21171-CA Pin Assignment (Alphanumeric)

Pin	Pin Name	Type	Pin	Pin Name	Type
F15	**addr_h<33>**	B	F16	**dack_h**	O
F17	**set_sel_h<0>**	O	F18	**set_sel_h<3>**	O
F19	**set_sel_h<11>**	O	F20	**set_sel_h<15>**	O
F21	**cas_h<1>**	O	F22	**GND**	P
G1	**GND**	P	G2	**iod_h<48>**	B
G3	**iod_h<46>**	B	G4	**iod_h<43>**	B
G5	**iod_h<34>**	B	G6	**iod_e_h<2>**	B
G17	**set_sel_h<1>**	O	G18	**set_sel_h<9>**	O
G19	**cas_h<0>**	O	G20	**cas_h<3>**	O
G21	**ras_h<1>**	O	G22	**GND**	P
H1	**PWR5**	P	H2	**iod_h<51>**	B
H3	**iod_h<50>**	B	H4	**iod_h<41>**	B
H5	**iod_h<39>**	B	H6	**iod_h<36>**	B
H17	**set_sel_h<7>**	O	H18	**set_sel_h<12>**	O
H19	**cas_h<2>**	O	H20	**ras_h<3>**	O
H21	**mem_en_h**	O	H22	**PWR5**	P
J1	**GND**	P	J2	**iod_h<54>**	B
J3	**iod_h<52>**	B	J4	**iod_h<49>**	B
J5	**iod_h<47>**	B	J6	**iod_h<45>**	B
J17	**set_sel_h<14>**	O	J18	**ras_h<0>**	O
J19	**ras_h<2>**	O	J20	**mem_addr_h<0>**	O
J21	**mem_addr_h<2>**	O	J22	**GND**	P
K1	**GND**	P	K2	**iod_h<57>**	B
K3	**iod_h<56>**	B	K4	**iod_h<58>**	B
K5	**iod_h<55>**	B	K6	**iod_h<53>**	B
K17	**mem_addr_h<1>**	O	K18	**mem_addr_h<3>**	O
K19	**mem_addr_h<6>**	O	K20	**mem_addr_h<4>**	O
K21	**mem_addr_h<5>**	O	K22	**PWR5**	P
L1	**PWR5**	P	L2	**iod_h<59>**	B
L3	**iod_h<60>**	B	L4	**pll_test_h<1>**	I
L5	**gru_sel_h** (Reserved)	O	L6	**gru_ack_h** (Reserved)	I
L17	**int_h**	O	L18	**sys_rst_l**	I
L19	**scan_in_h**	I	L20	**mem_addr_h<8>**	O
L21	**mem_addr_h<7>**	O	L22	**PWR5**	P

(continued on next page)

Table 2–9 (Cont.) DECchip 21171-CA Pin Assignment (Alphanumeric)

Pin	Pin Name	Type	Pin	Pin Name	Type
M1	**GND**	P	M2	**iod_h<61>**	B
M3	**iod_h<62>**	B	M4	**iod_h<30>**	B
M5	**iod_e_h<4>**	B	M6	**iod_e_h<6>**	B
M17	**fill_error_h**	O	M18	**mem_we_l<1>**	O
M19	**mem_addr_h<12>**	O	M20	**mem_addr_h<10>**	O
M21	**mem_addr_h<9>**	O	M22	**GND**	P
N1	**PWR5**	P	N2	**iod_h<63>**	B
N3	**iod_e_h<5>**	B	N4	**iod_h<24>**	B
N5	**iod_h<29>**	B	N6	**iod_h<27>**	B
N17	**addr_h<8>**	B	N18	**addr_h<4>**	B
N19	**tag_dirty_h**	O	N20	**mem_we_l<0>**	O
N21	**mem_addr_h<11>**	O	N22	**GND**	P
P1	**GND**	P	P2	**iod_e_h<7>**	B
P3	**iod_h<25>**	B	P4	**iod_h<26>**	B
P5	**error_h**	O	P6	**req_l**	O
P17	**addr_h<16>**	B	P18	**addr_h<12>**	B
P19	**addr_h<10>**	B	P20	**fill_id_h**	B
P21	**tag_ctl_par_h**	O	P22	**PWR5**	P
R1	**PWR5**	P	R2	**pll_test_h<2>**	I
R3	**pll_test_h<3>**	I	R4	**mem_ack_l**	O
R5	**par_h**	B	R6	**count_out_h**	O
R17	**addr_h<21>**	B	R18	**addr_h<18>**	B
R19	**addr_h<9>**	B	R20	**addr_h<6>**	B
R21	**addr_h<5>**	B	R22	**GND**	P
T1	**GND**	P	T2	**iod_h<28>**	B
T3	**debug_h<0>**	I	T4	**mem_req_l**	I
T5	**cbe_l<5>**	B	T6	**ad_h<62>**	B
T17	**addr_h<27>**	B	T18	**addr_h<23>**	B
T19	**addr_h<14>**	B	T20	**addr_h<11>**	B
T21	**addr_h<7>**	B	T22	**PWR5**	P
U1	**GND**	P	U2	**mem_cs_l**	I
U3	**gnt_l**	I	U4	**cbe_l<7>**	B
U5	**ack64_l**	B	U6	**ad_h<30>**	B

(continued on next page)

Table 2–9 (Cont.) DECchip 21171-CA Pin Assignment (Alphanumeric)

Pin	Pin Name	Type	Pin	Pin Name	Type
U7	**ad_h<60>**	B	U8	**ad_h<57>**	B
U9	**ad_h<21>**	B	U10	**ad_h<17>**	B
U11	**spare1**	I	U12	**ad_h<42>**	B
U13	**ad_h<6>**	B	U14	**ad_h<2>**	B
U15	**lock_l**	I	U16	**rst_l**	O
U17	**addr_h<30>**	B	U18	**addr_h<29>**	B
U19	**addr_h<19>**	B	U20	**addr_h<15>**	B
U21	**addr_h<13>**	B	U22	**GND**	P
V1	**GND**	P	V2	**par64**	B
V3	**cbe_l<6>**	B	V4	**req64_l**	B
V5	**debug_h<1>**	I	V6	**ad_h<61>**	B
V7	**ad_h<58>**	B	V8	**ad_h<24>**	B
V9	**ad_h<20>**	B	V10	**ad_h<16>**	B
V11	**irdy_l**	B	V12	**ad_h<43>**	B
V13	**ad_h<8>**	B	V14	**ad_h<4>**	B
V15	**ad_h<1>**	B	V16	**addr_h<26>**	B
V17	**devsel_l**	B	V18	**frame_l**	B
V19	**addr_h<24>**	B	V20	**addr_h<20>**	B
V21	**addr_h<17>**	B	V22	**PWR5**	P
W1	**PWR5**	P	W2	**cbe_l<4>**	B
W3	**serr_l**	B	W4	**GND**	P
W5	**ad_h<27>**	B	W6	**ad_h<56>**	B
W7	**ad_h<22>**	B	W8	**ad_h<23>**	B
W9	**ad_h<19>**	B	W10	**ad_h<46>**	B
W11	**trdy_l**	B	W12	**ad_h<9>**	B
W13	**ad_h<41>**	B	W14	**ad_h<5>**	B
W15	**ad_h<37>**	B	W16	**ad_h<3>**	B
W17	**ad_h<32>**	B	W18	**cbe_l<3>**	B
W19	**GND**	P	W20	**addr_h<25>**	B
W21	**addr_h<22>**	B	W22	**GND**	P
Y1	**GND**	P	Y2	**ad_h<63>**	B
Y3	**ad_h<31>**	B	Y4	**ad_h<59>**	B
Y5	**ad_h<25>**	B	Y6	**ad_h<54>**	B
Y7	**ad_h<52>**	B	Y8	**ad_h<50>**	B
Y9	**ad_h<49>**	B	Y10	**ad_h<47>**	B
Y11	**ad_h<45>**	B	Y12	**ad_h<44>**	B

(continued on next page)

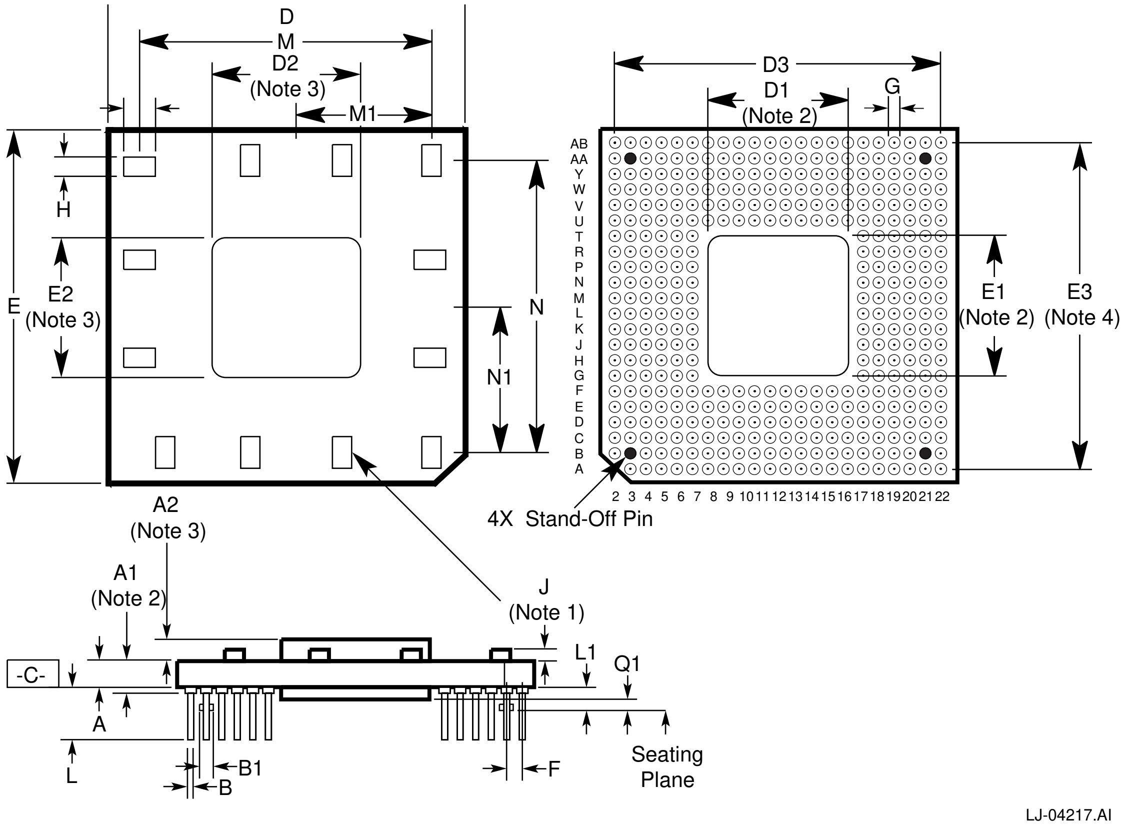
Table 2–9 (Cont.) DECchip 21171-CA Pin Assignment (Alphanumeric)

Pin	Pin Name	Type	Pin	Pin Name	Type
Y13	**ad_h<11>**	B	Y14	**ad_h<40>**	B
Y15	**ad_h<7>**	B	Y16	**ad_h<36>**	B
Y17	**ad_h<34>**	B	Y18	**ad_h<0>**	B
Y19	**stop_l**	B	Y20	**addr_h<28>**	B
Y21	**cbe_l<2>**	B	Y22	**PWR5**	P
AA1	**PWR5**	P	AA2	**ad_h<29>**	B
AA3	**ad_h<28>**	B	AA4	**ad_h<26>**	B
AA5	**ad_h<55>**	B	AA6	**ad_h<53>**	B
AA7	**ad_h<51>**	B	AA8	**ad_h<18>**	B
AA9	**ad_h<48>**	B	AA10	**ad_h<15>**	B
AA11	**ad_h<14>**	B	AA12	**ad_h<13>**	B
AA13	**ad_h<12>**	B	AA14	**ad_h<10>**	B
AA15	**ad_h<39>**	B	AA16	**ad_h<38>**	B
AA17	**ad_h<35>**	B	AA18	**ad_h<33>**	B
AA19	**cbe_l<0>**	B	AA20	**cbe_l<1>**	B
AA21	**perr_l**	B	AA22	**GND**	P
AB1	**PWR5**	P	AB2	**GND**	P
AB3	**GND**	P	AB4	**PWR5**	P
AB5	**GND**	P	AB6	**PWR5**	P
AB7	**GND**	P	AB8	**PWR5**	P
AB9	**GND**	P	AB10	**PWR5**	P
AB11	**PWR5**	P	AB12	**GND**	P
AB13	**GND**	P	AB14	**PWR5**	P
AB15	**GND**	P	AB16	**PWR5**	P
AB17	**GND**	P	AB18	**GND**	P
AB19	**GND**	P	AB20	**PWR5**	P
AB21	**GND**	P	AB22	**PWR5**	P

2.3 DECchip 21171-CA Mechanical Specifications

Figure 2–1 and Table 2–10 indicate the DECchip 21171-CA package dimensions.

Figure 2–1 DECchip 21171-CA Package Dimensions



Notes for Figure 2–1

The following notes apply to Figure 2–1:

Note 1. Chip capacitors envelope. Commercial grade chip caps; nominal capacitance 0.082 μF.
Note 2. Envelope for maximum lid (ceramic, epoxy seal).
Note 3. Envelope for integral heat slug.
Note 4. The drawing/dimensions are for reference only. For board layout, request an engineering drawing from Digital Semiconductor.

Table 2–10 DECchip 21171-CA Package Dimensions

Indicator	Minimum	Maximum
A	1.78 mm (0.070 in)	3.30mm (0.130 in)
A1	NA[1]	4.19 mm (0.165 in)
A2	NA	1.27 mm (0.050 in)
B	0.40 mm (0.016 in)	0.51 mm (0.020 in)
B1	1.17 mm (0.046 in)	1.37 mm (0.054 in)
D	57.30 mm (2.256 in)	57.51 mm (2.264 in)
D1	NA	24.89 mm (0.980 in)
D2	NA	24.13 mm (0.950 in)
E	57.30 mm (2.256 in)	57.51 mm (2.264 in)
E1	NA	24.89 mm (0.980 in)
E2	NA	24.13 mm (0.950 in)
G	1.78 mm (0.070 in)	2.04 mm (0.080 in)
H	NA	3.56 mm (0.140 in)
I	NA	4.06 mm (0.160 in)
J	NA	1.65 mm (0.064 in)
L	4.06 mm (0.160 in)	4.57 mm (0.180 in)
L1	1.12 mm (0.044 in)	1.42 mm (0.056 in)
Q1	0.23 mm (0.009 in)	NA

Indicator	Basic
D3	2.100 in (53.34 mm)
E3	2.100 in (53.34 mm)
F	1.0 in (2.54 mm)
M	2.050 in (52.07 mm)
M1	1.000 in (25.4 mm)
N	2.000 in (50.8 mm)
N1	0.950 in (24.13 mm)

[1]NA = Not applicable.

3 DECchip 21171-CA Architecture Overview

This chapter describes the DECchip 21171-CA (CIA) data paths with functional units, transactions, and architecture.

3.1 CIA Data Paths with Functional Units Description

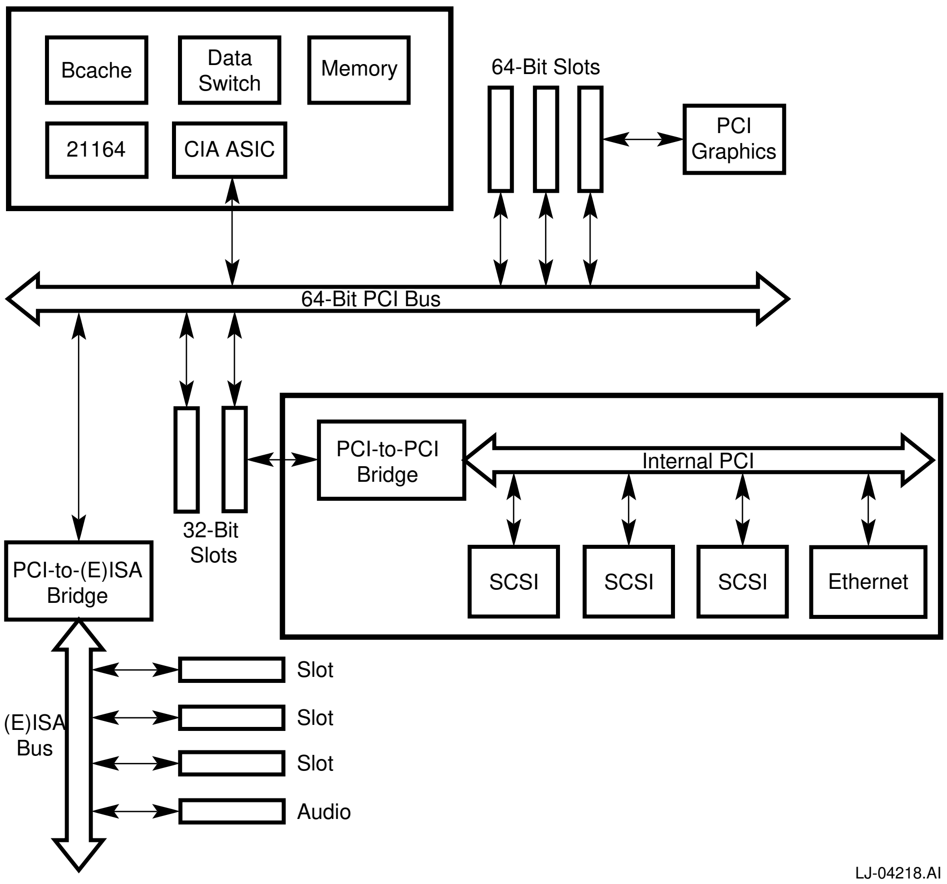
This section shows the CIA data path functional units and describes operation of the functional units.

3.1.1 System Functional Units

The CIA operates as the host bridge on a PCI-based system along with the 21164, four DSWs, Bcache, memory arrays, and support logic. A simplified description of the system data paths and functional units is presented here to show where the CIA operates within the system.

The CIA performs control, I/O, and address tasks within the DECchip 21171 Core Logic Chipset as shown in Figure 3–1. The host bridge module is shown plugged into a 64-bit PCI.

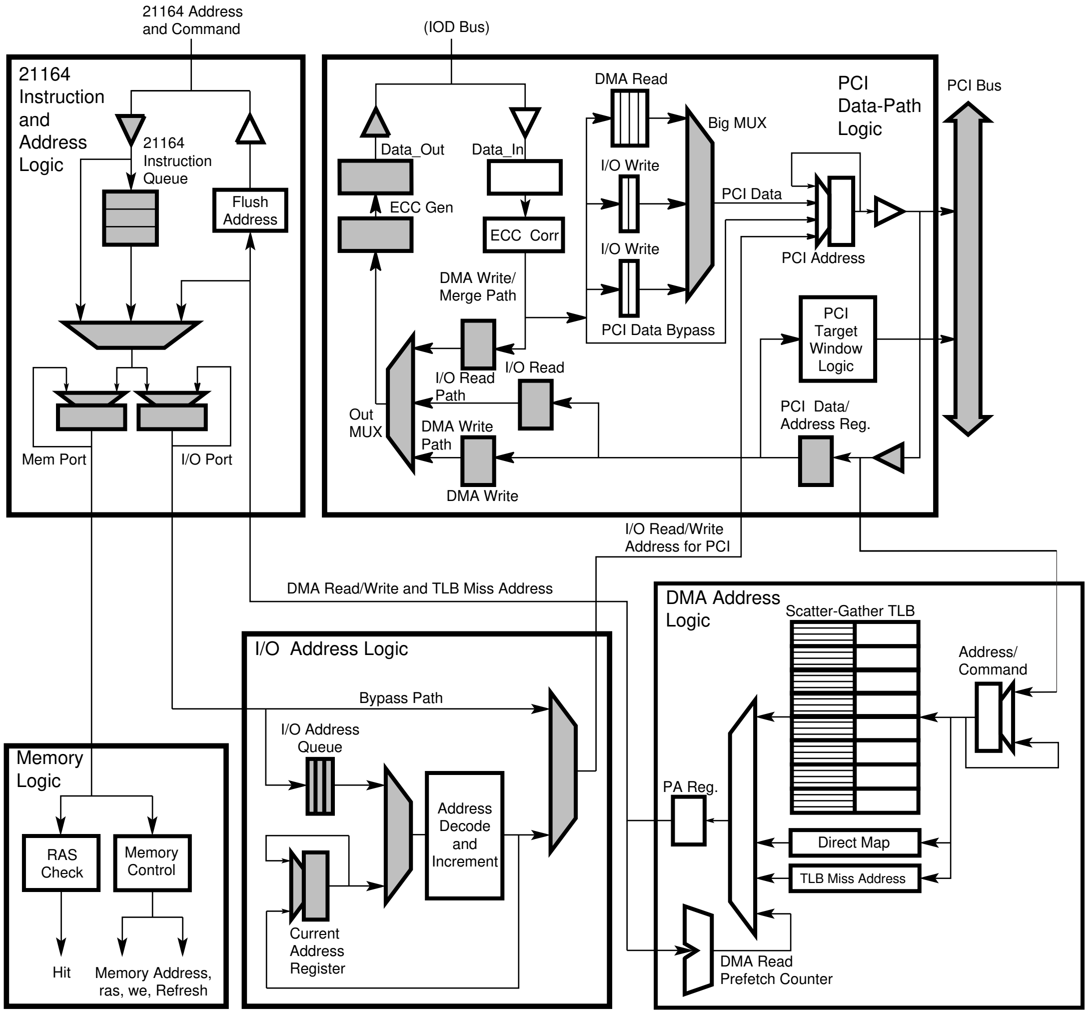
Figure 3–1 System Functional Block Diagram



3.1.2 CIA Functional Units

Figure 3–2 shows the CIA data paths and functional units.

Figure 3–2 CIA Functional Block Diagram



The CIA functional units are listed here and are described in Section 3.1.2.1 through Section 3.1.2.5:

- 21164 instruction and address logic
- PCI data path logic
- Memory logic
- I/O address logic
- DMA address logic

3.1.2.1 21164 Instruction and Address Logic

The 21164 instruction and address logic accepts commands from the 21164 and directs the instruction to the memory port or the I/O port. The DMA read/write address (or the scatter-gather TLB miss servicing address) also has access to the memory port through a multiplexer.

The 21164 instruction queue is provided to capture 21164 commands should the memory or I/O port be busy. The 21164 can issue, at most, two read miss transactions—with one having a victim. This maximum condition would require holding three addresses and commands, so a 3-entry buffer is needed.

The flush address register is used during DMA read or write transactions to the Bcache for the latest data. The Bcache is implemented as a write invalidate cache and so might have the only valid copy of the required data.

3.1.2.2 PCI Data Path Logic

Because the CIA is the PCI host bridge it generates and decodes PCI addresses and also supplies and receives data. The data path to and from the PCI goes through the CIA and the DSW. The ECC generation and check logic is excluded from the DSW because of the bit slice nature of the DSW and so is performed by the CIA.

Separate buffers are provided in the CIA for DMA and I/O read/write transactions. These buffers are either 32 bytes or 64 bytes in size, and are shown partitioned into 16-byte units, which matches the width of the 21164 data bus.

Later, when the DSW is examined, it will be seen that the DSW also has buffers for the DMA and I/O paths. Control logic simplification is the reason for this duplication.

Table 3–1 lists the PCI commands that the CIA responds to and sends.

Table 3–1 PCI Commands Recognized by the CIA

PCI cbe_l<3:0>	Command	CIA Slave	CIA Master
0000	Interrupt acknowledge	No	Yes
0001	Special cycle	No	Yes
0010	I/O read	No	Yes
0011	I/O write	No	Yes
0100	Reserved	—	—
0101	Reserved	—	—
0110	Memory read	Yes	Yes
0111	Memory write	Yes	Yes
1000	Reserved	—	—
1001	Reserved	—	—
1010	Configuration read	No	Yes
1011	Configuration write	No	Yes
1100	Memory read multiple	Yes	No
1101	Dual address cycle	Yes	No
1110	Memory read line	Yes	No
1111	Memory write and invalidate	Yes[1]	No

[1]Accepted as memory write

The CIA has these PCI features:

- Supports 64-bit PCI bus width
- Supports 64-bit PCI addressing (DAC cycles)
- Accepts PCI fast back-to-back cycles
- Issues PCI fast back-to-back cycles in dense address space

3.1.2.3 Memory Logic

Memory logic provides the row and column address for the memory banks and all the control signals (**ras_h**, **cas_h**, **mem_we_l<1:0>**, and **mem_en_h**). The memory logic provides control signals to the DSW to inform it when to send/strobe the data to/from memory. The CIA also controls memory refresh.

3.1.2.4 I/O Address Logic

The I/O address logic handles the I/O read and write addresses. The address decode logic extracts the PCI address from the dense/sparse space 21164 address encodings (see Chapter 6 for more details).

This logic also increments the current address of each data cycle for two reasons:

- In case of a PCI retry, which will require the transaction to be resumed later, we need to start with the address of the aborted data.
- We need to provide a pointer to the next data item to be loaded/sent from the I/O read/write buffers.

The CIA can queue up to six I/O write transactions. Two of the I/O write transactions can be waiting in the 21164 queue and four I/O write transactions can be sitting in the I/O address queue (a 3-entry I/O address queue is provided together with a single-entry current address register). This allows six I/O write transactions to be outstanding (the DSW provides four corresponding I/O write data buffers). The bypass path is enabled/disabled by CIA_CTRL[CSR_IOA_BYPASS] (CSR_CTRL<15>) being equal to 0/1 respectively.

The CIA provides two more buffers that are needed to sustain maximum bandwidth for memory copy to I/O space. A bypass path is provided for certain dense-space I/O write transactions, that is, the valid bits for the first 16 bytes of data are either 1111, 1011, 1010, and 1001—these cases are optimized for a 64-bit PCI with at least two PCI data cycles.

3.1.2.5 DMA Address Logic

The DMA address logic converts PCI addresses to CPU memory addresses. Two conversion methods are provided:

- A direct path where a base offset is concatenated with the PCI address
- A scatter-gather map, which maps any 8KB PCI page to any 8KB memory space page

The scatter-gather TLB is eight entries deep with each entry holding four consecutive PTEs. A TLB miss is handled by hardware but software is required to invalidate stale entries by writing to the translation buffer invalidate register (TBIA). See Chapter 6 for more information on address mapping.

The output of the physical address register (PA) has a a counter that generates the prefetch address for certain DMA read miss transactions. An 8KB detector prevents prefetching across page boundaries.

3.2 CIA Transactions

The CIA performs the transactions listed here:

- 21164 memory read miss
- 21164 memory read miss with victim
- 21164 I/O read
- 21164 I/O write
- DMA read
- DMA read (prefetch)
- DMA write

These transactions are described in detail in Section 3.2.1 through Section 3.2.8.

3.2.1 21164 Read Miss Transaction

The 21164 read miss command and address are sent to the CIA. If the CIA is idle, the command will be stored directly in the memory port register; otherwise, the command enters the 3-deep CPU instruction queue and remains there until the memory port is free. The CIA can accept another subsequent read miss command from the 21164 and store it in the 21164 instruction queue.

The three data paths to the memory port have to arbitrate for use of the memory port: (1) the direct path and (2) the 21164 instruction queue previously mentioned and (3) the DMA read/write address path. The DMA read/write address path is also the path for scatter-gather TLB miss addresses. Upstream of the multiplexer all instruction ordering from the 21164 is preserved. Downstream of the multiplexer ordering between I/O write transactions and 21164 memory transactions is lost. This ‘‘post and run’’ I/O write coherency issue is discussed in Section 3.3.2.

When the 21164 read miss command enters the memory logic, the memory controller generates **ras_h**, **cas_h**, and other address signals for the memory array. The memory controller then waits for the memory access delay before instructing the DSW to accept the memory data. The MEMDATA bus width is 256 bits (32 bytes), and so two memory cycles are required to access the 64-byte block.

The DSW synchronizes the data to the 21164 clock. The fixed 64-byte block size read data is returned to the 21164 in wrapped-order.

3.2.2 21164 Read Miss with Victim Transaction

The read miss with victim transaction is similar to the read miss transaction described in Section 3.2.1, except the 21164 will also use a Bcache victim command to send out the victim block. The victim block is the modified (dirty) block that is to be replaced in the Bcache by the read miss transaction data.

The read miss command is followed by the Bcache victim command and address. The read miss command will get to the memory port first and so the Bcache victim command and address are always written into the 21164 instruction queue. The victim block data is saved in the victim buffer in the DSW.

The CIA arbiter ensures that a read miss transaction and a victim write transaction are an atomic operation. The victim data is written to memory after the read data has been fetched from memory. The row address portion of the victim address is compared to the row address of the current read miss transaction. If they match, then no memory **ras_h** cycle is required; instead, only a **cas_h** cycle is needed. The address bits for the memory have been carefully interspersed to maximize the chance of a victim row address ‘‘hitting’’ the read address—this performance feature is transparent to software. The memory controller in the CIA generates the memory write pulses and instructs the DSW to send the victim data to memory.

The DSW victim buffer is invalidated if a DMA write transaction (or DMA read with lock transaction) ‘‘hits’’ in the victim buffer. Until the 21164 has its read miss transaction data returned, the victim block is still in the Bcache and is still ‘‘owned’’ by the 21164. It is possible for the read miss with victim transaction from the 21164 to be stalled behind a DMA write transaction. The DMA write transaction could ‘‘hit’’ the victim block—in this case, the DMA write transaction will cause a flush command to be issued to the 21164, which will result in the 21164 providing the victim data to the DSW and then the 21164 will invalidate the victim block in Bcache. Thus, the victim data waiting in the victim buffer is no longer valid and is marked invalid by logic in the CIA.

3.2.3 21164 Read Transaction to Noncacheable Space

A read transaction by the 21164 to noncacheable space can be to one of four address spaces:

- PCI memory space
- PCI I/O dense or sparse space
- PCI configuration space
- CIA control and status register (CSR) space

The read command to noncacheable space is accepted by the CIA like the read miss command, except the instruction goes to the I/O port. All read commands to noncacheable space go to the I/O address queue. There is no bypass path for dense space I/O read transactions.

Decode logic is provided on the output of the I/O address queue to extract the byte address for the PCI from the 21164's sparse space encoding, and to decode the address for the correct region (PCI memory, I/O, configuration, or CSR).

A read command to noncacheable space destined for the PCI is described here because it is the more interesting case. Logic increments the current longword /quadword address stored in the current address register each data cycle. This is needed in case of a PCI retry and is also used to index the next data item in the I/O write/read data buffers. The value in the current address register corresponds to the address of the data item on the PCI bus (it has the same canonical time as the PCI data-out register).

The I/O read command address is sent to the PCI after any prior I/O write transactions have completed (strict ordering is maintained). The PCI returns the requested data and places it in the I/O read buffer. The contents of this I/O read buffer are next copied to the I/O read buffer in the DSW and then sent to the 21164.

Why have the DSW I/O read buffers been replicated in the CIA? The I/O read buffers in the CIA are provided for the following reasons:

- The path to the DSW might be busy with the end of a prior DMA write transaction (especially if the data has to be merged with memory data to build it up to the ECC width).
- At least 64 bits of storage are required for ECC because the PCI can return 32 bits of data at a time.
- It is convenient to handle transaction retries forced by the PCI protocol. The I/O read buffer in the DSW was provided to build the data up from 64 bits (IOD bus data lines between the CIA and DSW) to the 128 bits required by the 21164.

I/O read transactions from the 21164 have 8-byte resolution and the CIA always returns 32 bytes. If smaller resolution is required, sparse space addressing must be used. Sparse space addressing is described in Chapter 6.

Data is returned in the appropriate byte lanes.

3.2.4 21164 Write Transaction to Noncacheable Space

The 21164 issues write transactions to noncacheable space with longword resolution. For smaller granularity, sparse space addressing must be used (see Chapter 6).

Data for I/O write transactions is captured in the I/O write buffer in the DSW. Four 32-byte buffers are available in the DSW and a further two in the CIA. This number of entries allows the CIA to sustain maximum bandwidth on a large copy operation from memory through the 21164-to-I/O space.

The data from these I/O write buffers is sent to the two I/O write buffers in the CIA: two buffers are provided, allowing one to be emptied to the PCI bus while the other is being filled. Each 32-byte buffer in the CIA requires a separate PCI transaction (no merging of the write buffers occurs).

Another benefit of the CIA's I/O write buffers is simpler data-flow management when a target PCI device stalls the I/O write transactions.

The address for an I/O write transaction is sent to the CIA I/O port. For a 32-byte aligned dense space write transaction, the I/O write command is either sent directly to the PCI bus via the bypass path, or queued up in the I/O address queue. The direct (bypass) path is used if:

- There are no outstanding I/O commands.
- The command is an I/O write command to dense space and the first 16 bytes are mostly valid (like 1111, 1011, 1010, 1001).

This command also goes into the I/O address queue, but only as a convenient path in the I/O addressing logic section, to get the address into the current address register.

A total of six I/O write addresses can be queued: the first in the current address register, two in the 21164 queue, and three in the I/O address queue. The I/O address queue maintains strict ordering for I/O transactions but does not maintain strict ordering of I/O write transactions relative to any memory read or write transactions (see Section 3.3).

3.2.5 DMA Transactions

The PCI address and command are captured in the address/command register and the data/address register. The address is compared against four address windows to determine if this PCI command should be accepted or ignored by the CIA.

Address windows are a requirement of the PCI specification and are software programmable—they are described in detail in Chapter 6. All PCI commands destined for memory are accepted by the CIA.

Note

Two registers capture incoming samples from the PCI bus.

- The address register captures the address and command, holding them for the duration of the transaction.
- The data-in register captures the data, but also captures the address.

The target window logic is attached to the data-in register. It is located there to capture a buffer of data (64 bytes) for use with a DMA write transaction scatter-gather TLB ‘‘miss.’’ The target window logic retries the master PCI device in case it has more data.

The CIA has released the PCI for further transactions but is busy servicing the TLB ‘‘miss’’ so must hold the virtual address in the address register. Should another PCI transaction occur while the CIA is servicing the TLB ‘‘miss,’’ the CIA will accept that address and compare it to the target window.

There are three registers associated with each of the four PCI windows:

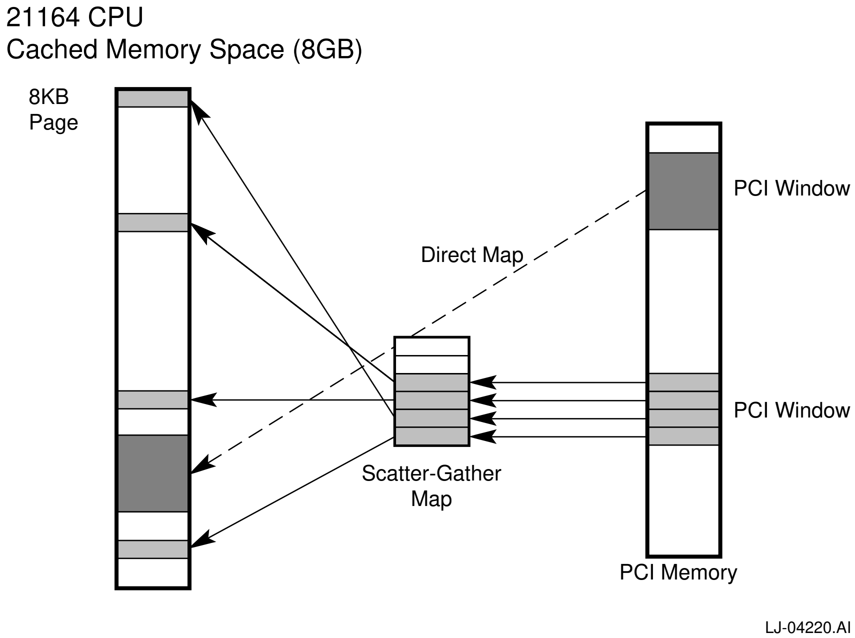
- Window base register 0–3 (W*n*_BASE)—defines the start of the target window. W*n*_BASE[W*n*_BASE_SG] determines if the scatter-gather map is used for the translation.
- Window mask register 0–3 (W*n*_MASK)—defines the size of the window.
- Translated base register (T*n*_BASE)—holds the base address used to relocate the PCI address in the 21164’s memory space (for direct mapping) and is also used to hold the scatter-gather map base address for scatter-gather mapping.

W*n*_BASE[W*n*_BASE_SG] determines how the PCI address is translated as follows:

- If W*n*_BASE[W*n*_BASE_SG] is clear, the address is directly mapped by concatenating T*n*_BASE[CSR_WR_DATA] with the address.
- If W*n*_BASE[W*n*_BASE_SG] is set, the address is mapped through the scatter-gather table, allowing any 8KB PCI address to map to any 8KB memory address.

Figure 3–3 illustrates the mapping. This scatter-gather table is located in memory, but the CIA provides an 8-entry TLB that caches the most recent scatter-gather table entries. Each TLB entry holds four consecutive scatter-gather table entries, mapping a contiguous 32KB of virtual PCI addresses to any four 8KB memory pages.

Figure 3–3 Address Space Overview



3.2.6 DMA Read Transaction

The translated address stored in the PA register is sent to the 21164 through the flush/read address register and to memory through the memory port. If the 21164 data cache contains valid modified data, then a copy is sent to the Bcache-buffer part of one of the two DMA buffers in the DSW and the valid bit is set. Memory data is always fetched and put in the memory buffer of the DMA buffer. The PCI-buffer part of the DMA buffer is not used for DMA read transactions.

When the first valid QW is available in the DSW's DMA buffer, it is sent to the DMA read buffer in the CIA through the multiplexers. Any ECC correction is done in the CIA. From the DMA buffer in the CIA, the data goes from a register onto the PCI a cycle later. The DMA read data also takes the bypass path around the multiplexer for as long as the PCI can accept this stream of data. Once the PCI stalls, then the buffers are used.

3.2.7 DMA Read Transaction (Prefetch)

The PCI supports three types of memory read commands, which are used as specified in the *PCI Local Bus Specification*:

- Read—used for small transfers (up to 1/2 a cache line)
- Read line—used for medium transfers (1/2 to 3 cache lines)
- Read multiple—used for large transfers (more than 3 cache lines)

In the PCI environment, most devices assume a processor cache of 32 bytes (compared to the CIA cache line of 64 bytes). The CIA prefetch strategy is shown in Table 3–2.

Table 3–2 CIA Prefetch Strategy

PCI Memory Read Command	Prefetch Operation
Read	None.
Read line	Prefetch one block.
Read multiple	Prefetch until end of transaction.

A counter is used to increment the memory block address for prefetching. The output of this counter goes to the Bcache and memory like a normal DMA read command address. The returned data is sent to the next free DMA buffer in the DSW. So as one buffer is being copied down to the CIA, the other is free to accept DMA prefetched data.

At the end of a transaction all prefetched data in the DMA read buffers is ignored (that is, no prefetch caching is performed). Prefetching does not continue over 8KB page boundaries.

The 64-byte DMA read buffer in the CIA acts like two 32-byte halves during DMA commands—as one half is emptying to the PCI bus, the other half is being filled.

3.2.8 DMA Write Transaction

The translated address stored in the PA register is sent to the Bcache through the flush/read address register and also to memory via the memory port. If the Bcache contains valid (modified) data, the data is sent to the Bcache-buffer portion of the DMA buffer in the DSW and the Bcache data is invalidated. Memory data is always fetched and is put in the memory-buffer part of the DMA buffer.

The DMA write data is taken from the PCI bus and is placed in the DMA write buffer. If the data is a complete quadword, then ECC is generated and the data is sent to the PCI-write part of the DMA buffer in the DSW. If the data is an incomplete quadword, then a merge operation has to be performed. The valid Bcache or memory quadword is sent to the CIA from the DMA buffer in the DSW, and the valid bytes of the DMA write data are merged. This merged quadword then has ECC generated and is sent to the PCI-write buffer portion of the DMA buffer in the DSW.

The DMA buffer in the DSW builds the data up to 64 bytes (the block size) before sending the data to memory. The memory logic generates the memory write pulses. The memory address is read from the PA register by way of a multiplexer.

Note

Should the I/O port be busy with an I/O write/read transaction, the memory port will be available for DMA write commands.

One reason for providing a copy of the DMA write buffer in the CIA is that the data path to the DSW may not be available at the start of a DMA write transaction It could be busy transferring I/O write transaction data from the DSW to the I/O write buffers if a 21164 I/O write command had been received concurrently with a PCI DMA write command.

DMA write transaction data is always written to memory, and is never stored (cached) in the DMA write buffers across transactions.

3.3 System Data Coherency Issues

The 21164 is the processor in systems using the DECchip 21171 Core Logic Chipset and the data coherency requirements of the 21164 dominate CIA data coherency issues. The CIA supports the flush-based data coherency protocol with a Bcache but with no duplicate tag stores for either the Scache or Bcache. Therefore the CIA uses the read and flush transactions to probe the 21164 cache.

3.3.1 Basic Properties of the System

The system has these basic properties:

1. The memory arrays do only one thing at a time.
2. A read or write transaction to memory is atomic. No operation of the memory arrays can occur between the read and write parts of a read-modify-write sequence.
3. The memory array sequence for a 21164 cache ‘‘miss’’ with victim transaction is atomic in the memory array. A cache miss with victim transaction causes the 21164 to perform a read miss transaction for the block required for the cache miss transaction and then a Bcache victim transaction to write the victim data to memory. No other memory access can come between the read miss transaction and the Bcache victim transaction.
4. When ‘‘transaction A’’ gains access to memory it is said to ‘‘own’’ memory. Once ‘‘transaction A’’ ‘‘owns’’ memory there will be no access for any other transaction until ‘‘transaction A’’ has completed all of its memory activity.
5. When an I/O TLB ‘‘miss’’ occurs, the CIA issues a read command to the 21164 for the TLB entry and a memory access is also made for this data. The read command is not issued to the 21164 until memory is owned for this access.

6. The CIA processes a PCI device read transaction to memory by issuing a read command to the 21164 for the data and making a memory access for the required data. The read command is not issued until memory is ''owned'' for the memory cycle.
7. A device write transaction to memory causes the CIA to issue a flush transaction to the 21164 for the data and to start a memory access for the data. The memory access serves to write data obtained from the 21164 and also serves to write the data from the PCI device to memory. The CIA does not issue the flush command until memory is ''owned'' for the memory write transaction.
8. We can conclude that the 2-step sequences described in 5, 6, and 7 are all mutually atomic. In each case, the 2-step sequence is a probe of 21164 and a memory cycle. For any pair of sequences of the types describe in 5, 6, and 7, both steps of one of the sequences happened entirely before either of the two steps of the other sequences.

3.3.2 Post and Run I/O Write Data Coherency

The Alpha architecture allows ''posted'' or ''buffered'' write transactions to I/O devices, allowing the CIA to buffer write transactions from the 21164 to a PCI device. Several read transactions from a PCI device to main memory might be completed while write transactions to the same PCI device are buffered. The usual way for a software programmer to ensure that the write data has reached the device is to read a register on the same device.

The CIA performs the following operations to implement the Alpha architecture post and run feature.

- Before a read transaction to an I/O location is processed, all preceding 21164 write transactions to devices are flushed out of the buffers.
- Before a read transaction from a PCI device to main memory is processed all preceding write transactions by the PCI device to main memory are flushed out of the write buffers.

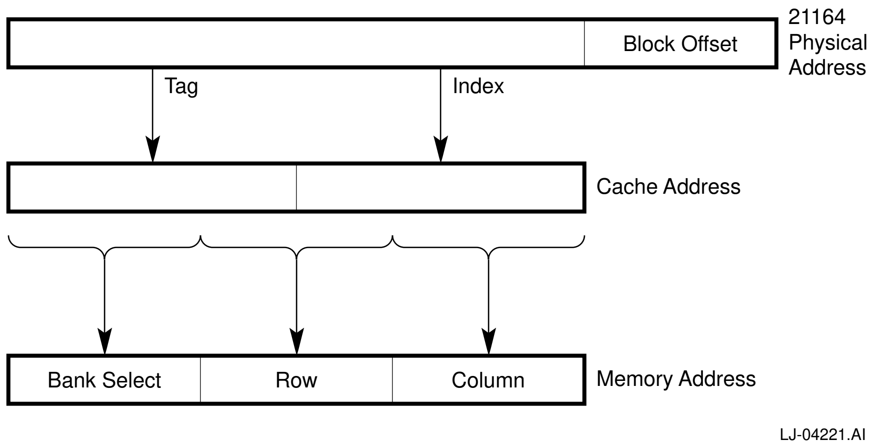
3.4 CIA Memory ras_h, cas_h, and Address Operation

The technique used by the CIA to eliminate **ras_h** cycles for victim write transactions is described here. The method is transparent to software.

3.4.1 Traditional Mapping of 21164 Address to Memory Address

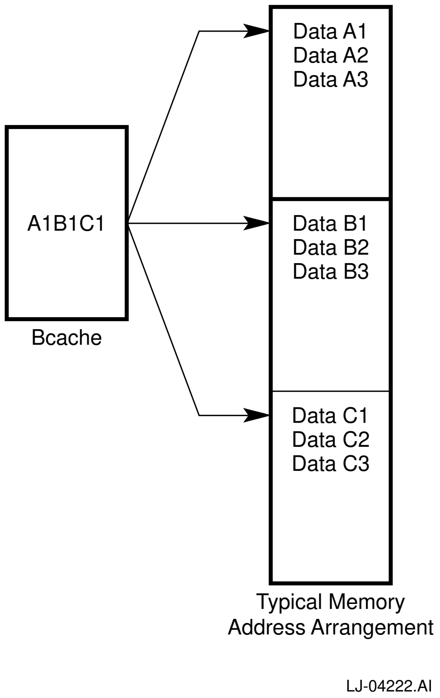
The traditional approach of mapping the 21164 address bit for bit to the memory address is shown in Figure 3–4.

Figure 3–4 Traditional Mapping to a Memory Address



Because a cache is much smaller than available memory, multiple memory locations will correspond to the same cache location. This correspondence is shown in the left-hand side of Figure 3–5 where data A1, data B1, and data C1 all map to the same cache location (A1B1C1).

Figure 3–5 Traditional Victim Cache to Memory Correspondence



Suppose that data A1 currently resides in the Bcache, but the 21164 wants to access data C1. A cache miss will occur that will fetch data C1 and overwrite A1 in the cache. If A1 is the only valid copy of the data (for example, a 21164 write transaction had previously updated A1), then A1 has to be written to memory before data C1 can be written over data A1. The displaced data A1 is referred to as a victim.

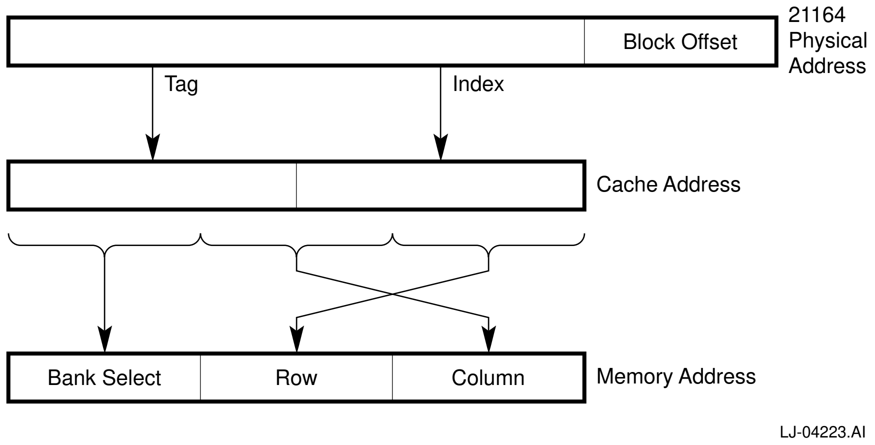
Look again at Figure 3–5 and notice that in the traditional memory addressing scheme, data C1 and the potential victim (data A1) are well separated in memory. They are separated by a multiple of *n* times the cache size. This sparse distribution of potential victim locations means a read fill block and its victim may not reside in the same memory page.

3.4.2 CIA Mapping of 21164 Address to Memory Address

The highest tag bits are used for memory bank selection in both the traditional and CIA address mapping schemes.

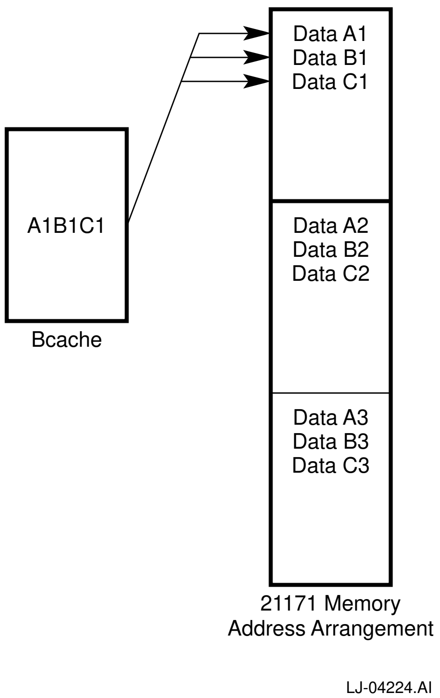
The CIA shuffles the memory address bits so that some of the high-order 21164 address bits (part of the Bcache tag) become the memory column address and the low-order CPU address bits (part of the cache index) become the memory row address as shown in Figure 3–6.

Figure 3–6 Modified Mapping to a Memory Address



Interchanging the memory row and column address bits, the write transaction target and the potential victim locations reside in the same SIMMs, even within the same row within the SIMMs, as shown in Figure 3–7. The chance of a cache block and its victim residing in the same memory page is greatly increased. This technique will remove all **ras_h** cycles from victim write transactions. In fact, all victims will be able to share an **ras_h** signal with the read miss transaction.

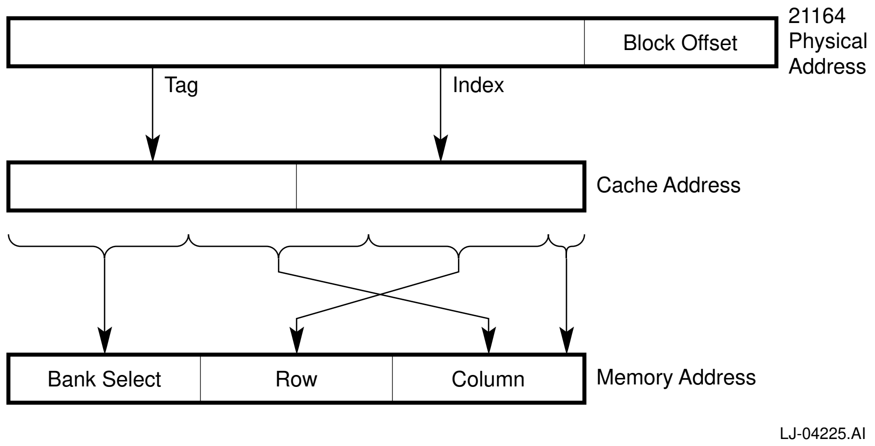
Figure 3–7 CIA Victim Cache to Memory Correspondence



However, this approach hinders DMA transactions. Consecutive blocks (data A1, data A2, and data A3), as shown in Figure 3–7, are scattered a page apart by using this memory addressing scheme. This implies that consecutive DMA read/write transaction blocks will require a **ras_h** cycle, reducing possible bandwidth. The traditional scheme does not suffer this problem.

A compromise between the two schemes, as shown in Figure 3–8, is used by the CIA. The four low-order bits of the index map straight to the low-order bits of the column address (just like in a traditional scheme). The remaining high-order index bits go to the memory row-address. The remainder of the column address uses the tag portion of the physical address.

Figure 3–8 CIA Mapping to a Memory Address



Using this compromise scheme, a DMA transaction will march through four 64-byte blocks (64 longwords), on average, before requiring a **ras_h** cycle. This constitutes a large DMA transfer, minimizing the **ras_h** penalty.

The logic that determines if a **ras_h** cycle is required is located in the memory logic block of Figure 3–2.

Figures 3–9 and 3–10 show the relationship between CPU/DMA address bits <28:04> and column and row address bits to memory. The memory address maps for the 128-bit data path and the 256-bit data path are shown in Tables 3–9 and 3–10, respectively. The user specifies row type in MBA*n*[ROW_TYPE] (MBA*n*<2:1>).

Figure 3–9 Memory Address Maps for 128-Bit Data Path

For 128-Bit-Wide Memory:

Row Type	DRAM Size	DRAM Configuration	28	27	26	25	24	23	22	21	20	19	18	17	16
00	4MB	10R x 10C	- -	- -	- -	- -	- -	- C8	- C7	- C6	- C5	R9 -	R8 -	R7 -	R6 -
01	16MB	11R x 11C 12R x 10C	- -	- -	- -	- C4	- C9	- C8	- C7	- C6	R10 -	R9 -	R8 -	R7 -	R6 -
10	64MB	12R x 12C 13R x 11C	- -	- C3	- C5	- C10	- C9	- C8	- C7	- C6	R10 -	R9 -	R8 -	R7 -	R6 -

Row Type	DRAM Size	DRAM Configuration	15	14	13	12	11	10	09	08	07	06	05*	04*
00	4MB	10R x 10C	R5 -	R4 -	R3 -	R2 -	R1 -	R0 -	- C4	- C3	- C2	- C1	- C0	- C9
01	16MB	11R x 11C 12R x 10C	R5 -	R4 -	R3 -	R2 -	R1 -	R0 -	R11 C10	- C3	- C2	- C1	- C0	- C5
10	64MB	12R x 12C 13R x 11C	R5 -	R4 -	R3 -	R2 -	R1 -	R0 -	R11 -	R12 C11	- C2	- C1	- C0	- C4

Note:
CPU tag index requires a minimum cache size of 2MB (CPU/DMA address bits <21:7> required).

LJ-04399.AI5

Figure 3–10 Memory Address Maps for 256-Bit Data Path

For 256-Bit-Wide Memory:

Row Type	DRAM Size	DRAM Configuration	28	27	26	25	24	23	22	21	20	19	18	17	16
00	4MB	10R x 10C	- -	- -	- -	- -	- C9	- C8	- C7	- C6	- C5	R9 -	R8 -	R7 -	R6 -
01	16MB	11R x 11C 12R x 10C	- -	- -	- C5	- C4	- C9	- C8	- C7	- C6	R10 -	R9 -	R8 -	R7 -	R6 -
10	64MB	12R x 12C 13R x 11C	- C4	- C3	- C5	- C10	- C9	- C8	- C7	- C6	R10 -	R9 -	R8 -	R7 -	R6 -

Row Type	DRAM Size	DRAM Configuration	15	14	13	12	11	10	09	08	07	06	05*	04*
00	4MB	10R x 10C	R5 -	R4 -	R3 -	R2 -	R1 -	R0 -	- C4	- C3	- C2	- C1	- C0	- -
01	16MB	11R x 11C 12R x 10C	R5 -	R4 -	R3 -	R2 -	R1 -	R0 -	R11 C10	- C3	- C2	- C1	- C0	- -
10	64MB	12R x 12C 13R x 11C	R5 -	R4 -	R3 -	R2 -	R1 -	R0 -	R11 -	R12 C11	- C2	- C1	- C0	- -

Note:
CPU tag index requires a minimum cache size of 2MB (CPU/DMA address bits <21:7> required).

LJ-04400.AI5

3.5 MB and LOCK Instructions

The MB (memory barrier) and LOCK instructions are described in this section.

3.5.1 MB Instruction

The MB instruction may or may not be disabled from leaving the 21164 by making the BC_CONTROL [EI_CMD_GRP3] bit clear. If BC_CONTROL [EI_CMD_GRP3] is clear, the CIA should never acknowledge receipt of an MB instruction.

CACK_EN [MB_ENABLE], in the CIA, enables or disables the CIA from acknowledging receipt of an MB instruction from the 21164. If CACK_EN [MB_ENABLE] is clear, the CIA will treat an MB instruction as a NOP. See Section 3.3.2.

3.5.2 LOCK Instruction

The CIA has to contend with locks originating on either the 21164 or the PCI. There are two cases that require that the 21164 internal lock state on a block be cleared (from a system point of view). The first is during a DMA write transaction to the locked block and the second is when a PCI device has established a lock.

The things to consider for system lock behavior are:

- DMA write transaction case—The 21164 maintains its lock flag and lock address in its bus interface unit (BIU). This lock status is correctly maintained, even if the locked block is displaced from the Scache, as long as the system sends all DMA write FLUSH commands to the 21164 (no duplicate cache tag).
- PCI lock established—A PCI device can only establish a lock on a 16-byte block of data during a DMA read transaction. Once the lock is established by the successful DMA read transaction, the PCI device has exclusive access to that 16-byte block (the PCI target may lock a larger block). This PCI lock allows the device to do an atomic read-modify-write on a 16-byte data unit.

The Alpha and PCI architectures have different mechanisms for establishing a lock:

- For PCI architecture, it is a successful read transaction (with the PCI LOCK signal asserted) that establishes a lock.
- For Alpha architecture, it is a successful write transaction (ST*n*_C) that establishes a lock.

So, a PCI read transaction with the PCI LOCK signal asserted must be treated by the 21164 as a write transaction (that is, for the Alpha architecture, it must have the same effect as a write transaction from some other processor). Example 3–1 shows an example of the 21164 and PCI competing for a memory flag.

Example 3–1 21164 and PCI Device Lock Contention

```
       21164 Atomic Update               |     PCI Device Atomic Update
       -------------------               |     ------------------------
                                         |
try_again:                               |
    LDQ_L R1, mem_flag                   |
    { modify memory flag }               |    READ/LOCK mem_flag
    STQ_C R1, mem_flag                   |    { modify mem_flag }
                                         |    WRITE/UNLOCK mem_flag
    BEQ   R,no_store                     |
    ------------------                   |
    ------------------                   |
no_store:                                |
    { check for excessive iterations}    |
    BR try_again                         |
                                         |
```

Both the 21164 and the PCI device are attempting to perform an atomic read-modify-write operation on a memory flag (a semaphore). The 21164 begins with a LDQ_L instruction but will not know if the update is successful until the companion STQ_C instruction completes successfully.

Assume that the PCI device performs a PCI read transaction with the PCI LOCK signal asserted, and obtains the lock to the memory flag before the 21164 STQ_C instruction is executed. The 21164's STQ_C instruction must fail, otherwise, the 21164 and the PCI device will both believe they have successfully modified the flag.

Before a PCI device establishes the PCI lock, the 21164 must clear its lock flag. The CIA must perform a flush transaction to the locked block, causing the 21164 to clear its lock flag. Unfortunately, the system must not only perform a DMA read transaction, but must also write the flushed block to memory. Once the CIA successfully sends the data to the PCI device, the PCI lock is established.

While the PCI lock is in effect, the 21164 may attempt to refill the flushed block. This will happen if a loop, as shown in Example 3–1, is repeatedly executing an LD*x*_L instruction. The requested block cannot be provided until the PCI lock has completed because once the block is in the 21164 internal cache, the CIA loses control of the block.

Initially, the CIA could prevent the 21164 from setting its internal lock flag for the block by using the 21164 input signal line **system_lock_flag_h**. The next time the LD*x*_L/ST*n*_C loop, the 21164 will have a cache hit on the block and will be unaware of the system lock state, so the loop will succeed this second time.

The CIA lock rules are:

- The 21164 **system_lock_flag_h** input signal line is not connected to the CIA and is always tied true.
- All DMA write transactions will result in the CIA performing a flush transaction to the 21164.
- All DMA read transactions with the PCI LOCK signal asserted will result in the CIA performing a flush transaction to the 21164. The CIA must write the flushed block to memory. The internal CIA lock address register will be set with the locking address.
- If the 21164 requests a fill that hits the CIA lock address register, then the fill is stalled until the PCI lock state is relinquished by the PCI device.
- The 21164 lock transaction will be treated as a NOP for systems with duplicate tag stores.
- The 21164 write transaction with lock will be treated as a write transaction.

3.5.3 Locks to Uncached Space

The CIA does not support LD*x*_L and ST*n*_C to noncacheable space. They are converted to LD*x* and ST*n*.

4

DECchip 21171-CA Control and Status Registers

This chapter describes the DECchip 21171-CA (CIA) control and status registers (CSRs). It also provides information about programming memory timing, configuring memory, and initializing the backup cache (Bcache).

Note

Programmers must write only zeros to register fields defined as reserved.

4.1 DECchip 21171-CA Registers

The DECchip 21171-CA CSRs are 32 bits wide and are located on NATURALLY ALIGNED 64-byte addresses. Write transactions to read-only registers could result in UNPREDICTABLE behavior while read transactions are nondestructive. Only zeros should be written to reserved bits. The register descriptions contain the initialized states.

The DECchip 21171-CA CSRs are presented here in seven groups:

- General registers
- Diagnostic registers
- Performance monitor registers
- Error registers
- System configuration registers
- PCI address and scatter-gather registers
- Address translation registers

The Alpha 21164 microprocessor CSRs, which are 64 bits wide, are described in the *Alpha 21164 Microprocessor Hardware Reference Manual*. The CSRs of most interest to uniprocessor system designers are:

- Scache control register (SC_CTL)
- Bcache control register (BC_CONTROL)
- Bcache configuration register (BC_CONFIG)

4.1.1 DECchip 21171-CA General Registers

Table 4–1 lists the DECchip 21171-CA (CIA) general registers and Section 4.2 shows and describes the individual registers.

Table 4–1 DECchip 21171-CA General Registers

Address	Mnemonic	Name
87.4000.0080	CIA_REV	CIA revision register
87.4000.00C0	PCI_LAT	PCI latency register
87.4000.0100	CIA_CTRL	CIA control register
87.4000.0400	HAE_MEM	Hardware address extension memory register
87.4000.0440	HAE_IO	Hardware address extension I/O register
87.4000.0480	CFG	Configuration register
87.4000.0600	CACK_EN	CIA acknowledgment enable register

4.1.2 DECchip 21171-CA Diagnostic Registers

Table 4–2 lists the DECchip 21171-CA (CIA) diagnostic registers and Section 4.3 shows and describes the individual registers.

Table 4–2 DECchip 21171-CA Diagnostic Registers

Address	Mnemonic	Name
87.4000.2000	CIA_DIAG	CIA diagnostic control register
87.4000.3000	DIAG_CHECK	Diagnostic check register

4.1.3 DECchip 21171-CA Performance Monitor Registers

Table 4–3 lists the DECchip 21171-CA (CIA) performance monitor registers and Section 4.4 shows and describes the individual registers.

Table 4–3 DECchip 21171-CA Performance Monitor Registers

Address	Mnemonic	Name
87.4000.4000	PERF_MONITOR	Performance monitor register
87.4000.4040	PERF_CONTROL	Performance control register

4.1.4 DECchip 21171-CA Error Registers

Table 4–4 lists the DECchip 21171-CA (CIA) error registers and Section 4.5 shows and describes the individual registers.

Table 4–4 DECchip 21171-CA Error Registers

Address	Mnemonic	Name
87.4000.8000	CPU_ERR0	CPU error information register 0
87.4000.8040	CPU_ERR1	CPU error information register 1
87.4000.8200	CIA_ERR	CIA error register
87.4000.8240	CIA_STAT	CIA status register
87.4000.8280	ERR_MASK	CIA error mask register
87.4000.8300	CIA_SYN	CIA syndrome register
87.4000.8400	MEM_ERR0	CIA memory port status register 0
87.4000.8440	MEM_ERR1	CIA memory port status register 1
87.4000.8800	PCI_ERR0	PCI error register 0
87.4000.8840	PCI_ERR1	PCI error register 1
87.4000.8880	PCI_ERR2	PCI error register 2

4.1.5 DECchip 21171-CA System Configuration Registers

Table 4–5 lists the DECchip 21171-CA (CIA) system configuration registers and Section 4.6 shows and describes the individual registers.

Table 4–5 DECchip 21171-CA System Configuration Registers

Address	Mnemonic	Name
87.5000.0000	MCR	Memory configuration register
87.5000.0600	MBA0	Memory base address register 0
87.5000.0680	MBA2	Memory base address register 2
87.5000.0700	MBA4	Memory base address register 4
87.5000.0780	MBA6	Memory base address register 6
87.5000.0800	MBA8	Memory base address register 8
87.5000.0880	MBAA	Memory base address register 10
87.5000.0900	MBAC	Memory base address register 12
87.5000.0980	MBAE	Memory base address register 14
87.5000.0B00	TMG0	Memory timing information register 0
87.5000.0B40	TMG1	Memory timing information register 1
87.5000.0B80	TMG2	Memory timing information register 2

4.1.6 DECchip 21171-CA PCI Address and Scatter-Gather Registers

Table 4–6 lists the DECchip 21171-CA (CIA) PCI address and scatter-gather registers, and Section 4.7 describes the individual registers.

Table 4–6 DECchip 21171-CA PCI Address and Scatter-Gather Registers

Address	Mnemonic	Name
87.6000.0100	TBIA	Scatter-gather translation buffer invalidate register
87.6000.0400	W0_BASE	Window base register 0
87.6000.0440	W0_MASK	Window mask register 0
87.6000.0480	T0_BASE	Translated base register 0
87.6000.0500	W1_BASE	Window base register 1
87.6000.0540	W1_MASK	Window mask register 1
87.6000.0580	T1_BASE	Translated base register 1
87.6000.0600	W2_BASE	Window base register 2
87.6000.0640	W2_MASK	Window mask register 2
87.6000.0680	T2_BASE	Translated base register 2
87.6000.0700	W3_BASE	Window base register 3
87.6000.0740	W3_MASK	Window mask register 3
87.6000.0780	T3_BASE	Translated base register 3
87.6000.07C0	W_DAC	Window Base DAC register

4.1.7 DECchip 21171-CA Address Translation Registers

Table 4–7 lists the DECchip 21171-CA (CIA) address translation registers and Section 4.8 describes the individual registers.

Table 4–7 DECchip 21171-CA Address Translation Registers

Address	Mnemonic	Name
87.6000.0800	LTB_TAG0	Lockable translation buffer Tag 0
87.6000.0840	LTB_TAG1	Lockable translation buffer Tag 1
87.6000.0880	LTB_TAG2	Lockable translation buffer Tag 2
87.6000.08C0	LTB_TAG3	Lockable translation buffer Tag 3
87.6000.0900	TB_TAG0	Translation buffer tag 0
87.6000.0940	TB_TAG1	Translation buffer tag 1
87.6000.0980	TB_TAG2	Translation buffer tag 2
87.6000.09C0	TB_TAG3	Translation buffer tag 3
87.6000.1000	TB0_PAGE0	Translation buffer 0 page 0
87.6000.1040	TB0_PAGE1	Translation buffer 0 page 1
87.6000.1080	TB0_PAGE2	Translation buffer 0 page 2
87.6000.10C0	TB0_PAGE3	Translation buffer 0 page 3
87.6000.1100	TB1_PAGE0	Translation buffer 1 page 0
87.6000.1140	TB1_PAGE1	Translation buffer 1 page 1
87.6000.1180	TB1_PAGE2	Translation buffer 1 page 2
87.6000.11C0	TB1_PAGE3	Translation buffer 1 page 3
87.6000.1200	TB2_PAGE0	Translation buffer 2 page 0
87.6000.1240	TB2_PAGE1	Translation buffer 2 page 1
87.6000.1280	TB2_PAGE2	Translation buffer 2 page 2
87.6000.12C0	TB2_PAGE3	Translation buffer 2 page 3

(continued on next page)

Table 4–7 (Cont.) DECchip 21171-CA Address Translation Registers

Address	Mnemonic	Name
87.6000.1300	TB3_PAGE0	Translation buffer 3 page 0
87.6000.1340	TB3_PAGE1	Translation buffer 3 page 1
87.6000.1380	TB3_PAGE2	Translation buffer 3 page 2
87.6000.13C0	TB3_PAGE3	Translation buffer 3 page 3
87.6000.1400	TB4_PAGE0	Translation buffer 4 page 0
87.6000.1440	TB4_PAGE1	Translation buffer 4 page 1
87.6000.1480	TB4_PAGE2	Translation buffer 4 page 2
87.6000.14C0	TB4_PAGE3	Translation buffer 4 page 3
87.6000.1500	TB5_PAGE0	Translation buffer 5 page 0
87.6000.1540	TB5_PAGE1	Translation buffer 5 page 1
87.6000.1580	TB5_PAGE2	Translation buffer 5 page 2
87.6000.15C0	TB5_PAGE3	Translation buffer 5 page 3
87.6000.1600	TB6_PAGE0	Translation buffer 6 page 0
87.6000.1640	TB6_PAGE1	Translation buffer 6 page 1
87.6000.1680	TB6_PAGE2	Translation buffer 6 page 2
87.6000.16C0	TB6_PAGE3	Translation buffer 6 page 3
87.6000.1700	TB7_PAGE0	Translation buffer 7 page 0
87.6000.1740	TB7_PAGE1	Translation buffer 7 page 1
87.6000.1780	TB7_PAGE2	Translation buffer 7 page 2
87.6000.17C0	TB7_PAGE3	Translation buffer 7 page 3

4.2 DECchip 21171-CA General Registers

This section defines and describes the DECchip 21171-CA general registers.

4.2.1 CIA Revision Register (CIA_REV)

Figure 4–1 CIA Revision Register (87.4000.0080)

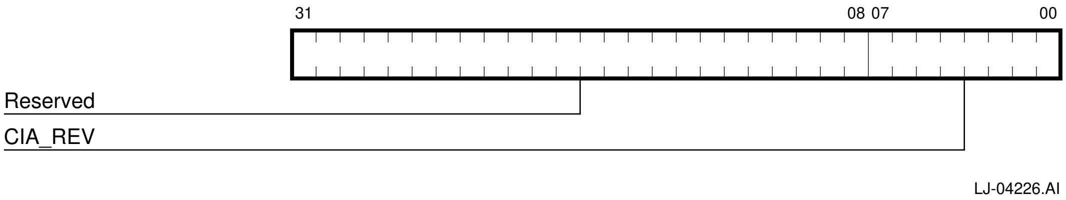


Table 4–8 CIA Revision Register

Field	Name	Type	Description
<7:0>	CIA_REV	RO,CIA Rev	CIA revision field. 0—Pass 1 CIA chip present. 1—Pass 2 CIA chip present.
<31:8>	Reserved	RO,0	—

4.2.2 PCI Latency Register (PCI_LAT)

Figure 4–2 PCI Latency Register (87.4000.00C0)

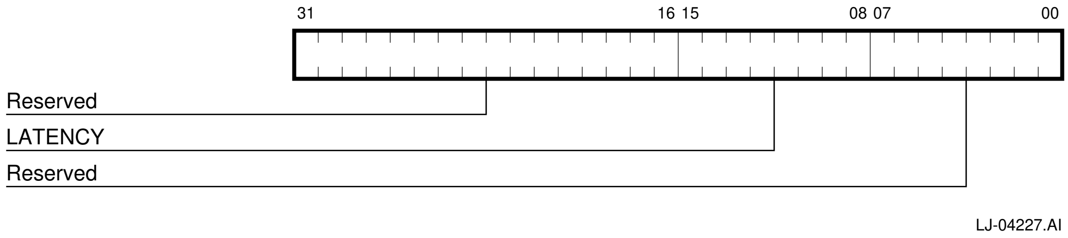


Table 4–9 PCI Latency Register

Field	Name	Type	Description
<7:0>	Reserved	RO,0	—
<15:8>	LATENCY	RW,0	PCI master latency timer in PCI clock cycles.
<31:16>	Reserved	RO,0	—

4.2.3 CIA Control Register (CIA_CTRL)

Figure 4–3 CIA Control Register (87.4000.0100)

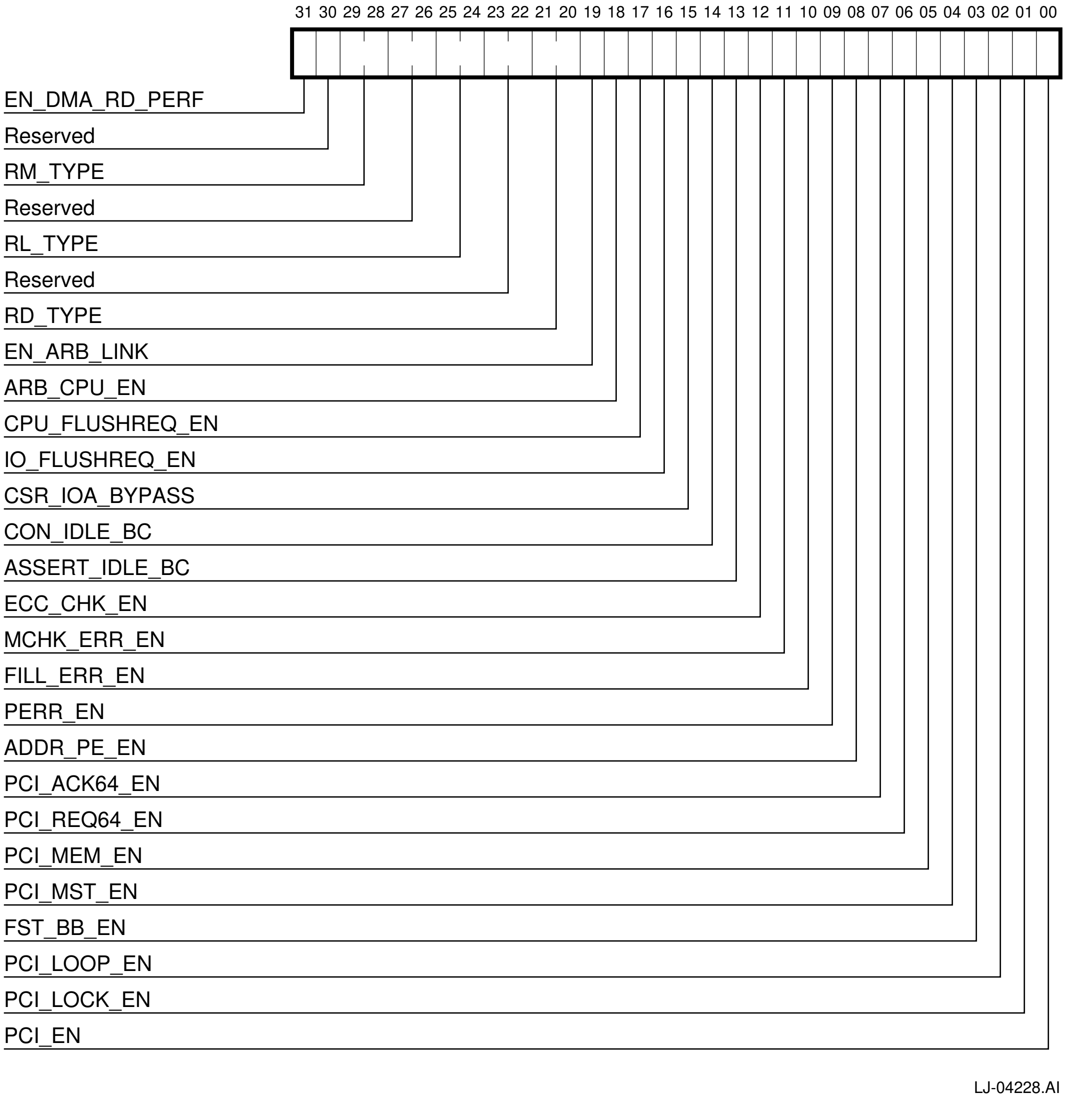


Table 4–10 CIA Control Register

Field	Name	Type	Description
<0>	PCI_EN	RW,0	0—CIA asserts reset to the PCI. 1—CIA does not assert reset to the PCI.
<1>	PCI_LOCK_EN	RW,0	0—CIA will not lock when the PCI tries to lock it. 1—CIA will lock when the PCI tries to lock it.
<2>	PCI_LOOP_EN	RW,0	0—CIA will not respond as a target when it is the master. 1—CIA will respond as a target when it is the master.
<3>	FST_BB_EN	RW,0	0—CIA will not initiate fast back-to-back PCI transactions. 1—CIA will initiate fast back-to-back PCI transactions.
<4>	PCI_MST_EN	RW,0	0—CIA will not initiate PCI transactions. 1—CIA will initiate PCI transactions.
<5>	PCI_MEM_EN	RW,0	0—CIA will not respond to PCI transactions. 1—CIA will respond to PCI transactions.
<6>	PCI_REQ64_EN	RW,0	0—CIA will not request 64-bit PCI data transactions. 1—CIA will request 64-bit PCI data transactions.
<7>	PCI_ACK64_EN	RW,0	0—CIA will not accept 64-bit PCI data transactions. 1—CIA will accept 64-bit PCI data transactions.
<8>	ADDR_PE_EN	RW,0	0—CIA will not check PCI address parity errors. 1—CIA will check PCI address parity errors.
<9>	PERR_EN	RW,0	0—CIA will not check PCI data parity errors. 1—CIA will check PCI data parity errors.
<10>	FILL_ERR_EN	RW,0	0—CIA will not assert **fill_error_h** on sysBus. 1—CIA will assert **fill_error_h** if an error occurs during a 21164 read miss transaction.
<11>	MCHK_ERR_EN	RW,0	0—CIA will not assert **error_h** on sysBus. 1—CIA will assert **error_h** to report system machine check conditions.
<12>	ECC_CHK_EN	RW,0	0— CIA will not check IOD data for ECC errors. 1—CIA will check IOD data for ECC errors.
<13>	ASSERT_IDLE_BC	RW,0	0—CIA will not assert **idle_bc_h** on sysBus when asserting **addr_bus_req_h**. 1—CIA will assert **idle_bc_h** when asserting **addr_bus_req_h**.

(continued on next page)

Table 4–10 (Cont.) CIA Control Register

Field	Name	Type	Description
<14>	CON_IDLE_BC	RW,0	0—CIA may generate a noncontiguous **idle_bc_h** signal. 1—CIA will guarantee that the **idle_bc_h** signal is contiguous.
<15>	CSR_IOA_BYPASS	RW,0	0—CIA will not bypass the I/O address queue. 1—CIA will bypass the I/O address queue. See Table 4–11.
<16>	IO_FLUSHREQ_ EN	RW,0	Used in combination with CPU_FLUSHREQ_EN. The two bits control the CIA's response to a PCI master, causing a flush request to the 21164.
<17>	CPU_FLUSHREQ_ EN	RW,0	See IO_FLUSHREQ_EN and Table 4–11.
<18>	ARB_CPU_EN	RW,0	0—Disable the bypass path from the 21164 into the memory and IOA queue. 1—Enable the bypass path from the 21164 into the memory and IOA queue.
<19>	EN_ARB_LINK	RW,0	0—Disable CPU memory read transactions from creating an arbitration link (sharing a common **ras** cycle). 1—Enable CPU memory read transactions to create an arbitration link.
<21:20>	RD_TYPE	RW,0	This field controls the prefetch algorithm used for PCI memory read commands. See Table 4–12.
<23:22>	Reserved	RO,0	—
<25:24>	RL_TYPE	RW,0	This field controls the prefetch algorithm used for PCI memory read line commands. See Table 4–12.
<27:26>	Reserved	RO,0	—
<29:28>	RM_TYPE	RW,0	This field controls the prefetch algorithm used for PCI memory read multiple commands. See Table 4–12.
<30>	Reserved	RO,0	—
<31>	EN_DMA_RD_ PERF	RW,1	0—Disable the DMA read performance logic. 1—Enable the DMA read performance logic.

Table 4–11 CPU and I/O Flush Request

IO_FLUSH_REQ_EN	CPU_FLUSH_REQ_EN	Response
0	0	Assert **mem_ack_l** immediately.
0	1	Assert **mem_ack_l** immediately and do not allow new CPU commands to proceed.
1	0	Illegal.
1	1	Wait until IOA queue is empty, then assert **mem_ack_l** but do not allow new CPU commands to proceed.

Table 4–12 PCI READ Prefetch Algorithm

RL_TYPE	Type	Description
0 0	Short	Fetch requested longword or quadword and the remainder of the cache block.
0 1	Medium	Prefetch next block and no more. Will not cross 8KB page boundary.
1 0	Long	Keep prefetching until PCI transaction completes. Will not cross 8KB page.
1 1	Reserved	—

4.2.4 Hardware Address Extension Memory Register (HAE_MEM)

This hardware address extension register is used to extend a PCI sparse-space memory address up to the full 32-bit PCI address.

In sparse-addressing mode, the 21164 address provides the low-order PCI address bits. The high-order PCI address bits <31:26> are obtained from either the hardware address extension register or the 21164 address, depending on sparse regions, as shown in Table 4–13.

Figure 4–4 Hardware Address Extension Memory Register (87.4000.0400)

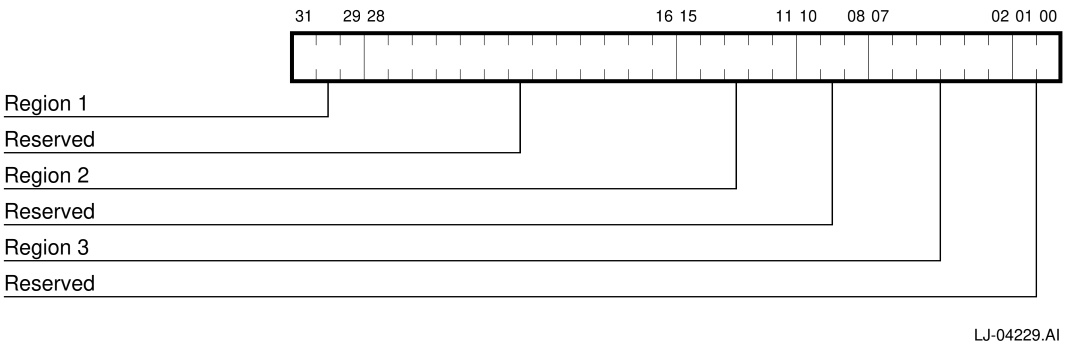


Table 4–13 HAE_MEM High-Order Sparse-Space Bits

Region[1]	PCI Address					
	<31>	<30>	<29>	<28>	<27>	<26>
1	HAE_MEM <31>	HAE_MEM <30>	HAE_MEM <29>	21164 <33>	21164 <32>	21164 <31>
2	HAE_MEM <15>	HAE_MEM <14>	HAE_MEM <13>	HAE_MEM <12>	HAE_MEM <11>	21164 <31>
3	HAE_MEM <7>	HAE_MEM <6>	HAE_MEM <5>	HAE_MEM <4>	HAE_MEM <3>	HAE_MEM <2>

[1]Region 1 is 80.0000.0000 to 83.FFFF.FFFF.
Region 2 is 84.0000.0000 to 84.FFFF.FFFF.
Region 3 is 85.0000.0000 to 85.7FFF.FFFF.

Table 4–14 lists the address range of each of the three regions.

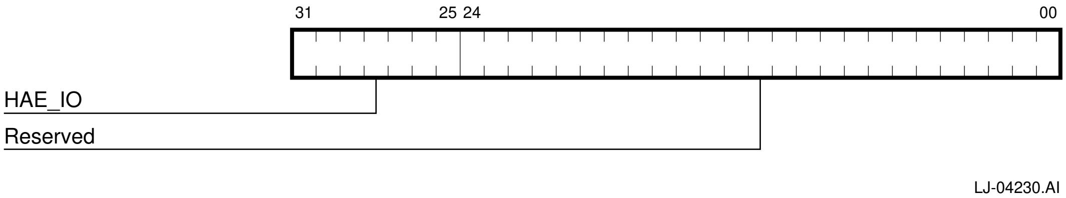
Table 4–14 Hardware Address Extension Memory Register

Field	Name	Type	21164 Address Range
<31:29>	Region 1	RW,0	80.0000.0000–83.FFFF.FFFF
<28:16>	Reserved	RW,0	—
<15:11>	Region 2	RW,0	84.0000.0000–84.FFFF.FFFF
<10:8>	Reserved	RW,0	—
<7:2>	Region 3	RW,0	85.0000.0000–85.7FFF.FFFF
<1:0>	Reserved	RW,0	—

Initializing this register to 0000.2028_{16} will make all three regions contiguous starting at PCI address 0.

4.2.5 Hardware Address Extension I/O Register (HAE_IO)

Figure 4–5 Hardware Address Extension I/O Register (87.4000.0440)



This hardware address extension register is used to extend a PCI sparse-space I/O address up to the full 32-bit PCI address. In sparse-addressing mode, the 21164 address provides the PCI address <24:0> and HAE_IO provides <31:25>.

At power-up 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 0200 0000 will make regions A and B consecutive in the lower 64MB of PCI I/O space.

Table 4–15 Hardware Address Extension I/O Register

Field	Name	Type	Description
<24:0>	Reserved	RO,0	—
<31:25>	HAE_IO	RW,0	In sparse-address mode, this field provides address bits <31:25>.

4.2.6 Configuration Register (CFG)

The CFG field bits are used as the low two address bits during an access to PCI configuration space.

Figure 4–6 Configuration Register (87.4000.0480)

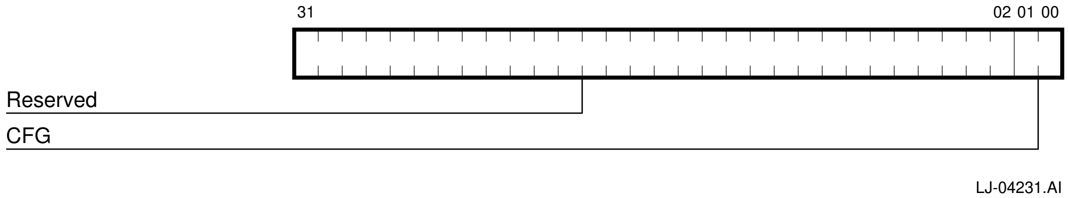


Table 4–16 Configuration Register

Field	Name	Type	Description
<1:0>	CFG	RW,0	PCI configuration space access type: **CFG<1:0>** — **Meaning** 0 0 — Type 0 configuration cycle 0 1 — Type 1 configuration cycle 1 x — Reserved
<31:2>	Reserved	RO,0	—

4.2.7 CIA Acknowledgment Enable Register (CACK_EN)

The CIA acknowledgment register (CACK_EN) allows software to decide if the CIA will acknowledge receipt any of four particular commands from the 21164 or to ignore any of these commands (NOP response).

Figure 4–7 CIA Acknowledgment Enable Register (87.4000.0600)

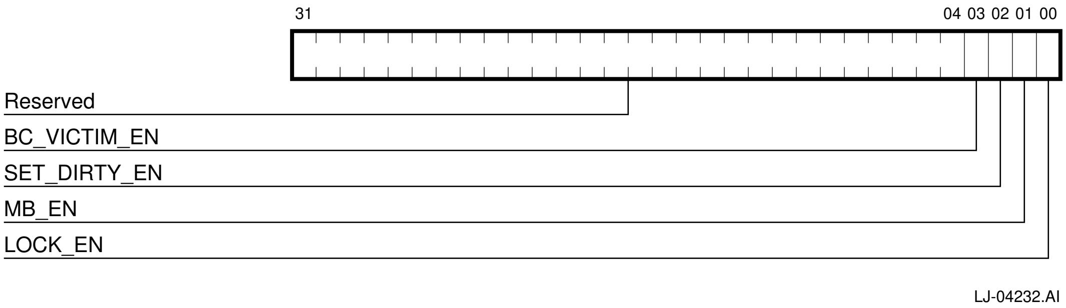


Table 4–17 CIA Acknowledgment Enable Register

Field	Name	Type	Description
<0>	LOCK_EN	RW,1	1—Enables CIA to acknowledge receipt of a LOCK command from the 21164.
<1>	MB_EN	RW,1	1—Enables CIA to acknowledge receipt of an MB command from 21164.
<2>	SET_DIRTY_EN	RW,1	1—Enables CIA to acknowledge receipt of a SET DIRTY command from the 21164.
<3>	BC_VICTIM_EN	RW,1	1—Enables CIA to acknowledge receipt of a BC_VICTIM command from the 21164.
<31:4>	Reserved	RO,0	—

4.3 DECchip 21171-CA Diagnostic Registers

This section defines and describes the DECchip 21171-CA diagnostic registers.

4.3.1 CIA Diagnostic Control Register (CIA_DIAG)

The CIA_DIAG is used as a diagnostic/debug tool, allowing various errors to be tested.

Figure 4–8 CIA Diagnostic Control Register (87.4000.2000)

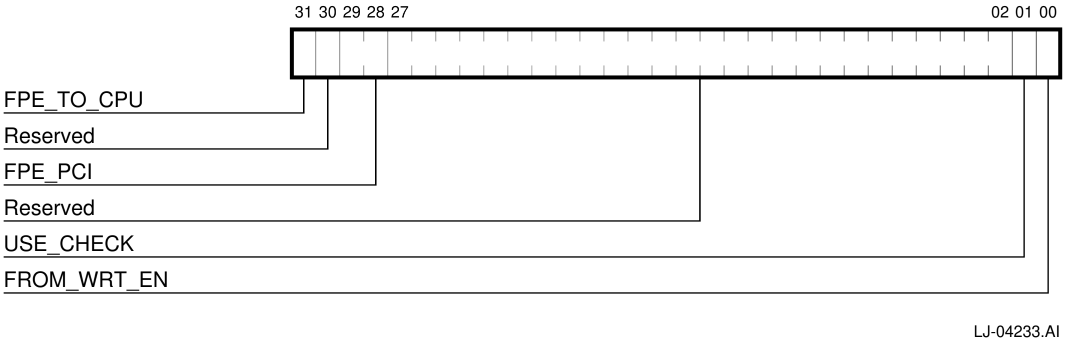


Table 4–18 CIA Diagnostic Control Register

Field	Name	Type	Description
<0>	FROM_WRT_EN	RW,0	0–Flash ROM cannot be programmed. A write transaction to flash ROM results in a system machine check. 1—The flash ROM can be programmed.
<1>	USE_CHECK	RW,0	1—Use DIAG_CHECK<7:0> for DMA write cycle ECC sent on the IOD bus.
<27:2>	Reserved	RO,0	—

(continued on next page)

Table 4–18 (Cont.) CIA Diagnostic Control Register

Field	Name	Type	Description
<29:28>	FPE_PCI	RW,0	This field allows bad parity to be forced on the PCI for diagnostic purposes: **FPE_PCI <1:0>** — **Meaning** 0 0 — Normal parity asserted on PCI. 0 1 — Bad parity is forced onto **ad_h<31:0>** during data cycles. 1 0 — Bad parity is forced onto **ad_h<63:32>** during data cycles. 1 1 — Bad parity is forced onto **ad_h<31:0>** and **ad_h<63:32>** during address and data cycles.
<30>	Reserved	RO,0	—
<31>	FPE_TO_CPU	RW,0	1—A parity error is forced on the 21164 address/command bus when the CIA is the bus master.

4.3.2 Diagnostic Check Register (DIAG_CHECK)

The DIAG_CHECK is used to cause diagnostic DMA write transactions to force a known ECC pattern into memory. This register provides the ECC value that is written to memory if CIA_DIAG[USE_CHECK] is set.

Figure 4–9 Diagnostic Check Register (87.4000.3000)

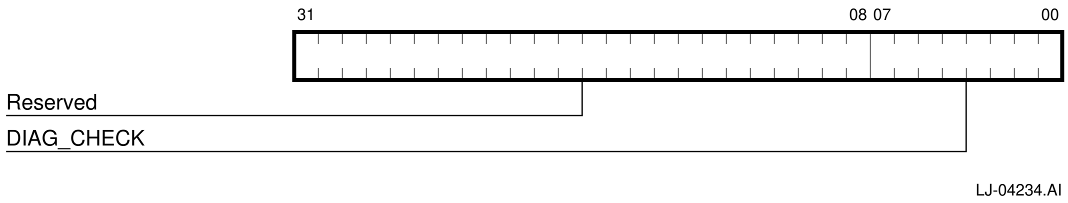


Table 4–19 Diagnostic Check Register

Field	Name	Type	Description
<7:0>	DIAG_CHECK	RW,UNDEFINED	This register provides the quadword ECC for diagostic DMA write transactions.
<31:8>	Reserved	RO,0	—

4.4 DECchip 21171-CA Performance Monitor Registers

This section defines and describes the DECchip 21171-CA (CIA) performance monitor registers.

4.4.1 Performance Monitor Register (PERF_MONITOR)

The performance monitor register is two 16-bit counters that can be programmed to count a variety of events. The performance control register is used to set up the counters. Each counter can be programmed to count events such as 21164 read miss transactions received by the CIA or DMA write transactions. This register can also be set up as a single 32-bit counter by directing the high counter to count the low counter overflow.

Figure 4–10 Performance Monitor Register (87.4000.4000)

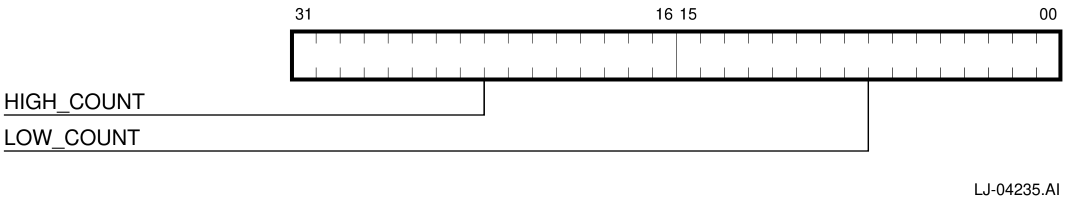


Table 4–20 Performance Monitor Register

Field	Name	Type	Description
<15:0>	LOW_COUNT	RO,0	Value of the low counter.
<31:16>	HIGH_COUNT	RO,0	Value of the high counter.

4.4.2 Performance Control Register (PERF_CONTROL)

Figure 4–11 Performance Control Register (87.4000.4040)

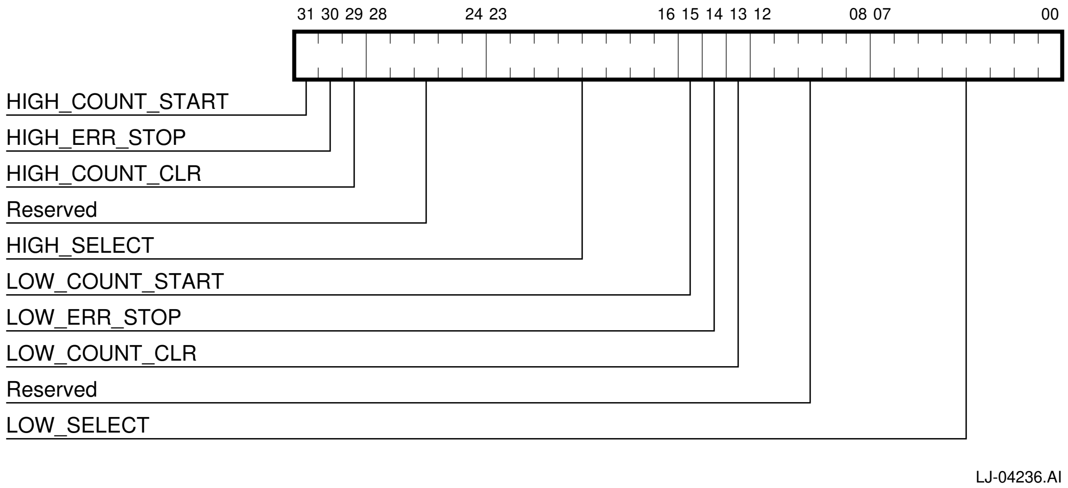


Table 4–21 Performance Control Register

Field	Name	Type	Description
<7:0>	LOW_SELECT	RW,0	See Table 4–22.
<12:8>	Reserved	RO,0	—
<13>	LOW_COUNT_CLR	WO,0	Write a one to clear the low counter.
<14>	LOW_ERR_STOP	RW,0	If the CIA encounters an error and this bit is set, then stop the low error counter.
<15>	LOW_COUNT_START	RW,0	0—Do not count. Keep current values. 1—Start counting.
<23:16>	HIGH_SELECT	RW,0	See Table 4–22.
<28:24>	Reserved	RO,0	—
<29>	HIGH_COUNT_CLR	WO,0	Write a one to clear the high counter.
<30>	HIGH_ERR_STOP	RW,0	If the CIA encounters an error and this bit is set, then stop the high error counter.

(continued on next page)

Table 4–21 (Cont.) Performance Control Register

Field	Name	Type	Description
<31>	HIGH_COUNT_START	RW,0	0—Do not count. Keep current values. 1—Start counting.

Table 4–22 PERF_CONTROL Low/High Select Encodings

Select[1]	Description
0000 0000	LOW_SELECT—The value is reserved. HIGH_SELECT—Make 32-bit counter.
0000 0001	Counting clock cycles—always increment.
0000 0010	Counting refresh cycles.
0001 0000	Counting number of 21164 commands acknowledged
0001 0001	Counting number of 21164 read commands (modify or not).
0001 0010	Counting number of 21164 read miss commands (not modify).
0001 0011	Counting number of 21164 read miss modify commands.
0001 0100	Counting number of 21164 Bcache victim commands that are acknowledged by the CIA.
0001 0101	Counting number of 21164 lock commands that are acknowledged by the CIA.
0001 0110	Counting number of 21164 memory barrier commands that are acknowledged by the CIA.
0001 0111	Counting number of 21164 FETCH or FETCH_M commands.
0001 1000	Counting number of 21164 write block commands (lock or not).
0010 0000	Counting number of 21164 memory commands.
0010 0001	Counting number of 21164 I/O commands.
0010 0010	Counting number of 21164 I/O read commands.
0010 0011	Counting number of 21164 I/O write commands.
0010 0100	Counting number of CIA system commands issued (read/flush).
0010 0101	Counting number of 21164 system read commands issued.
0010 0110	Counting number of 21164 system flush commands issued.
0010 0111	Counting number of times CIA received NOACK as a response.

[1]Select = HIGH_SELECT <23:16> and LOW_SELECT <7:0>.

(continued on next page)

Table 4–22 (Cont.) PERF_CONTROL Low/High Select Encodings

Select[1]	Description
0010 1000	Counting number of times CIA received Scache/ACK as a response.
0010 1001	Counting number of times CIA received Bcache/ACK as a response.
0011 0000	Counting number of DMA read commands (total).
0011 0001	Counting number of DMA read commands.
0011 0010	Counting number of DMA read commands (read line).
0011 0011	Counting number of DMA read commands (read multiple).
0011 0100	Counting number of DMA write commands (total).
0011 0101	Counting number of DMA write commands.
0011 0110	Counting number of DMA write commands (write and invalidate).
0011 0111	Counting number of DMA dual address cycles.
0011 1000	Counting number of DMA cycles that CIA responded to by issuing a retry.
0011 1001	Counting number of I/O cycles on which CIA received a retry response.
0100 0000	Counting number of times PCI bus lock was established.
0100 0001	Counting number of times 21164 tried to access block that was locked.
0100 0010	Counting number of times DMA commands (that caused a flush) hit on the victim address.
0101 0000	Counting number of times CIA had to refill the translation lookaside buffer (TLB).
0110 0000	Counting number of single-bit ECC errors detected.
All others	Reserved, unused, not counting.

[1]Select = HIGH_SELECT <23:16> and LOW_SELECT <7:0>.

4.5 DECchip 21171-CA Error Registers

This section defines and describes the DECchip 21171-CA (CIA) error registers.

4.5.1 CPU Error Information Register 0 (CPU_ERR0)

The low-order address bits of the sysBus are locked into this register when the CIA detects an error event. Clearing all the error bits in CIA_ERR unlocks this register. When CPU_ERR0 is not locked, its contents are UNDEFINED.

The information in CPU_ERR0 and CPU_ERR1 is only related to sysBus parity errors detected by the CIA.

Figure 4–12 CPU Error Information Register 0 (87.4000.8000)

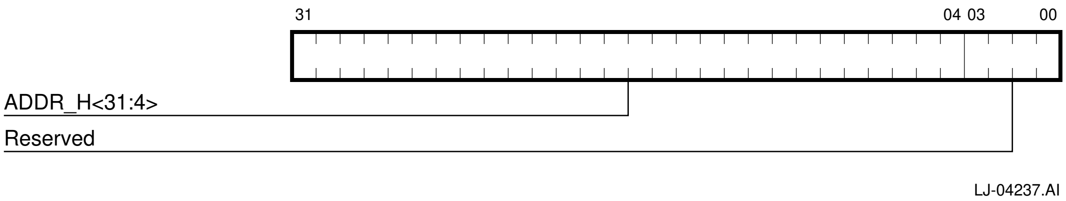


Table 4–23 CPU Error Information Register 0

Field	Name	Type	Description
<3:0>	Reserved	R0,0	—
<31:4>	ADDR_H<31:4>	RO,UNDEFINED	Contains address bits <31:4> of the current address on the sysBus when a sysBus error occurs.

4.5.2 CPU Error Information Register 1 (CPU_ERR1)

The mask and command field and the remaining address field on the sysBus are locked into CPU_ERR1 when the CIA detects a event error. Clearing all the error bits in CIA_ERR unlocks CPU_ERR1. When CPU_ERR1 is not locked, its contents are UNDEFINED.

The information in CPU_ERR0 and CPU_ERR1 is only related to sysBus parity errors detected by the CIA (CPU_ERR<2> = 1).

Figure 4–13 CPU Error Information Register 1 (87.4000.8040)

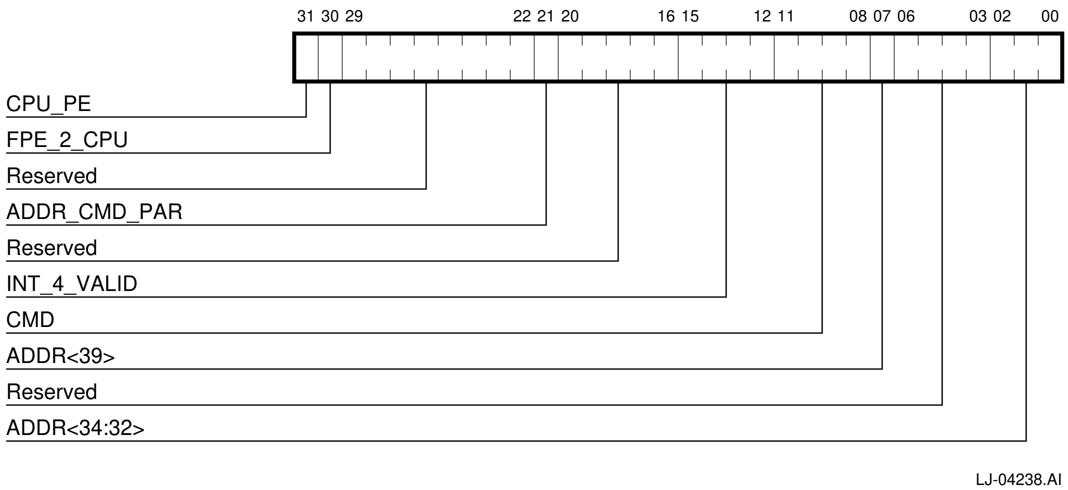


Table 4–24 CPU Error Information Register 1

Field	Name	Type	Description
<2:0>	ADDR<34:32>	RO, UNDEFINED	Contains **addr_h<34:32>** from the sysBus.
<6:3>	Reserved	RO,0	—
<7>	ADDR<39>	RO, UNDEFINED	Contains **addr_h<39>** from the sysBus.
<11:8>	CMD	RO, UNDEFINED	Contains the command from the sysBus.

(continued on next page)

Table 4–24 (Cont.) CPU Error Information Register 1

Field	Name	Type	Description
<15:12>	INT4_VALID	RO, UNDEFINED	Contains **int4_valid_h<3:0>** from the sysBus.
<20:16>	Reserved	RO,0	—
<21>	ADDR_CMD_PAR	RO, UNDEFINED	Contains the parity bit from the sysBus.
<29:22>	Reserved	RO,0	—
<30>	FPE_2_CPU	RO, UNDEFINED	Contains a copy of the CSR bit to force bad parity on the sysBus.
<31>	CPU_PE	RO, UNDEFINED	1—Indicates that the CIA has logged a sysBus error.

4.5.3 CIA Error Register (CIA_ERR)

The CIA error register is used by the CIA to log information about an error condition detected in the CIA. All bits of CIA_ERR, except the LOST bit, will be locked until the CIA_ERR register is cleared by a write transaction. The LOST bit will be set whenever an error is detected and CIA_ERR is already locked.

Figure 4–14 CIA Error Register (87.4000.8200)

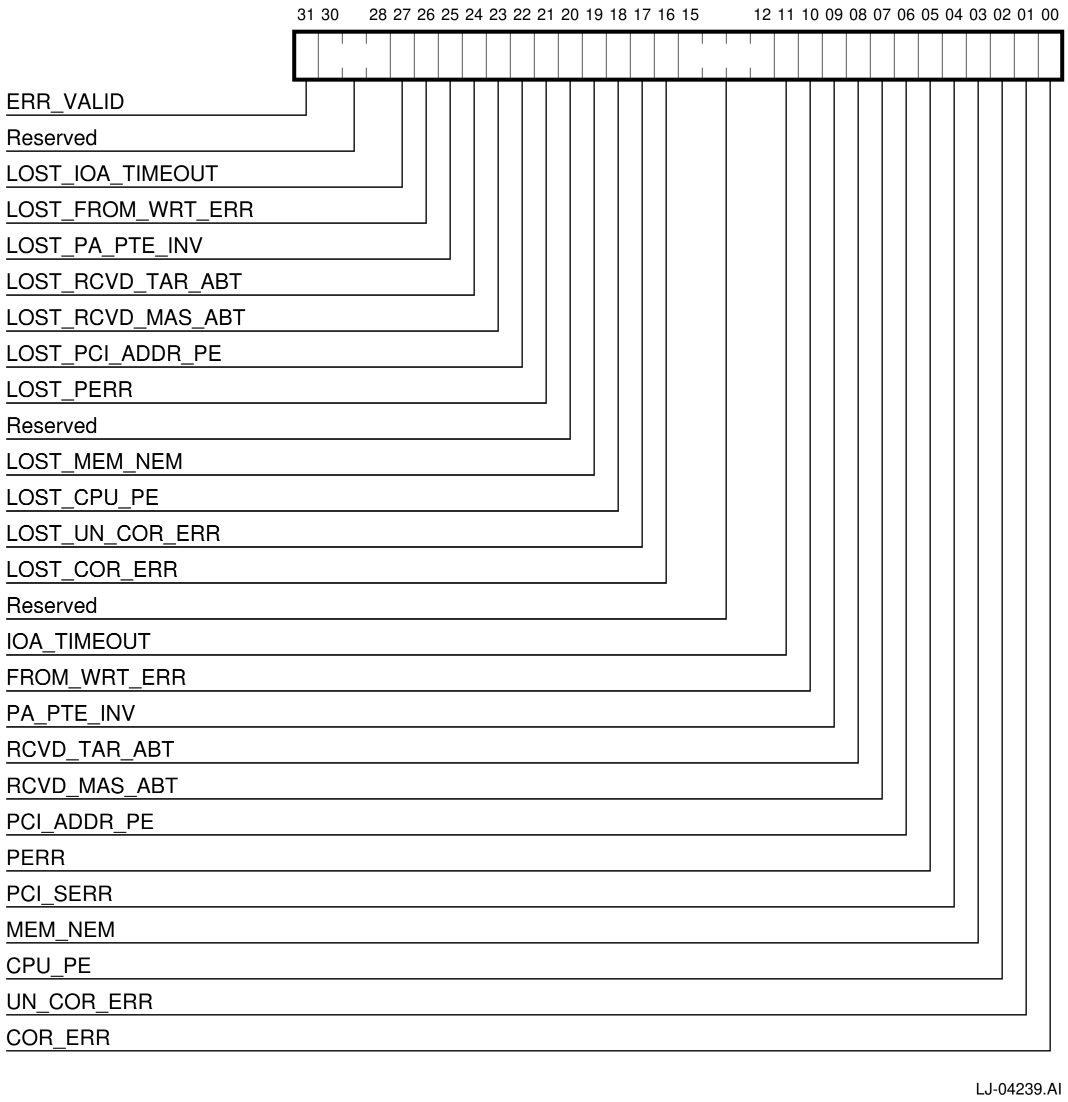


Table 4–25 CIA Error Register

Field	Name	Type	Description
<0>	COR_ERR	RW1C,0	Corrected single-bit ECC error detected. This error cannot occur for a 21164 to memory read/write: • 21164/memory read ECC errors are detected by the 21164. • 21164/memory write transactions are not checked. This error is applicable to a DMA transaction, a scatter-gather TLB miss, or an I/O write transaction from the 21164.
<1>	UN_COR_ERR	RW1C,0	Uncorrectable ECC error detected (double-bit error or single-bit error if correction disabled). This error cannot occur for a 21164-to-memory read/write transaction because: • 21164/memory read transaction ECC errors are detected by the 21164. • 21164/memory write transactions are not checked. This error is applicable to a DMA transaction, a scatter-gather TLB miss, or an I/O write transaction from the 21164.
<2>	CPU_PE	RW1C,0	sysBus parity error detected.
<3>	MEM_NEM	RW1C,0	Access to nonexistent memory detected.
<4>	PCI_SERR	RW1C,0	PCI bus system error (SERR) detected.
<5>	PERR	RW1C,0	PCI bus data parity error (PERR) detected.
<6>	PCI_ADDR_PE	RW1C,0	PCI bus address parity error detected.
<7>	RCVD_MAS_ABT	RW1C,0	PCI master state machine generated master abort.
<8>	RCVD_TAR_ABT	RW1C,0	PCI master state machine received target abort.
<9>	PA_PTE_INV	RW1C,0	Invalid page table entry (PTE) on scatter-gather access.
<10>	FROM_WRT_ERR	RW1C,0	Write to flash ROM attempted without setting CIA_DIAG[FROM_WRT_EN].

(continued on next page)

Table 4–25 (Cont.) CIA Error Register

Field	Name	Type	Description
<11>	IOA_TIMEOUT	RW1C,0	I/O timeout occurred. I/O read/write transaction failed to complete in 1 second.
<15:12>	Reserved	RO,0	—
<16>	LOST_COR_ERR	RO,0	A correctable error was detected while CIA_ERR was locked.
<17>	LOST_UN_COR_ ERR	RO,0	An uncorrectable error was detected while CIA_ERR was locked.
<18>	LOST_CPU_PE	RO,0	A sysBus parity error was detected while CIA_ERR was locked.
<19>	LOST_MEM_NEM	RO,0	An access to nonexistent memory was detected while CIA_ERR was locked.
<20>	Reserved	RO,0	—
<21>	LOST_PERR	RO,0	A PCI data parity error was detected while CIA_ERR was locked.
<22>	LOST_PCI_ADDR_ PE	RO,0	A PCI address parity error was detected while CIA_ERR was locked.
<23>	LOST_RCVD_ MAS_ABT	RO,0	The PCI master state machine generated a master abort while CIA_ERR was locked.
<24>	LOST_RCVD_TAR_ ABT	RO,0	The PCI master state machine received a target abort while CIA_ERR was locked.
<25>	LOST_PA_PTE_ INV	RO,0	An invalid page table entry on scatter-gather access occurred while CIA_ERR was locked.
<26>	LOST_FROM_ WRT_ERR	RO,0	A write transaction to flash ROM was attempted, with CIA_DIAG[FROM_WRT_EN] clear, while CIA_ERR was locked.
<27>	LOST_IOA_ TIMEOUT	RO,0	An I/O timeout occurred while CIA_ERR was locked. An I/O read/write transaction failed to complete in 1 second.
<30:28>	Reserved	RO,0	—
<31>	ERR_VALID	RO,0	An error has been detected and the CIA error registers are all locked.

4.5.4 CIA Status Register (CIA_STAT)

This register contains status information about the error stored in CIA_ERR. The register provides the state of the CIA when the error was detected.

Figure 4–15 CIA Status Register (87.4000.8240)

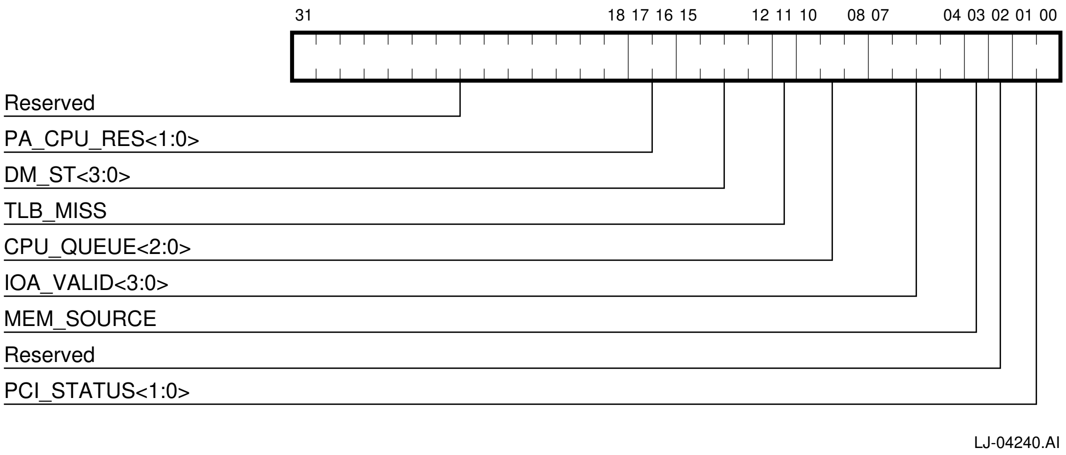


Table 4–26 CIA Status Register

Field	Name	Type	Description
<0>	PCI_STATUS<0>	RO,0	1—The PCI target state machine is active.
<1>	PCI_STATUS<1>	RO,0	1—The PCI master state machine is active.
<2>	Reserved	RO,0	—
<3>	MEM_SOURCE	RO,0	0—21164 is the source of the memory cycle. 1—PCI is the source of the memory cycle.
<7:4>	IOA_VALID<3:0>	RO,0	Valid bits for the I/O command/address queue.
<10:8>	CPU_QUEUE<2:0>	RO,0	Valid bits for the 21164 command/address queue.
<11>	TLB_MISS	RO,0	1—A TLB miss refill was in progress when the error occurred.

(continued on next page)

Table 4–26 (Cont.) CIA Status Register

Field	Name	Type	Description
<15:12>	DM_ST<3:0>	RO, UNDEFINED	This field represents the state of the logic that moves data between the CIA and the DSW: **DM_ST<3:0>** — **Definition** 0000 — Idle 0001 — Restarting DSW IOW buffers 0010 — I/O write transaction to the 64-bit PCI bus 0110 — DMA read transaction or TLB miss 0111 — DMA write transaction 1010 — I/O read transaction to an internal CIA CSR 1011 — I/O read transaction to the PCI bus (32-bit/64-bit). All others — Reserved
<17:16>	PA_CPU_RES<1:0>	RO,0	21164 response for the DMA: 00 = No response 01 = NOACK 10 = Scache hit 11 = Bcache hit
<31:18>	Reserved	RO,0	—

4.5.5 CIA Error Mask Register (ERR_MASK)

ERR_MASK is used to disable the logging and reporting of errors. A zero disables logging/reporting of errors while a one enables logging/reporting of errors.

Figure 4–16 CIA Error Mask Register (87.4000.8280)

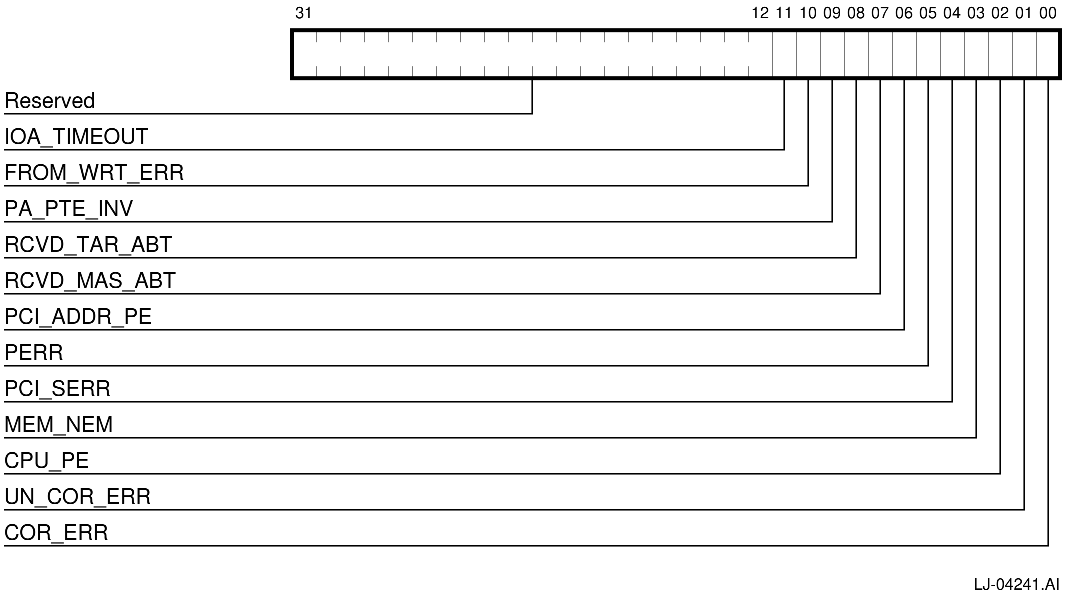


Table 4–27 CIA Error Mask Register

Field	Name	Type	Description
<0>	CORR_ERR	RW,0	Disable/enable error logging/reporting for correctable ECC errors.
<1>	UN_COR_ERR	RW,0	Disable/enable error logging/reporting for uncorrectable ECC errors.
<2>	CPU_PE	RW,0	Disable/enable error logging/reporting for 21164 parity errors.
<3>	MEM_NEM	RW,0	Disable/enable error logging/reporting for nonexistent memory access errors.

(continued on next page)

Table 4–27 (Cont.) CIA Error Mask Register

Field	Name	Type	Description
<4>	PCI_SERR	RW,0	Disable/enable error logging/reporting for PCI system errors.
<5>	PERR	RW,0	Disable/enable error logging/reporting for PCI data parity errors.
<6>	PCI_ADDR_PE	RW,0	Disable/enable error logging/reporting for PCI address parity errors.
<7>	RCVD_MAS_ABT	RW,0	Disable/enable error logging/reporting for PCI master abort errors.
<8>	RCVD_TAR_ABT	RW,0	Disable/enable error logging/reporting for PCI target abort errors.
<9>	PA_PTE_INV	RW,0	Disable/enable error logging/reporting for invalid PTE errors.
<10>	FROM_WRT_ERR	RW,0	Disable/enable error logging/reporting for flash ROM write errors.
<11>	IOA_TIMEOUT	RW,0	Disable/enable error logging/reporting for I/O timeout errors.
<31:12>	Reserved	RW,0	—

4.5.6 CIA Syndrome Register (CIA_SYN)

The CIA uses CIA_SYN to log syndrome information about an error detected by the ECC checkers. The syndrome information is locked into CIA_SYN when a CIA error occurs. Clearing all the error bits in CIA_ERR unlocks this register. When CIA_SYN is not locked its contents are UNDEFINED.

Figure 4–17 CIA Syndrome Register (87.4000.8300)

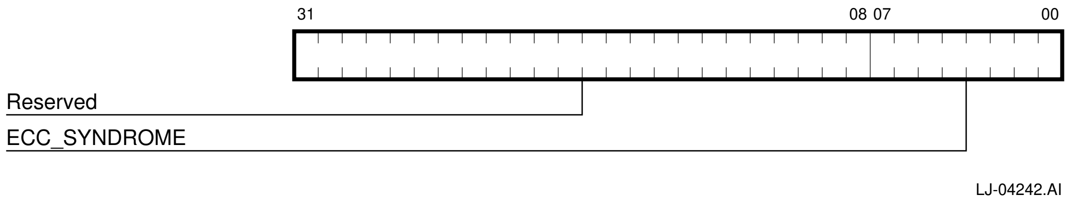


Table 4–28 CIA Syndrome Register

Field	Name	Type	Description
<7:0>	ECC_SYNDROME	RO, UNDEFINED	The syndrome is locked in ECC_SYNDROME <7:0> when an error is detected by the CIA.
<31:8>	Reserved	RO,0	—

Table 4–29 lists the ECC syndromes for single-bit errors.

Table 4–29 ECC Syndromes for Single-Bit Errors

Bit_{10}	$Syndrome_{16}$	Bit_{10}	$Syndrome_{16}$	Bit_{10}	$Syndrome_{16}$	Bit_{10}	$Syndrome_{16}$
Syndromes for Data Bits							
00	CE	01	CB	02	D3	03	D5
04	D6	05	D9	06	DA	07	DC
08	23	09	25	10	26	11	29
12	2A	13	2C	14	31	15	34
16	0E	17	0B	18	13	19	15
20	16	21	19	22	1A	23	1C
24	E3	25	E5	26	E6	27	E9
28	EA	29	EC	30	F1	31	F4
32	4F	33	4A	34	52	35	54
36	57	37	58	38	5B	39	5D
40	A2	41	A4	42	A7	43	A8
44	AB	45	AD	46	B0	47	B5
48	8F	49	8A	50	92	51	94
52	97	53	98	54	9B	55	9D
56	62	57	64	58	67	59	68
60	6B	61	6D	62	70	63	75
Syndromes for Check Bits							
00	01	01	02	02	04	03	08
04	10	05	20	06	40	07	80

4.5.7 CIA Memory Port Status Register 0 (MEM_ERR0)

The low-order address bits of the memory port address bus are locked into MEM_ERR0 upon a CIA-detected error. Clearing all the error bits in CIA_ERR unlocks this register. If the register is not locked, the contents are UNDEFINED.

Figure 4–18 CIA Memory Port Status Register 0 (87.4000.8400)

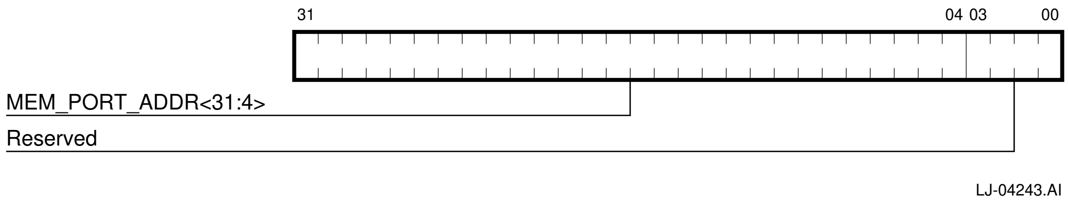


Table 4–30 CIA Memory Port Status Register 0

Field	Name	Type	Description
<3:0>	Reserved	RO, UNDEFINED	—
<31:4>	MEM_PORT_ ADDR<31:4>	RO, UNDEFINED	Contains address bits <31:4> of the current address in the memory port when the CIA detects an error.

4.5.8 CIA Memory Port Status Register 1 (MEM_ERR1)

The command memory mask (**int4_valid<3:0>** from the CPU), memory sequencer state, the source of the command, and encoded set select field, and the remaining address field are locked into MEM_ERR1 upon a CIA error. Clearing all error bits in CIA_ERR unlocks this register. If the register is not locked, the contents are UNDEFINED.

Figure 4–19 CIA Memory Port Status Register 1 (87.4000.8440)

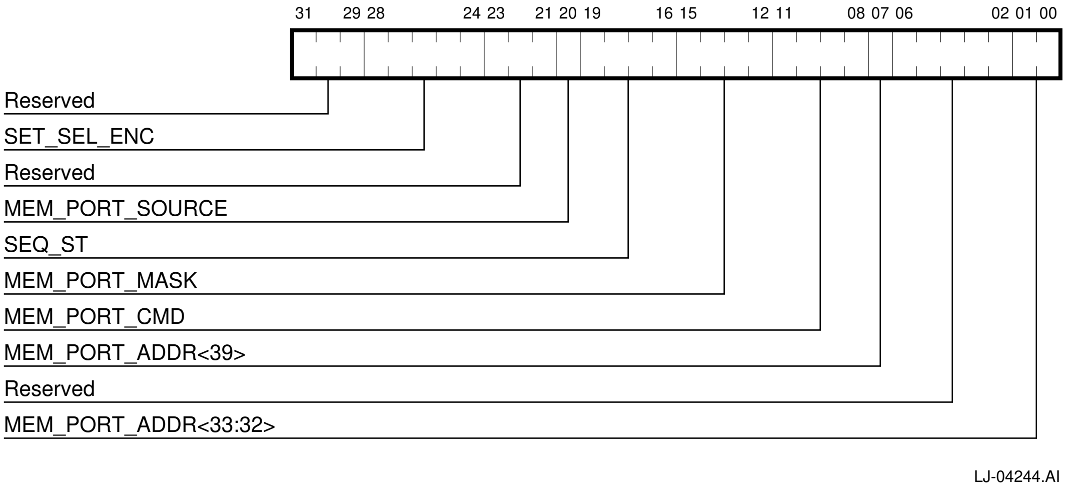


Table 4–31 CIA Memory Port Status Register 1

Field	Name	Type	Description
<1:0>	MEM_PORT_ ADDR<33:32>	RO, UNDEFINED	Captures memory port address bits <33:32> when the CIA detects an error.
<6:2>	Reserved	RO,0	—
<7>	MEM_PORT_ ADDR<39>	RW, UNDEFINED	Captures memory port address bit <39> when the CIA detects an error.
<11:8>	MEM_PORT_CMD	RO, UNDEFINED	Contains command field of the memory port when a CIA error is detected, as listed in Table 4–32.

(continued on next page)

Table 4–31 (Cont.) CIA Memory Port Status Register 1

Field	Name	Type	Description
<15:12>	MEM_PORT_MASK	RW, UNDEFINED	The mask bits when the error occurred.
<19:16>	SEQ_ST	RW, UNDEFINED	The memory sequencer state when the error occurred, as listed in Table 4–33.
<20>	MEM_PORT_SOURCE	RW, UNDEFINED	Source of the memory command: 0—21164 is source of memory command. 1—DMA is source of memory command.
<23:21>	Reserved	RO,0	—
<28:24>	SET_SEL_ENC	RW, UNDEFINED	Encoded set select indicates which memory set was active when the error occurred as listed in Table 4–34.
<31:29>	Reserved	RO,0	—

Table 4–32 Memory Port Command Field (MEM_PORT_CMD)

MEM_PORT_ SOURCE	MEM_PORT_CMD	Description
0	011X	21164 WRITE BLOCK or WRITE BLOCK LOCK
0	10XX	21164 READ MISS, READ MISS MODIFY
0	1100	21164 BC_VICTIM
0	111X	21164 READ MISS MODIFY
1	10XX	DMA READ, DMA READ MODIFY
1	001X	DMA WRITE

Table 4–33 Memory Sequencer State Field (SEQ_ST)

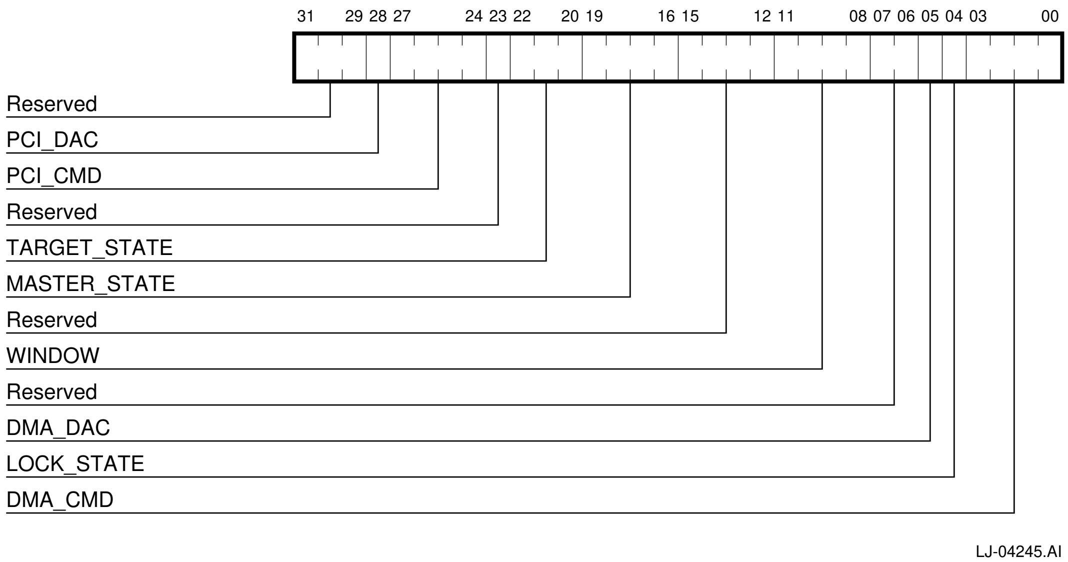
SEQ_ST	Description
0000	Idle.
0001	DMA read or write.
0010, 0011	21164 READ MISS (or READ MISS MDOIFY) with victim.
0100, 0101, 0110	21164 READ MISS (or READ MISS MODIFY) with no victim.
01F11, 1000, 1001	Refresh.
1100	Idle. Waiting for DMA pending read.
1110, 1111	Idle. **ras_h** precharge

Table 4–34 Encoded Set Select Field (SET_SEL_ENC)

SET_SEL_ENC	Description
00000	Set 0 selected
00001	Set 1 selected
00010	Set 2 selected
00011	Set 3 selected
00100	Set 4 selected
00101	Set 5 selected
00110	Set 6 selected
00111	Set 7 selected
01000	Set 8 selected
01001	Set 9 selected
01010	Set 1 selected
01011	Set B selected
01100	Set C selected
01101	Set D selected
01110	Set E selected
01111	Set F selected
10000	No set selected
11111	Refresh cyle.
—	All other codes reserved.

4.5.9 PCI Error Register 0 (PCI_ERR0)

Figure 4–20 PCI Error Register 0 (87.4000.8800)



PCI_ERR0 is used by the CIA to log information about the state of the PCI interface when an error is detected by the CIA. PCI_ERR0 is locked, as are all CIA error registers, when the CIA detects an error. PCI_ERR0 is unlocked when CIA_ERR is cleared. When the register is not locked, the contents are UNPREDICTABLE.

The data in the WINDOW, DMA_DAC, and DMA_CMD files is associated with the address stored in PCI_ERR1. This group and PCI_ERR1 hold information related to memory errors that occur while the CIA is handling a DMA transaction

- Correctable errors (CIA_ERR<0>)
- Uncorrectable errors (CIA_ERR<1>)
- Access to nonexistent memory (CIA_ERR<3>)
- Invalid page table entry (CIA_ERR<9>)

The data in the PCI_DAC, PCI_CMD, TARGET_STATE, and MASTER_STATE fields is associated with the address stored in PCI_ERR2. This group and PCI_ERR2 hold information about PCI bus errors

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

The LOCK_STATE field has general information about the current state of the CIA not specifically associated with either PCI_ERR1 or PCI_ERR2.

Table 4–35 PCI Error Register 0

Field	Name	Type	Description
<3:0>	DMA_CMD	RO, UNDEFINED	This field holds the current DMA command on the PCI.
<4>	LOCK_STATE	RO, UNDEFINED	1—The CIA was locked when the error was detected.
<5>	DMA_DAC	RO, UNDEFINED	1—The error occurred during a PCI dual address cycle (DAC).
<7:6>	Reserved	RO,0	—
<11:8>	WINDOW	RO, UNDEFINED	Indicates which window (if any) was selected by the PCI address. **WINDOW<3:0> — Window Selected** 0000 — No window active 0001 — Window 0 hit 0010 — Window 1 hit 0100 — Window 2 hit 1000 — Window 3 hit
<15:12>	Reserved	RO,0	—

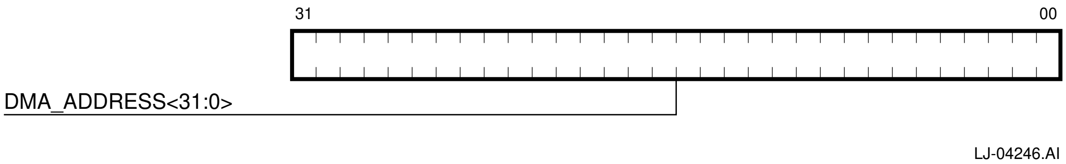
(continued on next page)

Table 4–35 (Cont.) PCI Error Register 0

Field	Name	Type	Description
<19:16>	MASTER_STATE	RO,0	Master state indicated here: 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
<22:20>	TARGET_STATE	RO,0	Target state indicated here: 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
<23>	Reserved	RO,0	—
<27:24>	PCI_CMD	RO, UNDEFINED	The current PCI command.
<28>	PCI_DAC	RO, UNDEFINED	1—The current PCI command is a dual address cycle command.
<31:29>	Reserved	RO,0	—

4.5.10 PCI Error Register 1 (PCI_ERR1)

Figure 4–21 PCI Error Register 1 (87.4000.8840)



The CIA uses PCI_ERR1 to log DMA **ad_h<31:0>** when an error condition is logged into PCI_ERR0. This register is locked whenever the CIA detects an error. This register always captures **ad_h<31:0>**—even for a DMA DAC cycle.

Signal **ad_h<39:32>** can be obtained from W_DAC; **ad_h<63:40>** must be zero for the CIA to hit on the DAC cycle.

PCI_ERR1 is unlocked when the error bits in CIA_ERR have all been cleared. The contents of PCI_ERR1 are UNPREDICTABLE when it is not locked

PCI_ERR1 and some fields in PCI_ERR0 (WINDOW, DMA_DAC, DMA_CMD) hold information related to the following errors, which are associated with the memory while the CIA is handling a DMA transaction:

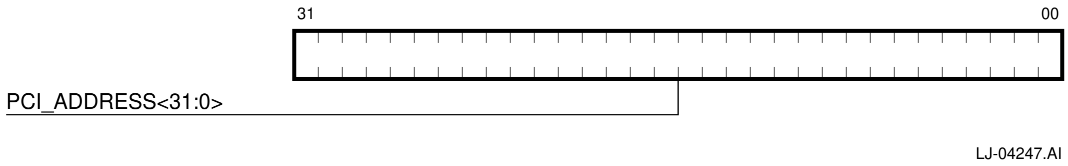
- Correctable ECC error (CIA_ERR<0>)
- Uncorrectable ECC error (CIA_ERR<1>)
- Access to nonexistent memory (CIA_ERR<3>)
- Invalid page table entry (CIA_ERR<9>)

Table 4–36 PCI Error Register 1

Field	Name	Type	Description
<31:0>	DMA_ADDRESS<31:0>	RO, UNDEFINED	Contains DMA **ad_h<31:0>**.

4.5.11 PCI Error Register 2 (PCI_ERR2)

Figure 4–22 PCI Error Register 2 (87.4000.8880)



The CIA uses PCI_ERR2 to log **ad_h<31:0>** when an error condition is logged into PCI_ERR0. This register is locked whenever the CIA detects an error. This register always captures **ad_h<31:0>**—even for a DMA DAC cycle.

Signal **ad_h<39:32>** can be obtained from W_DAC; **ad_h<63:40>** must be zero for the CIA to hit on the DAC cycle.

PCI_ERR2 is unlocked when the error bits in CIA_ERR have all been cleared. The contents of PCI_ERR1 are UNPREDICTABLE when it is not locked.

PCI_ERR2 and some fields in PCI_ERR0 (PCI_DAC, PCI_CMDK, TARGET_STATE, MASTER_STATE) hold information related to the following errors, which are associated with the PCI bus:

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

Table 4–37 PCI Error Register 2

Field	Name	Type	Description
<31:0>	PCI_ADDRESS<31:0>	RO, UNDEFINED	Contains PCI **ad_h<31:0>**.

4.6 DECchip 21171-CA System Configuration Registers

This section defines and describes the DECchip 21171-CA system configuration registers.

4.6.1 Memory Configuration Register (MCR)

MCR defines the system Bcache and memory configuration. The register contents are used to configure the CIA memory controller.

Figure 4–23 Memory Configuration Register (87.5000.0000)

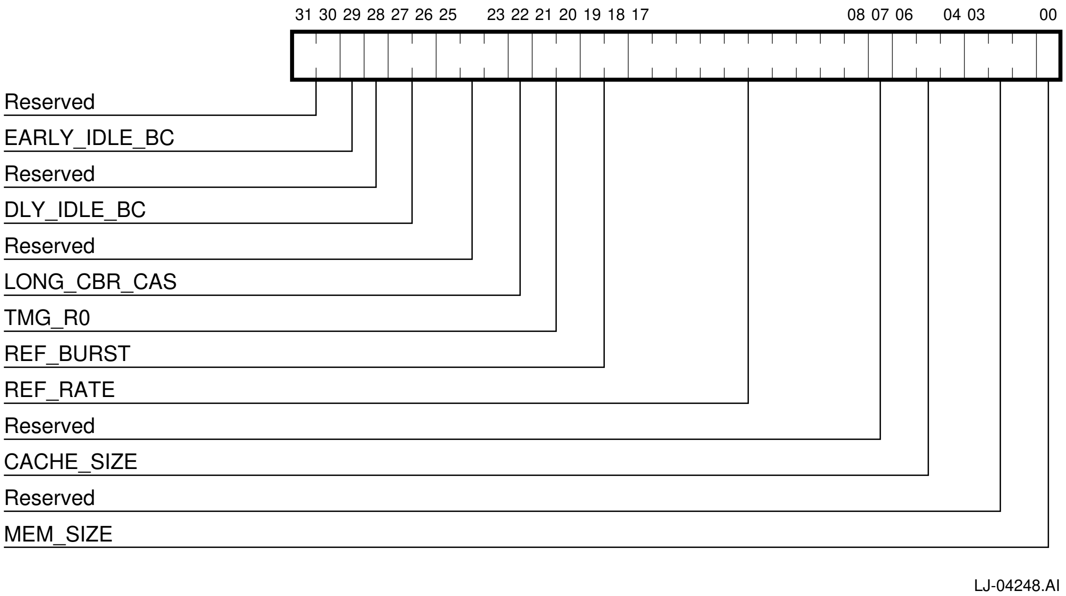


Table 4–38 Memory Configuration Register

Field	Name	Type	Description
<0>	MEM_SIZE	RW,0	This bit must be written by firmware. 0—The memory port is configured to a 128-bit data path. See Figure 3–9. 1—The memory port is configured to a 256-bit data path. See Figure 3–10. MMB0 and MMB1 must be populated identically or an illegal configuration is flagged.
<3:1>	Reserved	RO,0	—
<6:4>	CACHE_SIZE	RW,0	SROM code determines the cache size and writes these bits before any memory cycles are started.
<7>	Reserved	RO,0	—
<17:8>	REF_RATE	RW,0	This field controls the memory refresh rate. A 10-bit, free-running counter, starting at zero, counts up to the REF_RATE and resets to zero. Memory is refreshed every time the REF_RATE value is reached. Setting this field to zero will disable refresh cycles.

CACHE_SIZE <2:0>	Cache RAM Size
0 0 0	No cache present
0 0 1	Reserved
0 1 0	128K * X, 2MB—complete cache
0 1 1	256K * X, 4MB or larger—complete cache
100–111	Reserved

(continued on next page)

Table 4–38 (Cont.) Memory Configuration Register

Field	Name	Type	Description
<19:18>	REF_BURST	RW,0	The refresh state machine can be set up to perform all or half the single inline memory modules (SIMM) at once and also to do one or two refreshes when the REF_RATE counter rolls over.

REF_BURST<1:0>	Refresh Burst Size
0 0	**ras_h** to all SIMMs at once, single refresh
0 1	**ras_h** to all SIMMs at once, double refresh (burst)
1 0	**ras_h** to half of SIMMs at once, single refresh
1 1	**ras_h** to half of SIMMs at once, double refresh (burst)

Field	Name	Type	Description
<21:20>	TMG_R0	RW,0	Controls the row address nominal setup time.

TMG_R0<1:0>	Row Address Setup (nominal)
00	15 ns
01	30 ns
10	45 ns
11	60 ns

Field	Name	Type	Description
<22>	LONG_CBR_ CAS	RW,1	Refresh (**cas_h** before **ras_h**) **cas_h** pulse width. 0—Pulse width = 60 ns. 1—Pulse width = 90 ns.
<25:23>	Reserved	RO,0	—
<27:26>	DLY_IDLE_BC	RW,00	**Write only 00_2 to this field**. Writing any other value may cause UNPREDICTABLE behavior.
<28>	Reserved	RO,0	—
<29>	EARLY_IDLE_ BC	RW,1	**Write only 1_2 to this bit**. Writing a 0_2 may cause UNPREDICTABLE behavior.
<31:30>	Reserved	RO,0	—

4.6.2 Memory Base Address Register *n* (MBA*n*)

The eight memory base address registers correspond to the 16 possible banks of memory that the CIA can support. The bits within the MBA register provide a pattern that is compared with the incoming address to determine which bank is being accessed. The minimum bank size for an MBA register is 16MB.

The addresses for the eight memory base address registers are listed here.

MBA0—87.5000.0600	MBA2—87.5000.0680
MBA4—87.5000.0700	MBA6—87.5000.0780
MBA8—87.5000.0800	MBAA—87.5000.0880
MBAC—87.5000.0900	MBAE—87.5000.0980

Figure 4–24 Memory Base Address Register *n*

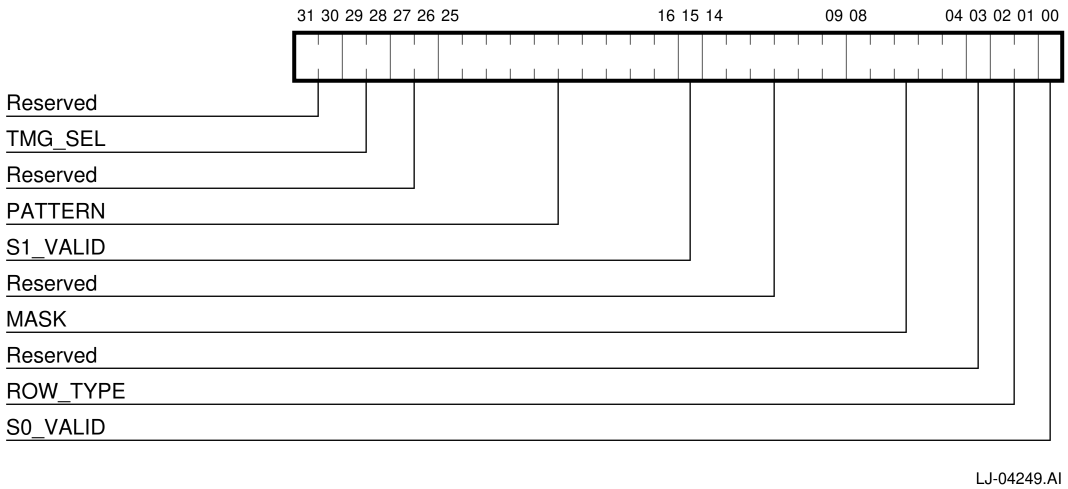


Table 4–39 Memory Base Address Register *n*

Field	Name	Type	Description
<0>	S0_VALID	RW,0	1—Indicates that side 0 for the bank is valid.
<2:1>	ROW_TYPE	RW,0	Specifies the row and column configuration of the physical bank being referenced. ROW_TYPE is used for generating the memory address map. See Figures 3–9 and 3–10.

ROW_TYPE<1:0>	Row/Column
0 0	10 row/10 column
0 1	12 row/10 column or 11 x 11
1 0	13 row/11 column or 12 x 12
1 1	Reserved

Field	Name	Type	Description
<3>	Reserved	RO,0	—
<8:4>	MASK	RW,0	Indicates bits to use when comparing MBA*n*[PATTERN] with the physical address. The MASK field implicitly indicates the size of the memory SIMMs in the bank corresponding to the MBA*n* register. For a 256-bit memory data bus the valid MASK fields are:

SIMM Size	MASK	Comparison Range
4MB or 8MB	00001	**addr<24>** is not used in the address comparison.
16MB or 32MB	00111	**addr<26:24>** are not used in the address comparison.
64MB or 128MB	11111	**addr<28:24>** are not used in the address comparison.

Field	Name	Type	Description
<14:9>	Reserved	RO,0	—
<15>	S1_VALID	RW,0	1—Indicates that side 1 for the bank is valid. Side 1 cannot be valid unless side 0 is also valid.
<25:16>	PATTERN	RW,0	MBA*n*[PATTERN] is compared with the incoming sysBus address **addr_h<33:24>** and then ORed with MASK<8:4> to determine if the bank corresponding to this MBA is being accessed. PATTERN indicates the base address of the memory bank while MASK<8:4> indicates the size of the bank.

(continued on next page)

Table 4–39 (Cont.) Memory Base Address Register *n*

<table>
<tr><th>Field</th><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td><27:26></td><td>Reserved</td><td>RO,0</td><td>—</td></tr>
<tr><td><29:28></td><td>TMG_SEL</td><td>RW,0</td><td>Timing register select:
<table>
<tr><th>TMG_SEL</th><th>TMGn Selected</th><th>Description</th></tr>
<tr><td>0 0</td><td>TMG0</td><td>Used to control timing of 50-ns and 60-ns SIMMs</td></tr>
<tr><td>0 1</td><td>TMG1</td><td>Used to control timing of 70-ns SIMMs</td></tr>
<tr><td>1 0</td><td>TMG2</td><td>Used to control timing of 80-ns SIMMs</td></tr>
<tr><td>1 1</td><td>Reserved</td><td>—</td></tr>
</table></td></tr>
<tr><td><31:30></td><td>Reserved</td><td>RO,0</td><td>—</td></tr>
</table>

4.6.3 Memory Timing Information Register *n* (TMG*n*)

The addresses for the three memory timing information registers are listed here.

TMG0—87.5000.0B00
TMG1—87.5000.0B40
TMG2—87.5000.0B80

The memory timing registers contain the system memory parameters that control operation of the CIA memory sequencer. See MBA*n*<29:28> in Section 4.6.2 for TMG*n* register selection.

Figure 4–25 Memory Timing Information Register *n*

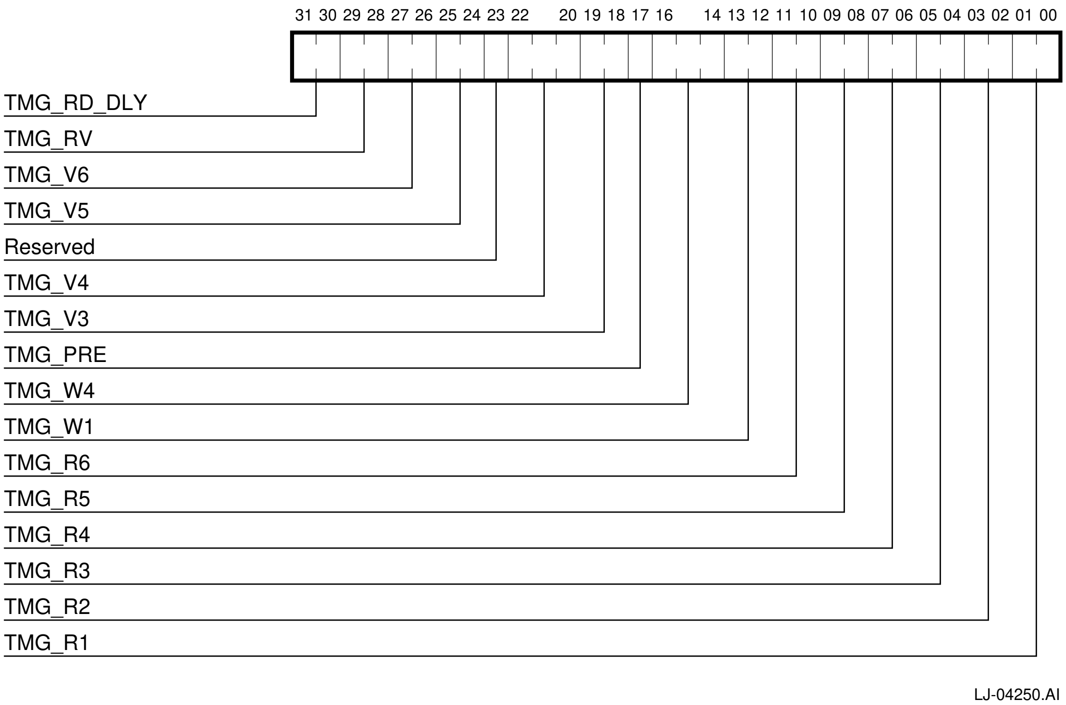


Table 4–40 Memory Timing Information Register *n*

Field	Name	Type	Description
<1:0>	TMG_R1	RW,0	Read starting data delay.
<3:2>	TMG_R2	RW,0	Row address hold.
<5:4>	TMG_R3	RW,0	Read cycle time.
<7:6>	TMG_R4	RW,0	Read **cas_h** assertion delay.
<9:8>	TMG_R5	RW,0	Read **cas_h** pulse width.
<11:10>	TMG_R6	RW,0	Read column address hold.
<13:12>	TMG_W1	RW,0	Write data delay.
<16:14>	TMG_W4	RW,0	Write **cas_h** assertion delay.
<17>	TMG_PRE	RW,0	**ras_h** precharge delay.
<19:18>	TMG_V3	RW,0	Write cycle time.
<22:20>	TMG_V4	RW,0	Linked victim, **cas_h** assertion delay.
<23>	Reserved	RO,0	—
<25:24>	TMG_V5	RW,0	Victim/write, **cas_h** pulse width.
<27:26>	TMG_V6	RW,0	Victim/write, column address hold.
<29:28>	TMG_RV	RW,0	Read-to-victim start delay.
<31:30>	TMG_RD_DLY	RW,0	Read data delay (affects DSW).

Table 4–41 lists the possible values of the encoded memory parameters.

Table 4–41 Memory Timing Parameters, Encoded Values

	Encoded Values							
Parameter	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
TMG_R1	30	—[1]	60	—	—	—	—	—
TMG_R2	30	45	60	75	—	—	—	—
TMG_R3	60	—	90	—	—	—	—	—
TMG_R4	30	45	60	75	—	—	—	—
TMG_R5	30	45	60	75	—	—	—	—
TMG_R6	30	45	60	75	—	—	—	—
TMG_W1	30	—	60	—	—	—	—	—
TMG_W4	60	75	90	105	120	135	—	—
TMG_PRE	0	30	—	—	—	—	—	—
TMG_V3	60	—	90	—	—	—	—	—
TMG_V4	60	75	90	105	120	135	—	—
TMG_V5	30	45	60	75	—	—	—	—
TMG_V6	30	45	60	75	—	—	—	—
TMG_RV[2]	0	0	30	30	—	—	—	—
TMG_RV[3]	15	30	45	60	—	—	—	—
TMG_RD_DLY	0	15	30	45	—	—	—	—

[1] Illegal parameters are indicated by —.

[2] Delay between a read and a linked victim.

[3] Delay after the read to assertion of **mem_en_h**.

Figures 4–26 and 4–27 demonstrate TMG*n* parameters applied to 21171 timing for read and write transactions. In these examples of 21171 timing, the 21171 accepts and uses a 33-MHz clock input. The 21171 doubles that frequency, generating a 66-MHz clock for its internal use.

Figure 4–26 Memory Read Timing Sample

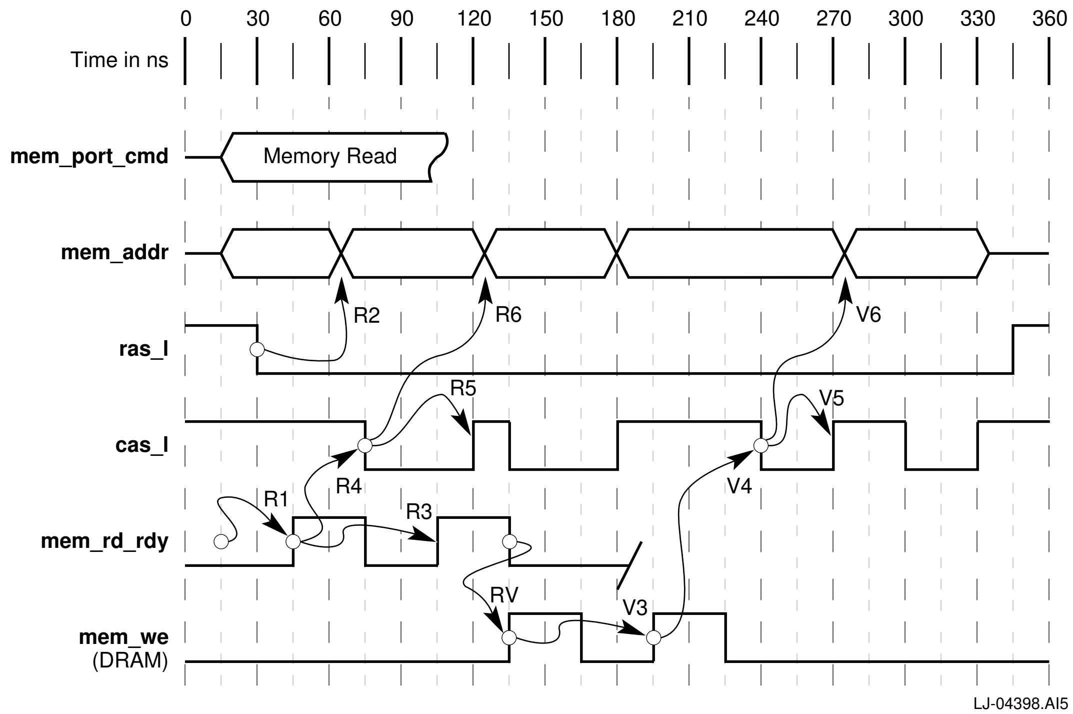
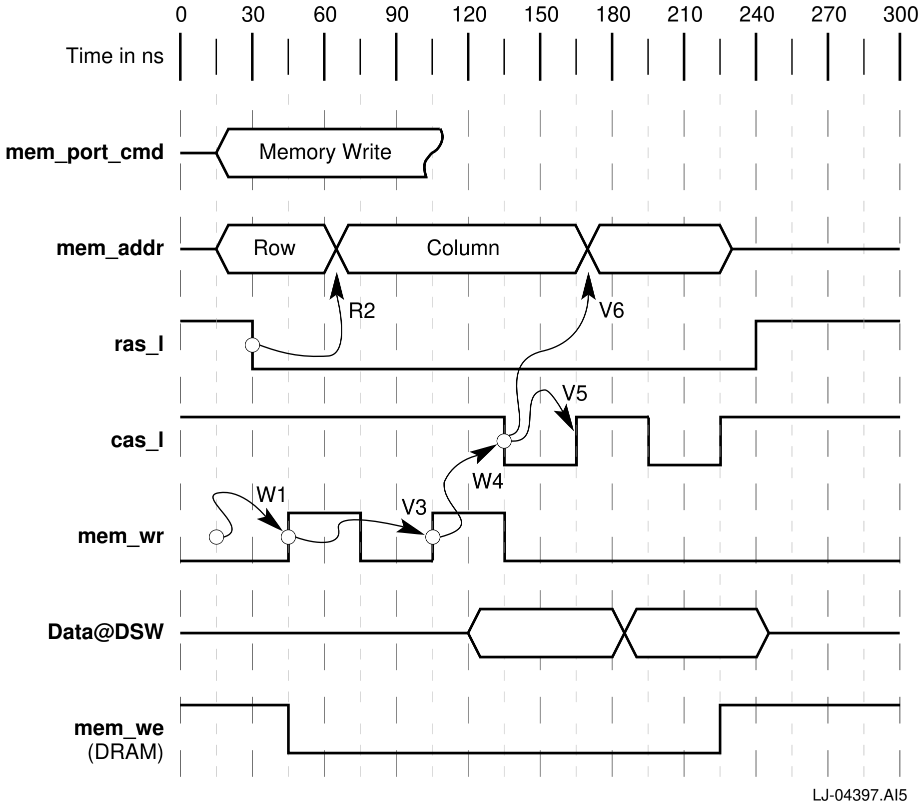


Figure 4–27 Memory Write Timing Sample



Tables 4–42 and 4–43 list the minimum and maximum timing limits for memory signals.

Table 4–42 Memory Control Signals Timing Limits

Signal Name	Minimum	Maximum
	ras_h Timing	
ras_h<0>	3.948 ns	7.030 ns
ras_h<1>	3.948 ns	7.030 ns
ras_h<2>	3.948 ns	6.996 ns
ras_h<3>	3.948 ns	7.030 ns
	cas_h Timing	
cas_h<0>	3.696 ns	6.466 ns
cas_h<1>	3.729 ns	6.466 ns
cas_h<2>	3.718 ns	6.466 ns
cas_h<3>	3.729 ns	6.466 ns
	set_sel<15:0> Timing	
set_sel_l<0>	5.806 ns	13.812 ns
set_sel_l<1>	5.969 ns	14.185 ns
set_sel_l<2>	6.141 ns	14.572 ns
set_sel_l<3>	6.517 ns	14.728 ns
set_sel_l<4>	5.985 ns	14.219 ns
set_sel_l<5>	6.470 ns	14.618 ns
set_sel_l<6>	6.368 ns	14.391 ns
set_sel_l<7>	6.659 ns	15.047 ns
set_sel_l<8>	6.078 ns	14.500 ns
set_sel_l<9>	6.353 ns	14.925 ns
set_sel_l<10>	6.170 ns	14.710 ns
set_sel_l<11>	6.371 ns	14.965 ns
set_sel_l<12>	5.938 ns	14.182 ns
set_sel_l<13>	6.729 ns	15.208 ns
set_sel_l<14>	5.971 ns	14.180 ns
set_sel_l<15>	5.987 ns	14.227 ns

(continued on next page)

Table 4–42 (Cont.) Memory Control Signals Timing Limits

Signal Name	Minimum	Maximum
	we_l Timing	
mem_we_l<0>	3.809 ns	6.646 ns
mem_we_l<1>	3.783 ns	6.576 ns
	mem_en_h Timing	
mem_en_h	3.330 ns	8.206 ns

Table 4–43 Memory Address Timing Limits

Signal Name	Minimum	Column Maximum	Row Maximum[1]
mem_addr_h<0>	3.839 ns	8.850 ns	23.166 ns
mem_addr_h<1>	3.843 ns	8.850 ns	22.946 ns
mem_addr_h<2>	3.842 ns	8.870 ns	22.926 ns
mem_addr_h<3>	3.846 ns	8.840 ns	23.086 ns
mem_addr_h<4>	3.861 ns	8.840 ns	23.106 ns
mem_addr_h<5>	3.843 ns	8.850 ns	23.186 ns
mem_addr_h<6>	3.846 ns	8.860 ns	23.266 ns
mem_addr_h<7>	3.842 ns	8.840 ns	23.186 ns
mem_addr_h<8>	3.846 ns	8.880 ns	23.506 ns
mem_addr_h<9>	4.260 ns	9.635 ns	23.616 ns
mem_addr_h<10>	3.880 ns	8.980 ns	23.356 ns
mem_addr_h<11>	3.883 ns	8.990 ns	23.196 ns
mem_addr_h<12>	4.338 ns	8.790 ns	23.586 ns

[1]Flow-through latch

Table 4–44 lists the data path timing limits for memory and CPU data.

Table 4–44 Memory Data Timing Limits

Signal Name	Setup Time	Hold Time	Clock-to-Out Time
cpu_dat_h<35:0>	7.54 ns	–0.11 ns	9.13 ns
mem_dat_h<71:0>	4.48 ns	0.30 ns	7.49 ns

The clock skew value for flip-flops triggered by the **clk_h** signal line (**iod_h**, **ioc_h**, **cmc_h**, and PCI signals) is 1500 ps. The clock skew value for flip-flops triggered by the **clk2x_h** signal line (**ras_h**, **cas_h**, **mem_we_l<1:0>**, **mem_addr_h<12:0>**, **mem_en_h**) is 2150 ps.

4.7 DECchip 21171-CA PCI Address and Scatter-Gather Registers

This section defines and describes the DECchip 21171-CA system configuration registers.

4.7.1 Scatter-Gather Translation Buffer Invalidate Register (TBIA)

Writing to the TBIA causes the specified group of scatter-gather TLB tags to be unlocked and marked invalid.

Figure 4–28 Scatter-Gather TBIA Register (87.6000.0100)

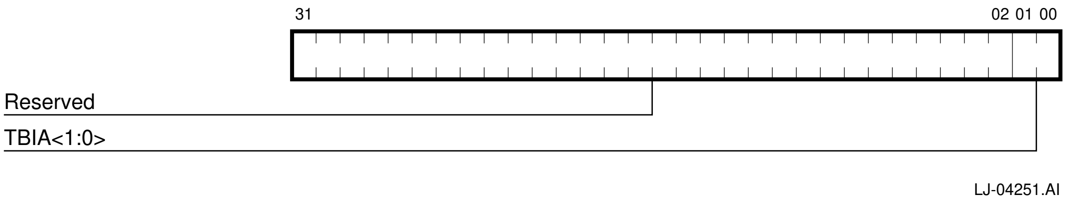


Table 4–45 Scatter-Gather Translation Buffer Invalidate Register

Field	Name	Type	Description	
<1:0>	TBIA<1:0>	WO,0	A write transaction to this register will invalidate the specified scatter-gather translation buffers.	
			TBIA<1:0>	**Operation**
			0 0	No operation.
			0 1	Invalidate and unlock the TLB tags that are currently locked.
			1 0	Invalidate the TLB tags that currently unlocked.
			1 1	Invalidate and unlock all TLB tags.
<31:2>	Reserved	RO,0	—	

4.7.2 Window Base Register (W*n*_BASE)

There are four window base registers, each providing the base address for a particular target window. The window base registers should not be modified unless software ensures that no PCI traffic is targeted for the window being modified.

The incoming PCI address bits <31:20> are compared with each of the four window base registers to determine if a hit occurs in the target window. The window mask register contents determine which bits are involved in the comparison (see Sections 6.3.4.1 and 6.3.4.2).

The target window is hit when the masked addresses match a valid window base register. If MEMCS_ENABLE is set, the hit is further qualified by the MEMCS input signal—this is used if PC compatibility holes are required in the CIA.

When W3_BASE[DAC_ENABLE] is set, W_DAC is used to compare PCI address bits <39:32> during a DAC cycle.

The addresses for the four window base address registers are listed here.

W0_BASE–87.6000.0400	W1_BASE–87.6000.0500
W2_BASE–87.6000.0600	W3_BASE–87.6000.0700

Figure 4–29 Window Base Register *n*

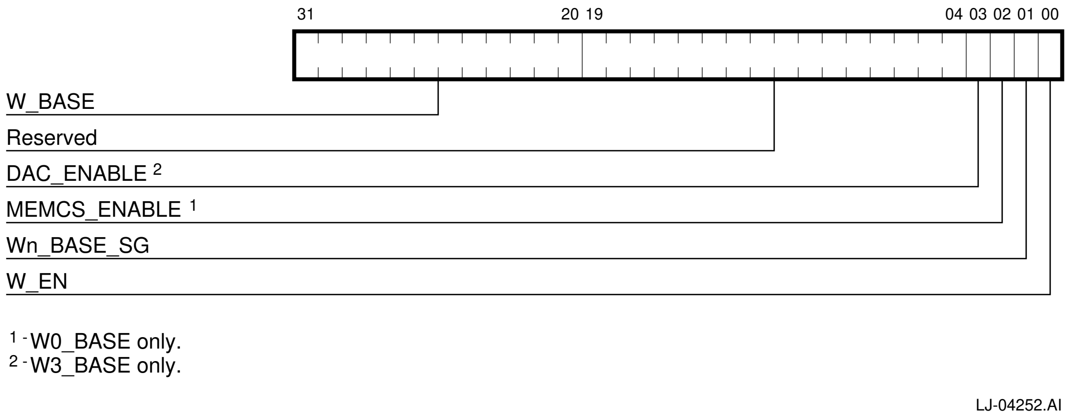


Table 4–46 Window Base Register *n*

Field	Name	Type	Description
<0>	W_EN	RW, UNDEFINED	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 is used to respond to PCI-initiated transfers that hit in the address range of the target window.
<1>	W*n*_BASE_SG	RW, UNDEFINED	0—The PCI target window uses direct mapping to translate a PCI address to a 21164 address. See Table 4–50. 1—The PCI target window uses scatter-gather mapping to translate a PCI address to a physical memory address. See Table 4–49.
<2>	MEMCS_ENABLE[1]	RW, UNDEFINED	1—The MEMCS signal from the PCI-to-EISA bridge is ANDed with the normal window hit.
<3>	DAC_ENABLE[2]	RW, UNDEFINED	1—W_DAC is compared with PCI address bits <39:32> for a PCI DAC cycle. If the compare is a hit, and the 32-bit portion of the PCI address is also hit, then a DAC cycle hit occurs.
<19:4>	Reserved	RO,0	—
<31:20>	W_BASE	RW, UNDEFINED	Specifies the PCI base address of the PCI target window and is used to determine a hit in the target window. See MEMCS_ENABLE and DAC_ENABLE.

[1]Only in W0_BASE.

[2]Only in W3_BASE.

4.7.3 Window Mask Register *n* (W*n*_MASK)

The window mask registers (W*n*_MASK) provide 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 2 by masking bits of an incoming PCI address with the window mask register bits <31:20> as shown in Table 4–47.

There are four window mask registers. The window mask registers should not be modified unless software ensures that no PCI traffic is targeted for the window being modified.

The incoming PCI address bits <31:20> are compared with each of the four window base registers to determine a hit in the target window. The window mask register contents determine which bits are involved in the comparison. The target window is hit when the masked addresses match a valid window base register.

The addresses of the window mask registers are listed here.

W0_MASK–87.6000.0440	W1_MASK–87.6000.0540
W2_MASK–87.6000.0640	W3_MASK–87.6000.0740

Figure 4–30 Window Mask Register *n*

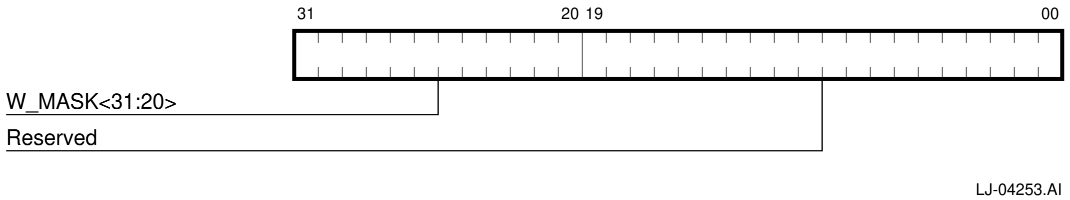


Table 4–47 Window Mask Register *n*

Field	Name	Type	Description
<19:0>	Reserved	RO,0	—
<31:20>	W_MASK	RW, UNDEFINED	This field specifies the size of the PCI target window. It is also used to mask out address bits not used when determining a PCI target window hit.

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

4.7.4 Translated Base Register *n* (T*n*_BASE)

There are four translated base registers (T*n*_BASE), one for each window. They are used to map PCI addresses into memory addresses. The addresses of the four translated base registers are listed here.

T0_BASE–87.6000.0480	T1_BASE–87.6000.0580
T2_BASE–87.6000.0680	T3_BASE–87.6000.0780

If the scatter-gather bit of the window base register is set, the T*n*_BASE provides the base address of the scatter-gather map for this window. If the scatter-gather bit is clear, the T*n*_BASE provides the base physical address of this window.

The T*n*_BASE register should not be modified unless software can ensure that no PCI traffic is targeted for the window being modified.

Figure 4–31 Translated Base Register *n*

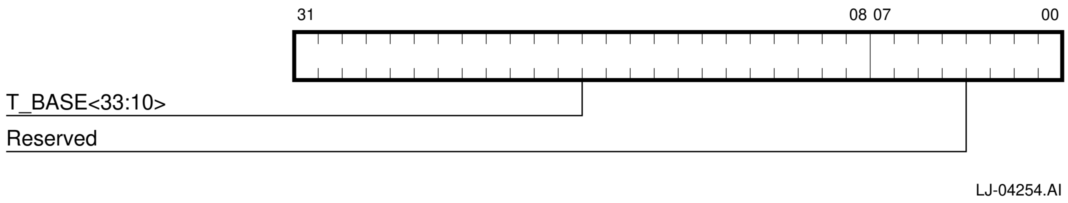


Table 4–48 Translated Base Register *n*

Field	Name	Type	Description
<7:0>	Reserved	RO,0	—
<31:8>	T_BASE<33:10>	RW, UNDEFINED	If scatter-gather mapping is enabled, this field specifies the base 21164 address of the scatter-gather map table for the PCI target window. See Table 4–49. If scatter-gather mapping is disabled, this field specifies the base 21164 address of the translated PCI address for the PCI target window. See Table 4–50.

W_MASK<31:20> sets the size of the PCI target window and the number of 8KB pages that fall into the window. Every 8KB page requires one 8-byte scatter-gather map entry. Table 4–49 shows the relationship of the W_MASK field to the size of the scatter-gather map in memory.

$$\frac{Size\ of\ the\ Window\ in\ Bytes}{8KB} = Number\ of\ Entries\ Required$$

$$Number\ of\ Entries * 8\ bytes = Size\ of\ the\ Scatter - Gather\ Table$$

Table 4–49 PCI Address Translation—Scatter-Gather Mapping Enabled

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

Note

Unused translated base must be zero for correct operation.

The quadword address used to index into the table is formed from concatenating the appropriate T_BASE and PCI address bits based on the size of the scatter-gather map. The PCI address forms the index into the table, while T_BASE forms the NATURALLY ALIGNED base of the table.

For example, there are 128 entries in the scatter-gather table, with a table size of 1KB, for a mask of 0000 0000 0000. Entries are quadwords, so the lower 3 bits, <2:0>, of the address are always zero. Now, mask off **ad_h<31:20>** because of W_MASK. Then use **ad_h<19:13>** (7 bits, $2^7 = 128$ entries in the table) as the table index. Use T_BASE <31:8> to get the other bits of the 34-bit address.

Table 4–50 PCI Address Translation—Scatter-Gather Mapping Disabled

W_MASK<31:20>	Translated Address addr_h<33:0>	Unused Translated Base Register Bits
0000 0000 0000	T_BASE<33:20> : ad_h<19:0>	T_BASE<19:10>
0000 0000 0001	T_BASE<33:21> : ad_h<20:0>	T_BASE<20:10>
0000 0000 0011	T_BASE<33:22> : ad_h<21:0>	T_BASE<21:10>
0000 0000 0111	T_BASE<33:23> : ad_h<22:0>	T_BASE<22:10>
0000 0000 1111	T_BASE<33:24> : ad_h<23:0>	T_BASE<23:10>
0000 0001 1111	T_BASE<33:25> : ad_h<24:0>	T_BASE<24:10>
0000 0011 1111	T_BASE<33:26> : ad_h<25:0>	T_BASE<25:10>
0000 0111 1111	T_BASE<33:27> : ad_h<26:0>	T_BASE<26:10>
0000 1111 1111	T_BASE<33:28> : ad_h<27:0>	T_BASE<27:10>
0001 1111 1111	T_BASE<33:29> : ad_h<28:0>	T_BASE<28:10>
0011 1111 1111	T_BASE<33:30> : ad_h<29:0>	T_BASE<29:10>
0111 1111 1111	T_BASE<33:31> : ad_h<30:0>	T_BASE<30:10>
1111 1111 1111	T_BASE<33:32> : ad_h<31:0>	T_BASE<31:10>

4.7.5 Window Base DAC Register (W_DAC)

W_DAC provides DAC_BASE<7:0> for comparison against PCI address bits <39:32> during a DAC cycle. PCI address bits <63:40> must be zero for a PCI window hit. This register is used with W3_BASE.

W_DAC is only applicable to window 3 and only if enabled by W3_BASE[DAC_ENABLE].

Determining a Hit in the Target Window

The target window is hit when the following is satisfied:

- The incoming PCI address bits <31:20> match one of the four window base registers. W*n*_MASK[W_MASK]<31:20> determines which bits are involved in the comparison.
- PCI address bits <63:40> are zero.
- PCI address bits <39:32> match DAC_BASE<7:0>.

Figure 4–32 Window Base DAC Register (87.6000.07C0)

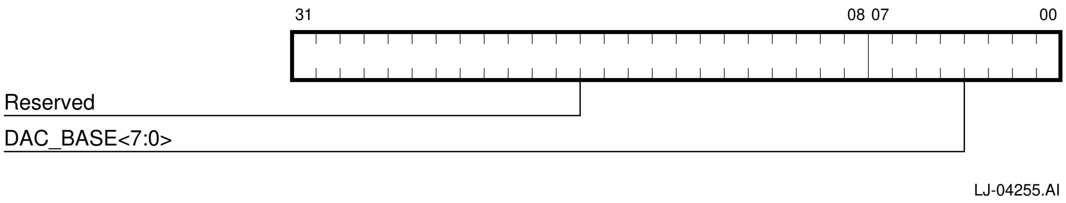


Table 4–51 Window Base DAC Register

Field	Name	Type	Description
<7:0>	DAC_BASE<7:0>	RW, UNDEFINED	Specifies bits <39:32> of the PCI base address used to determine a hit in the target window for a DAC cycle.
<31:8>	Reserved	RO,0	—

4.8 DECchip 21171-CA Address Translation Registers

This section defines and describes the DECchip 21171-CA address translation registers.

4.8.1 Lockable Translation Buffer Tag *n* Registers (LTB_TAG*n*)

Software can write to these four lockable translation buffer tag *n* registers. Hardware cannot evict the register contents when a scatter-gather TLB miss occurs if the LOCKED bit is set. The addresses of the LTB_TAG*n* registers are listed here.

LTB_TAG0–87.6000.0800	LTB_TAG1–87.6000.0840
LTB_TAG2–87.6000.0880	LTB_TAG3–87.6000.08C0

Determining a Hit in the Translation Buffer

If W*n*_BASE[W*n*_BASE_SG] true when a PCI address hits one of the W*n*_ BASE registers, the incoming PCI address bits <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 PCI address bits <14:13> and if it is valid, there is a translation buffer hit.

Operation on an Scatter-Gather TLB Miss

Hardware uses a round-robin algorithm to handle a scatter-gather TLB miss. A scatter-gather TLB entry is overwritten if it is not locked, but will not be evicted if it is locked. The hardware will write all four PTEs on a miss.

Figure 4–33 Lockable Translation Buffer Tag *n* Registers

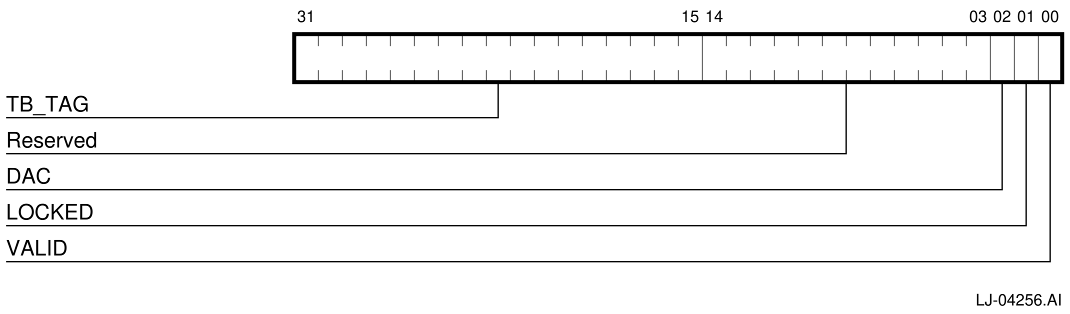


Table 4–52 Lockable Translation Buffer Tag *n* Registers

Field	Name	Type	Description
<0>	VALID	RW,0	If both VALID and the corresponding W*n*_BASE[W*n*_BASE_SG] are set, this entry will be used for address translation.
<1>	LOCKED	RW,0	1—The hardware will not evict this entry.
<2>	DAC	RW,0	0—This TAG entry belongs to a 32-bit PCI address. 1—This tag entry corresponds to a 64-bit PCI address.
<14:3>	Reserved	RO,0	—
<31:15>	TB_TAG	RW, UNDEFINED	The TAG for each translation buffer entry.

4.8.2 Translation Buffer Tag *n* Register (TB_TAG*n*)

The addresses for the four TB_TAGn registers are listed here.

TB_TAG0–87.6000.0900	TB_TAG1–87.6000.0940
TB_TAG2–87.6000.0980	TB_TAG3–87.6000.09C0

Software can write to these TLB tag entries, but the entries cannot be locked and so may be evicted by the hardware on a scatter-gather TLB miss.

The incoming PCI address bits <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 PCI address bits <14:13>, and if it is valid, then there is a translation buffer hit.

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 in the event of a scatter-gather TLB miss.

Figure 4–34 Translation Buffer Tag *n* Registers

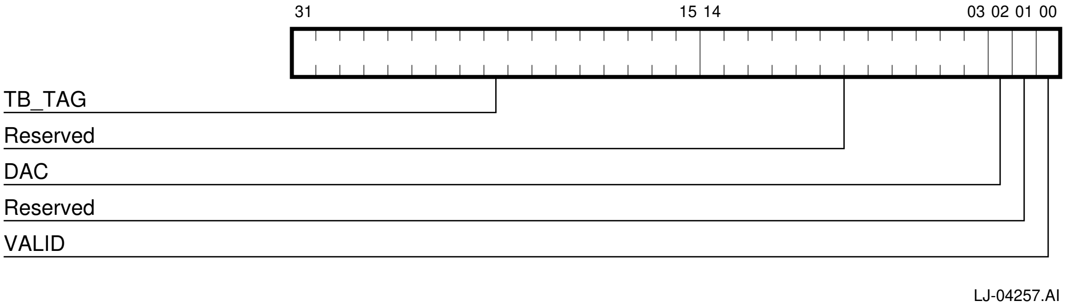


Table 4–53 Translation Buffer Tag *n* Registers

Field	Name	Type	Description
<0>	VALID	RW,0	If both VALID and the corresponding W*n*_BASE[W*n*_BASE_SG] are set, this entry is used for address translation.
<1>	Reserved	RO,0	—
<2>	DAC	RW,0	0—This TAG entry belongs to a 32-bit PCI address. 1—This TAG entry corresponds to a 64-bit PCI address.
<14:3>	Reserved	RO,0	—
<31:15>	TB_TAG	RW, UNDEFINED	The TAG for each translation buffer entry.

4.8.3 Translation Buffer *m* Page *n* Registers (TB*m*_PAGE*n*)

There are 32 translation buffer data 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 CIA hardware.

The addresses for the 32 TB*m*_PAGE*n* registers are listed here.

TB0_PAGE0–87.6000.1000	TB0_PAGE1–87.6000.1040
TB0_PAGE2–87.6000.1080	TB0_PAGE3–87.6000.10C0
TB1_PAGE0–87.6000.1100	TB1_PAGE1–87.6000.1140
TB1_PAGE2–87.6000.1180	TB1_PAGE3–87.6000.11C0
TB2_PAGE0–87.6000.1200	TB2_PAGE1–87.6000.1240
TB2_PAGE2–87.6000.1280	TB2_PAGE3–87.6000.12C0
TB3_PAGE0–87.6000.1300	TB3_PAGE1–87.6000.1340
TB3_PAGE2–87.6000.1380	TB3_PAGE3–87.6000.13C0
TB4_PAGE0–87.6000.1400	TB4_PAGE1–87.6000.1440
TB4_PAGE2–87.6000.1480	TB4_PAGE3–87.6000.14C0
TB5_PAGE0–87.6000.1500	TB5_PAGE1–87.6000.1540
TB5_PAGE2–87.6000.1580	TB5_PAGE3–87.6000.15C0
TB6_PAGE0–87.6000.1600	TB6_PAGE1–87.6000.1640
TB6_PAGE2–87.6000.1680	TB6_PAGE3–87.6000.16C0
TB7_PAGE0–87.6000.1700	TB7_PAGE1–87.6000.1740
TB7_PAGE2–87.6000.1780	TB7_PAGE3–87.6000.17C0

Figure 4–35 Translation Buffer *m* Page *n* Registers

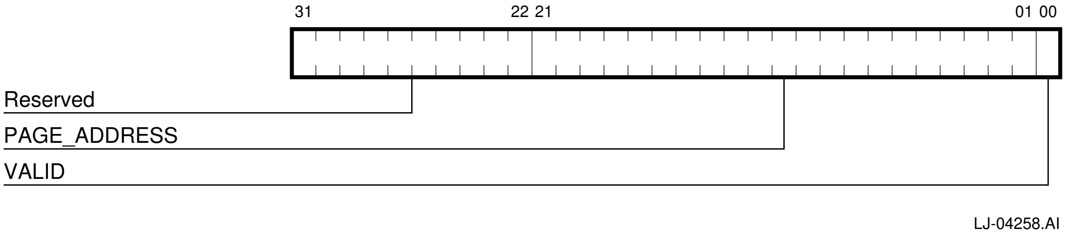


Table 4–54 Translation Buffer *m* Page *n* Registers

Field	Name	Type	Description
<0>	VALID	RW, UNDEFINED	1—This entry can be used for address translation.
<21:1>	PAGE_ADDRESS	RW, UNDEFINED	This field forms physical address<33:13>. PCI **ad_h<12:0>** forms physical address<12:0>.
<31:22>	Reserved	RO,0	—

Incoming PCI address bits <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 using PCI address bits <14:13>, and if it is valid, there is a translation buffer hit.

If the address bits do not match the tag, or the page entry is invalid, a TLB miss occurs. If the PTE, fetched by the hardware TLB-miss handler, is still invalid, the CIA_ERROR interrupt is asserted for a DMA write transaction.

5

DECchip 21171-CA Power-Up and Initialization

This chapter describes the behavior of the DECchip 21171-CA (CIA) chip on power-up and assertion of **reset_l**. It also describes the system level requirements and the various registers that must be initialized after **reset_l** is deasserted.

5.1 Power-Up

On power-up, the **reset_l** input of the CIA should be asserted. It should be kept asserted until the system clocks are up and running for eight cycles.

5.2 Internal Reset

The assertion and deassertion of the **reset_l** pin on the module is asynchronous to the CIA. The CIA generates an internal reset signal from **reset_l**. The internal reset signal is asserted asynchronously, as soon as **reset_l** is asserted, but is deasserted synchronously. Because of the synchronous deassertion of the internal reset, the CIA requires that no external transaction start until eight system clock cycles after the deassertion of **reset_l**.

5.3 State of Pins on Reset Assertion

The following are general rules and requirements for the behavior of the CIA pins during reset:

- All input control signals (except the clocks and **reset_l**) must be in the deasserted state as long as **reset_l** is asserted.
- All output and bidirectional signals are in the states listed in Table 5–1.

Table 5–1 lists the state of CIA pins while **reset_l** is asserted.

Table 5–1 CIA Pin State During Reset

Signal	Reset State	Signal	Reset State
addr_cmd_par_h	Z	**addr_h<39,34:4>**	Z
cmd_h<3:0>	Z	**addr_bus_req_h**	Z
cack_h	Z	**dack_h**	Z
fill_h	Z	**fill_id_h**	Z
fill_error_h	Z	**idle_bc_h**	Z
tag_dirty_h	Z	**tag_ctl_par_h**	Z
mem_addr<12:0>	x	**set_sel_h<15:0>**	0
ras_h<3:0>	0	**cas_h<3:0>**	0
mem_we_l<1:0>	11_2	**mem_en_h**	1
ad_h<63:0>	Z	**cbe_l<7:0>**	Z
par_h	Z	**par64_h**	Z
req64_l	Z	**ack64_l**	Z
frame_l	Z	**irdy_l**	Z
devsel_l	Z	**trdy_l**	Z
stop_l	Z	**perr_l**	Z
serr_l	Z	**req_l**	1
rst_l	0	**cmc_h<8:0>**	0
ioc_h<6:0>	52	**iod_h<63:0>**	Z
iod_e_h<7:0>	Z	**int_h**	0
error_h	0	**test_or_scan_out_h**	0

5.4 Configuration after Reset Deassertion

Table 5–2 lists the state of CIA pins after **reset_l** has been deasserted.

Table 5–2 CIA Pin State After Reset

Signal	Reset State	Signal	Reset State
addr_cmd_par_h	Z	**addr_h<39,34:4>**	Z
cmd_h<3:0>	Z	**addr_bus_req_h**	0
cack_h	0	**dack_h**	0
fill_h	0	**fill_id_h**	0
fill_error_h	0	**idle_bc_h**	0
tag_dirty_h	Z	**tag_ctl_par_h**	Z
mem_addr<12:0>	x	**set_sel_h<15:0>**	0
ras_h<3:0>	0	**cas_h<3:0>**	0
mem_we_l<1:0>	11_2	**mem_en_h**	1
ad_h<63:32>	Z	**cbe_l<7:4>**	Z
ad_h<31:0>	x or Z[1]	**cbe_l<3:0>**	x or Z[1]
par_h	x or Z[1]	**par64_h**	>
req64_l	0	**ack64_l**	Z
frame_l	Z	**irdy_l**	Z
devsel_l	Z	**trdy_l**	Z
stop_l	Z	**perr_l**	Z
serr_l	Z	**req_l**	1
rst_l	0	**cmc_h<8:0>**	0
ioc_h<6:0>	52	**iod_h<63:0>**	Z
iod_e_h<7:0>	Z	**int_h**	0
error_h	0	**test_or_scan_out_h**	0

[1]Dependent upon the state of PCI **gnt_l**.
If **gnt_l** is low, (CIA granted use of the PCI) the state of these signal lines is x.
If **gnt_l** is high, (CIA not granted use of the PCI) the state of these signal lines is Z.

After **reset_h** has been deasserted, the 21164 reads instructions from the serial ROM (SROM) into its internal Icache and then executes those instructions. The instructions from the SROM initialize the registers listed in Table 5–3 to the values shown.

Table 5–3 Registers Initialized by SROM Code

Register	Acronym	Initialized Value (Hex)
Memory configuration register	MCR	Depends upon memory configuration.
Memory base address registers 0,2,4,6,8,A,C,E	MBAN	Depends upon memory configuration.
Memory timing registers 0–2	TMG0, TMG1, TMG2	Depends upon memory configuration.

6

DECchip 21171-CA Address Mapping

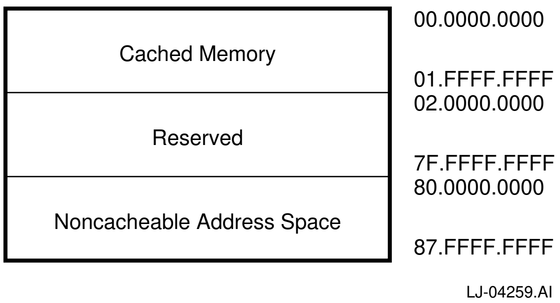
This chapter describes DECchip 21171-CA (CIA) mapping of 40-bit 21164 physical addresses to memory addresses and I/O space addresses. It also describes translation of a 21164-initiated address into a PCI address and translation of a PCI-initiated address into a physical memory address.

Topics include dense and sparse address space[1], PCI addressing, scatter-gather address translation for DMA operations, and Extended Industry Standard Architecture (EISA) requirements.

6.1 Address Mapping Overview

The 21164 address space is divided into two regions, as shown in Figure 6–1, using physical address bit **addr_h<39>**. If clear, the 21164 access is to the cacheable memory space (partly reserved), and if set, the 21164 access is to noncacheable address space.

Figure 6–1 21164 Address Space



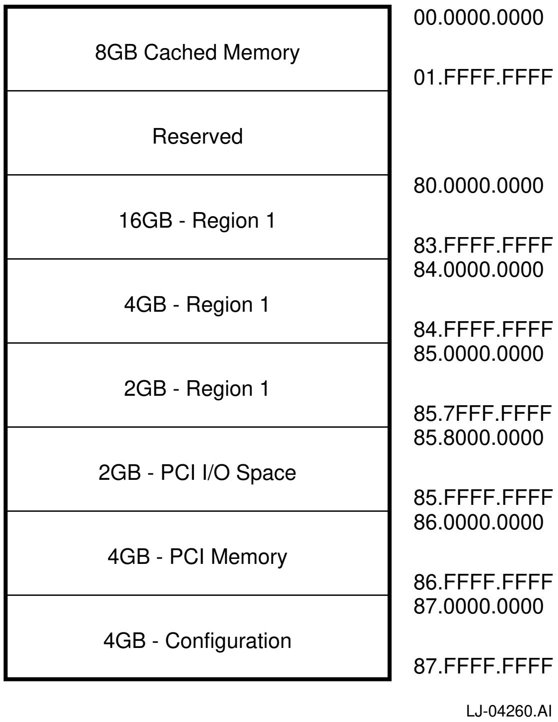
The CIA does not support use of **addr_h<38:35>**. Software must ensure that **addr_h<38:35>** is zero (even parity); otherwise, the CIA will cause a parity error interrupt.

[1] Dense and sparse address spaces are defined in Section 6.2.1 and Section 6.2.2, respectively.

6.1.1 21164 Address Space Configuration Supported by the CIA

As shown in Figure 6–2, the CIA supports only the first 8GB of cacheable memory space; the remainder is reserved. The cacheable memory space block size is fixed at 64 bytes. The CIA will send READ and FLUSH commands to the 21164 caches for DMA traffic to the 8GB memory address area.

Figure 6–2 21164 Address Space Configuration



The CIA supports 21164 access to memory-mapped I/O devices in noncacheable address space. The CIA defines four address spaces within the noncacheable space.

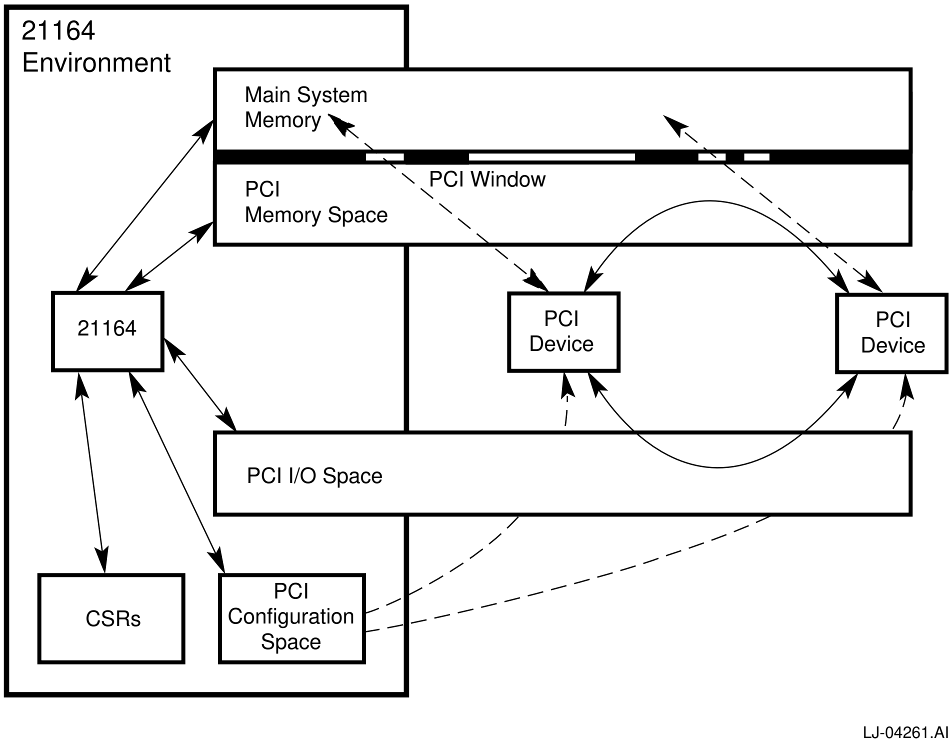
- 22GB PCI memory sparse space
- 2GB PCI I/O space
- 4GB PCI memory dense space
- 4GB that includes address space for:
  - PCI configuration

- Special/interrupt acknowledge cycles
- CIA control and status registers (CSR)
- Flash ROM and support logic registers

6.1.1.1 21164 Access to Address Space

The 21164 has access to the complete address space: it can access cached memory and CSRs, as well as all the PCI memory, I/O, and configuration regions, as shown in Figure 6–3.

Figure 6–3 Address Space Overview



The CIA sends only 32-bit addresses to the PCI (SAC[1]). The CIA does not generate 64-bit addresses (DAC[2]).

---

1 Single address cycle (PCI 32-bit address transfer).

2 Dual access cycle (PCI 64-bit address transfer)–used only if address bits **ad_h<63:32>** are non-zero.

6.1.1.2 PCI Access to Address Space

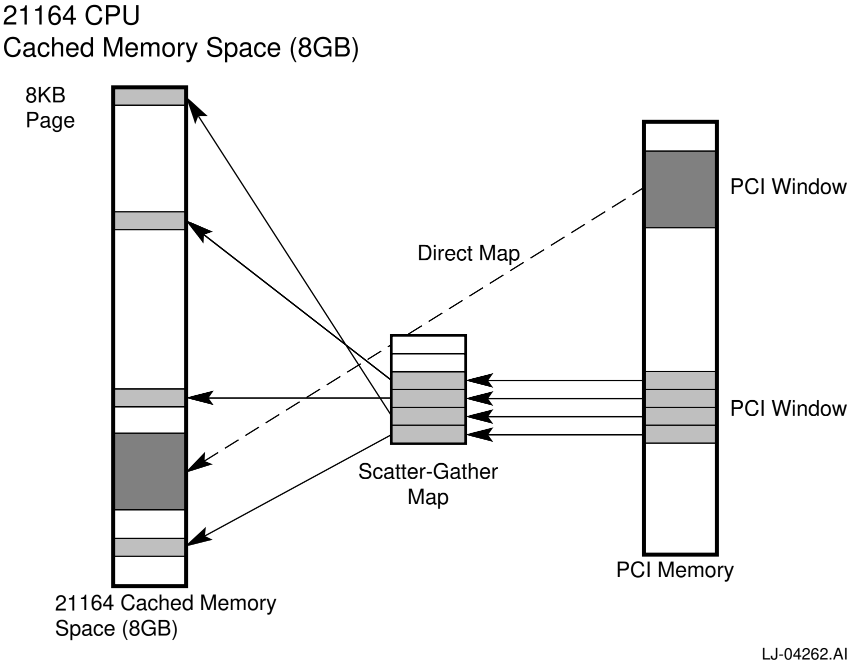
PCI devices have a restricted view of the address space. They can access any PCI device through the PCI memory or PCI I/O space, but they have no access to PCI configuration space.

Furthermore, the CIA restricts PCI access to the system memory (for DMA operations) to four programmable address windows in PCI memory space, as shown in Figure 6–3. Address windows are a PCI requirement. Each window location is defined by its base register and is implemented by all PCI devices.

DMA access to system memory is achieved through one of two access methods:

- The directly mapped method concatenates an offset to a portion of the PCI address.
- The virtually mapped method uses a scatter-gather translation map. The scatter-gather map allows any 8KB page of PCI memory address region to be redirected to an 8KB cached memory page, as shown in Figure 6–4.

Figure 6–4 Address Mapping Overview

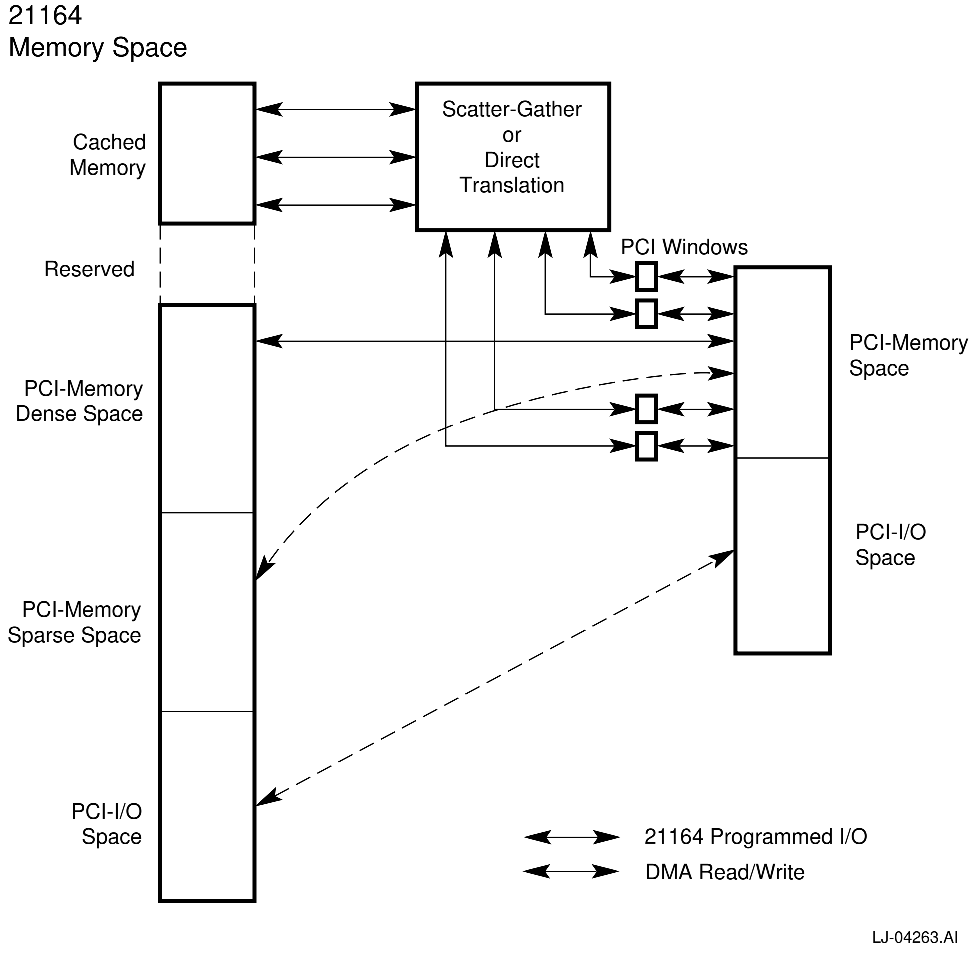


The CIA accepts both 64-bit address cycles (DAC) and 32-bit PCI address cycles (SAC). The following restrictions apply to PCI 64-bit address (DAC) support:

- The least significant 40 bits of the PCI address are used in the window comparison logic. PCI address bits **ad_h<63:40>** must be zero.
- Only one of the four PCI windows (window 3) can be enabled to accept a 64-bit (DAC) PCI address; windows 0 through 2 accept only 32-bit (SAC) address cycles.
- With a 4GB DAC window and one of the 4GB SAC windows, a PCI device can access all 8GB of memory supported by the CIA.

Figure 6–5 shows how the 21164 address map translates to the PCI address space. It also shows how PCI devices access 21164 memory space via DMAs. Notice that the PCI memory space is double mapped using dense and sparse space.

Figure 6–5 21164 and DMA Read and Write Transactions



6.2 21164 Address Space

This section lists and describes the 21164 address space regions. The requirements for using the address regions are also shown and described. Table 6–1 lists the 21164 address regions.

Table 6–1 21164 Address Space

21164 Address	Size (GB)	Selection
00.0000.0000–01.FFFF.FFFF	8	Main memory
80.0000.0000–83.FFFF.FFFF	16	PCI memory—512MB Sparse Space—Region 0
84.0000.0000–84.FFFF.FFFF	4	PCI memory—128MB Sparse Space—Region 1
85.0000.0000–85.7FFF.FFFF	2	PCI memory—64MB Sparse Space—Region 2
85.8000.0000–85.BFFF.FFFF	1	PCI I/O space—32MB Sparse Space—Region A
85.C000.0000–85.FFFF.FFFF	1	PCI I/O space—32MB Sparse Space—Region B
86.0000.0000–86.FFFF.FFFF	4	PCI memory, 4GB—Dense space
87.0000.0000–87.1FFF.FFFF	0.5	PCI configuration Sparse space
87.2000.0000–87.3FFF.FFFF	0.5	PCI special/interrupt acknowledge Sparse space
87.4000.0000–87.4FFF.FFFF	0.25	CIA main CSRs Pseudosparse[1]
87.5000.0000–87.5FFF.FFFF	0.25	CIA PCI memory control CSRs Pseudosparse[1]
87.6000.0000–87.6FFF.FFFF	0.25	CIA PCI address translation Pseudosparse[1]
87.7000.0000–87.FFFF.FFFF	2.25	Reserved

[1]Pseudosparse space is a hardware-specific restricted version of sparse space.

The reasons for using the 21164 I/O space address map are as follows:

- Limits the number of address pins sent to the CIA.
- Provides 4GB of dense space to completely map the 32-bit PCI memory space.
- Provides abundant PCI sparce memory space because sparse memory space has byte granularity and is the safest memory space to use (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 is wasteful of 21164 address space. For instance, 16GB of 21164 address space maps to only 512MB of PCI sparse space.

  The CIA supports three PCI sparse space memory regions, allowing 704MB of total sparse memory space. The three regions can be relocated using the HAE_MEM CSR, and 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 developers may find this region sufficient for their needs and can ignore the remaining two regions.
  - 128MB region, 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, some chips can access only 64KB. So most, if not all, PCI devices will not exceed 64KB for the foreseeable future. Therefore, the CIA supports 64MB of sparse I/O space.
- Supports two CIA PCI sparse space I/O regions. Region A contains 32MB and is fixed in PCI segment 0–32 MB. Region B also contains 32MB, but can be relocated using the HAE_IO register.

6.2.1 PCI Dense Memory Space

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

- Contiguous memory locations—Some software, like the default graphics routines of Windows NT, require memory-like access. These routines cannot use sparse space addresses, because sparse space addresses require PCI transactions to be at adjacent Alpha addresses, instead of being widely separated as in sparse space. As as result, if the user-mode driver uses sparse space for its frame-buffer manipulation, it cannot ‘‘hand over’’ the buffer to the common Windows NT graphics code.
- Higher bus bandwidth—PCI bus burst transfers are not usable in sparse space except for a two-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 accesses use separate read or write buffer entries. Dense space allows separate accesses to be collapsed in read and write buffers (this is exactly what the 21164 does).
- Few memory barriers–In general, sparse space accesses are separated by memory barriers to avoid read/write buffer collapsing. Dense space accesses only require barriers when explicit ordering is required by the software.

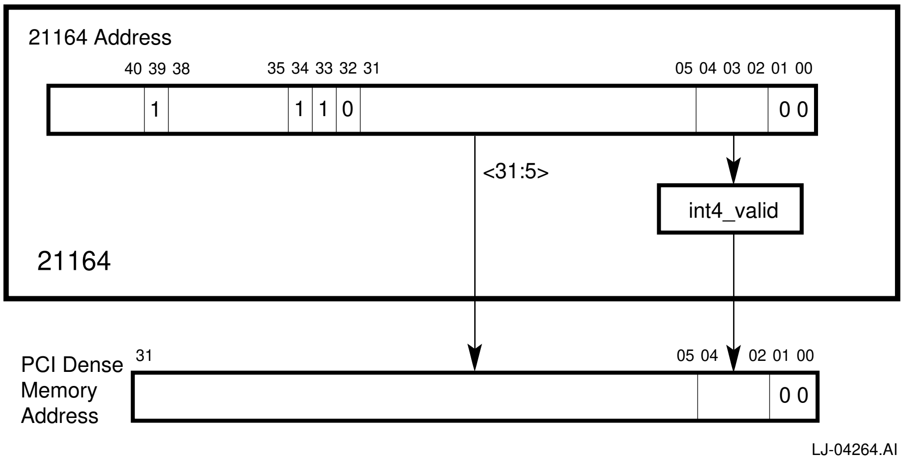
Dense space is provided for the 21164 to access PCI memory space, but not for accessing PCI I/O space. Dense space has the following characteristics:

- There is 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.
- The concept of dense space (and sparse space) is applicable only to 21164-generated addresses. There is no such thing as dense space (or sparse space) for PCI-generated address.
- Byte or word accesses are *not* possible in this 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 8 longwords on the PCI). Any combination of longwords may be valid on write transactions. Valid longwords surrounding invalid longwords (called a ‘‘hole’’) are required to be handled correctly by all PCI devices. The CIA 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 processor can request a longword but the CIA 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 thus coherency is not an issue. Note that a quadword read 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 allow another device to take control of the PCI bus. The CIA does not drive the PCI LOCK signal and thus the PCI cannot ensure atomicity. This is true of all current Alpha systems using the PCI.
- The 21164 merges noncached read transactions up to a 32-byte maximum. The largest dense space read transaction is thus 32 bytes from the PCI bus.
- Write transactions to this space are buffered in the 21164. The CIA supports a burst length of 8 on the PCI, corresponding to 32 bytes of data. In addition, the CIA provides four 32-byte write buffers to maximize I/O write performance. These four buffers are strictly ordered.

Address generation in dense space is shown in Figure 6–6.

Figure 6–6 Dense Space Address Generation



Address generation in dense space is described in the following list:

- **addr_h<31:5>** is directly sent out on the PCI as **ad_h<31:5>**.
- **addr_h<4:2>** is not sent from the 21164, but is inferred from **int4_valid_h<3:0>**.
- **ad_h<4:3>** is a copy of **addr_h<4:3>**.
- **ad_h<2>** differs for read and write transactions as follows:
  - For a read transaction, **ad_h<2>** is zero (minimum read resolution in noncached space is a quadword).
  - For a write transaction, **ad_h<2>** equals **addr_h<2>**.

6.2.2 PCI Sparse Memory Space

The CIA supports 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. Sparse address space maps a large piece of 21164 memory address space to a small PCI address space. For example, a 32-byte memory address might map to a single-byte PCI address.

A problem arises because 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, and quadword references. The CIA must also be able to emulate PCI transactions for PCI devices designed for systems that are capable of generating the UNALIGNED references.

The CIA accomplishes UNALIGNED PCI references by encoding the size of the data transfer (byte, word, and so on) and the byte enable information in the 21164 address. Address bits **addr_h<6:3>** are used for this purpose.

The PCI longword address **ad_h<26:3>** is generated using remaining address bits **addr_h<31:7>**.

Quadword address encoding is provided by **addr_h<6:3>** with **addr_h<7>** assumed to be zero by the CIA hardware. See Table 6–3.

The loss of address bits **addr_h<6:3>** has resulted in a 22GB ‘‘sparse’’ address space that maps to only 704MB of address space on the PCI.

The rules for accessing sparse space follow:

- Sparse space supports all the byte encodings that are expected to be generated in a system to ensure compatibility with existing and expected PCI devices/drivers. The results of some references are not explicitly defined (these are the missing entries in Table 6–3, such as word size with address **addr_h<6:5>** = 11). The hardware will complete the reference, but the reference is not required to produce any particular result nor will the CIA report an error.
- Software must use longword load or store instructions (LDL/STL) to perform a reference that is 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.
  - The hardware will do no byte shifting within the longword.
- Quadword load and store instructions must be used only to perform a quadword transfer. Use of STQ/LDQ instructions for any other references will produce UNPREDICTABLE results.
- Read-ahead (prefetch) is not performed in sparse space by the CIA hardware because the read-ahead might have unwanted side effects.
- Programmers are required to insert memory barrier (MB) instructions between sparse space accesses to prevent collapsing in the 21164 write buffer. However, this is not always necessary. For instance, 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/coherencey to a dense space address are to be maintained.
- On read transactions, the 21164 sends out **addr_h<4:3>** indirectly on the **int4_valid_h<3:0>**.
- Accesses with **addr_h<2:0>** non-zero will produce UNPREDICTABLE results.

- The relationship between **int4_valid_h<3:0>** and 21164 **addr_h<4:3>** for a sparse space write transaction is shown in Table 6–2. The important point is that all other **int4_valid_h<3:0>** patterns will produce UNPREDICTABLE results such as the result of collapsing in the 21164 write buffer or issuing an STQ instruction when an STL instruction was required.

Table 6–2 int4_valid_h<3:0> and addr_h<4:3> for Sparse Space Write Transactions

21164 Data Cycle	int4_valid_h<3:0> [1]	addr_h<4:3>
First	0 0 0 1	0 0
	0 0 1 0	0 0
	0 1 0 0	0 1
	1 0 0 0	0 1
Second	0 0 0 1	1 0
	0 0 1 0	1 0
	0 1 0 0	1 1
	1 0 0 0	1 1
	1 1 0 0 (STQ)[2]	1 1

[1] All other **int4_valid_h<3:0>** patterns cause UNPREDICTABLE results.

[2] Only one STQ case is allowed.

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

Table 6–3 PCI Memory Sparse Space Read/Write Encodings

Size	Byte Offset	21164	PCI		Data-In Register
addr_h<4:3>	addr_h<6:5>[1]	Instruction	ad_h<2:0>[2]	Byte[3] Enable	Byte Lanes [7:0]
Byte					
00	00	LDL, STL	addr_h<7>,00	1110	<0>
	01			1101	<1>
	10			1011	<2>
	11			0111	<3>
Word					
01	00	LDL, STL	addr_h<7>,00	1100	<1:0>
	01			1001	<2:1>
	10			0011	<3:2>
Tribyte					
10	00	LDL, STL	addr_h<7>,00	1000	<2:0>
	01			0001	<3:1>
Longword					
11	00	LDL, STL	addr_h<7>,00	0000	<3:0>
Quadword					
11	11	LDQ, STQ	000	0000	<7:0>

[1]Missing entries (such as word size with **addr_h<6:5>** = 11_2) cause UNPREDICTABLE results.

[2]In PCI sparse memory space, **ad_h<1:0>** is always equal to zero.

[3]Byte enable set to zero indicates that byte lane carries meaningful data.

The high-order PCI address bits **ad_h<31:26>** are obtained from either the hardware address extension register (HAE_MEM) or the 21164 address, depending on sparse space regions, as shown in Table 6–4.

Table 6–4 HAE_MEM High Order Sparse Space Bits

Region[1]	PCI Address					
	<31>	<30>	<29>	<28>	<27>	<26>
1	HAE_MEM <31>	HAE_MEM <30>	HAE_MEM <29>	21164 <33>	21164 <32>	21164 <31>
2	HAE_MEM <15>	HAE_MEM <14>	HAE_MEM <13>	HAE_MEM <12>	HAE_MEM <11>	21164 <31>
3	HAE_MEM <7>	HAE_MEM <6>	HAE_MEM <5>	HAE_MEM <4>	HAE_MEM <3>	HAE_MEM <2>

[1]Region 1 is 80.0000.0000 to 83.FFFF.FFFF.
Region 2 is 84.0000.0000 to 84.FFFF.FFFF.
Region 3 is 85.0000.0000 to 85.7FFF.FFFF.

HAE_MEM is located in the CIA and is described in Section 4.2.4. Figures 6–7 through 6–9 show mapping for the three regions.

Figure 6–7 PCI Memory Sparse Space Address Generation (Region 1)

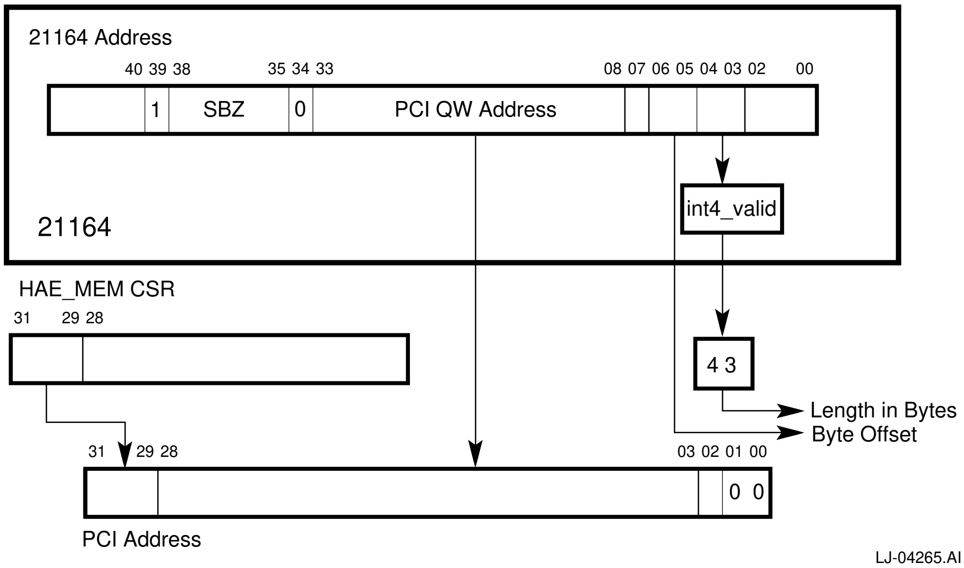


Figure 6–8 PCI Memory Sparse Space Address Generation (Region 2)

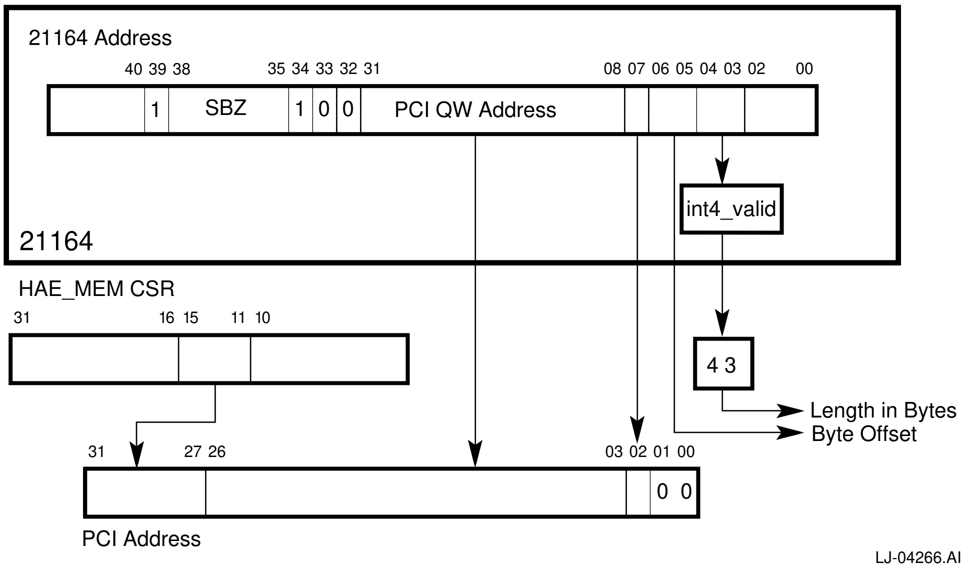
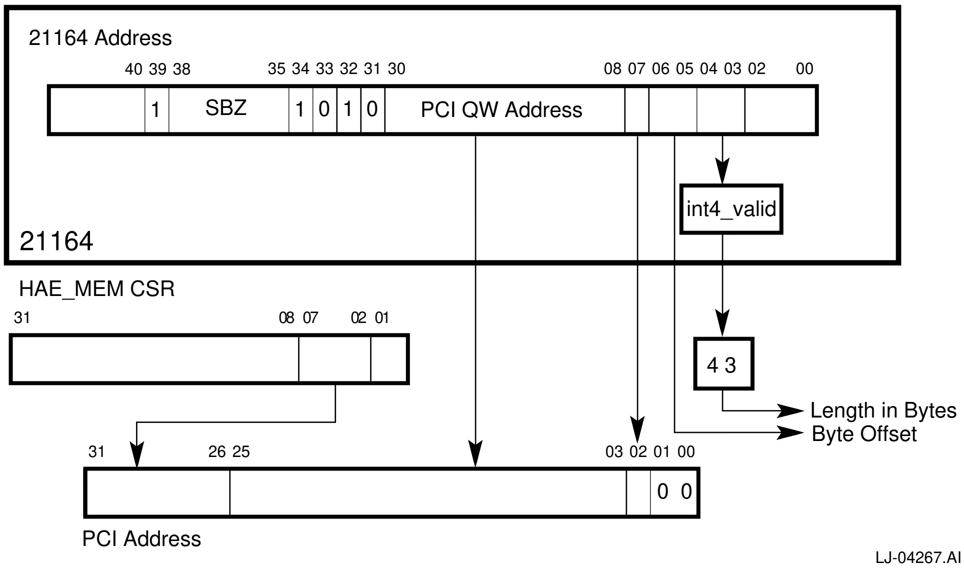


Figure 6–9 PCI Memory Sparse Space Address Generation (Region 3)



The 21164 provides six physical address bits <39:34> that can be used to backfill the ''lost'' sparse space bits. However, other 21164 platforms use these high-order bits in different ways, encoding multiple PCI ports for instance, and so for easier software portability, these bits are not used.

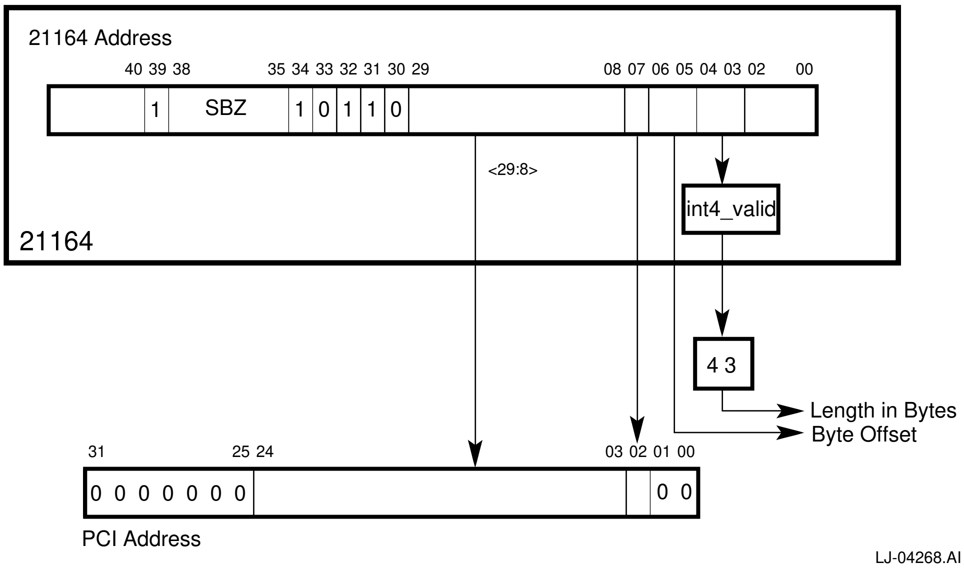
6.2.3 PCI Sparse I/O Space

PCI sparse I/O space has characteristics similar to the PCI sparse memory space. PCI sparse I/O space is located in the address range 85.8000.0000 to 85.FFFF.FFFF. 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 transaction.

The high-order PCI address bits for region A are handled as follows:

- This region has **addr_h<34:30>** equal to 10110_2 and addresses the lower 32MB of PCI sparse I/O space; thus, **ad_h<31:25>** are set to zero by the hardware. See Figure 6–10.
- **ad_h<24:3>** are derived from **addr_h<29:8>**.
- This region is used for (E)ISA addressing (the EISA 64KB I/O space cannot be relocated).
- **ad_h<2:0>** are defined in Table 6–5.

Figure 6–10 PCI Sparse I/O Space Address Translation (Region A)



The high-order PCI address bits for region B are handled as follows:

- This region has **addr_h<34:30>** equal to 10111_2 and addresses 32MB of PCI sparse I/O space that can be relocated.
- This 32MB segment is relocated by assigning **ad_h<31:25>** equal to HAE_IO<31:25>l as shown in Figure 6–11.

  HAE_IO is defined in Section 4.2.5. The power-on self-test (POST), running POST11 software, should initialize the contents of this register and the register should then remain unchanged.
- **ad_h<24:3>** are derived from **addr_h<29:8>**.
- **ad_h<2:0>** are defined in Table 6–5.

Figure 6–11 PCI Sparse I/O Space Address Translation (Region B)

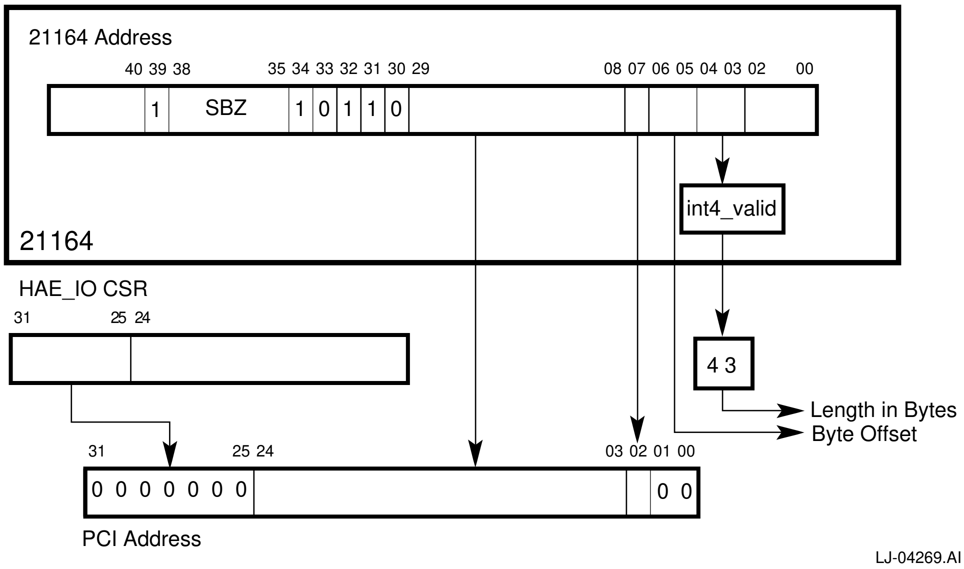


Table 6–5 PCI I/O Sparse Space Read/Write Encodings

Size	Byte Offset	21164	PCI		Data-In Register
addr_h<4:3>	addr_h<6:5>[1]	Instruction	ad_h<2:0>[1]	Byte[2] Enable	Byte Lanes [7:0]
Byte					
00	00	LDL, STL	addr_h<7>,00	1110	<0>
	01		addr_h<7>,01	1101	<1>
	10		addr_h<7>,10	1011	<2>
	11		addr_h<7>,11	0111	<3>
Word					
01	00	LDL, STL	addr_h<7>,00	1100	<1:0>
	01		addr_h<7>,01	1001	<2:1>
	10		addr_h<7>,10	0011	<3:2>
Tribyte					
10	00	LDL, STL	addr_h<7>,00	1000	<2:0>
	01		addr_h<7>,01	0001	<3:1>
Longword					
11	00	LDL, STL	addr_h<7>,00	0000	<3:0>
Quadword					
11	11	LDQ, STQ	000	0000	<7:0>

[1]Missing entries (such as word size with **addr_h<6:5>** = 11_2) cause UNPREDICTABLE results.

[2]Byte enable set to zero indicates that byte lane carries meaningful data.

The (E)ISA devices have reserved the lower 64KB of PCI I/O space (85.8000.0000 to 85.801F.FFFF). Hence, all PCI devices should be relocated above this region. Table 6–6 shows the PCI and (E)ISA I/O map for a 21164 system. Four (E)ISA slots are hardwired, allocating 4KB per slot (as per EISA standard). The first slot is reserved for the EISA system board (X-bus addressing, interrupt controller, and so forth).

Table 6–6 CIA PCI and (E)ISA I/O Map

21164 Address	PCI Address	Range (KB)	Selection
85.8000.0000	0000.0000	0–4	EISA system board
85.8002.0000	0000.1000	4–8	EISA slot 1
85.8004.0000	0000.2000	8–12	EISA slot 2
85.8006.0000	0000.3000	12–16	EISA slot 3
85.8008.0000	0000.4000	16–20	EISA slot 4
85.800A.0000	0000.5000	20–24	Reserved
85.800C.0000	0000.6000	24–28	Reserved
85.800E.0000	0000.7000	28–32	Reserved
85.8010.0000	0000.8000	32–36	Reserved
85.8012.0000– 85.801F.FFFF	0000.9000– 0000.FFFF	36–64	Reserved
85.8020.0000– 85.BFFF.FFFF	0001.0000– 01FF.FFFF	64KB– 32MB	PCI I/O area—fixed
85.C000.0000– 85.FFFF.FFFF	0200.0000– 03FF.FFFF	32MB	PCI I/O area—can be relocated

Note

A quadword access to the PCI sparse I/O space will result in a two-longword burst on the PCI. However, PCI devices might not support bursting in I/O space.

6.2.4 PCI Configuration Space

PCI configuration space is located in the address range: 87.0000.0000 to 87.1FFF.FFFF. Software designers are advised to clear CIA_CTRL [FILL_ERR_EN] when probing for PCI devices using configuration space read transactions. This will prevent the CIA from generating an ECC error if no device responds to the configuration cycle and UNPREDICTABLE data is read from the PCI bus.

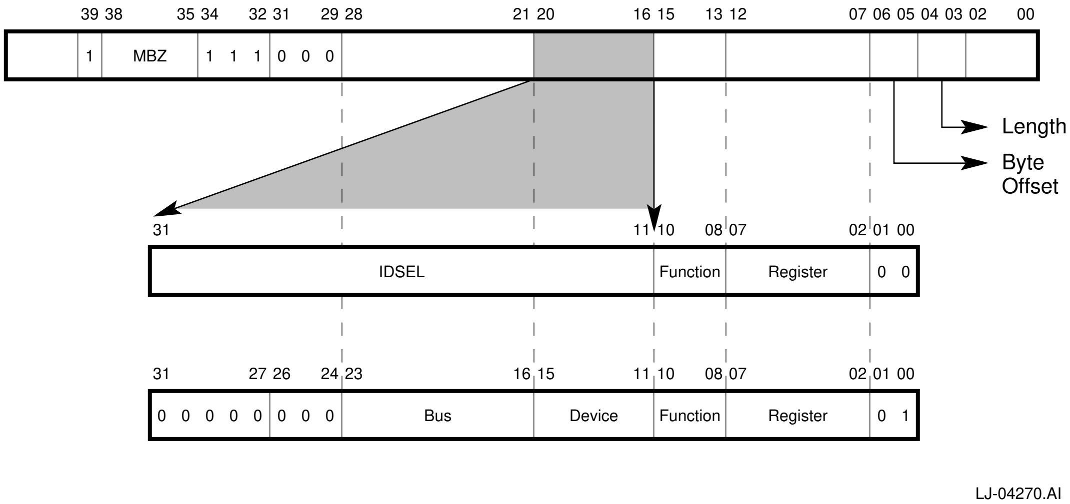
A 21164 read or write access to this address space causes a configuration read or write cycle on the PCI. The two types of targets selected, depending upon the value of the configuration register (CFG), are listed here and are shown in Figure 6–12.

- Type 0—These are targets on the primary 64-bit 21164 system PCI bus. These are selected by making CFG<1:0> equal to 00_2.
- Type 1—These are targets on the secondary 32-bit 21164 system PCI bus (that is, behind a PCI–PCI bridge). These are selected by making CFG<1:0> equal to 01_2.

Note

CFG<1:0> equal to 10_2 and 11_2 are reserved by the PCI specification.

Figure 6–12 PCI Configuration Space Definition



Software must program CFG before running a configuration cycle. Sparse address decoding is used.

Note

The CIA uses CFG<1:0> instead of unused **addr_h<38:35>** to be compatible with the DECchip 21071 core logic chipset. The DECchip 21071 core logic chipset is used with the Alpha 21064 microprocessor.

- The high-order PCI address bits **ad_h<31:24>** are always zero.
- Address bits **addr_h<28:7>** correspond to PCI **ad_h<23:2>** and provide the configuration command information (which device to select).
- Address bits **addr_h<6:3>** are used to generate both the length of the PCI transaction in bytes and the byte enables, as shown in Table 6–7.
- Address bits **ad_h<1:0>** are obtained from CFG<1:0>.

Table 6–7 PCI Configuration Space Read/Write Encodings

Size	Byte Offset	21164	PCI		Data-In Register
addr_h<4:3>	addr_h<6:5>[1]	Instruction	ad_h<1:0>[1]	Byte[2] Enable	Byte Lanes [7:0]
Byte					
00	00	LDL, STL	CFG<1:0>	1110	<0>
	01		CFG<1:0>	1101	<1>
	10		CFG<1:0>	1011	<2>
	11		CFG<1:0>	0111	<3>
Word					
01	00	LDL, STL	CFG<1:0>	1100	<1:0>
	01		CFG<1:0>	1001	<2:1>
	10		CFG<1:0>	0011	<3:2>
Tribyte					
10	00	LDL, STL	CFG<1:0>	1000	<2:0>
	01		CFG<1:0>	0001	<3:1>
Longword					
11	00	LDL, STL	CFG<1:0>	0000	<3:0>
Quadword					
11	11	LDQ, STQ	CFG<1:0>	0000	<7:0>

[1]Missing entries (such as word size with **addr_h<6:5>** = 11_2) cause UNPREDICTABLE results.

[2]Byte enable set to zero indicates that byte lane carries meaningful data.

6.2.4.1 Device Select (IDSEL)

Peripherals are selected during a PCI configuration cycle if these three statements are true:

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

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

Peripherals that integrate multiple functional units (like SCSI and Ethernet) can provide configuration space for each function. Address bits **ad_h<10:8>** can be decoded by the peripheral to select one of eight functional units. Address bits **ad_h<31:11>** are available to generate the IDSELs. (Note that IDSELs behind a PCI–PCI bridge are determined from the device field encoding of a type 1 access.)

The IDSEL pin of each device corresponds to a unique PCI address bit from **ad_h<31:11>**. The binary value of **addr_h<20:16>** is used to select an address that is asserted on **ad_h<31:11>**, as listed in Table 6–8.

Table 6–8 Generating IDSEL Pin Signals

addr_h<20:16>	ad_h<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[1]
.....	....
11111	0000 0000 0000 0000 0000 0[1]

[1]No device selected.

Note

If a quadword access is specified for the configuration cycle, then the least significant bit (LSB) of the register number field, **ad_h<2>**, must be zero. A quadword read/write transaction must access quadword-aligned registers.

If the PCI cycle is a configuration read or write cycle but **ad_h<1:0>** equals 01_2 (like a type 1 transfer), then a device on a hierarchical bus is being selected using a PCI–PCI bridge. This cycle is accepted by the PCI–PCI bridge for propagation to its secondary PCI bus. During this cycle, **ad_h<23:16>** selects a unique bus number; **ad_h<15:8>** selects a device on that bus (typically decoded by the PCI–PCI bridge to generate the secondary PCI address pattern for IDSEL); and **ad_h<7:2>** selects a longword in the device's configuration space.

Each PCI–PCI bridge can be configured by PCI configuration cycles on its primary PCI interface. Configuration parameters in the PCI–PCI bridge will identify the bus number for its secondary PCI interface, and a range of bus numbers that can exist hierarchically behind it.

If the bus number of the configuration cycle matches the bus number of the PCI–PCI bridge secondary PCI interface, it will accept the configuration cycle, decode it, and generate a PCI configuration cycle with address bits <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 PCI–PCI bridge passes the PCI configuration cycle on unmodified (address bits <1:0> = 01). It will be accepted by a bridge further downstream.

Figure 6–13 shows the PCI hierarchy for systems using the CIA. The IDSEL lines are significant in type 0 configuration cycles, and the PCI nodes are connected as tabulated in Table 6–9.

Figure 6–13 21164 System PCI Bus Hierarchy Example

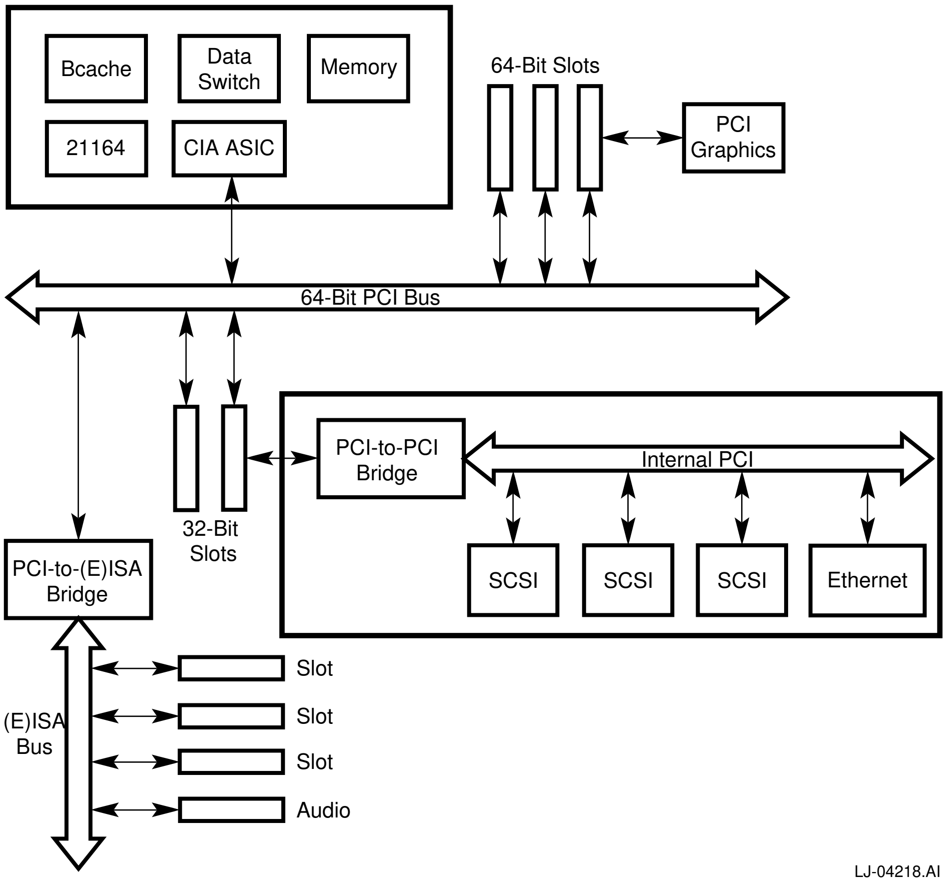


Table 6–9 Primary 64-Bit PCI IDSEL Mapping

Device Slot Assignment	IDSEL Bit	Module Reference	PCI Slot Number (Firmware)
64-bit PCI slot 0[1]	ad_h<18>	J11	7
64-bit PCI slot 1	ad_h<23>	J10	12
64-bit PCI slot 2	ad_h<22>	J9	11
32-bit PCI slot 3	ad_h<19>	J8	8
32-bit PCI slot 4[2]	ad_h<20>	J7	9
PCI-to-EISA bridge	ad_h<21>	—	10

[1]Next to memory board

[2]Next to EISA slots

6.2.4.2 PCI Special/Interrupt Acknowledge Cycles

PCI special/interrupt acknowledge cycle addresses 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 special cycle contains no explicit destination address, but is broadcast to all devices. The CIA will drive all zeros as the special cycle address. Each receiving device must determine if the message contained in the data field is applicable to it.

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

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

A read to address range 87.2000.0000 to 87.3FFF.FFFF will result in an interrupt acknowledge cycle on the PCI returning the vector data, which is provided by the PCI-to-EISA bridge, to the 21164.

6.2.4.3 Hardware-Specific and Miscellaneous Register Space

Hardware-specific and miscellaneous register space is located in the range 87.4000.0000 to 87.FFFF.FFFF. Table 6–10 lists the regions, with hardware registers, within this space.

Table 6–10 Hardware-Specific Register Address Space

addr_h<39:28>	Selected Region	addr_h<27:6>	addr_ h<5:0>
87.4[1]	CIA control, diagnostic, error registers	LW address	000000
87.5[1]	CIA memory control registers	LW address	000000
87.6[1]	CIA PCI adddress translation (S/G, windows, and so on)	LW address	000000
87.7 to 87.F	Reserved	—	—

[1]This address space is a hardware-specific variant of sparse space encoding. For the CSRs, **addr_h<27:6>** specify a longword address where **addr_h<5:0>** must be zero. All CIA registers are accessed with LW granularity. For more specific details on the CIA CSRs, refer to Chapter 4.

6.3 PCI to Physical Memory Addressing

This section describes direct and scatter-gather mapping through the use of windows.

6.3.1 Address Mapping Windows

PCI addresses coming into the CIA (32-bit or 64-bit) are mapped to the 21164 cached memory space (8GB). The CIA provides four programmable address windows that control access of PCI peripherals to system memory. Each window location is defined by its base register (W*n*_BASE) and its size is defined by its mask register (W*n*_MASK).

Mapping from the PCI address to a physical memory address can be direct (physical mapping with an address offset) or scatter-gather (virtual mapping).

The four PCI address windows are also referred to as the PCI target windows. Each window has three registers associated with it. They are:

- PCI base register (W*n*_BASE)
- PCI mask register (W*n*_MASK)
- Translation base register (T*n*_BASE)

In addition, there is an another register that is associated with window 3 only. It is the PCI window DAC base register (W_DAC). It is used for PCI 64-bit addresses (DAC).

W*n*_MASK provides a mask corresponding to bits <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 W*n*_MASK as shown in Table 6–11.

Table 6–11 PCI Target Window MASK Register (W*n*_MASK)

W_MASK<31:20>	Size of Window	Value of *n*[1]
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
All others	UNPREDICTABLE	—

[1]Only incoming **ad_h<31:*n*>** are compared with PCI base register <31:*n*> as shown in Figure 6–15. If *n*=32, no comparison is performed. Windows are not allowed to overlap.

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 W*n*_BASE. If one of the four W*n*_BASE addresses and the incoming PCI address match, then the PCI address has hit that PCI target window. Otherwise, it has missed the window.

A window enable bit, W*n*_BASE [W_EN], lets windows 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 CIA will respond to the PCI cycle by asserting **devsel_l**. The PCI target windows must be programmed so that their address ranges do not overlap. If they overlap, the compare results are UNDEFINED.

The window base address must be on a NATURALLY ALIGNED address boundary corresponding to the size of the window. For example, a 4MB window cannot start at address 1MB; it must start at address 4MB, or 8MB, or 12MB, and so on.

6.3.1.1 PCI Device Address Space

A PCI device specifies the amount of memory space it requires by using base registers in its configuration space. The registers are implemented such 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 of the address range it consumes, that is, the size of the PCI address window defined by the base address. Also, a PCI device need not 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 specify the address space that the PCI bridge will respond to in transactions.

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

These base and limit address registers are initialized by POST code at power-up. The CIA, as a PCI host–bridge device, does not have base and limit registers. Host bridges, because they are under system control, do not have to operate within the rules for other PCI devices. The CIA does respond to all the addresses within the four windows.

6.3.1.2 Address Mapping Example

Figure 6–14 shows how the DMA address ranges of a number of PCI devices are within the PCI window ranges. PCI devices are allowed to have multiple DMA address ranges, like device 2.

Figure 6–14 PCI DMA Address Example

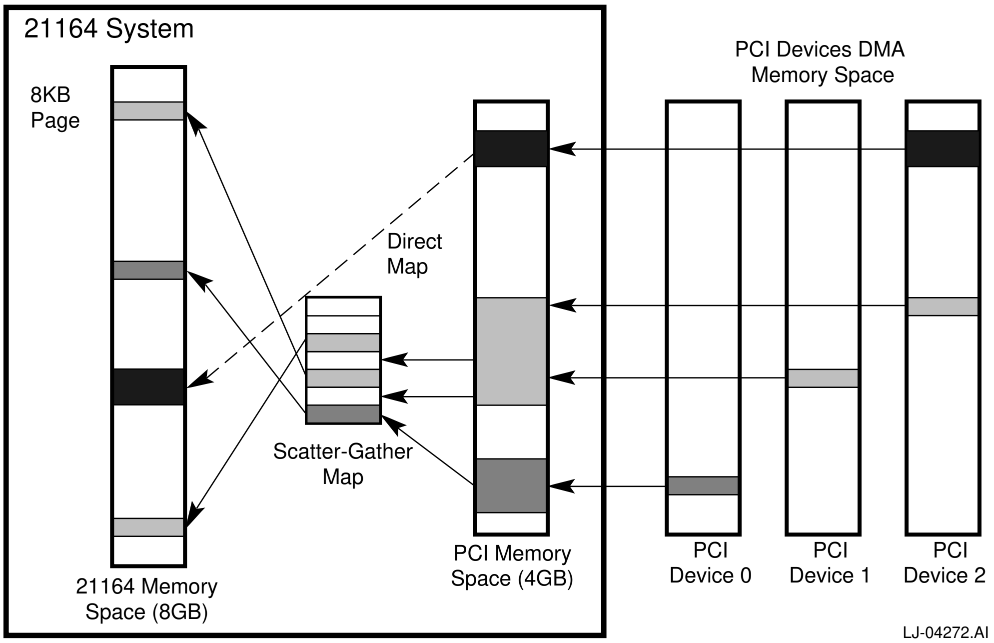


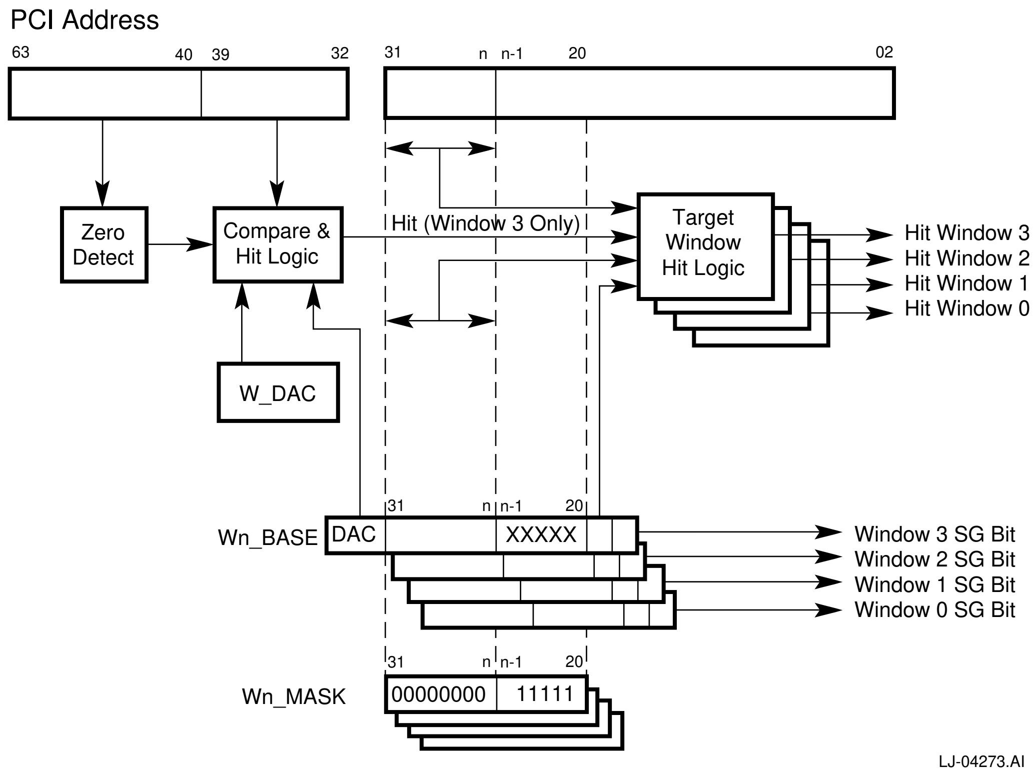
Figure 6–14 also shows that the CIA window can be larger than the corresponding device's DMA address range, as with device 0. Devices 1 and 2 have address ranges that are accepted by one CIA window. W*n*_BASE [W*n*_BASE_SG] for each window determines whether direct mapping or scatter-gather mapping is used to access physical memory.

PCI single and dual address cycles have the following characteristics:

- Dual address cycle (DAC)—issued only if <63:32> are non-zero in 64-bit PCI address mode, and only if enabled by W3_BASE[DAC_ENABLE].
- Single address cycle (SAC)—all 32-bit addresses. A PCI device must use SAC if bits <63:32> of a 64-bit address equal zero.

Figure 6–15 shows the PCI window comparison logic.

Figure 6–15 PCI Target Window Compare



The comparison logic associated with **ad_h<63:32>** is used only for DAC with window 3. The other windows recognize only 32-bit PCI addresses (SAC).

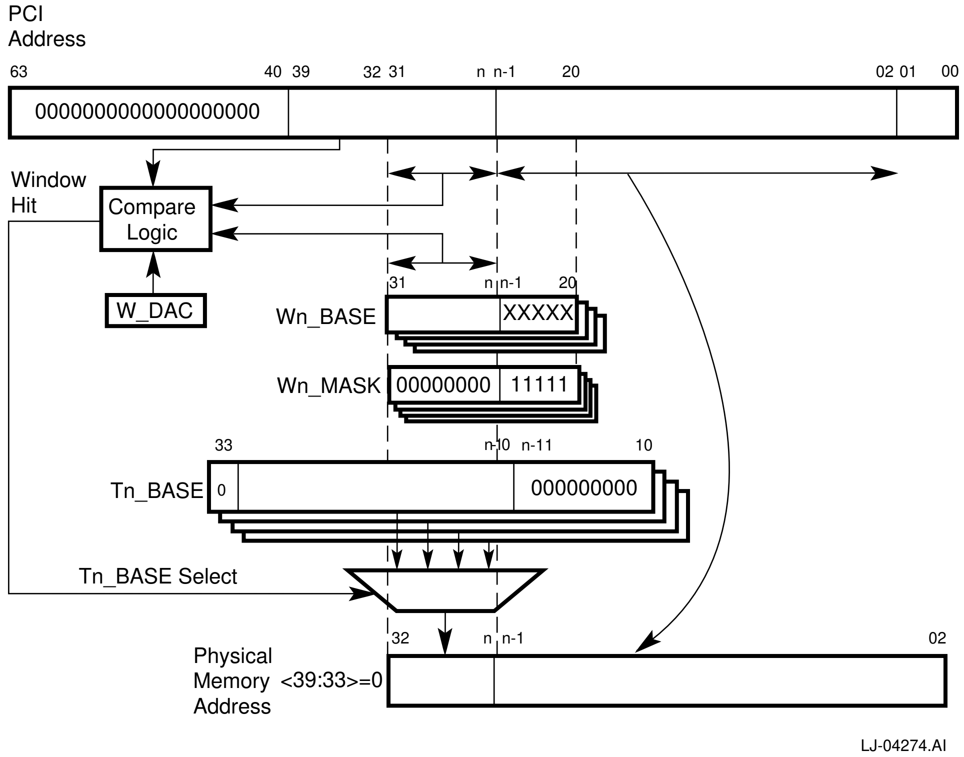
For a hit to occur on a DAC address, address bits <63:40> must be zero and **ad_h<39:32>** must match the W_DAC[DAC_BASE<7:0>]. The low-order address bits **ad_h<31:20>** must also hit. This scheme allows a NATURALLY ALIGNED 1MB to 4GB PCI window to be placed anywhere in the first 1TB of a 64-bit PCI address space.

When an address match occurs with a PCI target window, the CIA translates the 32-bit PCI address **ad_h<31:0>** to **memdata_h<33:0>**. The translated address is generated in one of two ways as determined by W*n*_BASE[W*n*_BASE_SG].

6.3.2 Direct Mapped Addressing

If W*n*_BASE [W*n*_BASE_SG] is clear, the DMA address is direct mapped. The translated address is generated by concatenating bits from the matching window's T*n*_BASE with bits from the incoming PCI address (**ad_h<31:0>**). This process is shown in Figure 6–16 with *n* being the LSB from the T*n*_BASE column of Table 6–12.

Figure 6–16 Direct Mapped Translation



The bits involved in the concatenation are defined by the window's W*n*_MASK, as shown in Table 6–12. Because memory is located in the lower 8GB of the 21164 address space, the CIA implicitly ensures that **addr_h<39:33>** are always zero. Because T*n*_BASE is simply concatenated to the PCI address, 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.

Table 6–12 Direct-Mapped PCI Target Address Translation

W_MASK <31:20>	Size of Window	Translated Address Source: T*n*_BASE[1]	Translated Address Source: PCI Address
0000 0000 0000	1MB	addr_h<32:20>	addr_h<19:2>
0000 0000 0001	2MB	addr_h<32:21>	addr_h<20:2>
0000 0000 0011	4MB	addr_h<32:22>	addr_h<21:2>
0000 0000 0111	8MB	addr_h<32:23>	addr_h<22:2>
0000 0000 1111	16MB	addr_h<32:24>	addr_h<23:2>
0000 0001 1111	32MB	addr_h<32:25>	addr_h<24:2>
0000 0011 1111	64MB	addr_h<32:26>	addr_h<25:2>
0000 0111 1111	128MB	addr_h<32:27>	addr_h<26:2>
0000 1111 1111	256MB	addr_h<32:28>	addr_h<27:2>
0001 1111 1111	512MB	addr_h<32:29>	addr_h<28:2>
0011 1111 1111	1GB	addr_h<32:30>	addr_h<29:2>
0111 1111 1111	2GB	addr_h<32:31>	addr_h<30:2>
1111 1111 1111	4GB	addr_h<32>	addr_h<31:2>

[1] Unused bits of T*n*_BASE must be cleared because the hardware performs an OR operation for the concatenation.

6.3.3 Scatter-Gather Addressing

When W*n*_BASE[W*n*_BASE_SG] is set, the translated address is generated using a table lookup. The table is referred to as a scatter-gather map.

The incoming PCI address is compared to the PCI window addresses for a hit. The T*n*_BASE of the window that was hit is used to specify the starting address of the scatter-gather map table in memory.

Part of the incoming PCI address is used as an offset from this starting address to access the scatter-gather page table entry (PTE). This PTE, together with the remaining least-significant PCI address bits, forms the 21164 memory address.

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

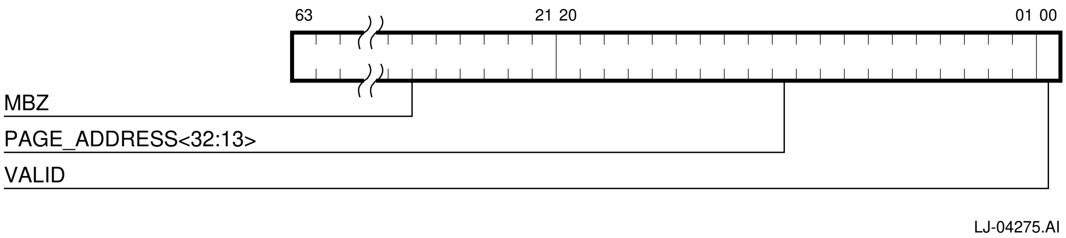
- Performance—ISA devices map to the lower 16MB of memory. The Windows NT operating system currently copies data from here to user space. The scatter-gather map avoids this copy operation.

- Address management—User I/O buffers might not be physically contiguous or contained within a page. Without scatter-gather maps, software has to manage the scattered nature of the user buffer.

In peripheral components architecture, the term scatter-gather is not an address translation scheme but instead is used to signify a DMA transfer list. An element of 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 the PCI-to-EISA bridge, support this form of scatter-gather process.

Each scatter-gather PTE is a quadword and has a valid bit in PTE<0> as shown in Figure 6–17. Page address bit <13> is at PTE<1>.

Figure 6–17 Scatter-Gather PTE Format



Because the CIA implements only valid memory addresses up to 8GB, scatter-gather PTE bits <63:21> must be set to zero. PTE bits <20:1> are used to generate the physical page address. This address is appended to **ad_h<12:5>** of the incoming PCI address to generate the memory address.

The size of the scatter-gather map table is determined by the size of the PCI target window defined by W*n*_MASK, as shown in Table 6–13. The number of entries is 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 T*n*_BASE and the PCI address, as shown in Table 6–13.

Table 6–13 Scatter-Gather Mapped PCI Target Address Translation

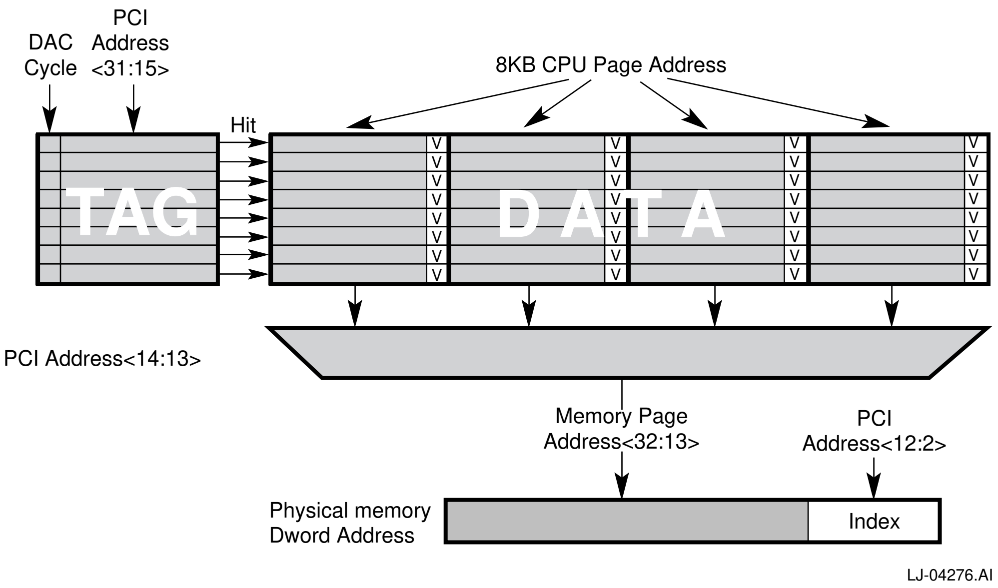
W_MASK	Size of	S-G Map	Scatter-Gather Map Address <33:3>	
<31:20>	Window	Table Size	T*n*_Base[1]	PCI Address
0000 0000 0000	1MB	1KB	<32:10 >	<19:13>
0000 0000 0001	2MB	2KB	<32:11>	<20:13>
0000 0000 0011	4MB	4KB	<32:12>	<21:13>
0000 0000 0111	8MB	8KB	<32:13>	<22:13>
0000 0000 1111	16MB	16KB	<32:14>	<23:13>
0000 0001 1111	32MB	32KB	<32:15>	<24:13>
0000 0011 1111	64MB	64KB	<32:16>	<25:13>
0000 0111 1111	128MB	128KB	<32:17>	<26:13>
0000 1111 1111	256MB	256KB	<32:18>	<27:13>
0001 1111 1111	512MB	512KB	<32:19>	<28:13>
0011 1111 1111	1GB	1MB	<32:20>	<29:13>
0111 1111 1111	2GB	2MB	<32:21>	<30:13>
1111 1111 1111	4GB	4MB	<32:22>	<31:13>

[1]Unused bits of T*n*_BASE must be zero for correct operation.

6.3.3.1 Scatter-Gather Translation Lookaside Buffer (TLB)

An 8-entry translation lookaside buffer (TLB) is provided in the CIA for scatter-gather PTEs. The TLB is a fully associative cache and holds the eight most recent scatter-gather map lookups. Four of these entries can be ''locked,'' preventing their displacement by the hardware TLB-miss handler. Each of the eight TLB entries holds a PCI address for the tag, and four consecutive 8KB page addresses as the TLB data, as shown in Figure 6–18.

Figure 6–18 Scatter-Gather Associative TLB



Each time an incoming PCI address hits in a PCI target window that has scatter-gather enabled, **ad_h<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 in the matching TLB entry. Address bits **ad_h<14:13>** select the correct 8KB page address from the four addresses fetched.

With a TLB hit, the scatter-gather map table lookup in memory is avoided, resulting in enhanced 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 (invalidates) to the TBIA. See Section 4.7.1.

The TAG portion of the TLB entry contains a DAC flag to indicate that PCI tag address bits <31:15> correspond to a 64-bit DAC address. Only one bit is required instead of the high-order PCI address bits **ad_h<39:32>** because only window 3 is assigned to a DAC cycle, and the window hit logic has already performed a comparison of the high-order address bits against W_DAC.

Figure 6–19 shows the entire translation process, from PCI address to physical address, on a window that implements scatter-gather. The MSB from the PCI address column of Table 6–13 equals *n*–1. Both paths are indicated: the path for a TLB hit is to the right, and the path for a TLB miss is to the left. The scatter-gather TLB is shown in a slightly simplified but functionally equivalent form.

6.3.3.1.1 Scatter-Gather TLB Hit Process The process for a scatter-gather TLB hit is as follows:

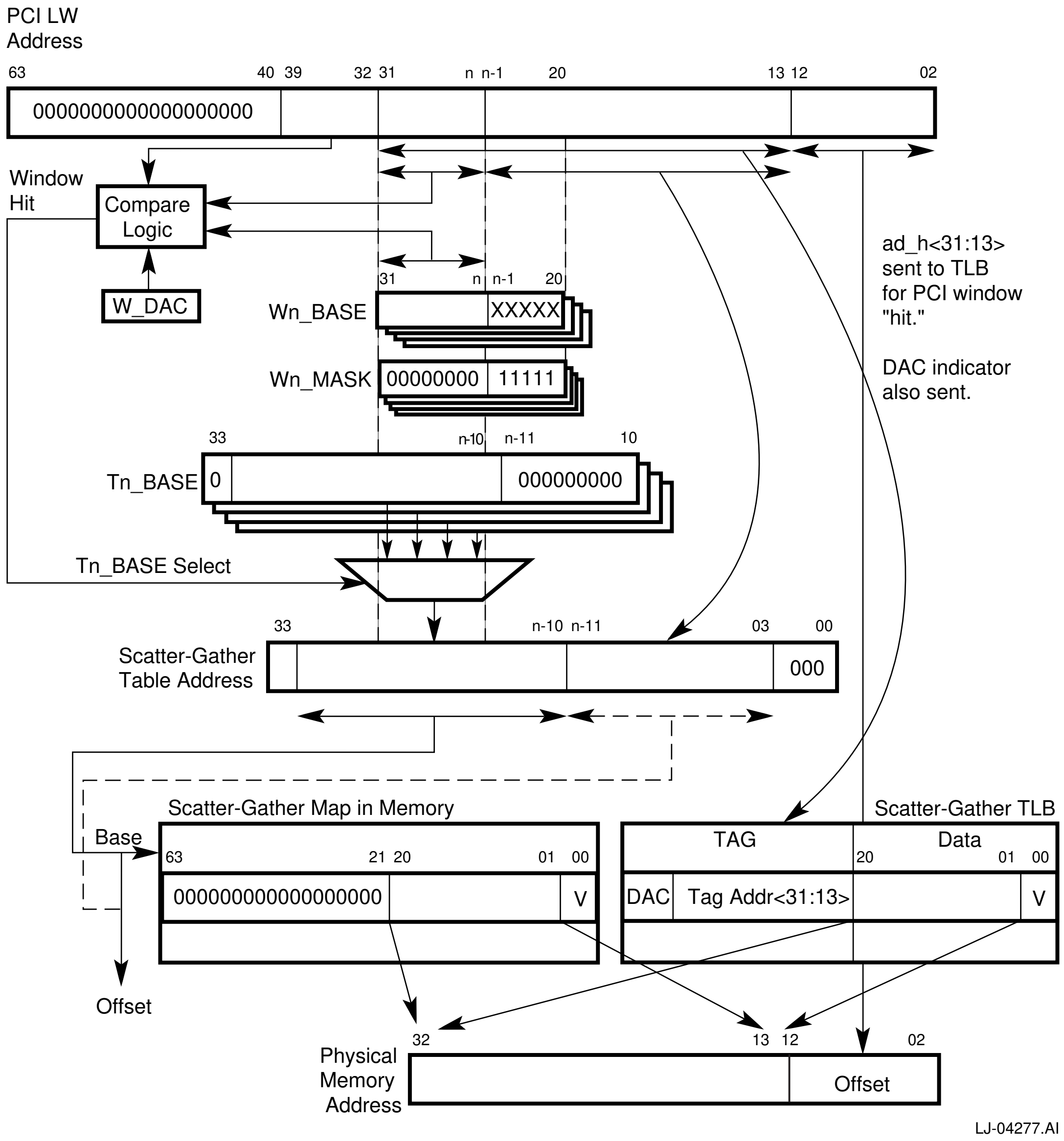
- The window compare logic determines if the PCI address has hit in one of the four windows and W*n*_BASE[W*n*_BASE_SG] determines if the scatter-gather path should be taken. If window 3 has DAC mode enabled, and the PCI cycle is a DAC cycle, then a further comparison is made between the high-order PCI bits and W_DAC.
- Address bits **ad_h<31:13>** are sent to the TLB associative tag together with the DAC hit indication. If the address and DAC bits match in the TLB, the corresponding 8KB page, 21164 memory 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_h<12:2>** to form the physical memory address. If the data entry is invalid or if the TAG compare failed, a TLB miss occurs.

6.3.3.1.2 Scatter-Gather TLB Miss Process The process for a scatter-gather TLB miss is as follows:

- The relevant bits of the PCI address (as determined by W*n*_MASK) are concatenated with the relevant T*n*_BASE bits to form the address used to access the scatter-gather PTE from a table located in main memory.
- Scatter-gather PTE<20:1> are used to generate the page address that is appended to the page offset to generate the physical memory address.

  At this point, the TLB is also updated (round-robin algorithm) with the four PTE entries that correspond to the 32KB PCI page 21164 memory 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.
- If the requested PTE is marked invalid (bit 0 clear), then a TLB invalid entry exception is taken.

Figure 6–19 Scatter-Gather Map Translation



6.3.4 Suggested Use of a PCI Window

Figure 6–20 shows a power-up PCI window assignment (as configured by firmware) and Table 6–14 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 6–20 Default PCI Window Allocation

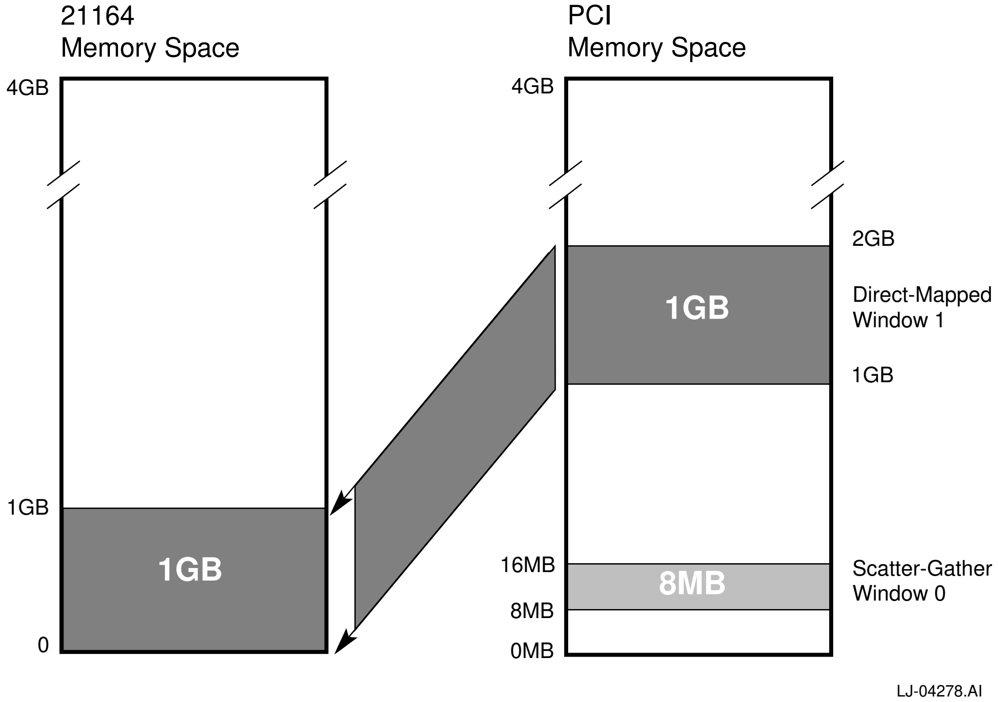


Table 6–14 PCI Window Power-Up Configuration

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	—	—

6.3.4.1 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 that are collectively known as peripheral component architecture compatibility holes.

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

6.3.4.2 Memory Chip Select Signal mem_cs_l

The PCI-to-EISA bridge set can be made using these two chips:

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

The PCI-to-EISA bridge provides address decode logic with 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 architecture compatibility holes.

This is known as main memory decoding in the PCEB 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 CIA uses **mem_cs_l** with W0_BASE.

In Figure 6–21, the two main holes are lightly shaded while the **mem_cs_l** range is darkly shaded.

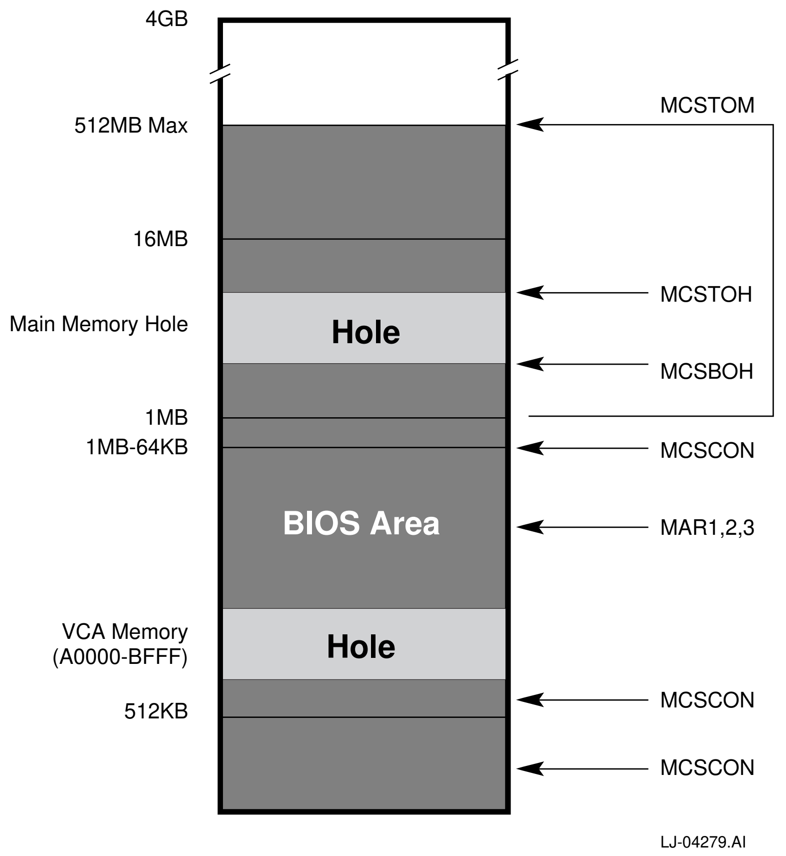
The **mem_cs_l** range of Figure 6–21 is subdivided into several portions (such as the BIOS area) that are individually enabled/disabled using CSRs as listed here:

- The MCSTOM (top of memory) register has 2MB granularity and can be programmed to select the regions from 1MB 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 MAR1, 2, and 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).

Note

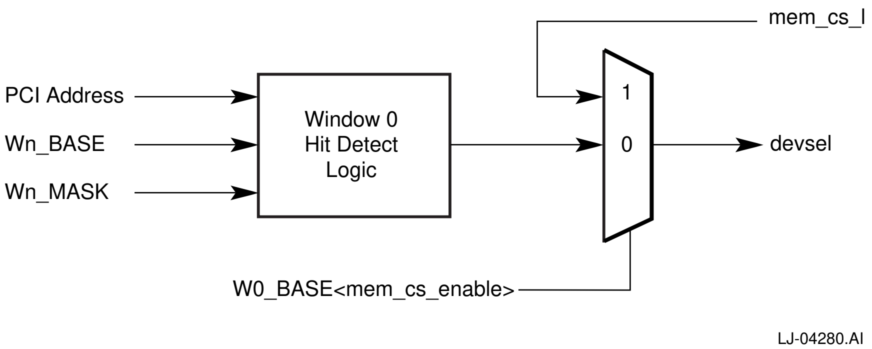
For more detail, please refer to the Intel 82375EB specification.

Figure 6–21 Memory Chip Select Signal (mem_cs_l) Decode Area



As shown in Figure 6–22, PCI window 0 in the CIA 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 **dev_sel_l** signal is asserted on the PCI.

Figure 6–22 Memory Chip Select Signal (mem_cs_l) Logic



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

- W0_BASE [W*n*_BASE_SG] determines if scatter-gather or direct mapping is applicable.
- W0_MASK size information must match the **mem_cs_l** size in order for the scatter-gather and direct-mapping algorithms to correctly use the translated base register.
- The **mem_cs_l** enable bit, W0_BASE [MEMCS_ENABLE], takes precedence over W0_BASE [W_EN].

Part II

DECchip 21171-BA (DSW) Information

Part II contains two chapters that provide information about the DECchip 21171-BA (DSW) chip. The following table lists the topics described in each chapter:

Chapter	Description
7	Pin signals
8	Architecture

7

DECchip 21171-BA Pin Descriptions

This chapter provides a description of the DECchip 21171-BA pin signals.

7.1 DECchip 21171-BA Pin List

Table 7–1 lists the pin signals grouped by function. The information in the Type column identifies a signal as input (I), output (O), bidirectional (B), or power (P).

Table 7–1 DECchip 21171-BA Pin List

Signal Name	Quantity	Type	Signal Name	Quantity	Type
		sysBus Signals (36 Total)			
cpu_dat_h<35:0>	36	B	—	—	—
		IOD Bus Signals (34 Total)			
cmc_h<8:0>	9	I	**ioc_h<6:0>**	7	O
iod_h<17:0>	18	B	—	—	—
		MEMDATA Bus Signals (73 Total)			
mem_data_h<71:0>	72	B	**mem_en_h**	1	I
		PLL Signals (6 Total)			
pll_agnd_h	1	O	**pll_clk_h**	1	I
pll_lp1_h	1	O	**pll_lp2_h**	1	I
pll_vdd_h	1	I	**pll_vss_h**	1	I

(continued on next page)

Table 7–1 (Cont.) DECchip 21171-BA Pin List

Signal Name	Quantity	Type	Signal Name	Quantity	Type
		Miscellaneous Signals (11 Total)			
config_h<1:0>	2	I	**spare1_or_count_ out_h**	1	O
spare<4:2>	3	I	**reset_l**	1	I
scan_in_h	1	I	**test_or_scan_out_h**	1	O
test_mode<1:0>	2	I	—	—	—

Pin Totals

Total input pins:	14
Total output pins:	20
Total bidirectional pins:	126
	—-
Total signal pins:	160
Total signal pins:	160
Total power pins:	24
Total ground pins:	24
	—-
Total pins:	208

7.2 DECchip 21171-BA Signal Descriptions

This section provides pin signal information, including a description of the signal; the clock edge on which the signal changes; and rules about signal usage during various sysBus transactions.

For simplicity, the rising edge of **sys_clk_out1_h** will be indicated by the name clk1R and the falling edge of **sys_clk_out1_h** will be indicated by the name clk1F. See Section 7.2.4 for more information about clock use on the DSW.

Signal descriptions are grouped by function and correspond to the pin list (see Table 7–1).

7.2.1 sysBus Signals

This section describes the sysBus data signal lines.

cpu_dat_h<35:0>

The **cpu_dat_h<35:0>** signals carry data between the 21164, Bcache, and the DSW. Using four DSW chips in parallel results in 144 signal lines that carry 128 data bits and 16 ECC bits. There are no specific data and ECC bits. All bits are treated the same in the DSW. ECC generation and checking is performed in the 21164 and CIA.

By default, **cpu_dat_h<35:0>** are in the tristate condition as a DSW internal signal; Drive_CPU_L, is in the high state. This allows data to be read into the DSW from the 21164 or Bcache. The data source is controlled by the 21164. Data is clocked into the DSW every CLKX cycle.

When Drive_CPU_L is in the low state, the DSW drives data onto **cpu_dat_h<35:0>** during each CLKX cycle.

7.2.2 IOD Bus Signals

This section describes the IOD bus signals.

iod<17:0>

The **iod<17:0>** signals carry data and ECC between the CIA and DSW on the IOD bus. Four DSW slices together drive and receive 72 data/ECC signal lines on the IOD bus between the DSW and the CIA.

The DSW slices treat all bits in the same manner and are unaware of the concept of ECC. No specific bits are dedicated to ECC nor data. All ECC generation and detection is performed in the 21164 and CIA.

The DSW puts these signal lines in tristate mode by default by deasserting an internal DSW signal, Drive_IOD_L. The DSW asserts Drive_IOD_L and so drives data to the CIA every CLKX cycle.

ioc_h<6:0>

The **ioc<6:0>** signals allow the CIA to control data flow on the IOD bus. The data transfers are between memory and the PCI bus. CIA control is encoded as shown in Table 7–2.

Table 7–2 ioc_h<6:0> Control Functions

ioc_h <6:4>	ioc_h <3>	ioc_h <2>	ioc_h <1>	ioc_h <0>	Function
000	Buffer *n*	x	x	x	Clear valid bits in PCI buffer *n*.
001	x	x	x	x	Load the LW register (Reserved).
010	Buffer *n*	Adr m2	Adr m1	Adr m0	Write PCI buffer *n*, adr m.
011	Mask3	Mask2	Mask1	Mask0	Write IOR (QW mask).
100	Buffer *n*	Adr m2	Adr m1	Adr m0	Read DMA buffer *n*, adr m.
101	Buffer *n*1	Buffer *n*0	Adr m1	Adr m0	Read IOW buffer *n*, adr m.
110	Buffer *n*1 Buffer *n*	Buffer *n*0 Adr m2	Adr m1	Adr m0	Read buffer (IOW or DMA) low longword.
111	Mask3	Mask2	Mask1	Mask0	Start IOW (buffer mask).

cmc_h<8:0>

The **cmc_h<8:0>** signal lines control flow of data between the 21164 and the memory arrays. They also start and stop loading of the free-running buffers (victim, flush, IOW buffers) in the DSW. In Table 7–3, A5 and A4 are address bits. The Var field encodes the transaction being performed as shown in Table 7–4, Table 7–5, and Table 7–6.

Table 7–3 cmc_h<8:0> Control Functions

cmc_h <8:5>	cmc_h <4>	cmc_h <3>	cmc_h <2>	cmc_h <1>	cmc_h <0>	Function
0000	Var 4[1]	Var 3	Var 2	Var 1	Var 0	Start/stop the buffer.
0010	x	A4[2]	Delay	Var 1	Var 0	Move fill data from memory to 21164.
0011	x	A4	x	Var 1	Var 0	Move IOR buffer data to 21164.
0100	A5[2]	A4	Delay	Var 1	Var 0	Move memory data to memory buffer 0.
0101	A5	A4	Delay	Var 1	Var 0	Move memory data to memory buffer 1.

[1]The variable field encodes the transaction being performed. See Table 7–4, Table 2–5, and Table 2–6.

[2]A5 and A4 are address bits.

(continued on next page)

Table 7–3 (Cont.) cmc_h<8:0> Control Functions

cmc_h <8:5>	cmc_h <4>	cmc_h <3>	cmc_h <2>	cmc_h <1>	cmc_h <0>	Function
0110	A5	A4	Delay	Stop IOW*n*1	Stop IOW*n*0	Move memory data to memory buffer 0, stop IOW buffer *n*.
0111	A5	A4	Delay	Stop IOW*n*1	Stop IOW*n*0	Move memory data to memory buffer 1, stop IOW buffer *n*.
1000	A5	A4	Var 2	Var 1	Var 0	Write victim buffer to memory.
1010	A5	A4	Var 2	Var 1	Var 0	Write DMA0 buffer to memory.
1011	A5	A4	Var 2	Var 1	Var 0	Write DMA1 buffer to memory.

Table 7–4 Var Encodings for CMC Command 0

Var<4:3>	Function
00	None.
01	Start victim.
10	Start flush buffer 0 and clear PCI 0 valid bits.
11	Start flush buffer 1 and clear PCI 0 valid bits.

Table 7–5 Var Encodings for CMC Commands 0, 8, A, and B

Var<2:0>	Function
000	None.
001	Stop victim.
010	Stop flush buffer 0.
011	Stop flush buffer 1.
100	Stop IOW 0.
101	Stop IOW 1.
110	Stop IOW 2.
111	Stop IOW 3.

Table 7–6 Var Encodings for CMC Commands 2, 3, 4, 5, 6, and 7

Var<1:0>	Function
00	None.
01	Stop victim.
10	Stop flush buffer 0.
11	Stop flush buffer 1.

7.2.3 MEMDATA Bus Signals

This section describes the MEMDATA bus signals.

mem_data_h<71:0>

The **mem_data_h<71:0>** signals carry data and ECC. Four DSW slices in parallel drive and receive 288 data/ECC signal lines on the MEMDATA bus between the DSW and memory arrays.

The DSW slices treat all bits in the same manner and are unaware of the concept of ECC. No specific bits are dedicated to ECC nor data. All ECC generation and detection is performed in the 21164 and CIA. By default, **mem_dat_h<35:0>** are in the active condition as a DSW internal signal; Drive_MEM_L is in the low state. This allows data to be driven onto the MEMDATA bus every CLKx cycle.

When Drive_MEM_L is in the high state, the DSW is in the tristate mode and data is read into the DSW from the memory arrays on the edge of each CLK2X cycle.

mem_en_h

When the CIA drives **mem_en_h** high, the DSW asserts Drive_MEM_L, enabling the DSW to drive the MEMDATA bus. When the CIA does not assert **mem_en_h**, the DSW deasserts Drive_MEM_L, enabling the memory arrays to drive the MEMDATA bus.

7.2.4 Phase-Locked Loop Signals

This section describes the DSW phase-locked loop signals.

pll_clk_h

The **pll_clk_h** signal is the system clock (or reference signal) input to the phase-locked loop (PLL) circuit. It supplies a clock with the same frequency as that supplied to the PCI.

pll_vdd_h

Signal **pll_vdd_h** is an input to the PLL. It is connected to the system board power.

pll_vss_h

Signal **pll_vss_h** is an input to the PLL. It is connected to the system board ground.

pll_lp2_h

Signal **pll_lp2_h** is an input to the PLL, driving a VCO input.

pll_lp1_h

Signal **pll_lp1_h** is an output of the PLL (phase detector output).

pll_a_h

Signal **pll_agnd_h** is an input to the PLL, connected to the system board ground.

7.2.5 Miscellaneous Signals

This section describes the miscellaneous DSW signals.

config_h<1:0>

Signals **config_h<1:0>** configure the data width of the IOD and MEMDATA buses as listed in Table 7–7.

Table 7–7 IOD Bus and MEMDATA Bus Data Width Selection

config<1:0>	IOD Bus Data Width	MEMDATA Bus Data Width
00[1]	32 bits	128 bits
01[1]	32 bits	256 bits
10	64 bits	128 bits
11	64 bits	256 bits

[1]Not implemented.

reset_l

Signal **reset_l** is the system reset input signal to the DSW. When **reset_l** is asserted, the DSW resets its internal state.

scan_in_h

Signal **scan_in_h** is the enable pin for the scan chain

spare1_or_count_out_h

Signal **spare1_or_count_out_h** is the **spare1** signal sharing a pin with the **count_out** signal. This pin is used during PLL tests.

test_mode_h<1:0>

Signals **test_mode_h<1:0>** define the four operating modes of the DSW as listed in Table 7–8.

Table 7–8 DSW Operating Modes

test_mode_h<1:0>	DSW Operating Mode
00	Tristate mode
01	Normal mode
10	Scan mode
11	PLL test mode

test_or_scan_out_h

Signal **test_or_scan_out_h** is the TEST_OUT output that is the end of the parametric NAND tree.

7.2.6 DECchip 21171-BA Pin Name List (Alphabetic)

Table 7–9 lists the DECchip 21171-BA (DSW) pins in alphabetical order. The following abbreviations are used in the Type column of the table:

- B = Bidirectional
- I = Input
- P = Power
- O = Output

Table 7–9 DECchip 21171-BA Pin Assignment List (Alphabetic)

Pin Name	Pin	Type	Pin Name	Pin	Type
cmc_h<0>	103	I	**cmc_h<1>**	102	I
cmc_h<2>	101	I	**cmc_h<3>**	100	I
cmc_h<4>	99	I	**cmc_h<5>**	116	I
cmc_h<6>	117	I	**cmc_h<7>**	118	I
cmc_h<8>	119	I	—	—	—
config_h<0>	49	I	**config_h<1>**	50	I
cpu_dat_h<0>	3	B	**cpu_dat_h<1>**	4	B
cpu_dat_h<2>	12	B	**cpu_dat_h<3>**	13	B
cpu_dat_h<4>	21	B	**cpu_dat_h<5>**	22	B
cpu_dat_h<6>	30	B	**cpu_dat_h<7>**	31	B
cpu_dat_h<8>	55	B	**cpu_dat_h<9>**	56	B
cpu_dat_h<10>	64	B	**cpu_dat_h<11>**	65	B
cpu_dat_h<12>	73	B	**cpu_dat_h<13>**	74	B
cpu_dat_h<14>	82	B	**cpu_dat_h<15>**	83	B
cpu_dat_h<16>	91	B	**cpu_dat_h<17>**	92	B
cpu_dat_h<18>	127	B	**cpu_dat_h<19>**	126	B
cpu_dat_h<20>	136	B	**cpu_dat_h<21>**	135	B
cpu_dat_h<22>	145	B	**cpu_dat_h<23>**	144	B
cpu_dat_h<24>	154	B	**cpu_dat_h<25>**	153	B
cpu_dat_h<26>	159	B	**cpu_dat_h<27>**	160	B
cpu_dat_h<28>	168	B	**cpu_dat_h<29>**	169	B
cpu_dat_h<30>	177	B	**cpu_dat_h<31>**	178	B
cpu_dat_h<32>	186	B	**cpu_dat_h<33>**	187	B
cpu_dat_h<34>	195	B	**cpu_dat_h<35>**	196	B
GND	5	P	**GND**	14	P

(continued on next page)

Table 7–9 (Cont.) DECchip 21171-BA Pin Assignment List (Alphabetic)

Pin Name	Pin	Type	Pin Name	Pin	Type
GND	23	P	**GND**	32	P
GND	44	P	**GND**	53	P
GND	62	P	**GND**	71	P
GND	80	P	**GND**	89	P
GND	98	P	**GND**	106	P
GND	111	P	**GND**	120	P
GND	129	P	**GND**	138	P
GND	147	P	**GND**	152	P
GND	161	P	**GND**	170	P
GND	179	P	**GND**	188	P
GND	197	P	**GND**	208	P
ioc_h<0>	45	I	**ioc_h<1>**	43	I
ioc_h<2>	42	I	**ioc_h<3>**	41	I
ioc_h<4>	40	I	**ioc_h<5>**	39	I
ioc_h<6>	38	I	—	—	—
iod_h<0>	2	B	**iod_h<1>**	11	B
iod_h<2>	20	B	**iod_h<3>**	29	B
iod_h<4>	54	B	**iod_h<5>**	63	B
iod_h<6>	72	B	**iod_h<7>**	81	B
iod_h<8>	90	B	**iod_h<9>**	128	B
iod_h<10>	137	B	**iod_h<11>**	146	B
iod_h<12>	155	B	**iod_h<13>**	158	B
iod_h<14>	167	B	**iod_h<15>**	176	B
iod_h<16>	185	B	**iod_h<17>**	194	B
mem_dat_h<0>	6	B	**mem_dat_h<1>**	7	B
mem_dat_h<2>	8	B	**mem_dat_h<3>**	9	B
mem_dat_h<4>	15	B	**mem_dat_h<5>**	16	B
mem_dat_h<6>	17	B	**mem_dat_h<7>**	18	B
mem_dat_h<8>	24	B	**mem_dat_h<9>**	25	B
mem_dat_h<10>	26	B	**mem_dat_h<11>**	27	B
mem_dat_h<12>	33	B	**mem_dat_h<13>**	34	B
mem_dat_h<14>	35	B	**mem_dat_h<15>**	36	B
mem_dat_h<16>	58	B	**mem_dat_h<17>**	59	B
mem_dat_h<18>	60	B	**mem_dat_h<19>**	61	B
mem_dat_h<20>	67	B	**mem_dat_h<21>**	68	B
mem_dat_h<22>	69	B	**mem_dat_h<23>**	70	B

(continued on next page)

Table 7–9 (Cont.) DECchip 21171-BA Pin Assignment List (Alphabetic)

Pin Name	Pin	Type	Pin Name	Pin	Type
mem_dat_h<24>	76	B	**mem_dat_h<25>**	77	B
mem_dat_h<26>	78	B	**mem_dat_h<27>**	79	B
mem_dat_h<28>	85	B	**mem_dat_h<29>**	86	B
mem_dat_h<30>	87	B	**mem_dat_h<31>**	88	B
mem_dat_h<32>	94	B	**mem_dat_h<33>**	95	B
mem_dat_h<34>	96	B	**mem_dat_h<35>**	97	B
mem_dat_h<36>	124	B	**mem_dat_h<37>**	123	B
mem_dat_h<38>	122	B	**mem_dat_h<39>**	121	B
mem_dat_h<40>	133	B	**mem_dat_h<41>**	132	B
mem_dat_h<42>	131	B	**mem_dat_h<43>**	130	B
mem_dat_h<44>	142	B	**mem_dat_h<45>**	141	B
mem_dat_h<46>	140	B	**mem_dat_h<47>**	139	B
mem_dat_h<48>	151	B	**mem_dat_h<49>**	150	B
mem_dat_h<50>	149	B	**mem_dat_h<51>**	148	B
mem_dat_h<52>	162	B	**mem_dat_h<53>**	163	B
mem_dat_h<54>	164	B	**mem_dat_h<55>**	165	B
mem_dat_h<56>	171	B	**mem_dat_h<57>**	172	B
mem_dat_h<58>	173	B	**mem_dat_h<59>**	174	B
mem_dat_h<60>	180	B	**mem_dat_h<61>**	181	B
mem_dat_h<62>	182	B	**mem_dat_h<63>**	183	B
mem_dat_h<64>	189	B	**mem_dat_h<65>**	190	B
mem_dat_h<66>	191	B	**mem_dat_h<67>**	192	B
mem_dat_h<68>	198	B	**mem_dat_h<69>**	199	B
mem_dat_h<70>	200	B	**mem_dat_h<71>**	201	B
mem_en_h	46	I	—	—	—
pll_agnd_h	109	I	**pll_clk_h**	105	I
pll_lp1_h	113	I	**pll_lp2_h**	107	I
pll_vdd_h	112	I	**pll_vss_h**	110	I
PWR5	1	P	**PWR5**	10	P
PWR5	19	P	**PWR5**	28	P
PWR5	37	P	**PWR5**	52	P
PWR5	57	P	**PWR5**	66	P
PWR5	75	P	**PWR5**	84	P
PWR5	93	P	**PWR5**	104	P
PWR5	108	P	**PWR5**	114	P
PWR5	125	P	**PWR5**	134	P

(continued on next page)

Table 7–9 (Cont.) DECchip 21171-BA Pin Assignment List (Alphabetic)

Pin Name	Pin	Type	Pin Name	Pin	Type
PWR5	143	P	**PWR5**	156	P
PWR5	157	P	**PWR5**	166	P
PWR5	175	P	**PWR5**	184	P
PWR5	193	P	**PWR5**	202	P
reset_l	115	I	**scan_in_h**	203	I
spare1_or_count_ out_h	204	O	**spare2**	205	I
spare3	206	I	**spare4**	207	I
test_or_scan_out_h	51	O	—	—	—
test_mode_h<0>	47	I	**test_mode_h<1>**	48	I

7.2.7 DECchip 21171-BA Pin Assignment List (Numeric)

Table 7–10 lists the DECchip 21171-BA (DSW) pins in numeric order. The following abbreviations are used in the Type column of the table:

- B = Bidirectional
- I = Input
- P = Power
- O = Output

Table 7–10 DECchip 21171-BA Pin Assignment (Numeric)

Pin Number	Pin Name	Type	Pin Number	Pin Name	Type
1	**PWR5**	P	2	**iod_h<0>**	B
3	**cpu_dat_h<0>**	B	4	**cpu_dat_h<1>**	B
5	**GND**	P	6	**mem_dat_h<0>**	B
7	**mem_dat_h<1>**	B	8	**mem_dat_h<2>**	B
9	**mem_dat_h<3>**	B	10	**PWR5**	P
11	**iod_h<1>**	B	12	**cpu_dat_h<2>**	B
13	**cpu_dat_h<3>**	B	14	**GND**	P
15	**mem_dat_h<4>**	B	16	**mem_dat_h<5>**	B
17	**mem_dat_h<6>**	B	18	**mem_dat_h<7>**	B
19	**PWR5**	P	20	**iod_h<2>**	B
21	**cpu_dat_h<4>**	B	22	**cpu_dat_h<5>**	B
23	**GND**	P	24	**mem_dat_h<8>**	B
25	**mem_dat_h<9>**	B	26	**mem_dat_h<10>**	B
27	**mem_dat_h<11>**	B	28	**PWR5**	P
29	**iod_h<3>**	B	30	**cpu_dat_h<6>**	B
31	**cpu_dat_h<7>**	B	32	**GND**	P
33	**mem_dat_h<12>**	B	34	**mem_dat_h<13>**	B
35	**mem_dat_h<14>**	B	36	**mem_dat_h<15>**	B
37	**PWR5**	P	38	**ioc_h<6>**	I
39	**ioc_h<5>**	I	40	**ioc_h<4>**	I
41	**ioc_h<3>**	I	42	**ioc_h<2>**	I
43	**ioc_h<1>**	I	44	**GND**	P
45	**ioc_h<0>**	I	46	**mem_en_h**	I
47	**test_mode_h<0>**	I	48	**test_mode_h<1>**	I
49	**config_h<0>**	I	50	**config_h<1>**	I
51	**test_or_scan_out_h**	O	52	**PWR5**	P
53	**GND**	P	54	**iod_h<4>**	B
55	**cpu_dat_h<8>**	B	56	**cpu_dat_h<9>**	B
57	**PWR5**	P	58	**mem_dat_h<16>**	B
59	**mem_dat_h<17>**	B	60	**mem_dat_h<18>**	B
61	**mem_dat_h<19>**	B	62	**GND**	P
63	**iod_h<5>**	B	64	**cpu_dat_h<10>**	B
65	**cpu_dat_h<11>**	B	66	**PWR5**	P
67	**mem_dat_h<20>**	B	68	**mem_dat_h<21>**	B
69	**mem_dat_h<22>**	B	70	**mem_dat_h<23>**	B
71	**GND**	P	72	**iod_h<6>**	B
73	**cpu_dat_h<12>**	B	74	**cpu_dat_h<13>**	B
75	**PWR5**	P	76	**mem_dat_h<24>**	B

(continued on next page)

Table 7–10 (Cont.) DECchip 21171-BA Pin Assignment (Numeric)

Pin Number	Pin Name	Type	Pin Number	Pin Name	Type
77	**mem_dat_h<25>**	B	78	**mem_dat_h<26>**	B
79	**mem_dat_h<27>**	B	80	**GND**	P
81	**iod_h<7>**	B	82	**cpu_dat_h<14>**	B
83	**cpu_dat_h<15>**	B	84	**PWR5**	P
85	**mem_dat_h<28>**	B	86	**mem_dat_h<29>**	B
87	**mem_dat_h<30>**	B	88	**mem_dat_h<31>**	B
89	**GND**	P	90	**iod_h<8>**	B
91	**cpu_dat_h<16>**	B	92	**cpu_dat_h<17>**	B
93	**PWR5**	P	94	**mem_dat_h<32>**	B
95	**mem_dat_h<33>**	B	96	**mem_dat_h<34>**	B
97	**mem_dat_h<35>**	B	98	**GND**	P
99	**cmc_h<4>**	I	100	**cmc_h<3>**	I
101	**cmc_h<2>**	I	102	**cmc_h<1>**	I
103	**cmc_h<0>**	I	104	**PWR5**	P
105	**pll_clk_h**	I	106	**GND**	P
107	**pll_lp2_h**	I	108	**PWR5**	P
109	**pll_agnd_h**	I	110	**pll_vss_h**	I
111	**GND**	P	112	**pll_vdd_h**	I
113	**pll_lp1_h**	I	114	**PWR5**	P
115	**reset_l**	I	116	**cmc_h<5>**	I
117	**cmc_h<6>**	I	118	**cmc_h<7>**	I
119	**cmc_h<8>**	I	120	**GND**	P
121	**mem_dat_h<39>**	B	122	**mem_dat_h<38>**	B
123	**mem_dat_h<37>**	B	124	**mem_dat_h<36>**	B
125	**PWR5**	P	126	**cpu_dat_h<19>**	B
127	**cpu_dat_h<18>**	B	128	**iod_h<9>**	B
129	**GND**	P	130	**mem_dat_h<43>**	B
131	**mem_dat_h<42>**	B	132	**mem_dat_h<41>**	B
133	**mem_dat_h<40>**	B	134	**PWR5**	P
135	**cpu_dat_h<21>**	B	136	**cpu_dat_h<20>**	B
137	**iod_h<10>**	B	138	**GND**	P
139	**mem_dat_h<47>**	B	140	**mem_dat_h<46>**	B
141	**mem_dat_h<45>**	B	142	**mem_dat_h<44>**	B
143	**PWR5**	P	144	**cpu_dat_h<23>**	B
145	**cpu_dat_h<22>**	B	146	**iod_h<11>**	B
147	**GND**	P	148	**mem_dat_h<51>**	B
149	**mem_dat_h<50>**	B	150	**mem_dat_h<49>**	B
151	**mem_dat_h<48>**	B	152	**GND**	P

(continued on next page)

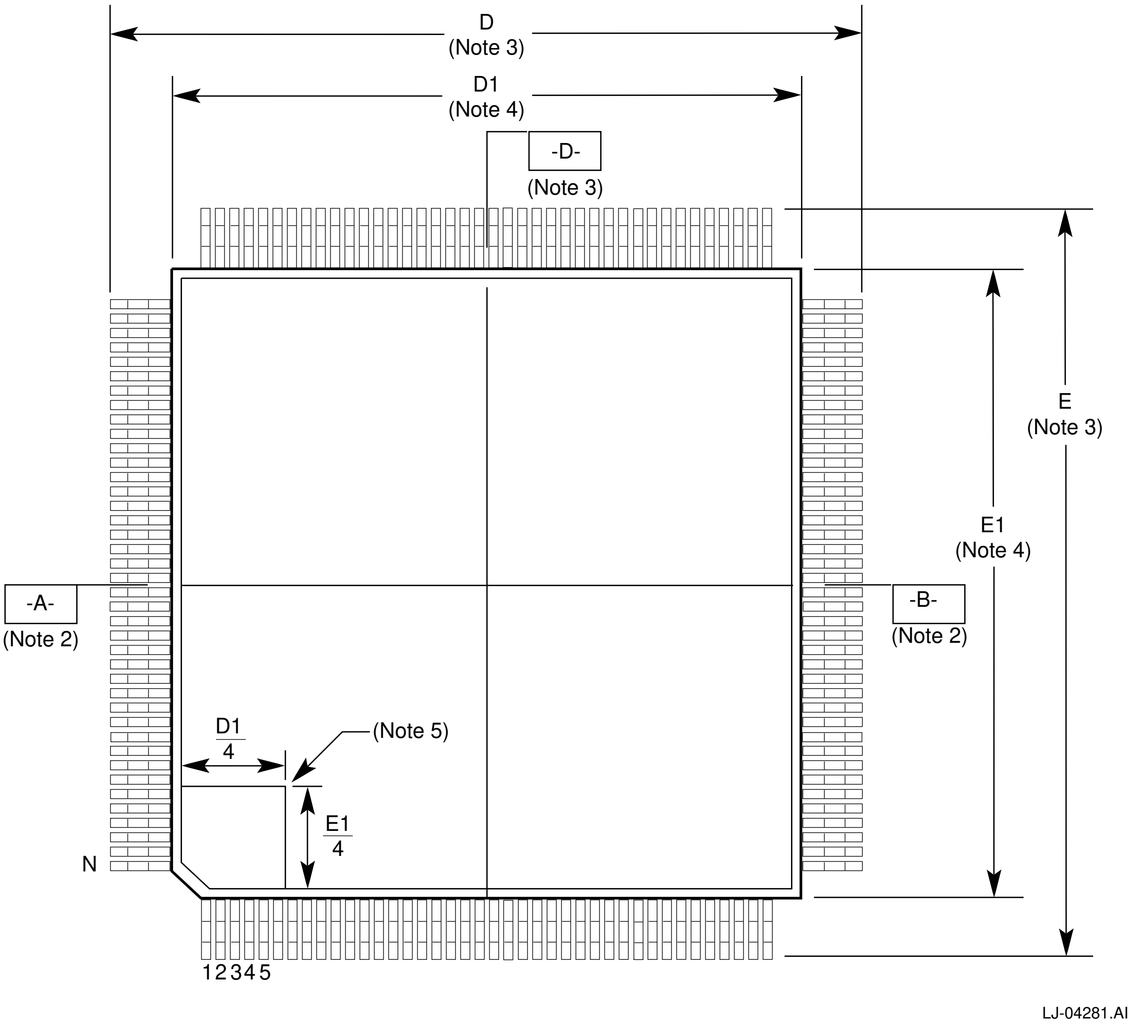
Table 7–10 (Cont.) DECchip 21171-BA Pin Assignment (Numeric)

Pin Number	Pin Name	Type	Pin Number	Pin Name	Type
153	**cpu_dat_h<25>**	B	154	**cpu_dat_h<24>**	B
155	**iod_h<12>**	B	156	**PWR5**	P
157	**PWR5**	P	158	**iod_h<13>**	B
159	**cpu_dat_h<26>**	B	160	**cpu_dat_h<27>**	B
161	**GND**	P	162	**mem_dat_h<52>**	B
163	**mem_dat_h<53>**	B	164	**mem_dat_h<54>**	B
165	**mem_dat_h<55>**	B	166	**PWR5**	P
167	**iod_h<14>**	B	168	**cpu_dat_h<28>**	B
169	**cpu_dat_h<29>**	B	170	**GND**	P
171	**mem_dat_h<56>**	B	172	**mem_dat_h<57>**	B
173	**mem_dat_h<58>**	B	174	**mem_dat_h<59>**	B
175	**PWR5**	P	176	**iod_h<15>**	B
177	**cpu_dat_h<30>**	B	178	**cpu_dat_h<31>**	B
179	**GND**	P	180	**mem_dat_h<60>**	B
181	**mem_dat_h<61>**	B	182	**mem_dat_h<62>**	B
183	**mem_dat_h<63>**	B	184	**PWR5**	P
185	**iod_h<16>**	B	186	**cpu_dat_h<32>**	B
187	**cpu_dat_h<33>**	B	188	**GND**	P
189	**mem_dat_h<64>**	B	190	**mem_dat_h<65>**	B
191	**mem_dat_h<66>**	B	192	**mem_dat_h<67>**	B
193	**PWR5**	P	194	**iod_h<17>**	B
195	**cpu_dat_h<34>**	B	196	**cpu_dat_h<35>**	B
197	**GND**	P	198	**mem_dat_h<68>**	B
199	**mem_dat_h<69>**	B	200	**mem_dat_h<70>**	B
201	**mem_dat_h<71>**	B	202	**PWR5**	P
203	**scan_in_h**	I	204	**spare1_or_count_out_h**	O
205	**spare2**	I	206	**spare3**	I
207	**spare4**	I	208	**GND**	P

7.3 DECchip 21171-BA Mechanical Specifications

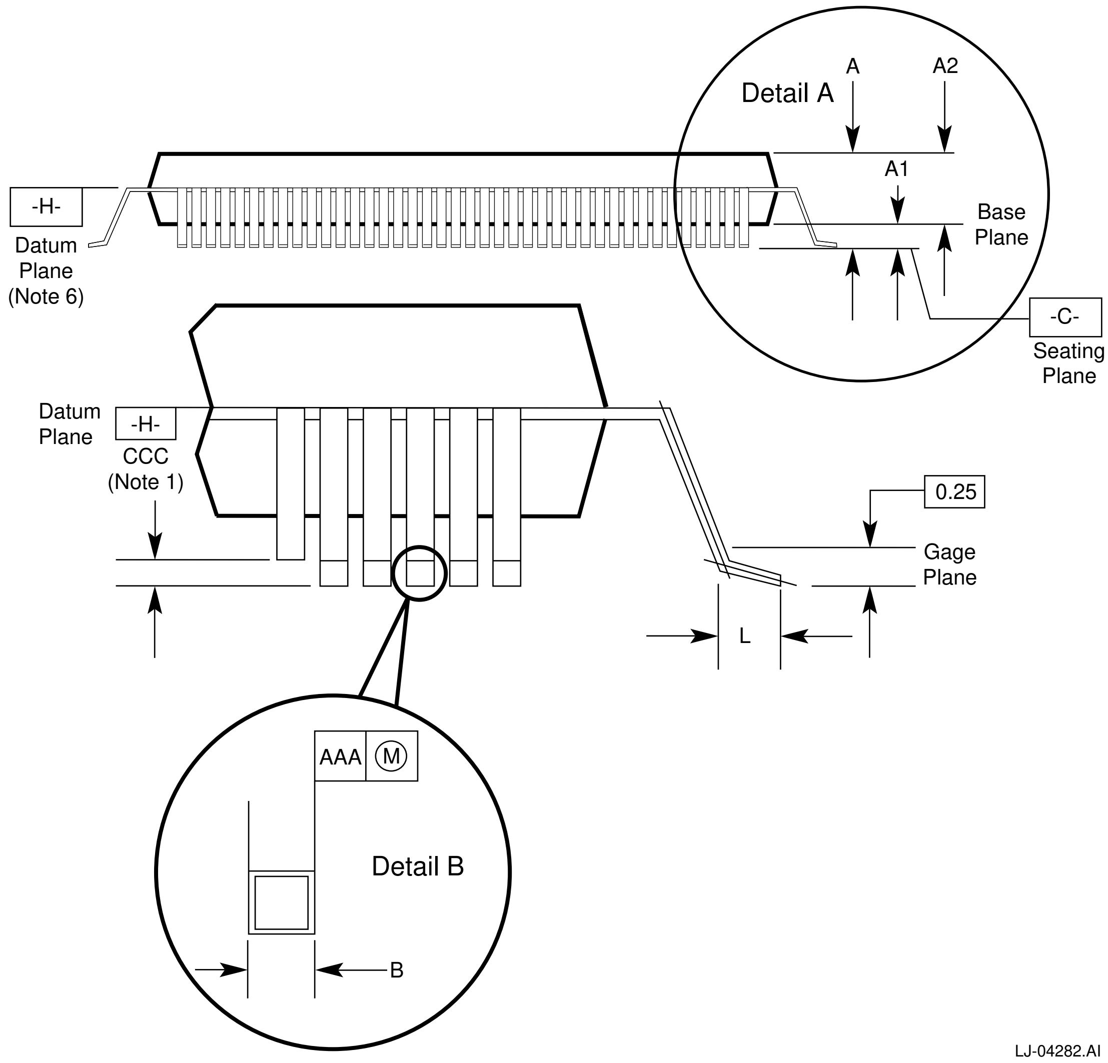
Figure 7–1 shows the DECchip 21171-BA (DSW) package dimensions.

Figure 7–1 DECchip 21171-BA Package Dimensions



(continued on next page)

Figure 7–1 (Cont.) DECchip 21171-BA Package Dimensions



Notes for Figure 7–1

The following notes apply to Figure 7–1:

Note 1. Coplanarity is the difference between the highest lead and the seating plane -H-.
Note 2. Datums A-B and -D- to be determined at Datum plane -H-.

Note 3. To be determined at seating plane -C-.
Note 4. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm (0.010 in) per side. Dimensions D1 and E1 do not include mold mismatch and are determined at Datum plane -H-.
Note 5. Details of pin 1 identifier are optional but must be located within the zone indicated.
Note 6. Date plane -H- is located at the mold parting line and is coincident with the bottom of the leads where the lead exits the plastic body.

Note

The drawing/dimensions are for reference only. For board layout, request a detailed engineering drawing from Digital Semiconductor.

Table 7–11 provides the maximum and minimum dimensions for the 21171-AB (DSW).

Table 7–11 DECchip 21171-BA Package Dimensions

Indicator	Minimum	Maximum
A	NA[1]	4.20 mm (0.165 in)
A1	0.25 mm (0.010 in)	NA
A2	3.17 mm (0.125 in)	3.67 mm (0.144 in)
B	0.17 mm (0.007 in)	0.28 mm (0.011 in)
D	30.35 mm (1.195 in)	30.85 mm (1.215 in)
D1	27.90 mm (1.098 in)	28.10 mm (1.106 in)
E	30.35 mm (1.195 in)	30.85 mm (1.215 in)
E1	27.90 mm (1.098 in)	28.10 mm (1.106 in)
L	0.40 mm (0.016 in)	0.80 mm (0.031 in)
aaa	NA	0.08 mm (0.003 in)
ccc	NA	0.08 mm (0.003 in)

Indicator	Basic
E	0.50 mm (0.020 in)

[1]NA = Not applicable.

8
DECchip 21171-BA Architecture Overview

This chapter describes the DECchip 21171-BA (DSW) architecture.

8.1 DSW Functional Units

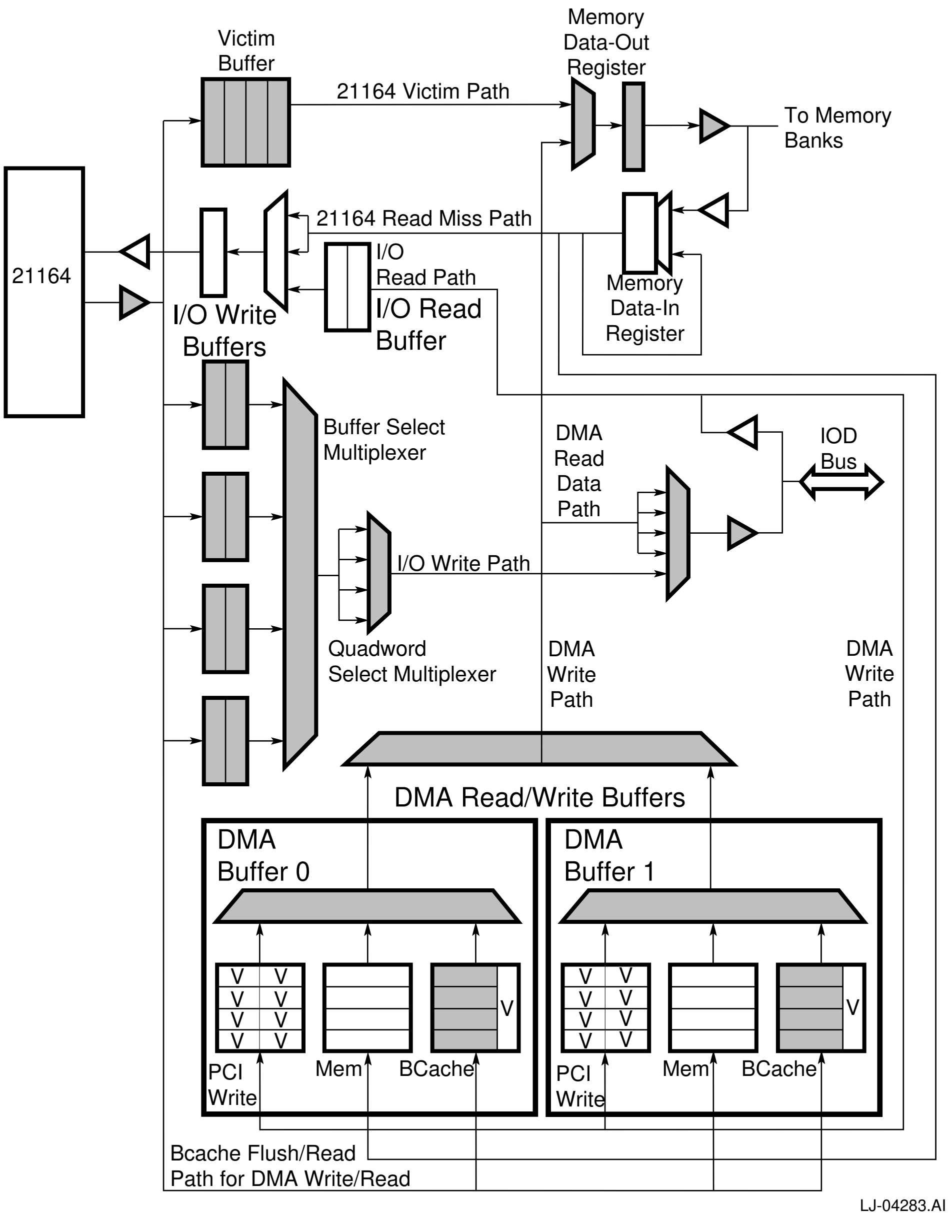
The DSW connects the data paths between the memory, the 21164/Bcache, and the CIA (for PCI data).

- The memory data path, on the MEMDATA bus, has 128 bits or 256 bits of data and 16 bits or 32 bits of ECC. The width of the data path, 128 bits or 256 bits, is selected using input pins **config_h<1:0>**.
- The 21164/Bcache data path, on the sysBus, has 128 bits of data and 16 bits of ECC.
- The CIA data path, on the IOD bus, has 64 bits of data and 8 bits of ECC.

The data paths supported by the DSW require four DSW slices to accommodate the data path widths. Each DSW slice is identical and there are no restrictions on how the bits of any bus are assigned.

The DSW is composed of mainly data buffers and multiplexers, as shown in Figure 8–1. All control for the DSW is supplied by the CIA (though some encoding and simple sequencing is performed by the DSW).

Figure 8–1 DSW Functional Block Diagram



The DSW functional units are:

- Victim buffer—64 bytes
- I/O read buffer—32 bytes
- Four I/O write buffers—4 * 32 bytes
- Two DMA buffer sets are provided for read and write transactions. Each set contains:
  - Memory buffer
  - Flush buffer
  - PCI buffer

8.1.1 Victim Buffer—64 Bytes

The victim buffer in each DSW slice has four 36-bit entries. Using four DSW slices results in a victim buffer with four 128-bit data entries and four 16-bit ECC entries.

The victim buffer is a free-loading first-in/first-out buffer (FIFO). The values in **cmc_h<8:0>** ''start'' and ''stop'' loading of the victim buffer. When buffer loading is ''started'' data is shifted into the buffer during every CLKX cycle. When buffer loading is ''stopped,'' the contents of the buffer are held and loading is inhibited. Normally buffer loading is ''stopped'' before data is read out to the MEMDATA bus.

The MEMDATA bus can be configured, using **config_h<1:0>**, for 128-bit or 256-bit width. The buffer can be addressed in two ways:

- 128-bit—The victim buffer data has addresses 0, 1, 2, or 3. The data will be sent to memory by using the low-order memory bus bits.
- 256-bit—The victim buffer data has addresses 0 or 2 and two octawords of data plus ECC are sent to the memory in one cycle.

Data is sent to memory from a flip-flop loaded at CLKX.

8.1.2 I/O Read Buffer (IOR)—32 Bytes

The IOR buffer in each DSW slice has two 36-bit entries. Using four DSW slices results in an IOR buffer with two 128-bit data entries and two associated 16-bit ECC entries.

The IOR buffer is loaded from the IOD bus by using a quadword mask. Any or all of the quadwords can be loaded in one cycle. Only one quadword of data comes from the IOD bus during one cycle so if more than a single quadword is selected, the same data will be written into each IOR buffer quadword.

One instruction to the DSW controls writing the entire IOR buffer. The instruction specifies which quadword to write first. Data is read from the IOR buffer and written into the IOR buffer on the edge of CLKX.

8.1.3 I/O Write (IOW) Buffers—4 * 32 Bytes

The IOW buffer in each DSW slice has four banks, with each bank containing two 36-bit registers. Using four DSW slices results in an IOW buffer with four banks, with each bank containing two 144-bit registers. The 144 bits contain 128 bits of data and 16 bits of ECC.

The IOW buffer is free-loading. Signals **ioc_h<6:0>** and **cmc_h<8:0>** ‘‘start’’ and ‘‘stop’’ loading of the IOW buffer.

When the IOW buffer is ‘‘started,’’ new data is loaded every CLKX cycle from the sysBus data path (the same data is loaded into each bank). When the IOW buffer is ‘‘stopped,’’ loading is inhibited and the current buffer contents are held.

Data is read from the IOW buffer to the IOD bus a quadword at each CLKX cycle by the CIA issuing an instruction on **ioc_h<6:0>**. The instruction specifies the bank number and the bank address of the quadword.

Quadword valid bits are provided for the IOW buffer. These valid bits are used to merge the appropriate IOW buffer quadword with the memory/Bcache data. Finer granularity merging (such as bytes) is performed in the CIA (because of ECC) by looping the appropriate memory/Bcache quadword through the CIA and merging the valid write bytes.

No data is preserved in the IOW buffers across transactions (no I/O read prefetched data or posted I/O write data). Consequently, there are no coherency issues associated with I/O read/write data lingering in the buffers.

8.1.4 DMA Buffer Sets

Two DMA buffer sets are provided for read and write transactions. Each buffer set consists of three buffers:

- Memory buffer for memory data
- Flush buffer for sysBus data
- PCI buffer for PCI DMA write data (not used during DMA read transactions)

The sysBus and memory data could have shared one buffer, because only one of these two will provide valid data, but two buffers simplified the control logic.

The memory buffer (memory buffers 0 and 1) in each DSW slice has four 36-bit entries. Using four DSW slices results in a 4-entry memory buffer with 144 bits per entry (includes 128 data bits and 16 ECC bits).

The memory buffer is loaded from the MEMDATA bus on a CLKX cycle. Only the entry addressed by the instruction on **cmc_h<8:0>** is loaded.

Data is read from the memory buffer by addressing an octaword or quadword location in the DMA buffer while the flush buffer and PCI buffer are *not* valid. Octawords are addressed for data to the memory arrays, and quadwords are addressed for data to the IOD bus.

8.1.4.1 Flush Buffer

The flush buffer (both flush 0 and flush 1) in each DSW slice has four 36-bit entries. Using four DSW slices results in a flush buffer with four 144-bit entries (128 bits of data and 16 bits of ECC).

The flush buffer is a free-loading FIFO. Signal **cmc<8:0>** ''starts'' and ''stops'' loading of the flush buffer. While it is ''started,'' it is loaded from the sysBus every CLKX cycle. When the flush buffer is ''stopped,'' loading is inhibited and the current contents are held.

''Stopping'' the flush buffer makes the buffer contents valid. When the flush buffer is valid, its contents will be selected over the contents of the memory buffer during a DMA read transaction.

Octawords are addressed for data sent to the memory arrays and quadwords are addressed for data sent to the IOD bus.

8.1.4.2 PCI Buffer

The PCI buffer (both PCI 0 and PCI 1) in each DSW slice has eight 18-bit entries. Using four DSW slices results in a PCI buffer of eight 72-bit entries (64 bits data and 8 bits ECC).

The PCI buffer is loaded from the IOD bus with a quadword address under the control of **ioc_h<6:0>**. Each quadword has a valid bit that is set when the quadword is written.

A DMA write transaction causes data to be read from the PCI buffer. When the write address points to valid data in the PCI buffer that data will be written to the memory arrays.

A

Technical Support and Ordering Information

A.1 Technical Support

If you need technical support or help deciding which literature best meets your needs, call the Digital Semiconductor Information Line:

United States and Canada	**1–800–332–2717**
Outside North America	**+1–508–628–4760**

A.2 Ordering Digital Semiconductor Products

To order the Alpha 21164 microprocessor and evaluation boards, contact your local distribtor.

You can order the following semiconductor products from Digital:

Product	Order Number
Alpha 21164 300-MHz Microprocessor	21164–BA
Alpha 21164 266-MHz Microprocessor	21164–AA
DECchip 21171 Core Logic Chipset	21171–AA
Alpha 21164 Microprocessor Evaluation Board Kit—266 MHz (Supports Windows NT operating systems.)	21A04–01

Note

Heat sink assemblies for the DECchip 21164 microprocessors are available from United Thermotechnology at (603)-642-3601.

A.3 Ordering Associated Third-Party Literature

You can order the following third-party literature directly from the vendor:

Title	Vendor
PCI Local Bus Specification PCI System Design Guide	PCI Special Interest Group 1–800–433–5177 (US) 1–503–797–4207 (International) 1–503–234–6762 (FAX)

Index

D

E

I