DECchip 21064 and DECchip 21064A Alpha AXP PCI Evaluation Board

User's Guide

Order Number: EC–N0640–72

Revision/Update Information: This is a new manual.

Digital Equipment Corporation
Maynard, Massachusetts

April 1994

This document was prepared using VAX DOCUMENT Version 2.1.

Contents

3 Functional Description

4 System Address Mapping

5 Board Requirements and Parameters

A System Register Descriptions

B SROM Initialization

C PCI Address Maps

Figures

Tables

Preface

This guide describes the DECchip 21064 and DECchip 21064A PCI Evaluation Board (EB64+). The board is an evaluation and development module for computing systems based on the DECchip 21064 or 21064A microprocessor.

Audience

This guide is written for system designers and others who use the EB64+ to design or evaluate computer systems based on the DECchip 21064 or 21064A microprocessor.

Scope

This guide describes the features, configuration, functional operation, and interfaces of the EB64+. Additional information is available in the EB64+ schematics and program source files. A list of related documentation and technical support information is provided at the end of this document.

Document Content

This guide contains the following content:

- Chapter 1, Introduction to the EB64+, introduces the EB64+, including its components, uses, and features.
- Chapter 2, System Jumpers and Connectors, describes the user environment configuration, board connectors and functions, jumper functions and logic states, and identifies jumper and connector locations.
- Chapter 3, Functional Description, provides a functional description of the board including the DECchip 21072 chipset, Bcache and memory subsystems, systems interrupts, clock and power subsystems, and PCI and ISA devices.
- Chapter 4, System Address Mapping, describes the mapping of the 34-bit processor address space into memory and I/O space addresses.

- Chapter 5, Power and Environmental Requirements, describes the board power and environmental requirements, and identifies major board components.
- Appendix A, System Register Descriptions, describes the control and status registers of the DECchip 21071-CA and DECchip 21071-DA.
- Appendix B, SROM Initialization, describes the general SROM, Bcache, and memory initialization steps and associated parameters. Also included are the firmware interface, timing considerations, SROM header, and configuration jumper descriptions.
- Appendix C, PCI Address Maps, provides the EB64+ PCI operating register address space maps.
- Appendix D, Technical Support, Ordering, and Associated Literature, describes how to obtain DECchip information and technical support, and order DECchip products and associated literature.

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 can not invoke UNDEFINED operations.

Ranges and Extents

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 are specified by a pair of numbers in angle brackets (<>) separated by two periods (..) 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 nonzero value. If the processor encounters a nonzero 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. Nonzero 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 Type Notation

Notation	Description
RW	A read-write bit or field. The value may be read and written by software, microcode, or hardware.
RO	A read-only bit or field. The value may be read by software, microcode, or hardware. It is written by hardware; software or microcode writes are ignored.

(continued on next page)

Table 1 (Cont.) Register Field Type Notation

Notation	Description
WO	A write-only bit. The value may be written by software and microcode. It is read by hardware. Reads by software or microcode return an UNPREDICTABLE result.
WZ	A write-only bit or field. The value may be written by software or microcode. It is read by hardware, and reads by software or microcode return a zero.
WC	A write-to-clear bit. The value may be read by software or microcode. Software or microcode writes of a 1 cause the bit to be cleared by hardware. Software or microcode writes of a 0 do not modify the state of the bit.
RC	A read-to-clear field. The value is written by hardware and remains unchanged until read. The value may be read by software or microcode, at which point hardware may 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 ignored on a write.
1	A 1 in a bit position indicates a register bit that is read as a 1 and ignored on a write.
x	An x in a bit position indicates a register bit that does not exist in hardware. The value is UNPREDICTABLE when read, and ignored on a write.

Bit Notation

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

Cautions

Cautions indicate potential damage to equipment or data.

Data Units

The following table defines the data unit terminology used through this guide.

Table 3 Data Units

Term	Words	Bytes	Bits	Other
Word	One	Two	16	
Longword	Two	Four	32	
Quadword	Four	Eight	64	
Octaword	Eight	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.

Schematic References

Logic schematics are included in the EB64+ design package. In this guide, references to schematic pages are printed in italics. For example, the following specifies schematic page 3:

‘‘. . . the 300 MHz oscillator (*eb64+.3*) supplies . . .’’

In some cases, more than one schematic page is referenced. For example, the following specifies schematic pages 17 through 20:

‘‘. . . the DRAM buffers (*eb64+.17-20*) . . .’’

Signal names in text are printed in boldface lowercase. Mixed-case and uppercase signal naming conventions are ignored. For example:

Table 4 Signal References

This Guide	Other Documentation	
cwmask7	cWMask7	CWMASK7

In addition, where a group of signals are referenced the signal names are combined. For example: the following would be combined as: **b0_l0-2_we_l**:

b0_l0_we_l
b0_l1_we_l
b0_l2_we_l

1

EB64+ Introduction

This chapter provides an overview of the EB64+, its components, features, and uses.

1.1 Introduction

The DECchip 21064 (21064) and DECchip 21064A (21064A) PCI Evaluation Board (EB64+) is an evaluation and development module for computing systems based on the 21064 or 21064A microprocessor. The EB64+ provides a single-board hardware and software development platform for the design, integration, and analysis of supporting logic and subsystems. The board also provides a platform for PCI I/O device hardware and software development.

Note

If you are not familiar with the 21064 or 21064A, you are encouraged to read the following system and memory/cache design application notes:

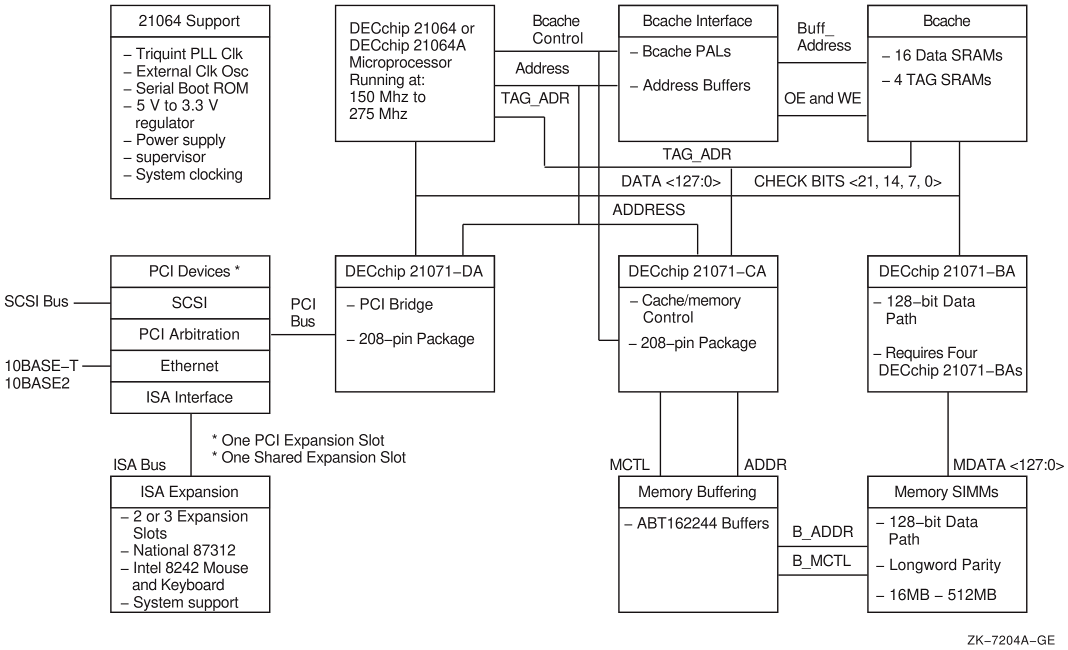
- Designing a System with the DECchip 21064 and DECchip 21064A Microprocessor
- Designing a Memory/Cache Subsystem for the DECchip 21064 and DECchip 21064A Microprocessor

Ordering information and a list of related documentation is provided at the end of this document.

1.2 System Components and Features

The EB64+ is implemented in industry-standard parts and uses a 21064 or 21064A CPU running at 150 MHz to 275 MHz. The functional components are shown in Figure 1–1 and introduced in the following subsections.

Figure 1–1 EB64+ Functional Block Diagram



1.2.1 Memory Subsystem

The DRAM memory can provide 16 MB to 512 MB with a 128-bit data bus. The memory is contained in two banks of four commodity single inline memory modules (SIMMs). Each SIMM is 36 bits wide, with 32 data bits, 1 parity bit, and 3 unused bits with 70 nsec or less access. The following SIMM sizes are supported:

- 1M x 36
- 2M x 36
- 4M x 36
- 8M x 36

- 16M x 36

1.2.2 DECchip 21072 Support Chipset

The 21064 and 21064A are supported by a DECchip 21072 ASIC chipset (21072), with a 128-bit memory interface. The chipset consists of the following three chips:

- DECchip 21071-CA (21071-CA) provides the interface from the 21064 or 21064A to cache and main memory, and includes the cache and memory controller.
- DECchip 21071-BA (21071-BA) provides a 32-bit data path from the 21064 or 21064A to main memory and I/O. Four chips provide the 128-bit interface.
- DECchip 21071-DA (21071-DA) provides an interface from the 21064 or 21064A to the Peripheral Component Interconnect (PCI).

The chipset includes the majority of functions required to develop a high-performance PC or workstation, requiring minimum discrete logic on the module. It provides flexible and generic functions to allow its use in a wide range of systems.

1.2.3 PAL Control Set

The EB64+ contains a five PAL control set and includes the following:

- Two 20V8-5 PALs provide Bcache output enable and write enable functions.
- One 22V10-10 PAL and one 22V10-15 PAL provide PCI arbitration.
- One Mach 210-20 PAL provides the PCI and ISA interrupts.

1.2.4 Bcache Subsystem Overview

The external backup cache (Bcache) subsystem provides 512-KB, 1-MB, or 2-MB cache sizes using a 128-bit data bus. The Bcache size can be reconfigured through on-board hardware and software jumpers.

The Bcache supports 12, 128K x 8 SRAMs and 4, 128K x 9 SRAMs. The cache tag supports 4, 64K x 4 SRAMs. All Bcache and tag SRAMs have a 15 nsec access time.

1.2.5 Clock Subsystem Overview

The clock subsystem provides clocks to the 21072 chipset and PCI devices. Four oscillators provide clocks for the Ethernet, SCSI, ISA, and combination chip functions.

1.2.6 PCI Interface Overview

The PCI interface provides a selectable PCI speed between 25 MHz and 33 MHz (based on 21064 or 21064A clock divisors,) and the following interface functions:

- PCI to ISA bridge provided through an Intel 82378IB or 82378ZB Saturn-I/O chip (SIO)
- Ethernet link provided by a PCI to Ethernet chip supporting 10Base-T (twisted pair) and 10Base2 (ThinWire)
- SCSI interface provided by an NCR 53C810 chip

The PCI has one dedicated slot and one shared slot with the ISA.

1.2.7 ISA Interface Overview

The ISA provides an expansion bus and the following system support functions:

- Mouse and keyboard controller functions provided through an Intel 8242 chip
- National 87312 chip used as the combination chip providing a diskette controller, two universal, asynchronous receiver/transmitters (UARTs), and a bidirectional parallel port
- Time of year (TOY) function provided by a Dallas DS1287A (DS1287A) chip
- An 8 KB nonvolatile RAM for operating system support provided by a DS1225Y chip

The ISA has two dedicated expansion slots and one shared expansion slot with the PCI.

1.2.8 Frame Buffer Support

Two 100-pin connectors are available to support a frame buffer. The buffer resides on the memory bus and supports a 128-bit interface.

1.2.9 Software Support

Software support includes an industry-standard 512-KB UVPROM containing debug monitor code and a console interface. Source code listings for all software (including boot code and diagnostic ROM monitor) are provided. The monitor provides functions such as:

- Download files through serial and Ethernet ports and diskette
- Load data from a ROM through the debug monitor
- Examine and deposit the EB64+ system register, 21064 and 21064A IPRs, and I/O mapped registers
- Examine and modify DRAM and I/O mapped memory
- Disassemble CPU instructions in memory
- Transfer control to programs in memory
- Perform native debugging including breakpoints and single stepping
- Perform full source level debugging using DECladebug running on a host communicating through an Ethernet connection
- Development code can be generated on a host system and loaded into the EB64+ through the serial line, Ethernet port, diskette, or ROM socket
- Full design database and user documentation

A serial ROM (SROM) contains the 21064 or 21064A initialization code, which is loaded into the instruction cache (Icache). When **reset** is deasserted, the contents of the SROM are read into the Icache. The code is executed from the Icache to perform initialization. Following initialization, control is transferred to the console and diagnostic ROM.

Development code can be generated on a host system and loaded into the EB64+ through the serial line, Ethernet port, or floppy diskette.

1.2.10 Design Support

The full database, including schematics and source files, is supplied. User documentation is also included. The database allows designers with no previous Alpha AXP experience to successfully develop a working Alpha AXP system with minimal help.

1.2.11 Evaluation Board Uses

The EB64+ has a remote debug capability and a software debug monitor for loading code into the system and performing other software debug functions such as memory read, memory write, and instruction breakpoint.

When combined with a hardware interface, the debug monitor can be used to write and debug software (for example, device drivers) for workstation and PC-type products. The monitor can also be used for embedded control products such as laser printers, communications engines (such as bridges and routers), and video products.

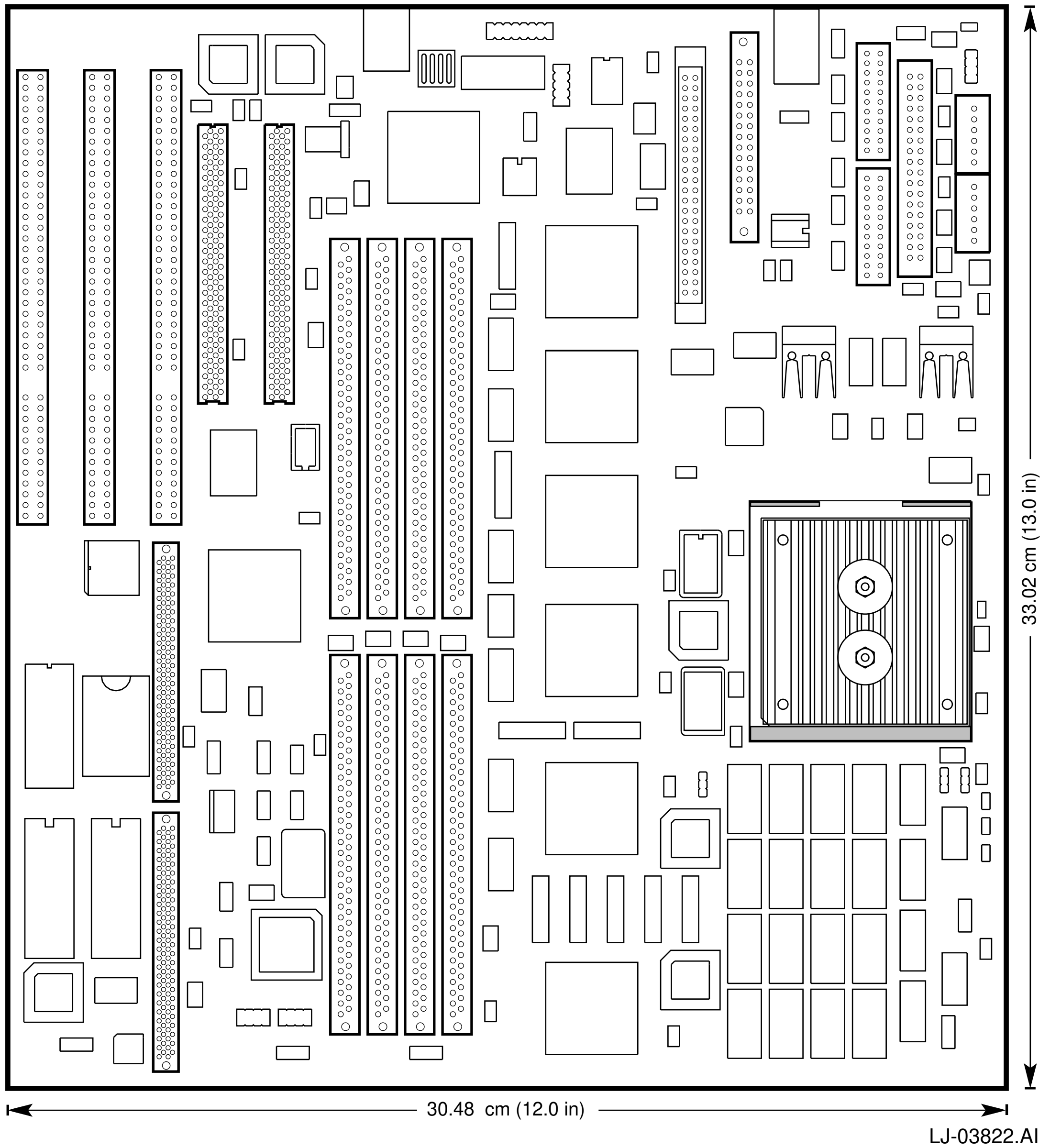
The EB64+ has user-configurable flexibility in Bcache size and DRAM access time. Performance benchmarks can be run to determine the effects of these characteristics on actual programs.

Different coding techniques can be tested and combined with the hardware tradeoffs available to optimize system performance.

The EB64+ provides a basis for a high-performance, low-cost deskside PC or workstation.

Figure 1–2 shows the EB64+ board component layout and board dimensions.

Figure 1–2 EB64+ Component Layout and Board Dimensions



2

System Jumpers and Connectors

2.1 Introduction

The EB64+ uses jumpers to implement variations in timing and speed, Bcache size, and network type selection. These jumpers must be configured for the user's environment. Onboard connectors are provided for the I/O and network interfaces, memory SIMMs, frame buffer, and serial and parallel peripheral ports.

After the module is configured, power can be applied, and the debug monitor can be run. The debug monitor and its commands are described in the *DECchip 21064 Evaluation Board Debug Monitor User's Guide*. Information about other software design tools is provided at the end of this document.

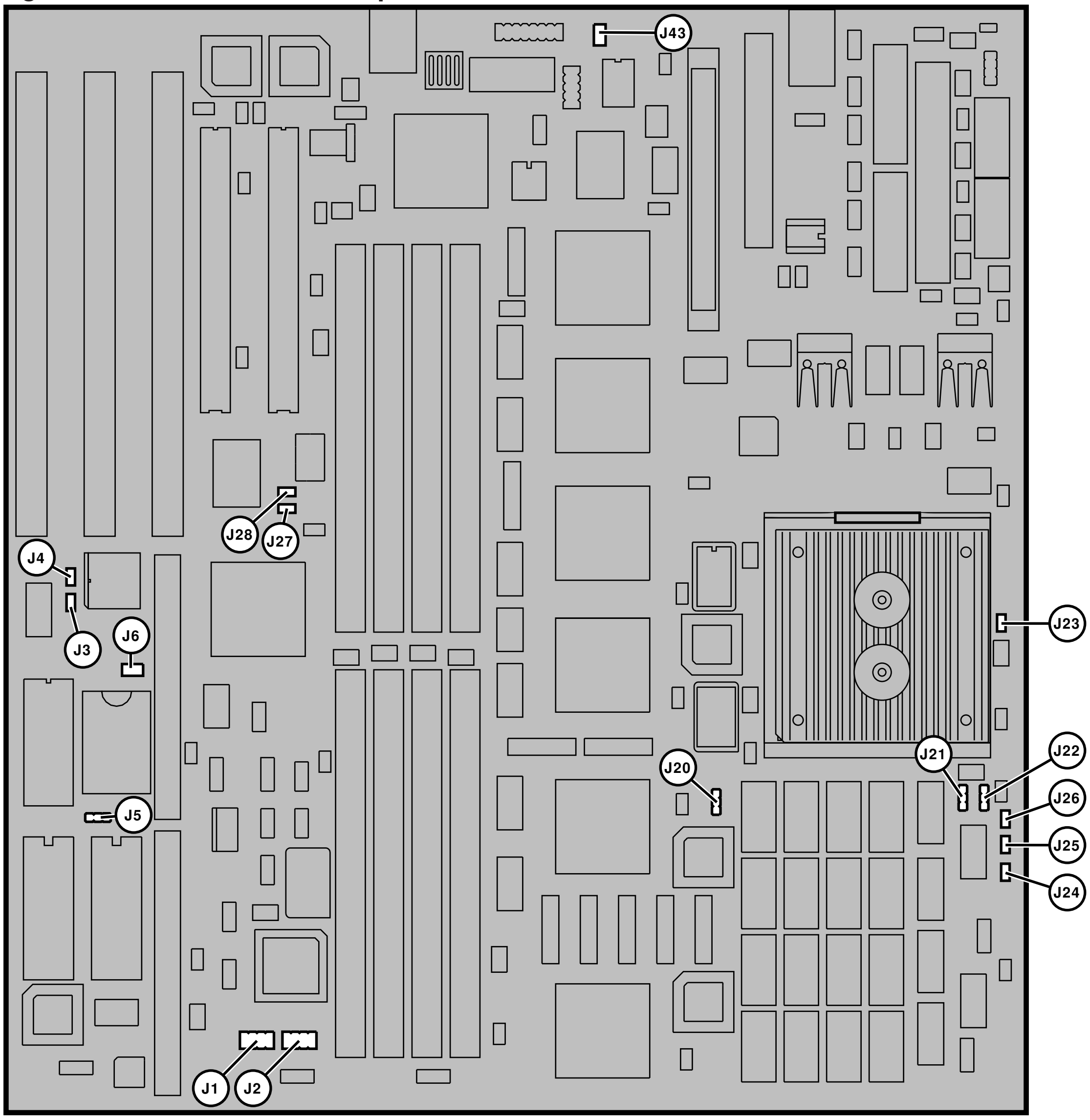
2.2 Configuration Jumpers

The software and hardware configuration jumpers are identified in Figure 2–1 and Figure 2–3, and described in the following tables.

2.2.1 Software Configuration Jumpers

The software configuration jumpers are completely programmable. The SROM code defines the software configuration jumpers as shown in Figure 2–1 (see Appendix B. Each jumper position is described in Table 2–1

Figure 2–1 EB64+ Board Jumpers



LJ-03825.AI

Figure 2–2 Software Configuration Jumpers

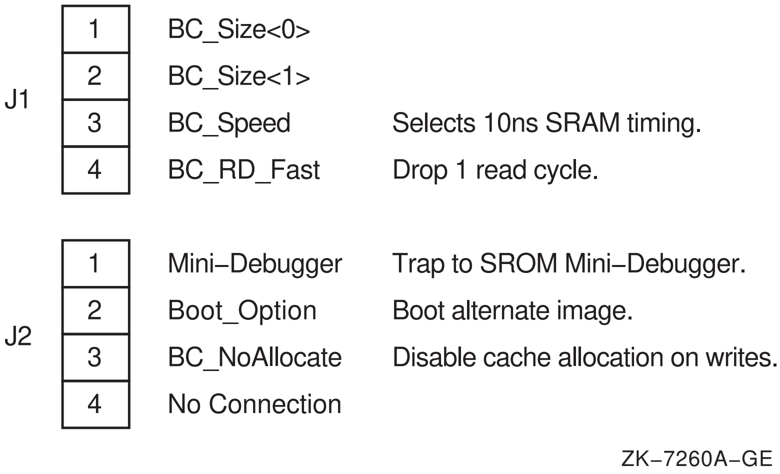


Table 2–1 Jumper Position Descriptions

Jumper Position	Position Description
BC_Size<1..0>	These jumpers force the Bcache to emulate a smaller Bcache as specified in the following table. When both positions are jumpered the Bcache is disabled. These jumpers are changed in conjunction with the appropriate hardware jumpers J21 and J22.

BC_Size<1:0>	Bcache
In, In	Disabled
In, Out	512KB
Out, In	1MB
Out, Out	2MB

(continued on next page)

Table 2–1 (Cont.) Jumper Position Descriptions

<table>
<tr><th>Jumper Position</th><th>Position Description</th></tr>
<tr><td>BC_Speed</td><td>This jumper selects Bcache timing parameters used to compute the BIU_CTL register value. The BC_Speed jumper (in) selects the appropriate timing for SRAMs with 10ns access time. This defaults to 15ns SRAM timing without a jumper installed.
<table>
<tr><th>BC_Speed</th><th>Bcache speed</th></tr>
<tr><td>In</td><td>Selects 10ns SRAM timing</td></tr>
<tr><td>Out</td><td>Selects 15ns SRAM timing</td></tr>
</table></td></tr>
<tr><td>BC_RD_Fast</td><td>This jumper forces a read speed setting of 1 cycle faster than nominal.
<table>
<tr><th>BC_RD_Fast</th><th>Bcache speed</th></tr>
<tr><td>In</td><td>Make Read Speed 1 cycle faster</td></tr>
<tr><td>Out</td><td>Nominal Read Speed</td></tr>
</table></td></tr>
<tr><td>Mini-Debugger</td><td>The SROM Mini-Debugger is provided in the SROM. This jumper (In) causes the SROM initialization to trap to the Mini-Debugger after all initialization is complete, but before starting the execution of the System ROM code.</td></tr>
<tr><td>Boot_Option</td><td>This jumper (In) selects the second (alternate) image from the System ROM. By default, the first image is loaded. The SROM allows the System ROM to contain two different ROM images each with its own header. The header informs the SROM where to load the image, and if it has been compressed with the MAKEROM tool.

For System ROMs which contain a single image, the header is optional. If the header does not exist, the entire 512-KB System ROM is loaded and executed at physical address zero.</td></tr>
<tr><td>BC_NoAllocate</td><td>This Jumper (In) disables Bcache allocates on writes to cacheable memory.</td></tr>
</table>

2.2.2 Hardware Configuration Jumpers

Hardware configuration jumpers are shown in Figure 2–1 and described in Table 2–2.

Table 2–2 EB64+ Board Jumpers

Connector	Pins	Description
		8242 Controller
J3	2	Reserved. (color_8242 (*eb64+.42*))
J4	2	Reserved. (test_8242 (*eb64+.42*))
		UVPROM Select
J5	3	UVPROM Select (*eb64+.44*)

Select	Jumper Pins
U21 ROM	1 to 2
U22 ROM	2 to 3

Connector	Pins	Description
		TOY Clear
J6	2	TOY Clear (*eb64+.44*)

Select	Jumper Pins
Clear NVRAM	Jumper Installed (0)
Default	Default (1)
	Note: The TOY jumper should only be installed or removed with the **power off**.

(continued on next page)

Table 2–2 (Cont.) EB64+ Board Jumpers

Connector	Pins	Description
		Bcache Control
J20	3	Bcache write (*eb64+.9*)

Select	Jumper Pins
Fast write	1 to 2
Slow write	2 to 3

Connector	Pins	Description
J21/J22	3	Adr20/Adr19 Bcache (*eb64+.10*)

Jumper	512 KB	1 MB	2 MB	Reserved
J21	1 to 2	1 to 2	2 to 3	2 to 3
J22	1 to 2	2 to 3	2 to 3	1 to 2

Connector	Pins	Description
		System Clock Functions
J23	2	System clock divisor. Used only with DECchip 21064A. (*eb64+.2*)
J26/J25/J24	2	21064 CPU clock divisor selection. (*eb64+.4*)

J26 IRQ2	J25 IRQ1	J24 IRQ0	Divisor
0	0	0	2
0	0	1	3
0	1	0	4
0	1	1	5
1	0	0	6
1	0	1	7
1	1	0	8
1	1	1	8

(continued on next page)

Table 2–2 (Cont.) EB64+ Board Jumpers

Connector	Pins	Description
J26/J25/J24	2	21064A CPU clock divisor selection. (*eb64+.4*)

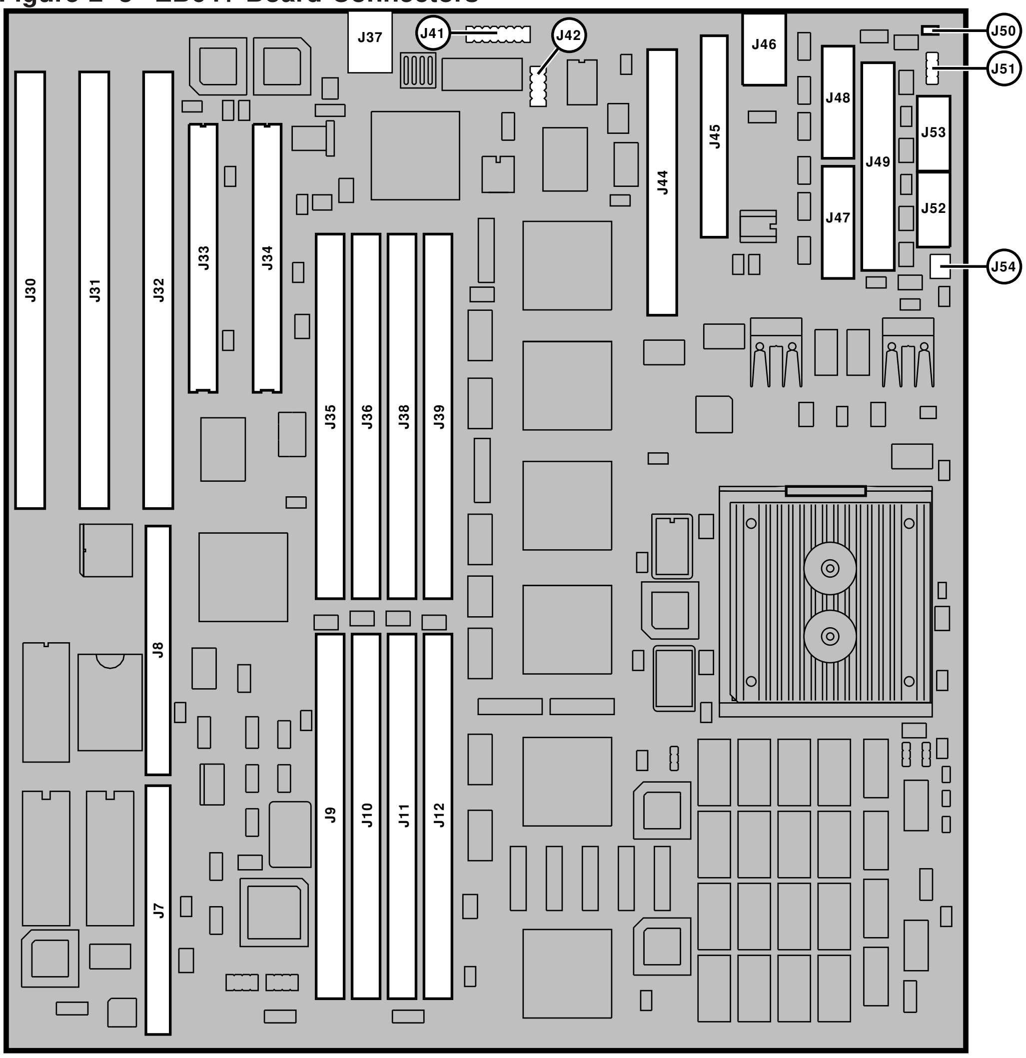
J23 (sysClkDiv_h)	J26 IRQ2	J25 IRQ1	J24 IRQ0	Divisor
0	0	0	0	2
0	0	0	1	3
0	0	1	0	4
0	0	1	1	5
0	1	0	0	6
0	1	0	1	7
0	1	1	0	8
1	1	1	1	9
1	0	0	0	10
1	0	0	1	11
1	0	1	0	12
1	0	1	1	13
1	1	0	0	14
1	1	0	1	15
1	1	1	0	16
1	1	1	1	17

Connector	Pins	Description
		Parallel Port Control
J27	2	Reserved. (PDIR Jumper (*eb64+.40*))
J28	2	Reserved. (POE Jumper (*eb64+.40*))
		Ethernet Select
J43	2	Reserved. (**aui_tp** (*eb64+.30*))

2.2.3 EB64+ Board Connectors

The module connectors are shown in Figure 2–3 and described in Table 2–3.

Figure 2–3 EB64+ Board Connectors



LJ-03824.AI

Table 2–3 Module Connector Descriptions

Connector	Pins	Description
		Frame Buffer Connectors
J7	100	Frame buffer connector (*eb64+.25*)
J8	100	Frame buffer connector (*eb64+.25*)
		Memory SIMMS
J12	72	Bank 0, DRAM 0 SIMM (*eb64+.21*)
J11	72	Bank 0, DRAM 1 SIMM (*eb64+.21*)
J39	72	Bank 0, DRAM 2 SIMM (*eb64+.22*)
J38	72	Bank 0, DRAM 3 SIMM (*eb64+.22*)
J10	72	Bank 1, DRAM 0 SIMM (*eb64+.23*)
J9	72	Bank 1, DRAM 1 SIMM (*eb64+.23*)
J36	72	Bank 1, DRAM 2 SIMM (*eb64+.24*)
J35	72	Bank 1, DRAM 3 SIMM (*eb64+.24*)
		ISA Connectors
J32	98	ISA expansion connector 0 (*eb64+.38*)
J31	98	ISA expansion connector 1 (*eb64+.38*)
J30	98	ISA expansion connector 2 (*eb64+.38*)
		PCI Connectors
J34	124	PCI expansion connector 0 (*eb64+.34*)
J33	124	PCI expansion connector 1 (*eb64+.34*)

(continued on next page)

Table 2–3 (Cont.) Module Connector Descriptions

Connector	Pins	Description
		Keyboard and Mouse
J37	12	Keyboard and mouse connector (*eb64+.42*) **Note** Mouse is the top connector.
		Ethernet Connectors
J41	16	Ethernet connector, ThinWire (10Base2) (*eb64+.31*)
J42	10	Ethernet connector, twisted pair (10Base-T) (*eb64+.31*)
		SCSI Bus
J44	50	SCSI bus connector (*eb64+.33*)
		National 87312 Connectors
J45	26	Combination chip parallel port connector (*eb64+.40*)
		SROM Test
J46	6	SROM test data serial port input connector (*eb64+.3*) **Note** This connector can be used as a terminal port for the Mini-debugger.

(continued on next page)

Table 2–3 (Cont.) Module Connector Descriptions

Connector	Pins	Description
J47	10	Combination chip serial communication port 1 connector (*eb64+.41*) **Note** This connector can be used as a terminal port for the Debug Monitor.
J48	10	Combination chip serial communication port 2 connector (*eb64+.41*)
J49	34	Combination chip diskette drive connector (*eb64+.41*)
		System Reset
J50	2	System reset switch connector (*eb64+.47*)
		Speaker Connector
J51	4	Speaker should be connected to pins 1 and 3 (*eb64+.49*)
		Power Connectors
J52	6	Module power connector (ground, -5 V, +5 V (VDD)), (*eb64+.48*)
J53	6	Module power connector (ground, +12 V, -12 V, +5 V (VDD), **p_dcok** (*eb64+.48*)) **Note:** Power for the EB64+ is provided by a user-supplied, standard PC power supply. Digital does not provide this power supply.
J54	3	CPU fan power and sensor (*eb64+.48*) **Caution: Fan sensor required** The fan *must* have a built-in sensor that drives a signal if the airflow stops. The sensor is connector to J54. The fan supplied with the EB64+ includes an airflow sensor.

3 Functional Description

This chapter describes the functional operation of the EB64+. The description introduces the ASIC support chipset and describes its implementation with the 21064 or 21064A microprocessor and its supporting memory and I/O devices.

Information, such as bus timing and protocol, found in other specifications, data sheets, and reference documentation is not duplicated. A list of supporting documents and order numbers is provided at the end of this document.

Note

For detailed descriptions of chipset logic, operations, and transactions refer to the *DECchip 21071-AA and DECchip 21072-AA Chipset Data Sheet.*

For details of the PCI interface refer to the *PCI System Design Guide.*

3.1 Chipset Introduction

The DECchip 21072-AA chipset provides a cost-competitive solution for designers developing uniprocessor systems using the 21064 or 21064A processor. The chipset provides a 128-bit memory interface, and includes the following three gate arrays:

- DECchip 21071-CA (21071-CA): cache and memory controller—208-pin PQFP
- DECchip 21071-DA (21071-DA): PCI interface—208-pin PQFP
- DECchip 21071-BA (21071-BA): Data path—208-pin PQFP

3.1.1 21071-CA Chip Introduction

The 21071-CA chip provides the interface from the 21064 or 21064A to and main memory. It also provides the system interface to the cache. The chip controls and moves data to and from banks of main memory.

It responds to commands from the CPU and 21071-DA and arbitrates between them. It also provides support for control of the Bcache RAMs during CPU cache miss, and DMA transactions.

On the EB64+ the 21071-CA controls two banks of DRAM SIMMs. The SIMMs can range in size from 1M x 36 to 16M x 36. Each bank can accommodate four 36-bit SIMMs to support a 128-bit data path with longword parity.

Figure 3–1 shows the maximum and minimum SIMM bank layouts.

The chip provides support for a single video bank of dual-port RAM (VRAM). This bank may have 128K, 256K, 512K, or 1M locations. Each location consists of octaword data for the 128-bit interface. VRAM capacity can vary from 1 MB to 16 MB.

The components of the cache and memory subsystem are distributed between the 21071-CA and the 21071-BA. Together, the chips serve as an interface between the sysBus and memory subsystem. (See Figure 3–2.)

The CPU, 21071-DA, cache, and memory communicate with each other through the sysBus. The sysBus is essentially the processor pinbus with additional signals for DMA transaction control, arbitration, and cache control. The 21071-CA chip performs the Bcache and memory control functions. The following list summarizes the major features of the 21071-CA:

- Provides control for filling the Bcache and extracting victims on CPU-initiated transactions.
- Provides control for probing the Bcache on DMA transactions and invalidating the Bcache on DMA write hits.
- Provides arbitration between the CPU and the 21071-DA for control of the sysBus.
- Stores addresses for the four cache line memory write buffer.
- Controls the loading of the I/O write buffer and the DMA read buffer.

Figure 3–1 Maximum SIMM Bank Layout

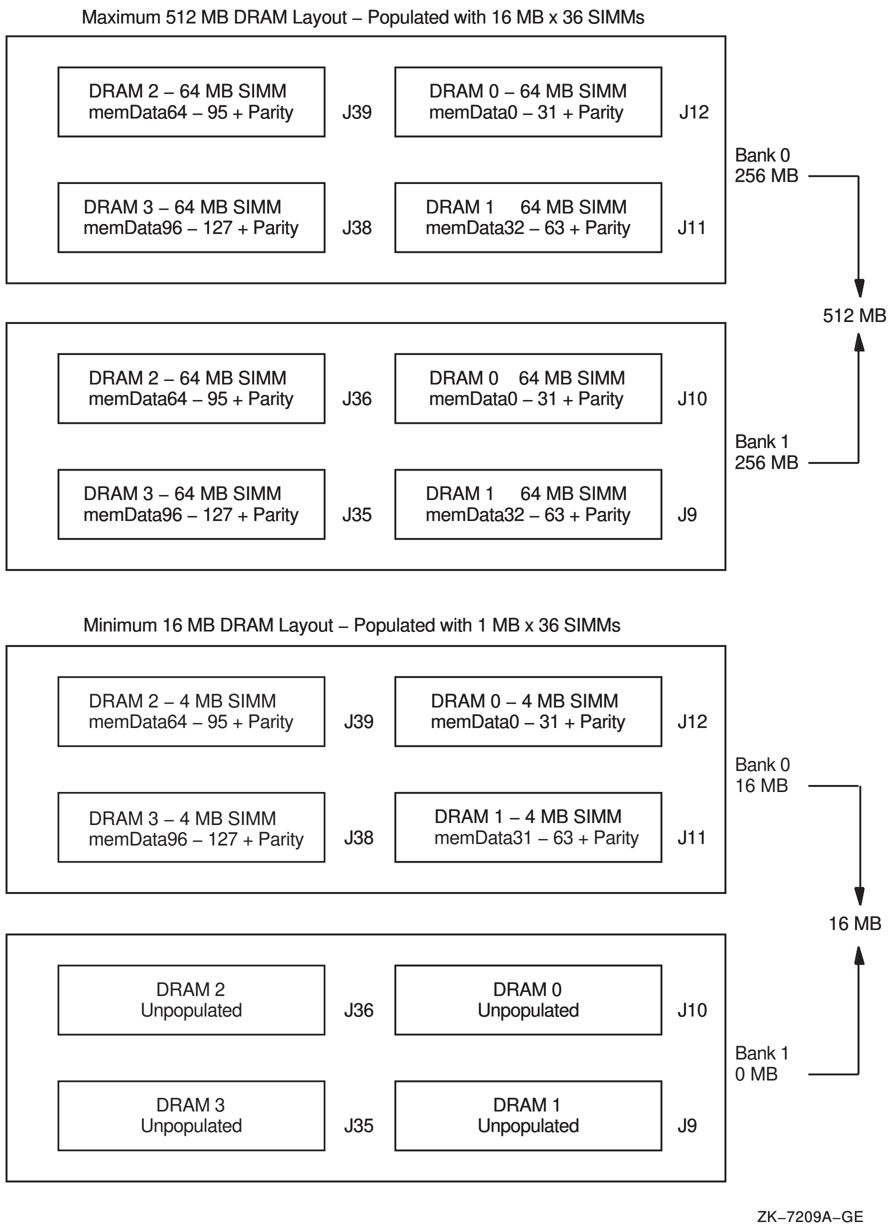
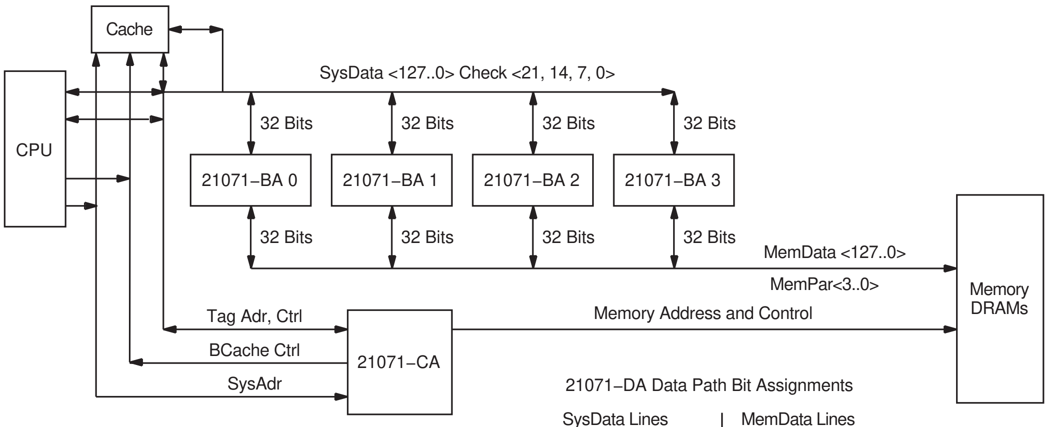


Figure 3–2 Basic Cache and Memory Subsystem Address and Data Paths



SysData Lines	MemData Lines
21071−BA 0 <31..0>	MemData <31..0>
21071−BA 1 <63.32>	MemData <63.32>
21071−BA 2 <95..64>	MemData <95..64>
21071−BA 3 <127..96>	MemData <127..96>

- Uses fast-page mode on the DRAMs to improve performance on DMA burst reads and memory writes.
- Supports a frame buffer on the memory data bus.

3.1.2 21071-BA Introduction

The 21071-BA chip provides a 32-bit data path from the 21064 or 21064A to main memory and I/O. Four chips are required for the 128-bit interface.

The chip contains the cache and memory interface data path which includes buffers for victim, noncacheable write, and DMA write operations. It also contains the I/O subsystem data path which provides buffering for DMA read and write data, and I/O read and write data.

The chip interfaces to the cache and CPU using the CPU sysBus (pin bus). It interfaces with the 21071-DA through the 32-bit epiBus (communications path between the 21071-DA and 21071-BA). The 21071-BA functions as the data path for the cache, memory, and I/O subsystem, and contains the following data path functions:

Error Detection Logic: The 21071-BA supports longword parity on the 128-bit memory interface. Error checking and generation is performed only on DMA-initiated transactions; error checking and generation on CPU-initiated transactions is performed by the CPU.

Memory Write Buffer: The memory write buffer has four entries; each entry is a cache line (32 bytes). The buffer is distributed across the four 21071-BA chips in the system. Data stored in this buffer has passed all cache coherency checks and is written to memory in the order it was received on the sysBus.

Memory Read Buffer: The memory read buffer is a one cache line temporary holding buffer used to store data written by the CPU on memory writes, or to store data read from the PCI bus on CPU reads.

I/O Write Buffer: The I/O write buffer has two entries: one entry acts as a write buffer for CPU I/O writes to the 21071-BA or PCI bus, the other acts as a holding buffer.

DMA Read Buffer: The DMA read buffer stores data that is being read from the memory by a device on the PCI bus. This buffer consists of two cache lines and is distributed across the 21071-BA chips.

DMA Write Buffer: The DMA write buffer stores four cache lines of PCI memory write data. Each entry is unloaded after the necessary cache coherency checks have been performed.

3.1.3 21071-DA Introduction

The 21071-DA chip functions as the bridge between the PCI, and the CPU and its Bcache and memory. (See Figure 3–3.) The chip interface protocol is compliant with the PCI local bus. With the exception of a few pipeline registers and the parity tree, all the data path functions required to support the PCI reside in the chip.

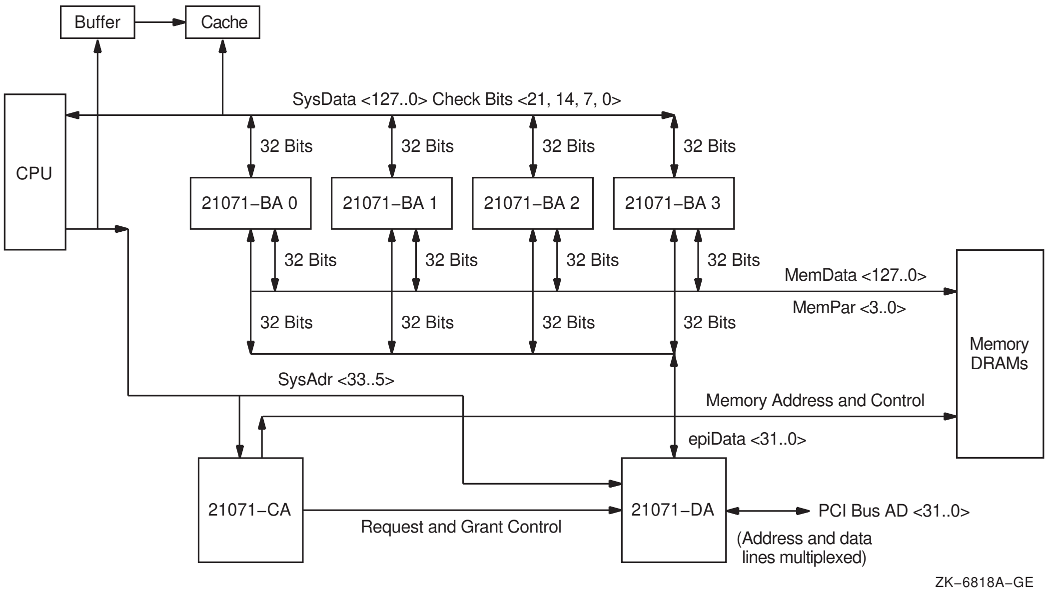
The 21071-DA provides all controls and interfaces to the PCI and sysBus and contains the following components and functions:

- sysBus interface state machine
- sysBus address decoder and translator
- epiBus arbitration and control
- PCI interface, state machines, and parity generation
- PCI address decoder and translator

The following list summarizes the major features of the 21071-DA:

- Scatter gather mapping from the 32-bit PCI address to the 34-bit physical address, with an onchip, 8-entry translation lookaside buffer (TLB) for fast address translations. To reduce cost, the scatter gather tables are stored in

Figure 3–3 Basic I/O Subsystem Address and Data Paths



memory and are automatically read by the 21071-DA when a translation misses in the TLB.

- Supports a maximum PCI burst length of 16 longwords on PCI memory reads and writes.
- Supports two types of addressing regions on CPU-initiated transactions to PCI space.
  - Sparse space for accesses with byte and word granularities, and a maximum burst length of two.
  - Dense space for burst lengths from one to eight writes and a burst length of two on reads. This region can be used for memory-like structures, such as frame buffers, which require high bandwidth accesses.
  - Stores address information for the DMA write buffer; controls the loading of the DMA write buffer and I/O read buffer.
  - Stores address information for the I/O write buffer and controls the unloading of the I/O write buffer and DMA read buffer.

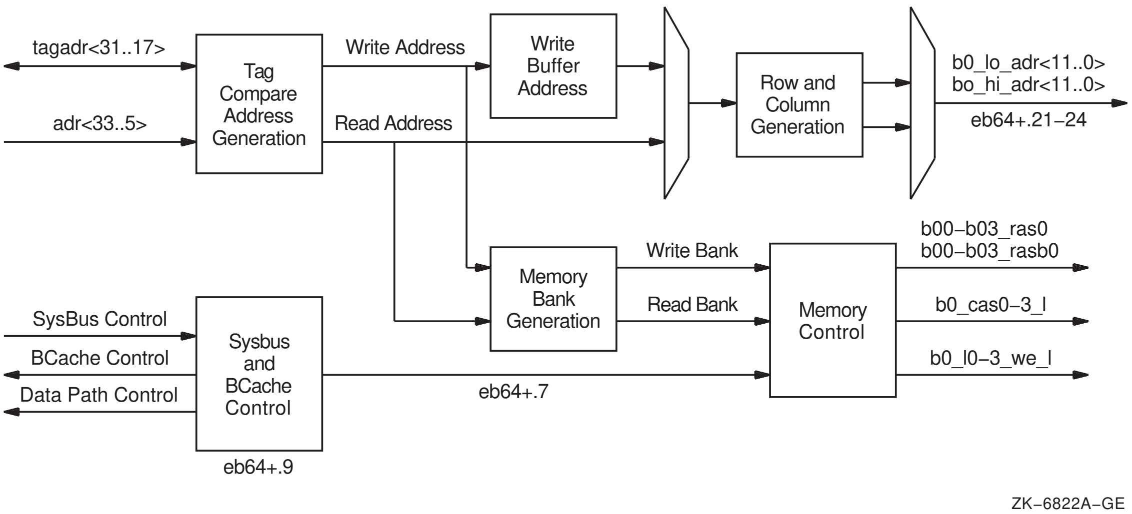
Note

The 21071-DA is not a PCI peripheral; it is a bridge between the PCI peripherals and the CPU/system memory. The chip implements functions of a host bridge that are not sufficient to interface the chip as a PCI peripheral component.

3.2 21071-CA Functional Overview

The 21071-CA (*eb64+.7*) provides a second-level cache and memory control functions. It also controls the cache and memory data path located on the 21071-BA. Figure 3–4 shows a block diagram of the 21071-CA.

Figure 3–4 21071-CA Block Diagram



3.2.1 sysBus Interface

The CPU, 21071-DA (*eb64+.29*), cache, and 21071-CA communicate with each other through the sysBus. The sysBus is essentially the processor pinbus with additional signals for DMA transaction control, arbitration, and cache control.

3.2.1.1 sysBus Arbitration

The 21071-CA arbitrates between the CPU and 21071-DA, which requests use of the sysBus and the Bcache when they have a transaction to perform. The CPU has default ownership of the sysBus so that it can access the Bcache whenever the 21071-DA is not requesting the bus.

3.2.1.2 Bcache Control

The Bcache controller provides control for the secondary cache on CPU-initiated memory read and write transactions that miss, and on all CPU-initiated memory LDx_L and STx_C transactions (hits and misses).

On DMA-initiated transactions, the Bcache controller provides control for probing the cache and extracting or invalidating the cache line when required. The 21071-CA supports a write-back cache.

Figure 3–5 shows the implementation of a cache subsystem with a 2-MB cache. Note that the 21071-CA supports a 128-bit secondary cache interface.

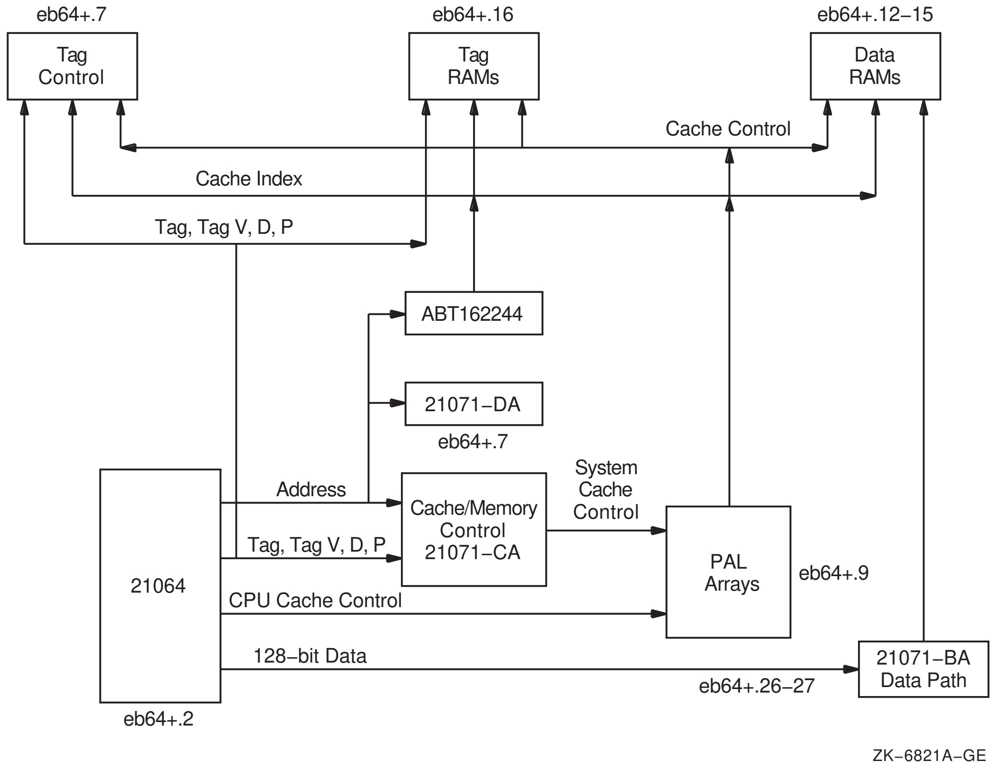
3.2.1.3 sysBus Controller

The sysBus controller consists of a sequencer that receives CPU and DMA command fields for decode, results from the sysBus arbiter logic, and status from the memory controller logic. The sequencer supplies states that are used to generate Bcache control, read requests to the memory controller, load data from the sysBus into the read and merge and write buffers, and acknowledge cycles to the CPU and 21071-DA. The sysBus controller supports wrapping on the sysBus.

3.2.1.4 Address Decoding

The 21071-CA sysBus interface logic decodes the sysBus address for both CPU and DMA requests to determine the action to take. It supports cacheable and noncacheable memory accesses, as well as accesses to its CSR space. (See Chapter 4 for address space mapping.)

Figure 3–5 Cache Subsystem for a 2-MB Cache



3.2.1.5 Error Handling

During CPU and DMA transactions, the 21071-CA detects the following errors:

- Bcache tag address parity error
- Bcache tag control parity error
- Nonexistent memory error

When one or more errors are detected on a transaction, the 21071-CA signals the errors to the CPU or the 21071-DA at the end of the transaction by acknowledging **hard error** on the **cack<2..0>** or **iocack<1..0>** field. The current **sysadr<33..5>** is logged in the error address register and error status is logged in the error and diagnostics status register. These CSRs are locked until the CPU clears all the error status bits by writing the CSR.

If errors occur on a transaction while the error address and status are locked, the transaction is acknowledged with **hard error** on the **cack<2..0>** or **iocack<1..0>** command fields. The **LostErr** bit in the error and diagnostics status register is set, and the lost error address and status are not recorded.

The **hard error** overrides STx_C fail. The lock bit is unpredictable after LDx_L transactions with errors.

3.2.2 Memory Controller

This section summarizes memory organization and memory controller features.

3.2.2.1 Memory Organization

The 21071-CA supports up to:

- Eight banksets of DRAM (banksets0..7), where one bankset = four SIMMs
- One bankset (bankset8) of VRAM

Each bankset can be made up of one or two banks. A bank of memory refers to one width of DRAMs, implemented with SIMMs. The SIMM implementation requires more than one SIMM to form one memory bank. For example, four 33-bit SIMMs are required to form the 128-bit bank width. On the EB64+ the 21071-CA supports 16 MB to 512 MB of DRAM, and an additional 1 MB to 8 MB of dual port random access memory (VRAM).

Memory is accessed at 128 bits. Because the EB64+ uses longword parity, 132 bits are required.

3.2.2.2 Memory Address Generation

The programmable base address of a bankset must be aligned to the natural size boundary. For example, an 8 MB bankset must start on an 8 MB boundary. The hardware allows for holes in memory with badly programmed addresses.

Each bankset has a programmable base address and size. The incoming physical address is compared in parallel with the memory ranges of all banksets present. Depending on the size of the bankset, a variable number of physical address and base address bits from the CSR are compared.

3.2.2.3 Memory Page Mode Support

The 21071-CA supports page mode optimization on the memory banks within a transaction. Page mode between transactions is supported on DMA read burst transactions and on memory write transactions.

3.2.2.4 Read Latency Minimization

To minimize the read latency seen by devices on the sysBus , the memory controller performs certain optimizations in the way transactions are selected. In general, because writes can go into a deep write buffer, reads are given priority over writes (that is, to the extent that in some cases the memory controller waits for a read to execute even if there are writes queued in the write buffer).

3.2.2.5 Transaction Scheduler

The memory interface does memory refresh, cache-line reads and writes, and shift register loads to VRAM bankset8. The memory controller has a scheduler that prioritizes transactions and selects one to be serviced. If the selected transaction is waiting for RAS precharge, and another higher priority transaction is initiated, the scheduler deselects the previously chosen transaction and selects the higher priority transaction.

3.2.2.6 Programmable Memory Timing

The memory control state machine sequences through all memory transactions. On memory read and write transactions, it communicates with the 21071-BAs so that data may be latched from the memData bus, or driven onto the memData bus respectively.

The memory control state machine is actually two state machines (master, and read and write). The master state machine performs the RAS, CAS assertions, and controls when the other state machine starts. The read and write state machine performs the sequencing for generating the **memcmd** to read or write memory data. The read and write state machine is started by the master and sequences independently.

3.2.2.7 Presence Detect Logic

The 21071-CA supports loading the status of 32 presence pins into a register after reset. The 32 bits are loaded into a shift register on the module and then shifted one bit at a time into the 21071-CA.

3.2.2.8 Frame Buffer and Video Support Logic

As shown in Figure 3–6, all address and control signals (except **vframel** and **vrefreshl**) are buffered in the DRAM buffers (*eb64+.18-.19*) before being applied to the external frame buffer connectors. Memory data and its parity are applied to the connectors directly from the DRAMs. Parity may also be disabled on the frame buffer interface.

Figure 3–6 Frame Buffer Data and Signal Layout

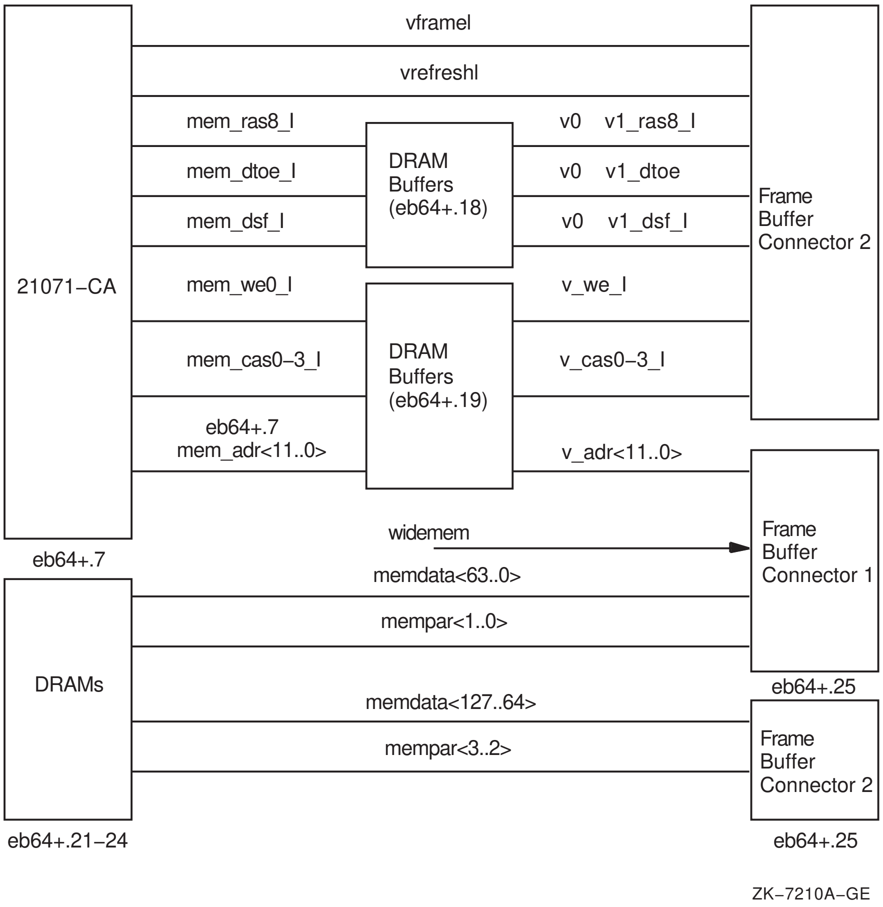


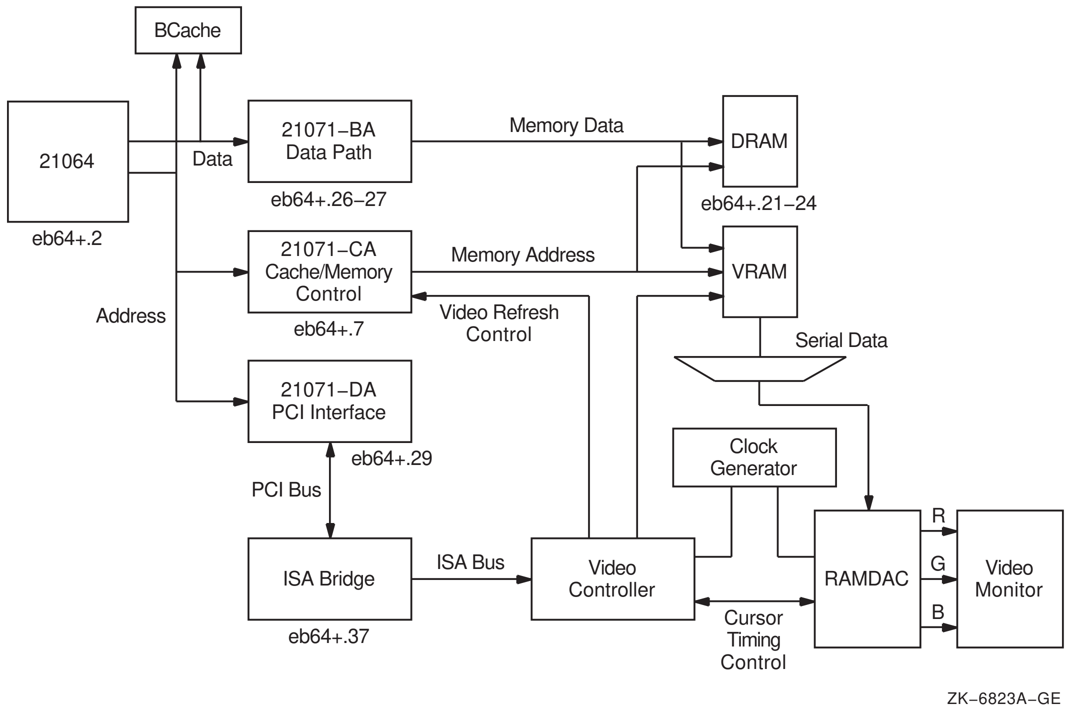
Table 3–1 describes the frame buffer connector signals.

Table 3–1 Frame Buffer Connector Signal Descriptions

Signal Name	Type	Description
Frame Buffer Connector 1		
Vdd	Input	+5 Vdc
gnd	—-	Ground
memdata<63..0>	Input	Low-order memory data sourced directly from the DRAM SIMMs.
mempar1	Input	Longword parity for memdata<63..32>.
mempar0	Input	Longword parity for memdata<31..0>.
v_adr<11..0>	Input	Buffered address from the 21071-CA.
widemem	Input	Connected to +5 Vdc, 128-bit mode selected
Frame Buffer Connector 2		
Vdd	Input	+5 Vdc
gnd	—-	Ground
memdata<127..64>	Input	High-order memory data sourced directly from the DRAM SIMMs.
mempar3	Input	Longword parity for memdata<127..96>.
mempar2	Input	Longword parity for memdata<95..64>.
v0-v1_dtoe_l	Input	Buffered data transfer enable.
v0-v1_ras8_l	Input	Buffered row address strobe.
v0-v1_dsf	Input	Buffered special function. With **dtoe** deasserted, **dsf** determines whether a full (**dsf** = 0) or split (**dsf** = 1) VRAM shift register load will be executed.
v0-v1_we_l	Input	Buffered VRAM write enable function.
v_cas0-3_l	Input	Buffered column address strobe.
vframel	Output	Load video display pointer and load video RAM shift register to CPU.
vrefresh	Output	Increment video display pointer and load video RAM shift register.

The 21071-CA provides the logic and control to perform full and split serial register loads to the VRAM bankset8. Figure 3–7 shows a basic video subsystem with frame buffer.

Figure 3–7 Video Subsystem and Frame Buffer



The 21071-CA performs CPU and DMA accesses to the random port of bankset8 if the address matches the bank base addressed. In addition, the 21071-CA performs serial register loads in response to **vframel** or **vrefreshl** pin assertions.

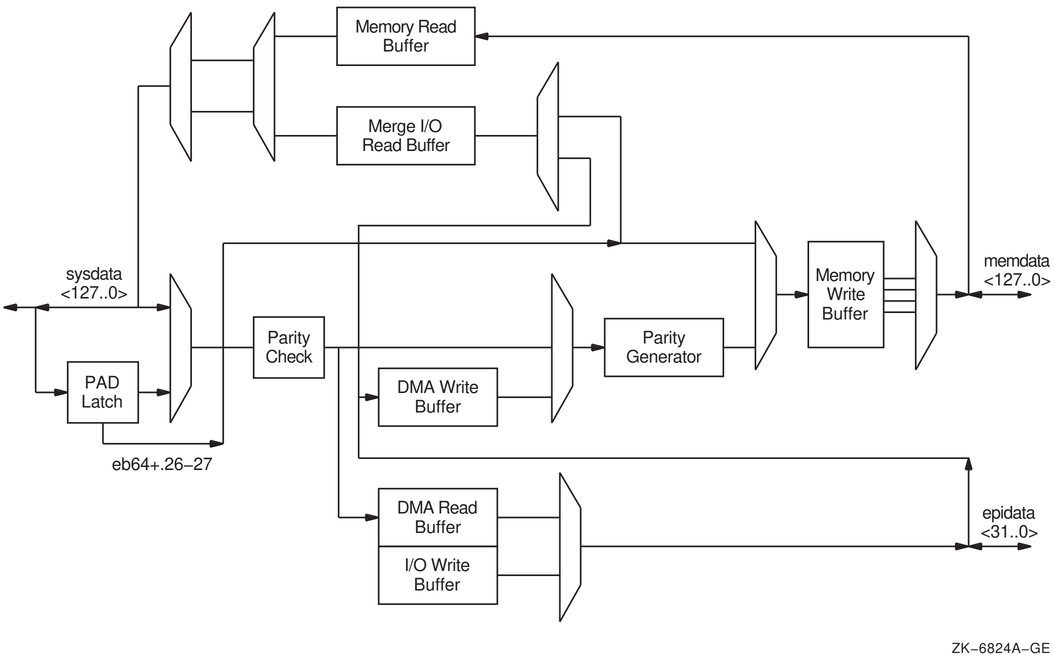
When the 21071-CA performs a serial register load, the VRAM latches the data in the accessed row into its serial register. Other external logic then shifts out the serial register through the VRAM serial port. The 21071-CA does not provide any support for unloading the serial port of the VRAM.

In a full serial register load, the entire RAM row specified by the row address is latched into the serial register. In a split serial register load, only half the row is latched into the serial register. The MSB of the column address specifies whether the upper or lower half of the row is to be latched.

3.3 21071-BA Functional Overview

This section describes the data bus configurations and provides a functional overview of the 21071-BA. Figure 3–8 shows a block diagram of the 21071-BA.

Figure 3–8 DECchip 21071-BA Block Diagram



3.3.1 sysData Bus

With the 21072-AA configuration, only the lower 32 bits of the sysData bus are used:

- 21071-BA #0 connects to **data<31..0>** (longword 0)
- 21071-BA #1 connects to **data<63..32>** (longword 1)
- 21071-BA #2 connects to **data<63..32>** (longword 2)

- 21071-BA #3 connects to **data<127..96>** (longword 3)

3.3.2 memData Bus

With a memData bus of 128 bits, four 21071-BAs are required, that is:

- 21071-BA #0 connects to longword 0 (**memdata<31..0>**)
- 21071-BA #1 connects to longword 1 (**memdata<63..32>**)
- 21071-BA #2 connects to longword 2 (**memdata<95..64>**)
- 21071-BA #3 connects to longword 3 (**memdata<127..64>**)

3.3.3 epiData Bus

Each 21071-BA has 32 epiData bus pins. The epiData pins of the 21071-BAs are tied together to form a 32-bit epiData bus.

3.3.4 Memory Read Buffer

The memory read buffer stores data that is read from memory before it is returned to the CPU on the sysBus, or to DMA in the DMA read buffer.

Each 21071-BA stores four longwords of data and corresponding parity bits in the memory read buffer. The 4-chip system contains an additional eight longwords of storage; however, the extra storage is not usable.

3.3.5 I/O Read Buffer and Merge Buffer

On CPU-initiated memory transactions; the buffer performs the merge buffer functions. On CPU-initiated I/O reads addressed to, or through the 21071-DA, the buffer acts as the I/O read buffer. Loading of data into the buffer is controlled by the 21071-CA and 21071-DA.

Each 21071-BA contains four longwords of data and corresponding parity bits. The parity bits are meaningful only for merge data. The parity bits are UNPREDICTABLE for I/O read data.

3.3.6 I/O Write Buffer and DMA Read Buffer

This buffer can store up to four entries of data, where each entry has four longwords for each 21071-BA. Data from this buffer is sent out on the epiData bus. In the 21072-AA implementation, two entries of this buffer are allocated for I/O write data storage, and two entries are allocated for DMA read data storage. The four 21071-BA system uses only two of the four longwords of each entry. However, the extra storage is not accessible. Loading of each entry can be controlled separately, allowing maximum flexibility in allocating the buffer entries to the 21071-DA. Loading of the buffer is handled by the 21071-CA, with the address provided by the 21071-DA on **iolinesel<1:0>**. The 21071-DA controls unloading of the buffer.

3.3.7 DMA Write Buffer

The DMA write buffer has four entries. Each entry contains four longwords for each 21071-BA and corresponding byte masks. However, only half the storage for each entry is used. The extra storage is not accessible.

The DMA write buffer is loaded by the 21071-DA and is unloaded by the 21071-CA during a DMA write transaction on the sysBus. The byte masks are used to merge the valid bytes of data written in the DMA write buffer with the background data from the cache line, which may be obtained from the Bcache or memory.

3.3.8 Memory Write Buffer

The memory write buffer has four entries. Each entry contains four longwords of data for each 21071-BA, and corresponding parity bits. The memory write buffer is loaded by the 21071-CA sysBus interface, and unloaded by the 21071-CA memory controller.

3.3.9 Error Checking

The 21071-BA performs error checking on DMA transactions. When memory or Bcache data is read because of a DMA transaction (DMA read or a DMA write masked), the data is checked for parity errors.

If the data contains a parity error, the 21071-DA is notified (for a DMA read), or bad parity is written back into memory (for a DMA write).

In case of a DMA write masked, parity is generated for the merged data going into the memory write buffer.

3.3.10 epiBus Data Path

The epiBus may be used to load the I/O read buffer or the DMA write buffer. In addition to write data, byte masks are stored in the DMA write buffer.

The epiBus may also be used to unload the DMA read buffer (which also serves as the I/O write buffer).

3.3.11 sysBus Output Selectors

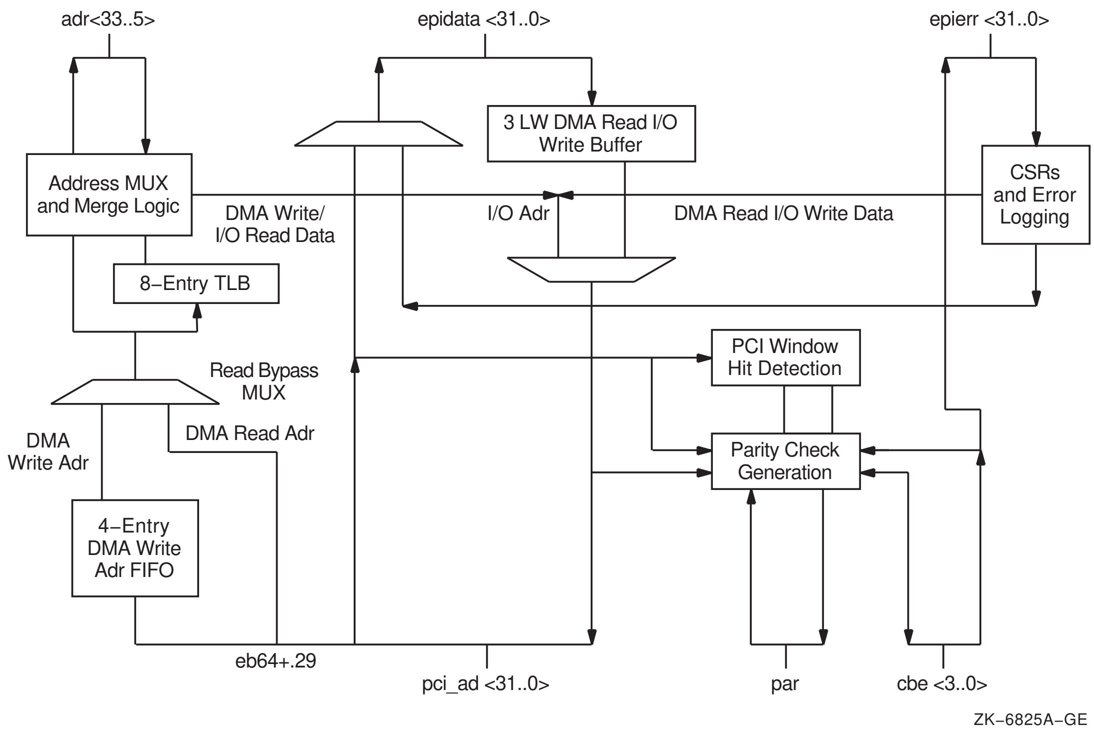
Two levels of multiplexers select the output for the sysData bus. The first level selects the source for each longword of data and parity bits. The second level selects the two longwords to be driven on the sysData bus.

3.4 21071-DA Functional Overview

The 21071-DA is a bridge between the PCI local bus and the 21064 microprocessor and its Bcache and memory. The 21071-DA contains all control functions of the bridge and some data path functions. Other data path functions reside in the 21071-BA.

The 21071-DA can be divided into two major sections: the sysBus (processor, memory) interface and the PCI interface. The following sections describe the sysBus and PCI interface features. Figure 3–9 shows a block diagram of the 21071-DA.

Figure 3–9 DECchip 21071-DA Block Diagram



3.4.1 sysBus Interface

The sysBus interface includes the sysBus control state machine, the address decode for CPU-initiated transactions, buffering for CPU-initiated transactions, and the 21071-DA control and status registers.

3.4.1.1 Address Decode

The 21071-DA provides logic for translating and extending between the 21064 or 21064A 34-bit physical address space, and the 32-bit PCI address space. The address decode in the 21071-DA uses the address mapping and translation scheme described in Chapter 4 to generate PCI addresses on CPU-initiated transactions. All systems using the 21071-DA are required to follow this address mapping scheme.

3.4.1.2 I/O Write Transaction Buffering

The 21071-DA supports *write-and-run* I/O write transactions (see Intel PCI Specification) and implements a one-entry write buffer. The address and control mechanism is in the 21071-DA; the corresponding data is stored in the 21071-BA.

3.4.1.3 I/O Read Data Buffering

The 21071-DA provides data buffering for one I/O read transaction initiated by the CPU. The I/O read buffer resides in the 21071-BA, but is controlled by the 21071-DA. The I/O read buffer is a temporary holding buffer, and is invalidated at the end of each I/O read transaction.

3.4.1.4 Wrapping Mode

The 21071-DA supports wrapped mode only on transactions initiated by the 21064 or 21064A. The requested quadword is the only one that is returned on I/O read transactions. To function correctly, the CPU must be configured in wrap mode.

3.4.2 PCI Interface

The PCI interface of the 21071-DA is a fully compliant PCI host bridge. It acts as a master on the PCI on CPU-initiated transactions, and is a target on memory space transactions initiated by PCI masters.

3.4.2.1 DMA Address Translation

The PCI interface supports direct and scatter gather mapping from the 32-bit PCI address to the 34-bit physical address space. It provides two windows that can be mapped to regions within the PCI address space. Each address region can be independently programmed to be direct mapped or scatter gather mapped.

3.4.2.2 DMA Write Buffer

The PCI interface has a write buffer for buffering DMA write data. The DMA write buffer is made up of four entries. Each entry contains the cache line address, eight longwords of data, the byte enables corresponding to each longword, and a valid bit for the entry. The untranslated PCI address is stored in the DMA write buffer. Address translation is performed when the particular entry is unloaded from the DMA write buffer. The address and valid bits are stored in the 21071-DA, and corresponding data and byte enables are stored in the 21071-BA.

3.4.2.3 DMA Read Buffer

The 21071-DA controls the DMA read buffer located in the 21071-BA. The buffer stores up to 16 longwords of data organized as two cache lines. A valid bit is implemented with each longword. Data received from the sysBus (memory or cache) is loaded into the DMA read buffer by the sysBus interface, and the corresponding valid bit is set. The data is unloaded by the PCI interface.

3.4.2.4 PCI Burst Length and Prefetching

The PCI interface supports a maximum burst length of 16 longwords on PCI write transactions directed toward main memory. If the PCI write transaction starts on an even cache line boundary with (**pci_ad<5>** = 0 and **pci_ad<4..2>** = 0), a full burst of 16 longwords is supported. The transaction will be terminated using a PCI disconnect after the sixteenth longword has been received. In all other cases, the actual burst will be less than 16 longwords.

On DMA read transactions, a maximum burst length of eight longwords is supported if DMA prefetching is not enabled in the 21071-DA and a PCI read multiple command was not used by the requesting device. A maximum burst length of 16 longwords is supported if DMA prefetching is enabled in the 21071-DA or a PCI read multiple command was used by the requesting device.

On CPU-initiated read transactions, when the 21071-DA is a master on the PCI, a maximum burst length of two is supported.

On CPU-initiated write transactions, when the 21071-DA is a master on the PCI, a maximum burst length of two is supported in sparse memory and I/O spaces, and a maximum burst length of eight is supported in dense memory space.

3.4.2.5 PCI Burst Order

pci_ad<1..0> of the PCI address are used to specify the burst ordering requested by the master during memory transactions. When the 21071-DA is a master of the PCI it will always indicate a linear incrementing burst order (**pci_ad<1..0>** = 0) on read and write transactions.

On DMA transactions, the 21071-DA supports burst transfers only when a linear incrementing burst order is specified. If the master specifies a burst order other than that (**pci_ad<1..0>** is nonzero), the PCI interface disconnects the transaction after one data transfer.

3.4.2.6 PCI Parity Support

The 21071-DA complies with the specification that all PCI devices generate parity across **pci_ad<31..0>** (data and address lines) and **cbe#<3..0>** (command/byte enables). When it is master of the PCI, it also checks the incoming parity on I/O reads, interrupt vector reads, and configuration reads during data phases. When the 21071-DA is a target on the PCI, it checks parity during the address phase, and during data phases on memory write transactions.

3.4.2.7 PCI Exclusive Access

The 21071-DA supports the PCI Exclusive Access protocol using the **lock_l** signal. A locked transaction to main memory on the PCI causes the PCI interface to lock out all nonexclusive main memory accesses initiated by PCI masters. This is done by disconnecting the PCI transaction without completing any data transfers. Until the lock is cleared on the PCI, only the PCI master that locked main memory is allowed to complete transactions to main memory. (See the *PCI Local Bus Specification.*)

On the sysBus side, the PCI lock causes the system lock flag to be cleared by using the **ioclrlock** command encoded on the **iocmd<2..0>**. The system lock flag is held cleared until all locked DMA reads and writes to memory have been completed on the sysBus and the lock is cleared on the PCI.

As a master on the PCI, the 21071-DA does not initiate locked transactions.

3.4.2.8 PCI Bus Parking

When no devices are requesting bus mastership, it is recommended that the system arbiter grant default bus ownership to the 21071-DA by asserting its **iogrant** signal. This will reduce the latency for CPU-initiated transfers to the PCI when the bus is idle. Granting the PCI to a device when no requests are pending is referred to as *bus parking* in the *PCI Local Bus Specification*. If the 21071-DA is granted the bus when it is not requesting the PCI, it will drive the **pci_ad<31..0>**, **cbe_l<3..0>**, and the **par** signals.

The 21071-DA also supports PCI bus parking during reset. If the **iogrant** signal is asserted by the PCI arbiter (**req_l** is always tristated by the 21071-DA during reset), the 21071-DA will drive **pci_ad<31..0>**, **cbe<3..0>** and (one clock cycle later) **par**. When **iogrant** is deasserted, the 21071-DA tristates these signals.

3.4.2.9 PCI Retry Timeout

The 21071-DA implements a timeout mechanism to terminate CPU-initiated transactions that do not complete on the PCI because of too many disconnects or retries. When it initiates a CPU transaction on the PCI, the 21071-DA counts the number of times it is retried or disconnected. If the number exceeds 2^{24} it flags an error to the CPU and aborts the transaction.

3.4.2.10 PCI Master Timeout

The PCI protocol specifies a mechanism to limit the duration of a master's burst sequence. The mechanism requires a PCI master to implement a latency timer that counts the number of cycles since the assertion of **frame#**. If the master latency timer has expired and the master's grant has been taken away, the master is required to surrender the bus.

This mechanism is intended to prevent masters from holding bus ownership for extended periods of time and trades off high throughput for low latency. The 21071-DA implements a programmable master latency timer.

3.4.2.11 Address Stepping in Configuration Cycles

To provide flexibility and reduce design complexity when using the address stepping feature, the 21071-DA performs address stepping on configuration reads and write transactions. For these transactions, the 21071-DA will drive the PCI bus for two clock cycles during the address phase for the **idsel#** pins of all PCI devices to reach a valid logic level. The 21071-DA does not perform address or data stepping in any other case.

3.4.2.12 Data Coherency

There are generally two agents in the system where data transfer actions must be synchronized: CPU and a remote PCI device. The 21071-DA maintains data coherency and synchronization between the agents using the following mechanisms:

- The 21071-DA preserves strict ordering of DMA writes initiated on the PCI.
- DMA reads can bypass writes that are not to the same address (double cache line). Strict ordering is maintained between reads and writes to the same address.

- I/O transfers from the CPU to the PCI or to 21071-DA CSRs are performed in order. This policy guarantees a coherent view of PCI I/O space from the CPU.
- The 21071-DA flushes DMA write data to memory before acknowledging a barrier command from the CPU. Because explicit ordering commands are absent on the PCI, the MB instruction is used to order CPU and DMA accesses.
- The 21071-DA also flushes the I/O write buffer to the PCI before acknowledging a barrier command. This preserves the order between CPU I/O accesses and CPU memory accesses.
- The 21071-DA clears the system lock flag on PCI exclusive reads and writes to system memory.

3.4.2.13 Deadlock Resolution

There are two major buses allocated for use during data transfers: the sysBus and PCI. Some data transfers require the use of both of these buses to complete. In particular, CPU I/O transfers to or from the PCI require ownership of the sysBus followed by ownership of the PCI. Similarly, PCI DMA transfers to or from the memory subsystem require ownership of the PCI followed by ownership of the sysBus.

Because of the nonpended nature of these buses, during read transfers (I/O or DMA), both buses must be held at the same time for the transfer to complete. Generally, because the 21071-DA features write-and-run style buffering, only one bus must be held at a time during write transfers (I/O or DMA). However, when a write buffer is full, both buses must be held at the same time so that some data from the write buffer can be flushed before new data is accepted.

The 21071-DA resolves deadlock by forcing the CPU to relinquish ownership of the sysBus, thereby giving priority to the PCI agent. By giving priority to the PCI agent, the 21071-DA provides the system designer more flexibility in choice of PCI devices. That is, it supports devices that use the PCI disconnect in handling deadlock situations. The 21071-DA forces the CPU to relinquish the sysBus by using a preempt request while arbitrating for the sysBus.

3.4.2.14 Guaranteed Access Time Mode Support

The Intel 82378ZB or 82378IB ISA bridge provides three sideband signals (**flushreq_l**, **memreq_l**, and **sat_memack_l**) to provide mechanisms for flushing system write buffers and to allow a guaranteed access time of 2.1 μs to a master on the ISA bus. The **flushreq_l** and **memreq_l** signals are outputs from the bridge; **sat_memack_l** is an input to the bridge.

The 21071-DA provides its own mechanism for deadlock prevention by preempting CPU transactions to allow DMA transactions to complete. It does not need to support the deadlock prevention mechanism of the bridge, but does implement the **flushreq_l** protocol.

3.5 Interrupts

The 21071-DA interrupts the CPU using the **int_hw0** signal when it has errors to report. The 21071-DA does not distinguish between hard and soft errors when asserting the interrupt signal.

The 21071-DA does not provide an interval timer interrupt. This functionality is should be provided to the CPU by some other device in the system. In addition, interrupts from other PCI devices or from a PCI interrupt controller must be sent directly to the CPU without intervention.

The 21071-DA participates in the interrupt acknowledge process. It responds to CPU read block commands directly to the interrupt acknowledge address space, which triggers the 21071-DA to perform an Interrupt Acknowledge transaction on the PCI. The interrupt vector returned on the PCI is returned to the CPU through the sysBus by the 21071-DA.

3.6 Error Handling

The setting of CSR error bits and the locking of the associated error address register, assume that another error that locks that error address register is not set. If another error occurs, only the lost error bit is set, and **int_hw0** is asserted to interrupt the processor. The **int_hw0** signal is held asserted as long as the corresponding error bit is set.

The PCI error address register (PEAR) logs addresses sent out or received on the PCI. The sysBus error address register (SEAR) logs the address that was sent out or received on the sysBus.

The 21071-DA returns **hard_error** in the **iocmd<2..0>** field on I/O read transactions with errors. No interrupt is asserted in this case, because the 21064 has been notified that the read had an error. There is no case where the 21071-DA asserts **soft_error** on an I/O read transaction.

I/O writes are acknowledged with OK (100 on **cack<2..0>**) because of the *write and run* feature for I/O writes in the 21071-DA. The transaction is acknowledged on the sysBus before it is initiated on the PCI. Interrupt **int** is asserted to notify the 21064 that an error has occurred on the PCI during the I/O write.

All DMA transaction errors are flagged by interrupting the processor with **int** when the error occurs.

3.7 Clock Subsystem

The system clocks can be divided into three areas:

- Input clocks required by the 21064 or 21064A CPU
- CPU clock distribution to the system logic
- Miscellaneous oscillators and clocks required for the peripheral interfaces and functions

The 21064 or 21064A CPU clock input is provided by a Triquint phase lock loop (PLL) clock oscillator

3.7.1 Triquint PLL Clock Oscillator

As shown in Figure 3–10, the Triquint PLL clock oscillator is composed of two stages: oscillator and frequency multiplier, with the multiplier producing the CPU clock input (**clkin_h** and (**clkin_h**). Depending on the PLL clock part selected, the oscillator stages can be programmed to operate from 25 MHz to 50 MHz, with the multiplier stages producing output frequencies of 250 MHz to 700 MHz. (See Table 3–2.)

Figure 3–10 Triquint Clock Generator

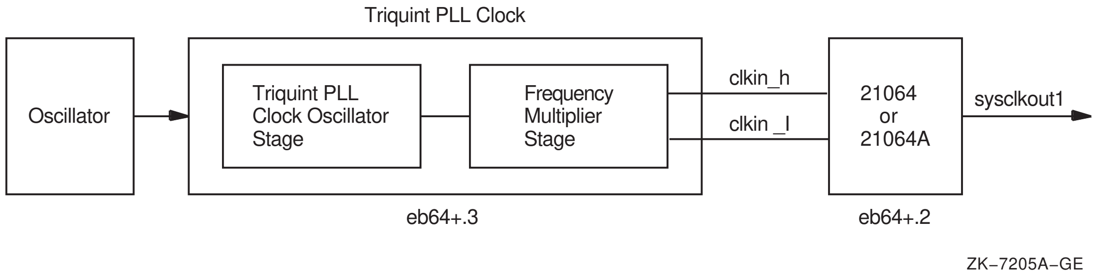


Table 3–2 Triquint Operating Frequencies

PLL Part	Operating Frequencies
TQ2059	Oscillator running from 25 MHz to 35 MHz, X10 multiplier output from 250 MHz to 350 MHz

(continued on next page)

Table 3–2 (Cont.) Triquint Operating Frequencies

PLL Part	Operating Frequencies
TQ2060	Oscillator running from 35 MHz to 50 MHz, X10 multiplier output from 350 MHz to 500 MHz
TQ2061	Oscillator running from 25 MHz to 35 MHz, X20 multiplier output from 500 MHz to 700 MHz

Assume a Triquint 500 MHz differential clock is supplied to the CPU. The CPU divides the clock by 2, generating its internal clock operating at 250 MHz.

The internal clock is further divided by the CPU to generate the system clock (**sysclkout1**). The system clock divisor can be programmed over a range from 2 to 17 as specified in Table 3–3.

Table 3–3 Clock Divisor Range

J26 IQR2	J25 IQR1	J24 IQR0	Divisor
0	0	0	2
0	0	1	3
0	1	0	4
0	1	1	5
1	0	0	6
1	0	1	7
1	1	0	8
1	1	1	9
0	0	0	10
0	0	1	11
0	1	0	12
0	1	1	13
1	0	0	14
1	0	1	15
1	1	0	16
1	1	1	17

Note

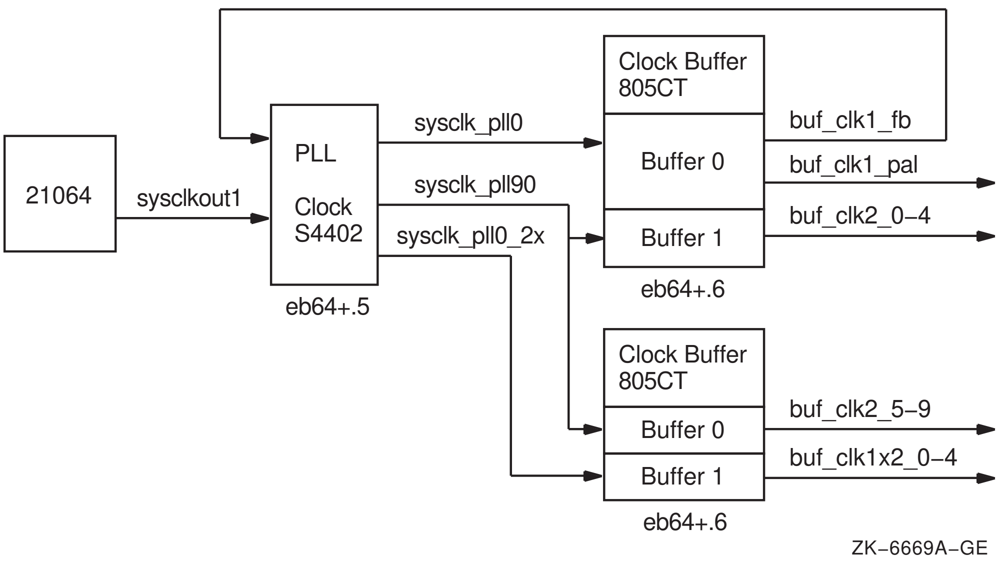
For other clocks generated by the CPU and not used on the board, refer to the *DECchip 21064 Microprocessor Hardware Reference Manual (EC–N0079–72)*.

3.7.2 System Clock Distribution

Figure 3–11 shows the primary clock distribution network for both clock oscillators. The major module clock is **sysclkout1**. It is sent to a phase lock loop (PLL) clock driver (S4402). The driver generates three copies of **sysclkout1**:

- **sysclk_pll0**—has the same frequency and is in phase with **sysclkout1**
- **sysclk_pll0_x2**—has twice the frequency and is in phase with **sysclkout1**
- **sysclk_pll90**—has the same frequency as **sysclkout1** but is 90 degrees out of phase

Figure 3–11 Primary Clock Distribution Network



The clock driver outputs are applied to a pair of clock buffers (805CT). Each clock buffer is organized as two, 1-to-5 buffers. Each of these buffers is capable of producing 10 copies of its input. However, not all buffer copies are used.

As shown in Figure 3–11 the clock buffer for **sysclk_pll0** produces **buf_clk1_fb** as a reference feedback clock to the PLL clock (S4202), and **buf_clk1_pal** to the Bcache PAL.

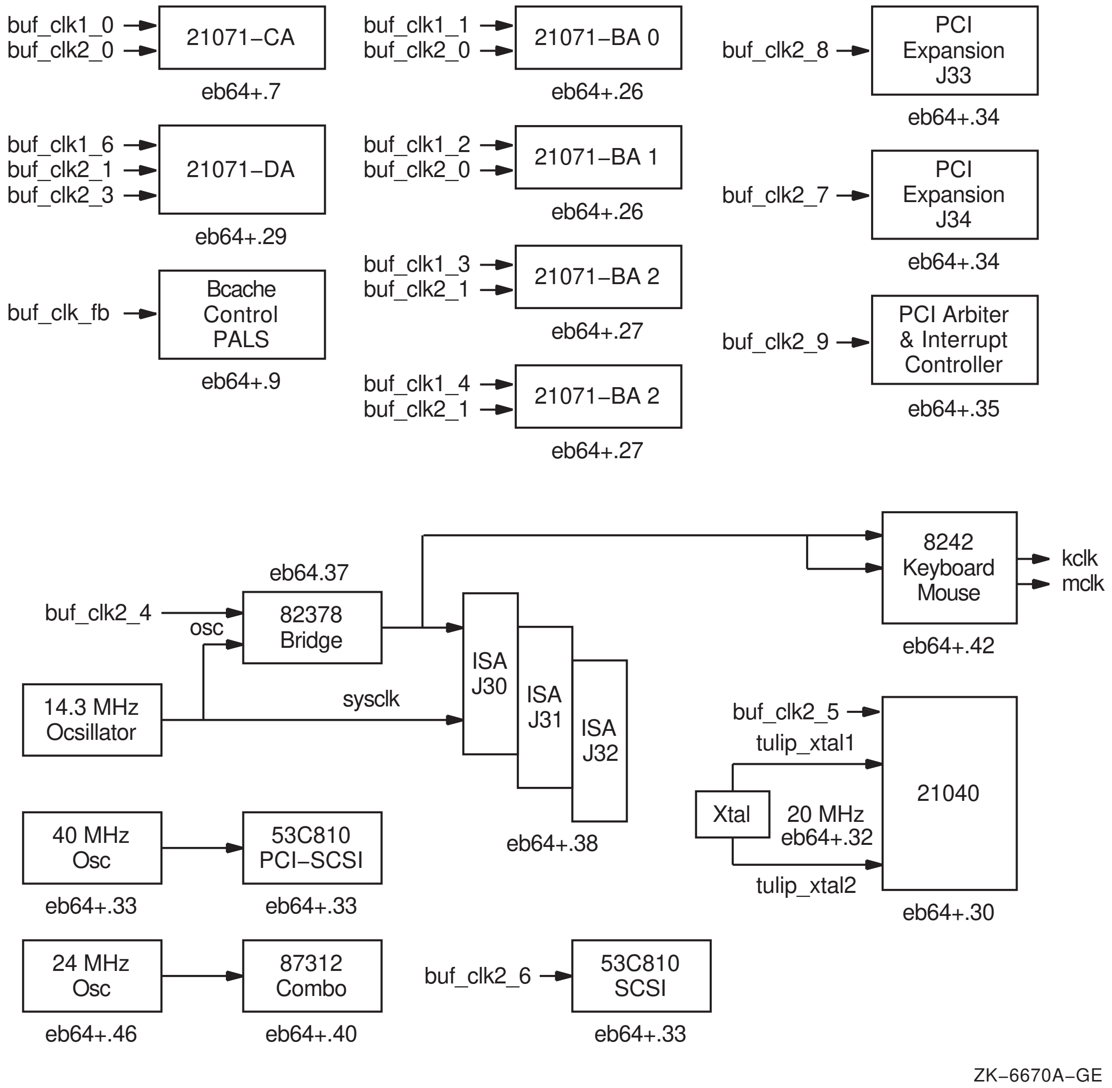
The clock buffer for **sysclk_pll0_2x** produces five copies of **buf_clkx2_x** distributed (see Figure 3–12) as follows:

- **buf_clk1x2_0** to the 21071-CA (*eb64+.7*) and the 21071-DA (*eb64+.29*)
- **buf_clk1x2_1** to the 21071-BA0 (*eb64+.26*)
- **buf_clk1x2_2** to the 21071-BA1 (*eb64+.26*)
- **buf_clk1x2_3** to the 21071-BA2 (*eb64+.27*)
- **buf_clk1x2_4** to the 21071-BA3 (*eb64+.27*)

The clock buffer for **sysclk_pll90** produces 10 copies of **buf_clk2_x** distributed as follows:

- **buf_clk2_0** to the 21071-CA (*eb64+.7*), 21071-BA0, and 21071-BA1 (*eb64+.26*)
- **buf_clk2_1** to the 21071-DA (*eb64+.29*), 21071-BA2, and 21071-BA3 (*eb64+.27*)
- **buf_clk_2** to the Bcache write enable PAL (*eb64+.9*)
- **buf_clk_3** to the 21071-DA (*eb64+.29*)
- **buf_clk2_4** to the Intel 82378ZB or 82378IB chip to synchronize the PCI to ISA bridge PCI bus transactions (*eb64+.37*)
- **buf_clk2_5** to the 21040 PCI to Ethernet chip to synchronize the Ethernet controller PCI bus transactions (*eb64+.30*)
- **buf_clk2_6** to the NCR 53C810 to synchronize the SCSI controller PCI bus transactions (*eb64+.33*)
- **buf_clk2_7** to PCI expansion connector J34 (*eb64+.34*)
- **buf_clk2_8** to PCI expansion connector J33 (*eb64+.34*)
- **buf_clk2_9** to the PCI arbiter PALs *eb64+.35*

Figure 3–12 Buffered Clock Distribution Network



In addition four oscillators provide clocks for PCI, combination chip, and Ethernet functions.

A 14.3 MHz oscillator (69.9 ns period) (*eb64+.46*) output is buffered and produces **osc**, which is routed to the PCI to ISA bridge (*eb64+.37*) and **sysclk**, which is routed to the three ISA slots (*eb64+.38*). This is the standard 14.31818 MHz ISA clock.

A 20 MHz crystal (50 ns period) (*eb64+.30*) connected to the Ethernet controller (21040) produces signals **tulip_xtal1** and **tulip_xtal2** for Ethernet packet timing.

A 24 MHz oscillator (41.7 ns period) (*eb64+.46*) produces **osc24**, which provides clocking for the National 87312 combo chip (*eb64+.40*).

A 40 MHz oscillator (25 ns period) (*eb64+.33*) produces the SCSI clock **scsi_clk**, which is used to derive all SCSI related timing.

3.8 PCI Interrupts and Arbitration

The following subsections describe the PCI interrupt and arbitration (arbiter) logic.

3.8.1 PCI Interrupt Logic

Figure 3–13 shows the EB64+ interrupt logic. The PCI and SIO interrupts are implemented in one Mach 210 interrupt controller.

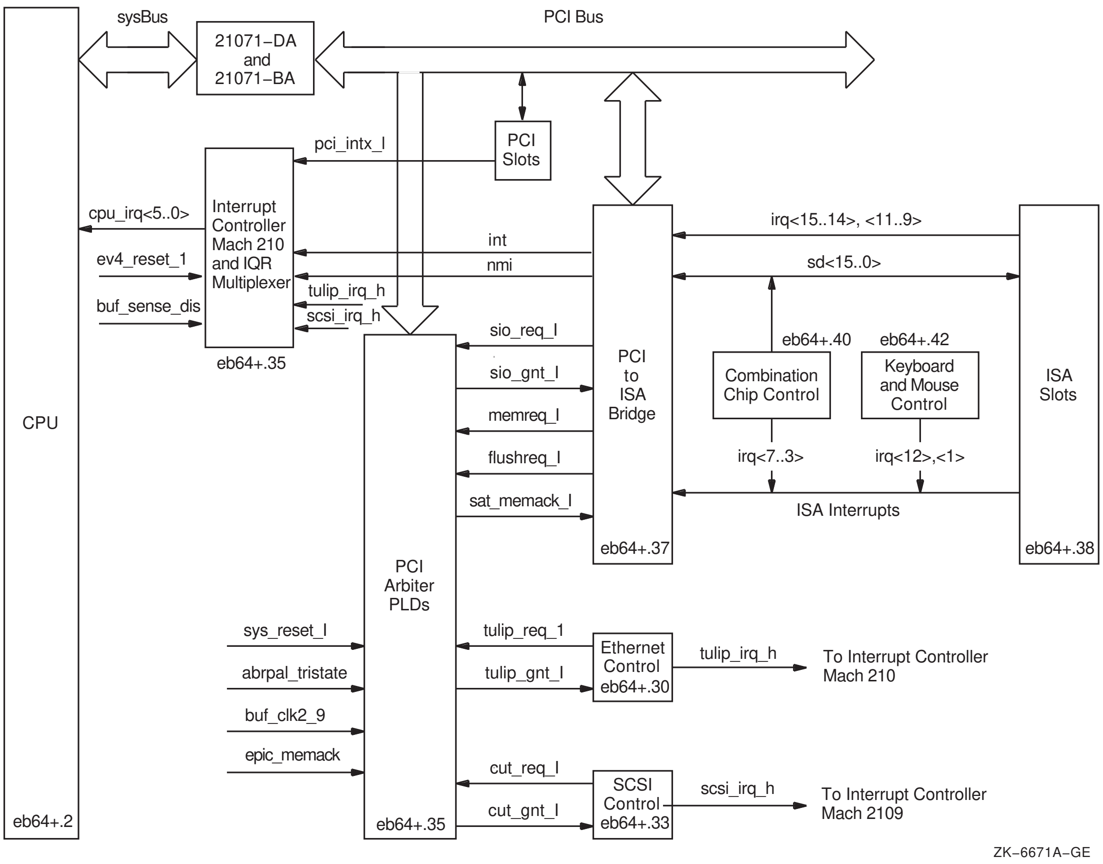
The Mach 210 acts as an 8-bit I/O slave on the ISA bus at addresses 26_{16} and 27_{16}. This is accomplished through a decode of 11 ISA address lines **sa<11>** and **sa<9..0>**. Because the NVRAM is located at addresses 800_{16} - $8FF_{16}$, **sa<11>** is required as part of the decoding.

The Mach 210 controls 11 PCI interrupts:

- Four from PCI slot 0 (PCI_INTx)
- Four from PCI slot 1 (PCI_INTx)
- One from the PCI-ISA bridge (INT)
- One from the Ethernet controller (TULIP_IRQ)
- One from the SCSI controller (SCSI_IRQ)

Each interrupt can be individually masked by setting the appropriate bit in one of the two mask registers (see Figure 3–14 and Figure 3–15). The SIO controls the ISA interrupts.

Figure 3–13 Interrupt and Arbitration Logic



An interrupt is disabled by writing a '1' to the mask register. An interrupt is enabled by writing a '0'. For example, bit **7** set in Interrupt Mask Register 0 indicates that the SCSI interrupt is disabled.

An I/O read at addresses 26 and 27 returns the state of the eleven PCI interrupts rather than the state of the masked interrupts. On reads, a '1' means that one of the interrupt sources shown in Figure 3–14 or Figure 3–15 has asserted its interrupt. The mask registers are write-only and can be updated by writing to addresses 26 or 27.

Figure 3–14 Mask Register 0—Address 26_{16}

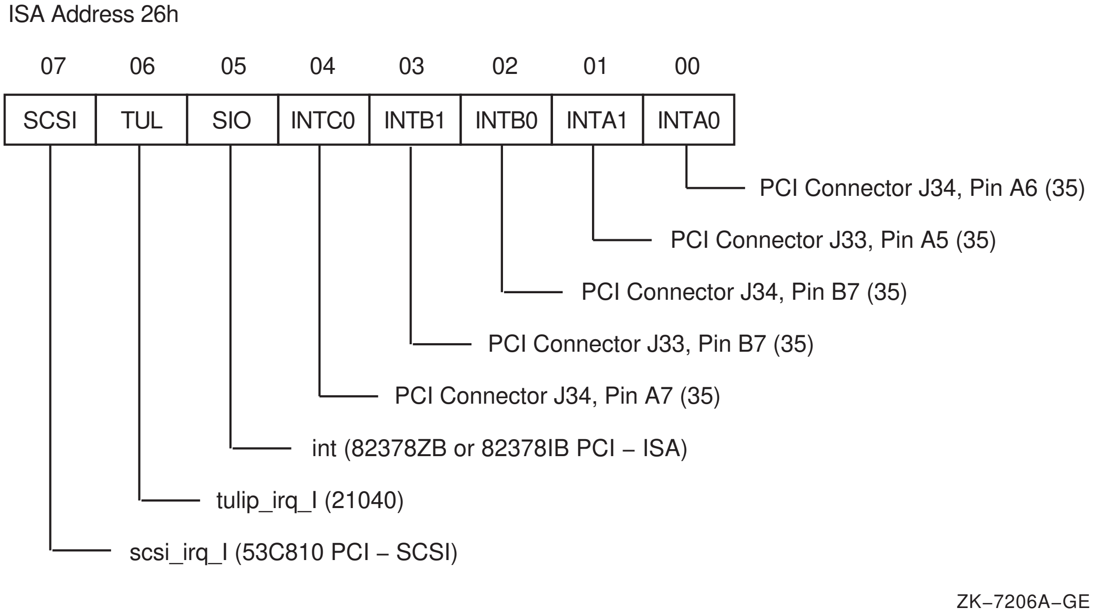


Figure 3–15 Mask Register 1—Address 27_{16}

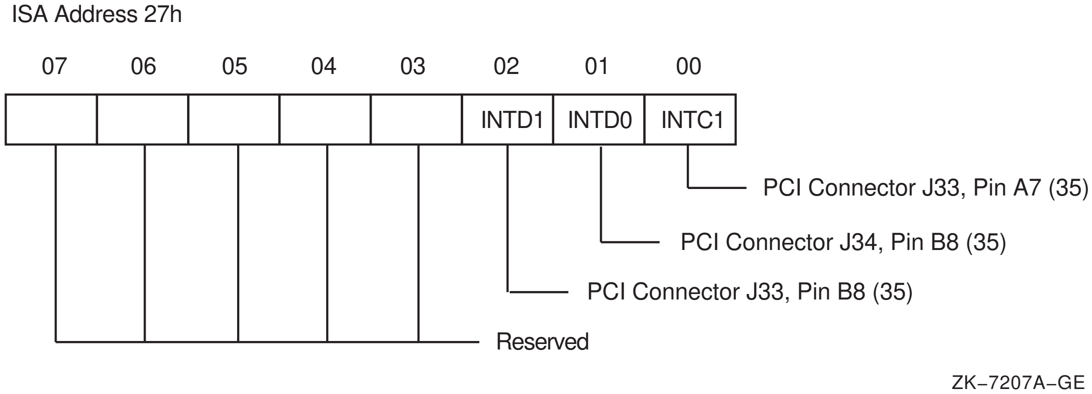
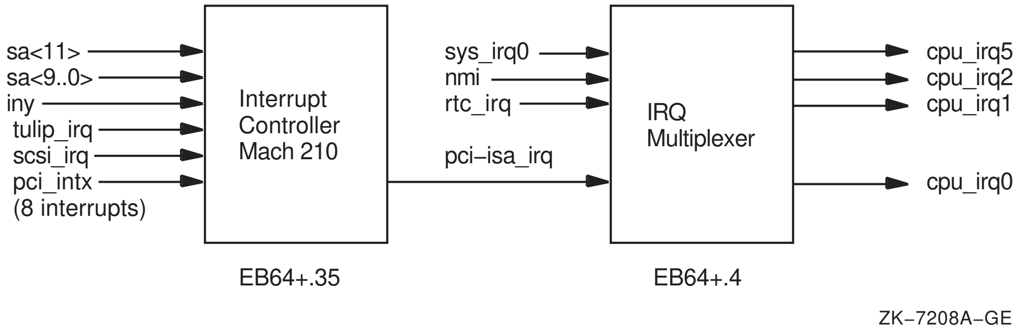


Figure 3–16 shows the functional layout of the Mach 210 controller and the associated IRQ multiplexer. All PCI interrupts are combined into a single Mach 210 output (PCI_ISA_IRQ).

Figure 3–16 Mach 210 Controller and Multiplexer



The CPU has six interrupt inputs (**cpu_irq0 through cpu_irq5**); however, **cpu_irq3** and **cpu_irq4** are not implemented. With **ev4_reset_l** deasserted, the values of the interrupts are reflected on their associated line. The interrupt signals are asserted until some action is taken by the 21064 or 21064A (for example: interrupt vector reads, error bit reads, and so on). Table 3–4 describes each CPU interrupt and its source.

Table 3–4 CPU Interrupt Descriptions

CPU Interrupts	Source Interrupt	Source Interrupt Description
CPU_IRQ0	PCI-ISA_IRQ	Summary interrupt representing interrupts from the following devices: INT: PCI-ISA bridge that includes: – IRQ1: Mouse – IRQ<7..3>: Combination chip including diskette controller, two serial ports, and one parallel port – IRQ<11:9>: ISA expansion slots – IRQ<12>: Keyboard – IRQ<15..13>: ISA expansion slots TULIP_IRQ: Ethernet controller (21040)

(continued on next page)

Table 3–4 (Cont.) CPU Interrupt Descriptions

CPU Interrupts	Source Interrupt	Source Interrupt Description
		SCSI_IRQ: SCSI controller (53C810)
		PCI_INTA0: PCI connector J34
		PCI_INTB0: PCI connector J34
		PCI_INTA1: PCI connector J33
		PCI_INTB1: PCI connector J33
		PCI_INTC0: PCI connector J34
		PCI_INTD0: PCI connector J34
		PCI_INTC1: PCI connector J33
		PCI_INTD1: PCI connector J33
CPU_IRQ1	RTC_IRQ	Real-time clock (DS1287A - TOY)
CPU_IRQ2	NMI	Nonmaskable interrupt from the SIO
CPU_IRQ3	NA	Not used. Tied to ground through a resistor.
CPU_IRQ4	NA	Not used. Tied to ground through a resistor.
CPU_IRQ5	SYS_IRQ	21071-DA interrupt

3.8.2 Arbitration Logic

The PCI arbitration (arbiter) logic is implemented in two cascaded programmable logic devices (PLDs) (*eb64+.35*). The two PLDs arbitrate and determine the priorities of the I/O (PCI and ISA) device request and grant functions.

A round-robin arbitration scheme is used among the following devices:

- PCI slot 0
- PCI slot 1
- Ethernet controller
- SCSI controller
- PCI to ISA bridge (SIO)
- 21071-DA

To make the PCI arbitration fair, a round-robin protocol ensures every requesting device is given an opportunity to acquire the bus. To ensure fairness, the following two round-robin implementation rules are observed:

1. Priority among the devices is based on the current task entry or robin. Requesters at any given time are prioritized based on their priority position from the current robin. Table 3–5 shows the round-robin priority.

Table 3–5 PCI Arbiter Round-Robin Priority

Current Robin	Priority (Highest to Lowest)					
PCI slot 0	Slot 0	Slot 1	Ethernet	SCSI	SIO	21071-DA
PCI slot 1	Slot 1	Ethernet	SCSI	SIO	21071-DA	Slot 0
Ethernet controller	Ethernet	SCSI	SIO	21071-DA	Slot 0	Slot 1
SCSI controller	SCSI	SIO	21071-DA	Slot 0	Slot 1	Ethernet
SIO bridge	SIO	21071-DA	Slot 0	Slot 1	Ethernet	SCSI
21071-DA	21071-DA	Slot 0	Slot 1	Ethernet	SCSI	SIO

Use the following table as an example. Assume that the current robin is Ethernet, and that PCI slots 0 and 1 and the 21071-DA have outstanding requests.

Device	Status	Priority
Slot 0	Request	–
Slot 1	Request	Lowest
Ethernet	Current robin	Highest
SCSI	–	–
SIO	–	–
21071-DA	Request	–

The PCI arbiter issues a grant to the next highest requesting device to the current robin. In this example, it is the 21071-DA.

2. The current robin is changed only when the highest requesting device (based on the current robin) has had a chance to use the bus (through assertion of PCI signal **frame_l**). When **frame_l** is asserted, the current robin is assigned to the last grantee +1. This ensures that whatever device used the bus last, will become the lowest priority device in the arbitration for the next arbitration interval.

   Continuing with the previous example, once the 21071-DA has had a chance to use the bus (**frame_l** asserted), the next robin is assigned to slot 0 (the last grantee +1), making the 21071-DA the lowest priority during the next arbitration interval.

   The following table shows the arbitration status after the 21071-DA has used the bus:

Device	Status	Priority
Slot 0	Current Robin	Highest
Slot 1	Request	–
Ethernet	–	–
SCSI	–	–
SIO	–	–
21071-DA	–	Lowest

Implementation Notes

- When the PCI bus and devices are reset, the current robin is assigned to PCI slot 0. This implementation is fair even for address stepping agents (that is, configuration cycles) because the robin is not changed until ownership of the bus takes place (assertion of **frame_l**). Eventually, the address stepping device will become the highest requesting device for a given robin and gain access (**frame_l**) to the bus.
- Bus parking is implemented to the 21071-DA. The 21071-DA arbitration latency will still be one cycle on an idle bus.
- There is no way to ensure fairness for a device that removes its request prematurely. To ensure a device gains the bus, it must assert request and keep it asserted until it receives a grant.
- To work in a system with the SIO, the sideband signaling protocol involving signals **flushreq_l**, **memreq_l**, and **sat_memack_l** must be followed to

ensure ISA master guaranteed access time (GAT). The SIO rules are as follows:

- If **memreq_l** and **sat_memack_l** are asserted once the SIO is granted the PCI bus, the arbiter will not remove **sio_gnt_l** until the SIO removes its request. This allows the ISA master a means to retain the bus, so that PCI retries will not force the ISA master off the PCI. PCI arbitration is still fair, because the SIO must still compete for the PCI bus. Only when it has been granted the bus, is it allowed to retain it.
- When an ISA master or DMA transfer requests the ISA bus, the SIO asserts **flushreq_l**. The **flushreq_l** signal is an indication to the system to flush all posted write buffers destined for the PCI bus. The SIO also flushes its own posted write buffers. Once the posted write buffers have been flushed and disabled, the system asserts **sat_memack_l**. This is to avoid any potential deadlock between the ISA and PCI. The SIO handles the deadlock cases through other mechanisms, such as software memory barriers, and therefore does not acknowledge **flushreq_l** with **sat_memack_l**. However, the SIO requires **sat_memack_l** be asserted whenever **flushreq_l** is asserted; therefore, the arbiter acknowledges **flushreq_l**.

Note

A fast, back-to-back master communicating with a zero wait state target (FAST DEVSEL) has the potential to execute a second transaction (if burst size=1 longword), because the detection of **frame_l** changes the round robin pointer. The arbitration is still fair, and after the second transaction, the master's priority becomes the lowest priority request.

The cascaded PCI arbiter PLD pair (*eb64+.35*) has 13 inputs and 7 outputs, as specified in Table 3–6.

Table 3–6 PLD Inputs and Outputs

Signal	Purpose
	Arbiter Inputs
buf_clk2_9	PCI clock

(continued on next page)

Table 3–6 (Cont.) PLD Inputs and Outputs

Signal	Purpose
	Arbiter Inputs
sys_reset3_l	Reset (active low)
pci_req_slot0_l	PCI request from expansion slot 0
pci_req_slot1_l	PCI request from expansion slot 1
tulip_req_l	Ethernet request
cut_req_l	SCSI request
sio_req_l	SIO request
cpu_req_l	21071-DA request
epic_memack_l	Memack from 21071-DA
frame_l	PCI bus transaction in progress
flushreq_l	SIO flush request
memreq_l	SIO memory request
arbpal_tristate_l	Tristate all outputs (pulled up in EB64+)
	Arbiter Outputs
pci_gnt_s0_l	PCI grant for slot 0
pci_gnt_s1_l	PCI grant for slot 1
tulip_gnt_l	Ethernet grant
cut_gnt_l	SCSI grant
sio_gnt_l	SIO ISA grant
cpu_gnt_l	21071-DA grant
sat_memack_l	Memack to SIO

Arbiter signals **prior0**, **prior1**, and **prior2** are used for inter-PLD communication.

3.9 PCI Devices

The EB64+ uses the PCI bus as the main I/O bus for the majority of peripheral functions. The board implements the ISA bus as an expansion bus for system support functions and peripheral devices.

3.9.1 SCSI Controller

The NCR 53C810 SCSI I/O processor is used as the interface from the PCI to SCSI. The chip supports both SCSI-1 and SCSI-2. It is packaged in a 100-pin PQFP package.

The 53C810 is designed to implement a multithread I/O algorithm free of microprocessor intervention except at the end of an I/O transfer. because the 53C810 connects to the PCI without interfacing logic, a SCSI solution including the oscillator, termination, and external connector can be implemented on less than 4 square inches of module space.

Refer to NCR document No. T159351 *NCR 53C810 PCI-SCSI I/O Processor Data Manual* for additional information. Refer to NCR document No. E25931I *NCR 53C810/53C820 PCI-SCSI Processor Design In Guide* for information on SCSI design applications and NCR TolerANT® SCSI system design.

3.9.2 Ethernet Controller

The Ethernet controller (DECchip 21040) provides the interface from the PCI to Ethernet. It is implemented using Digital's CMOS4 process and is packaged in a 120-pin PQFP package.

The DECchip 21040 is an Ethernet LAN controller that interfaces directly to the PCI bus. It communicates with the microprocessor by means of onchip control and status registers (CSRs) and a shared memory area, which is set up during initialization. This minimizes CPU involvement in operation during reception and transmission. The DECchip 21040 is capable of functioning in a full duplex network environment.

The SROM interface provides the Ethernet ID.

On the serial side of the chip, the DECchip 21040 provides both an AUI interface (10BASE2) for Digital's ThinWire and DECconnect applications and a twisted-pair interface (10BASE-T). This enables a low chip count connection to the two Ethernet interfaces. An auxiliary Halo Electronics adapter is used between the EB64+ and the connectors to form a simpler interface.

Refer to *DECchip 21040 Ethernet Controller Reference Manual* for additional information.

3.9.3 Intel Saturn IO Chip

The SIO provides the bridge between the PCI bus and the industry standard architecture (ISA) bus. The SIO incorporates the logic for:

- A PCI interface (master and slave)
- An ISA interface (master and slave)
- Enhanced 7-channel DMA controller that supports fast DMA transfers and scatter gather, and data buffers to isolate the PCI bus from the ISA bus
- PCI and ISA arbitration

Note

Although the SIO has a built-in PCI arbiter, it must be disabled for correct system operation.

The EB64+ implements the PCI arbitration logic and PLD devices. This allows use of the Intel 82378IB or 82378ZB

- A 14-level interrupt controller
- A 16-bit BIOS timer
- Three programmable timer counters
- Nonmaskable interrupt (NMI) control logic
- Decoding and control for utility bus peripheral devices
- Speaker driver

Refer to Intel document No. 290483 *82420/82430 PCIset ISA and EISA Bridges* for additional information.

3.9.4 PCI Expansion Slots

Two PCI bus expansion slots are available on the EB64+, with one slot shared with the ISA. Both slots use the Intel standard 5V PCI connector and pinout for 32-bit implementation. This allows the system designer to add additional PCI options.

3.9.5 PCI Graphics Interface

The DECchip 21030 8bpp Evaluation Board (EB30) can be used as a high-performance graphics processor. This optional add-on occupies one PCI slot. An ISA-based graphics board may be used if slower graphics are desired.

The microprocessor memory controller has the capability of performing simple graphics-oriented data manipulations. An embedded graphics memory connector is provided in line with one of the ISA slots to support a plug-in frame buffer module.

3.10 ISA Devices

Figure 3–17 shows the EB64+ ISA bus implementation with peripheral devices and connectors. Also shown is the utility bus with system support devices.

3.10.1 Combination Controller

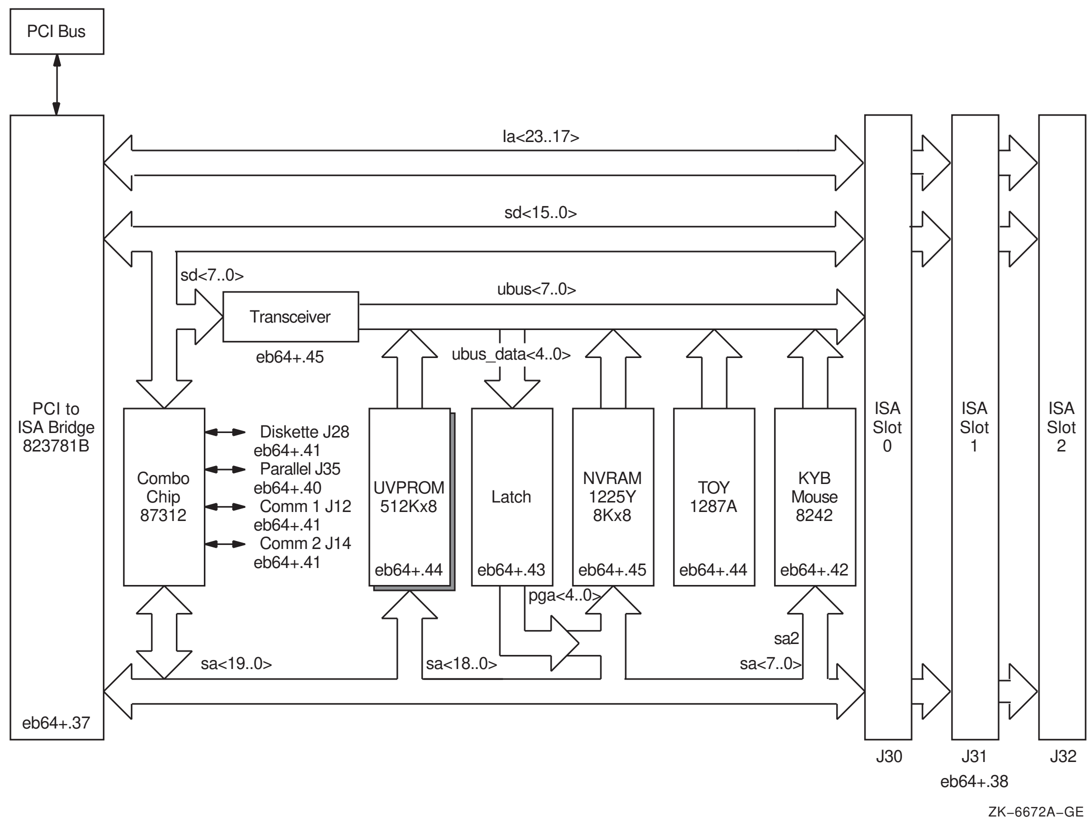
The EB64+ uses the National Semiconductor PC87312 as the combination controller chip. (See Figure 3–17.) It is packaged in a 100-pin PQFP configuration. The chip provides the following ISA peripheral functions:

- Diskette controller—Software compatible to the Intel PC8477 (contains a superset of the Intel DP8473 and NEC *u*PD765)) and the Intel N82077 FDC functions. The onchip analog data separator requires no external filter components and supports the 4 MB drive format and other standard diskette drives used with 5.25 inch and 3.5 inch media. FDC data and control lines are brought out to a standard 34-pin connector. A ribbon cable interfaces the connector to one or two diskette drives.
- Serial ports—Two UARTs with modem control, compatible with NS16450 or PC16550, are brought out to separate onboard, 10-pin connectors. The lines can be brought out through 25-pin female D-sub connectors on the bulkhead of a standard PC enclosure.
- Parallel Port—The bidirectional (jumper-controlled) parallel port is brought out to an onboard 26-pin connector. It can be brought out through a 25-pin female D-sub connector on the bulkhead of a standard PC enclosure.

An onboard 24 MHz oscillator supplies a reference clock for the diskette data separator and serial ports.

Refer to National Semiconductor document No. 11362 *PC87311/PC87312 (SuperI/O™ II/III) Floppy Disk Controller with Dual UARTs, Parallel Port, and IDE Interface* for additional information.

Figure 3–17 ISA Devices



3.10.2 Keyboard and Mouse Controller

The Intel N8242 located on the ISA utility bus provides the keyboard and mouse controller functions. It is packaged in a 44-pin PLCC configuration.

The 8242 is an Intel UPI™-42AH universal peripheral interface. It is an 8-bit slave microcontroller with 2 KB of ROM and 256 bytes of RAM that has been preprogrammed with a keyboard BIOS for standard scan codes.

Refer to Intel document No. 210393 *UPI™-41AH/42AH Universal Peripheral Interface 8-bit Slave Microcontroller* for additional information.

3.10.3 Time-of-Year Clock

The Dallas DS1287A chip, located on the ISA utility bus, provides the time-of-year (TOY) function. It is packaged in a plastic 24-pin DIP configuration. The DS1287A is designed with onchip RAM, a lithium energy source, a quartz crystal, and write protection circuitry. (See Figure 3–17.)

The functions available to the user include:

- A nonvolatile time of day clock
- An alarm
- 100-year calendar
- Programmable interrupt
- Square wave generator
- 50 bytes of nonvolatile static RAM

The time of day and memory are maintained in the absence of power through the lithium energy source.

The DS1287A includes three separate, fully automatic interrupt sources for a processor. The alarm interrupt can be programmed to occur at rates from one per second to one per day. The periodic interrupt can be selected for rates from 122*u*s to 500 ms. The update-ended interrupt can be used to indicate to the program that an update cycle has completed. The device interrupt line is presented to the system interrupt PLD.

3.10.4 Utility Bus Memory Devices

The EB64+ utility bus drives two different memory types: NVRAM and UVPROM. The following paragraphs summarize the devices and their applications.

NVRAM

The DS1225Y chip, located on the ISA utility bus, provides an additional 8 KB of nonvolatile CMOS RAM for operating system support. Read access time is 170 ns. The chip is packaged in a plastic 28-pin DIP configuration.

NVRAM data is accessed through 13 address inputs. The high-order five bits **pga<4..0>** are latched from **ubus_data<4..0>** and the low-order eight bits are driven by ISA bus **sa<7..0>**.

Data is maintained in the absence of **Vdd** (+5Vdc) without any additional support circuitry because the chip constantly monitors **Vdd**. Should **Vdd** decay, the NVRAM automatically write protects itself by disregarding RAM inputs and driving all outputs to high impedance. As **Vdd** falls, the internal power switching circuit connects the lithium energy source to the RAM to retain the data. During power up when **Vdd** rises, the power switching circuit connects external **Vdd** to the RAM and disconnects the lithium energy source. Normal RAM operations can resume after Vcc exceeds 4.5 volts.

UVPROM

Two industry standard UVPROMs are provided on the EB64+. The UVPROMs are individually jumper selectable; only one is selected at a time. One UVPROM contains the debug monitor code; the other is provided for user-specific code development.

3.10.5 ISA Expansion Slots

Three ISA expansion slots are provided for plug-in ISA peripherals. One of the slots is shared with the PCI and can be used for a PCI or ISA device.

3.11 Serial ROM

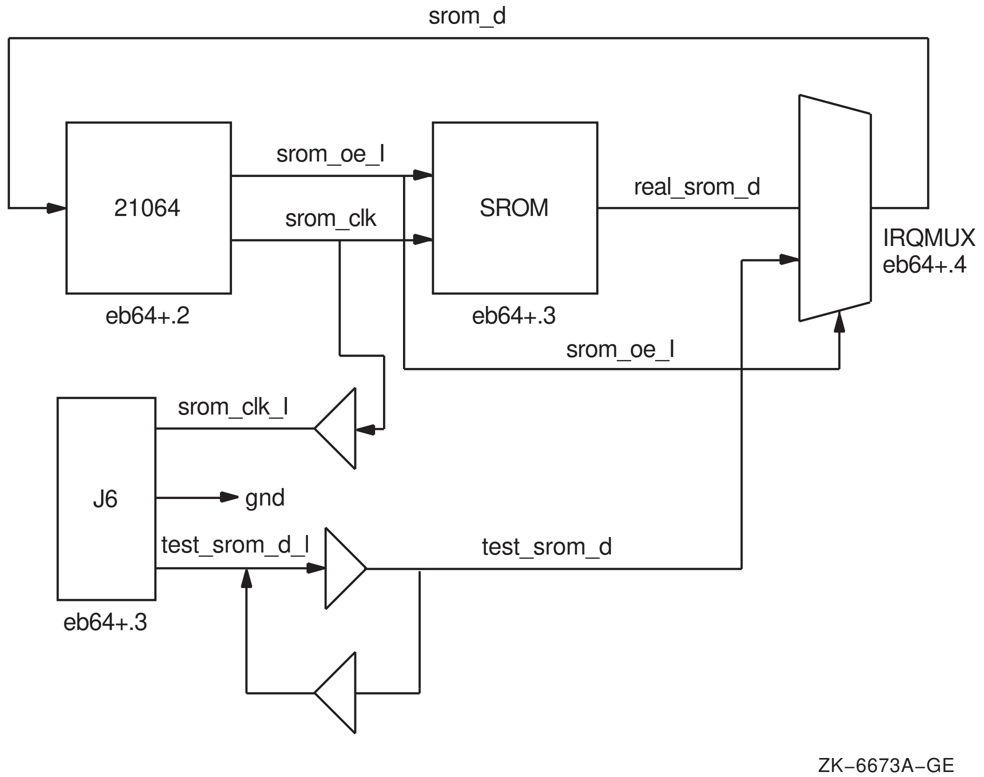
The 21064 or 21064A use a serial ROM (SROM) for its initialization code. When **reset** is deasserted, the contents of the SROM are read into the Icache of the 21064. The code is then executed from the Icache. The general steps performed by the SROM initilization are:

1. Initialize the CPU's internal processor registers (IPRs).
2. Perform the minimum I/O subsystem initialization necessary to access the Real Time Clock (RTC) and the System ROM.
3. Detect CPU speed by polling the Periodic Interrupt flag (PIF) in the RTC.
4. Set up memory and/or Backup cache (Bcache) parameters based on the speed of the CPU.
5. Wake up the DRAMs.
6. Initialize the Bcache.
7. Copy the contents of the entire memory to itself to insure good memory data parity.
8. Scan System ROM for special header which specifies where and how System ROM firmware should be loaded.
9. Copy the contents of the System ROM to memory and begin code execution.

10. Pass parameters up to the next level of firmware to provide a predictable firmware interface.

Figure 3–18 is a simplified block diagram of the SROM serial port logic. The multiplexer (IRQMUX - 74F257) (*eb64+.4*) provides a multiplex function for the SROM **real_srom_d** and serial port **test_srom_d** data inputs to the 21064. Signal **srom_oe_l** provides the multiplexer input select function.

Figure 3–18 SROM Serial Port

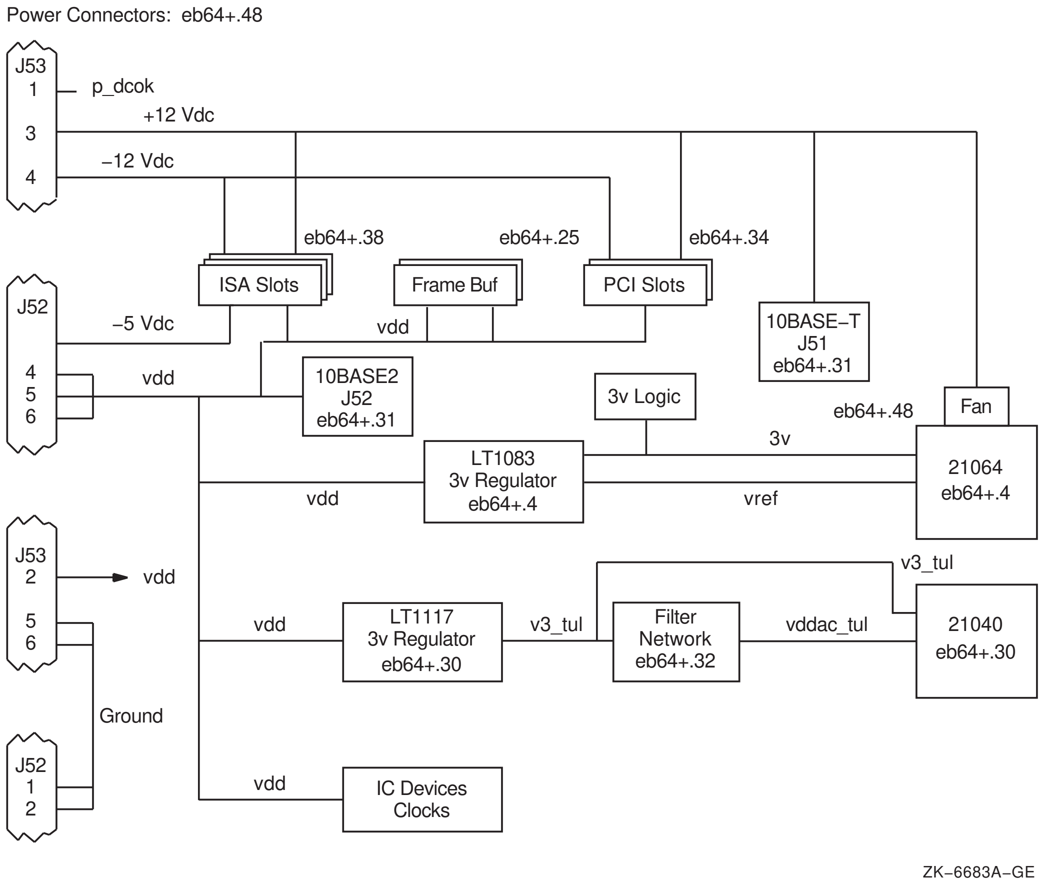


After the SROM code has been read into the Icache, the 21064 SROM port can be used as a software controlled serial port. The serial port can be used for such things as diagnosing system problems when the only working devices are the 21064, SROM, and the logic required for their direct support.

3.12 Dc Power Distribution

The EB64+ derives its system power from a user-supplied, industry-standard PC power supply. The power supply must provide +12 and -12 Vdc, -5 Vdc, and **Vdd** (+5Vdc). Dc power is supplied through power connectors J52 and J53 (*eb64+.48*). (See Figure 3–19.) Power is distributed to the board logic through dedicated power planes within the six-layer board structure.

Figure 3–19 Dc Power Distribution



A shown in Figure 3–19 the +12 and -12 Vdc, and -5 Vdc are supplied to ISA connectors J30, J31, and J32 (*eb64+.38*). The +12 and -12 Vdc are supplied to PCI connectors J33 and J34 (*eb64+.31*). The +12 Vdc is also supplied to the Ethernet 10BASE-T connector J51 (*eb64+.31*) and the CPU fan connector J54 (*eb64+.48*). **Vdd** is supplied to ISA connectors J30, J31, and J32, PCI connectors J33 and J34, and the 10BASE2 connector J41 (*eb64+.31*).

Vdd is supplied to a pair of LT1083 linear regulators to produce +3.3 Vdc (+3V) required by the CPU. The +3V output is supplied to the 21064 or 21064A and the 3V logic. The +3V output is also applied to a voltage divider to produce **vref** which is supplied to the 21064 or 21064A. A TL7702B power monitor senses the +3V output to ensure that it is stable before the 21064 inputs and I/O pins are driven. Any device that drives the 21064 or 21064A has a tristate output controlled by the power monitor output.

Should the +3V output fail, the power monitor enables **sense_dis** which is applied to the reset logic (*eb64+.47*). The reset logic generates a group of reset functions to the 21064 and the remainder of the system including PCI devices. (See Section 3.13.)

Vdd is also supplied to an LT1117 linear regulator (*eb64+.30*) producing **v3_tul**, which is applied to the 21064 and 21040 controller. The **v3_tul** signal is also applied to a filter network producing **vddac_tul**, which is supplied to the 21040. A second TL7702B power monitor senses the +3.3 Vdc level of the controller and produces **3v_tul_bad** if the level fails.

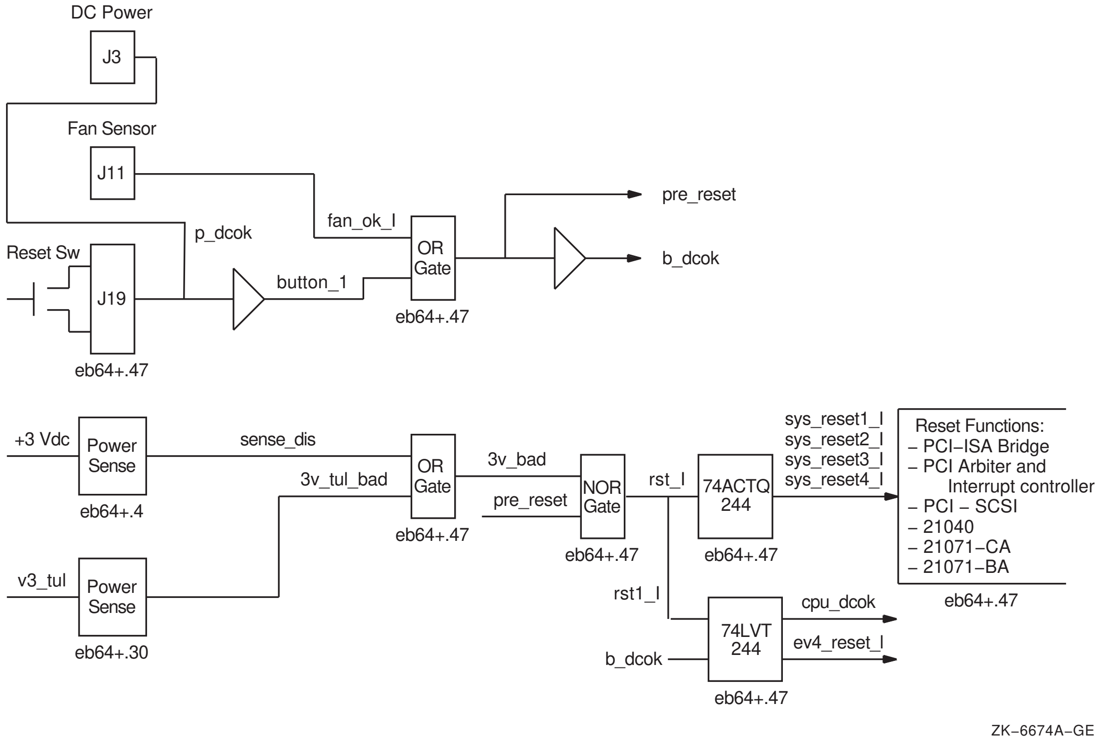
3.13 Reset and Initialization

An external switch can be connected to J19 (*eb64+.47*) to control the **reset** and **dcok** signals. (See Figure 3–20). The **reset** function initializes the 21064 or 21064A and the system logic, but does not send an initialization pulse to the ISA devices. The **dcok** signal provides a full system initialization, equivalent to a power off and power on cycle.

In addition, the fan sense signal (**fan_ok_l**) is ORed with the reset switch output and when enabled, will drive **cpu_dcok** indicating a fan failure.

The **3v_tul_bad** signal from the Ethernet chip power monitor is ORed with the power monitor reference **buf_sense_dis** (*eb64+.4*) to produce **3v_bad**. In turn, **3v_bad** and **pre_reset** are applied to a pair of NOR gates and produce: **rst1_l** and **rst_l**.

Figure 3–20 System Reset and Initialization



The **rst1_l** signal drives the 74LVT244 to enable **ev4_reset** to the 21064 or 21064A. The **rst_l** signal drives the 74ACTQ244 to enable a set of **sys_reset** signals to reset the remainder of the system including PCI and ISA devices.

3.14 System Software

The EB64+ software is divided into the following categories:

- Serial ROM code
- Debug and monitor ROM code
- Windows NT and OSF/1 operating systems

3.14.1 Serial ROM Code

The serial ROM code is contained in the Xilinx XC1765D serial configuration ROM (*eb64+.3*). The code is executed by the 21064 or 21064A on power up as described in Section 3.11. The serial ROM code initializes the system, then transfers control to the debug and monitor ROM.

The mini-debugger is also resident in the SROM. A jumper can be set in J2 to trap to the mimi-debugger. Connector J46 provides a terminal port for the mini-debugger.

3.14.2 Debug and Monitor ROM

The EB64+ includes an industry standard 512-KB UVPROM that contains the debug monitor code (*eb64+.44*). A second blank 512 KB UVPROM is provided for user-specific code development and can be selected by means of a jumper (J5). The user can develop code on a host system and load it into the EB64+ through the serial or Ethernet ports.

The monitor provides functions such as:

- File load
- Read and write memory and registers
- Memory image dump
- Transfer control to program
- Breakpoints

The user kit includes the full source code listing for all SROM and debug and monitor ROM software.

3.14.3 Operating Systems

The EB64+ is designed to run either the OSF/1 or Windows NT operating systems.

4 System Address Mapping

This chapter describes the mapping of the 34-bit processor physical address space into memory and I/O space addresses. Also included are the translations of the processor-initiated address into a PCI address, and PCI initiated addresses into physical memory addresses.

4.1 CPU Address Mapping to PCI Space

The 34-bit physical sysBus address space is composed:

- Memory address space
- Local I/O space, for registers residing on the sysBus (that is, registers in the 21071-CA and 21071-DA chips)
- PCI space

Note

The sysBus represents the 21064 pin bus as well as control signals between the 21071-CA and 21071-DA chips.

The PCI defines three physical address spaces: PCI memory (for memory residing on the PCI), PCI I/O space, and PCI configuration space. In addition to these address spaces, the sysBus I/O space is also used to generate PCI Interrupt Acknowledge cycles and PCI special cycles. Figure 4–1 shows the address space. Table 4–1 provides a summary description of the spaces.

Figure 4–1 sysBus Address Map

Cacheable Memory Space	0 0000 0000 0 FFFF FFFF
Noncacheable Memory Space	1 0000 0000 1 7FFF FFFF
21071–CA CSR Space	1 8000 0000 1 9FFF FFFF
21071–DA CSR Space	1 A000 0000 1 AFFF FFFF
PCI Interrupt Acknowledge Special Cycle Space	1 B000 0000 1 BFFF FFFF
PCI Sparse I/O Space	1 C000 0000 1 DFFF FFFF
PCI Configuration Space	1 E000 0000 1 FFFF FFFF
PCI Sparse Memory Space	2 0000 0000 2 7FFF FFFF
PCI Dense Memory Space	3 0000 0000 3 FFFF FFFF

ZK–6520A–GE

Table 4–1 sysBus Address Space Description

sysadr[33..32]	sysadr[31..28]	Address Space	Description
00	xxxx	Cacheable memory space	Accessed by the CPU instruction stream (Istream) or data stream (Dstream). Accessed by DMA. The 21071-DA does not respond to addresses in this space.
01	0xxx	Noncacheable memory space	Accessed by the CPU (Istream or Dstream). Accessed by DMA. Can be used for a frame buffer on the DRAM bus. The 21071-DA does not respond to addresses in this space.
01	100x	21071-CA CSRs	The 21071-CA responds to all addresses in this space, Dstream access only. The 21071-DA does not respond to addresses in this space.
01	1010	21071-DA CSRs	The 21071-DA responds to all addresses in this space, Dstream access only.
01	1011	PCI interrupt acknowledge or PCI special cycle	The 21071-CA expects the 21071-DA to respond to all addresses in this space. A read causes a PCI interrupt acknowledge; a write causes a special cycle. Dstream access only.
01	110x	PCI sparse I/O space	16 MB of PCI space. The lower 256 KB of this space must be used for addressing the PCI and ISA devices. The remainder of the space can be used for other devices. Dstream access only.

(continued on next page)

Table 4–1 (Cont.) sysBus Address Space Description

sysadr[33..32]	sysadr[31..28]	Address Space	Description
01	111x	PCI configuration space	See Section 4.1.7. Dstream access only.
01	xxxx	PCI sparse memory space	128 MB addressable PCI space. The lower address bits are used to determine byte masks and transaction length information. The 4 GB space is reduced to a 128 MB sparse space. Must use this space when byte or word granularity is required. Read or write length is no more than a quadword. Reading other than the requested data is harmful. Prefetching read data is prohibited. Dstream access only.
11	xxxx	PCI dense memory space	4 GB of PCI space. Used for devices with access granularity greater than one longword. Reads do not have side effects; prefetching data from PCI devices is allowed. Typically used for data buffers. Dstream access only.

4.1.1 Cacheable Memory Space—0 0000 0000 .. 0 FFFF FFFF

The 21071-CA recognizes the 4 GB of quadrant 0 (corresponding to **sysBus33..32 = 00**) as cacheable memory space. The 21071-CA responds to all read and write accesses in this space. Some or all of main memory can be programmed to be in cacheable space.

4.1.2 Noncacheable Memory Space—1 0000 0000 .. 1 7FFF FFFF

The 21071-CA recognizes the lower 2 GB of quadrant 1 (corresponding to **sysBus<33..32> = 01**) as noncacheable memory space. The Bcache is bypassed by the 21071-CA on accesses to this space. Some or all of main memory can be programmed to be in this space. If a frame buffer is supported in system memory, it should be addressed in this space.

4.1.3 DECchip 21071-CA CSR Space—1 8000 0000 .. 1 9FFF FFFF

DECchip 21071-CA responds to all CSR accesses in this space. Table 4–2 specifies the registers and associated register addresses. Register descriptions are contained in Appendix A.

Table 4–2 DECchip 21071-CA CSR Register Addresses

Address$_{16}$	Register Name
1 8000 0000	General Control Register
1 8000 0020	Reserved
1 8000 0040	Error and Diagnostic Status Register
1 8000 0060	Tag Enable Register
1 8000 0080	Error Low Address Register
1 8000 00A0	Error High Address Register
1 8000 00C0	LDx_L Low Address Register
1 8000 00E0	LDx_L High Address Register
1 8000 0200	Global Timing Register
1 8000 0220	Refresh Timing Register
1 8000 0240	Video Frame Pointer Register
1 8000 0260	Presence Detect Low Data Register
1 8000 0280	Presence Detect High Data Register
1 8000 0800	Bank 0 Base Address Register
1 8000 0820	Bank 1 Base Address Register
1 8000 0840	Bank 2 Base Address Register
1 8000 0860	Bank 3 Base Address Register
1 8000 0880	Bank 4 Base Address Register
1 8000 08A0	Bank 5 Base Address Register
1 8000 08C0	Bank 6 Base Address Register
1 8000 08E0	Bank 7 Base Address Register
1 8000 0900	Bank 8 Base Address Register

(continued on next page)

Table 4–2 (Cont.) DECchip 21071-CA CSR Register Addresses

$Address_{16}$	Register Name
1 8000 0A00	Bank 0 Configuration Register
1 8000 0A20	Bank 1 Configuration Register
1 8000 0A40	Bank 2 Configuration Register
1 8000 0A60	Bank 3 Configuration Register
1 8000 0A80	Bank 4 Configuration Register
1 8000 0AA0	Bank 5 Configuration Register
1 8000 0AC0	Bank 6 Configuration Register
1 8000 0AE0	Bank 7 Configuration Register
1 8000 0B00	Bank 8 Configuration Register
1 8000 0C00	Bank 0 Timing Register A
1 8000 0C20	Bank 1 Timing Register A
1 8000 0C40	Bank 2 Timing Register A
1 8000 0C60	Bank 3 Timing Register A
1 8000 0C80	Bank 4 Timing Register A
1 8000 0CA0	Bank 5 Timing Register A
1 8000 0CC0	Bank 6 Timing Register A
1 8000 0CE0	Bank 7 Timing Register A
1 8000 0D00	Bank 8 Timing Register A
1 8000 0E00	Bank 0 Timing Register B
1 8000 0E20	Bank 1 Timing Register B
1 8000 0E40	Bank 2 Timing Register B
1 8000 0E60	Bank 3 Timing Register B
1 8000 0E80	Bank 4 Timing Register B
1 8000 0EA0	Bank 5 Timing Register B
1 8000 0EC0	Bank 6 Timing Register B
1 8000 0EE0	Bank 7 Timing Register B
1 8000 0F00	Bank 8 Timing Register B

4.1.4 DECchip 21071-DA CSR Space—1 A000 0000 .. 1 AFFF FFFF

The DECchip 21071-DA responds to all accesses in this space. Table 4–3 specifies the registers and associated register addresses. Register descriptions are contained in Appendix A.

Table 4–3 DECchip 21071-DA CSR Register Addresses

$Address_{16}$	Register Name
1 A000 0000	21071-DA Control and Status Register (DCSR)
1 A000 0020	PCI Error Address Register (PEAR)
1 A000 0040	SysBus Error Address Register (SEAR)
1 A000 0060	Dummy Register1
1 A000 0080	Dummy Register2
1 A000 00A0	Dummy Register3
1 A000 00C0	Translated Base 1 Register
1 A000 00E0	Translated Base 2 Register
1 A000 0100	PCI Base 1 Register
1 A000 0120	PCI Base 2 Register
1 A000 0140	PCI Mask 1 Register
1 A000 0160	PCI Mask 2 Register
1 A000 0180	Host Address Extension Register 0 (HAXR0)
1 A000 01A0	Host Address Extension Register 1 (HAXR1)
1 A000 01C0	Host Address Extension Register 2 (HAXR2)
1 A000 01E0	PCI Master Latency Timer Register
1 A000 0200	TLB Tag 0 Register
1 A000 0220	TLB Tag 1 Register
1 A000 0240	TLB Tag 2 Register
1 A000 0260	TLB Tag 3 Register
1 A000 0280	TLB Tag 4 Register
1 A000 02A0	TLB Tag 5 Register
1 A000 02C0	TLB Tag 6 Register
1 A000 02E0	TLB Tag 7 Register

(continued on next page)

Table 4–3 (Cont.) DECchip 21071-DA CSR Register Addresses

$Address_{16}$	Register Name
1 A000 0300	TLB 0 Data Register
1 A000 0320	TLB 1 Data Register
1 A000 0340	TLB 2 Data Register
1 A000 0360	TLB 3 Data Register
1 A000 0380	TLB 4 Data Register
1 A000 03A0	TLB 5 Data Register
1 A000 03C0	TLB 6 Data Register
1 A000 03E0	TLB 7 Data Register
1 A000 0400	TBIA (Translation Buffer Invalidate All Register)

4.1.5 PCI Interrupt Acknowledge/Special Cycle Space—1 B000 000 .. 1 BFFF FFFF

A read access to this space causes an interrupt acknowledge cycle on the PCI. Bits **sysBus<6..3>** are used to generate the byte enables on the PCI as specified in Table 4–4. Bits **sysBus<26..7>** are *don't care* during this transaction.

A write access to this space causes a special cycle on the PCI. The address and byte enables are *don't care* during this transaction.

Note

Software must use an STL instruction to initiate these transactions. An STQ instruction will result in a 2-longword burst on the PCI, which is illegal.

4.1.6 PCI Sparse I/O Space—1 C000 0000 .. 1 DFFF FFFF

The PCI sparse I/O space is similar to the PCI sparse memory space. This 512 MB SysBus address space maps to 16 MB of PCI I/O address space. A read or write to this space causes a PCI I/O read or PCI I/O write command respectively.

Bits **sysBus<33..29>** identify the various address spaces on the SysBus. Bits **sysBus<6..3>** generate the length of the PCI transaction in bytes, the byte enables, and **ad<2..0>** on the PCI. (See Table 4–4).

Bits **sysBus<28..8>** correspond to the quadword PCI addresses and are sent out on **ad<23..3>** during the address phase on the PCI. Bits **ad<31..24>** are obtained from one of two host address extension registers (HAXR0 and HAXR2). The HAXR0 register (which is hard coded as 0) is used for SysBus addresses between 1 C000 0000 and 1 C07F FFFF (that is, when **sysBus<28..23>** are 0).

The HAXR2 register maps SysBus addresses between 1 C080 0000 and 1 DFFF FFFF (that is, when **sysBus<28..23>** are nonzero anywhere in the PCI address space). The HAXR2 register is a CSR in the 21071-DA chip and is fully programmable. This allows ISA devices that require their I/O space to be in the lower 256 KB to coexist with other devices not having that restriction. The lower 256 KB have a fixed mapping (HAXR0) to 0, and the remaining 64 MB—64 KB can be programmed anywhere in PCI space.

Figure 4–2 shows the SysBus to PCI I/O address translation. Table 4–4 describes the generation of the byte enables, and PCI **ad<2..0>** from **SysBus<6..3>**.

Figure 4–2 PCI Sparse I/O Space Address Translation

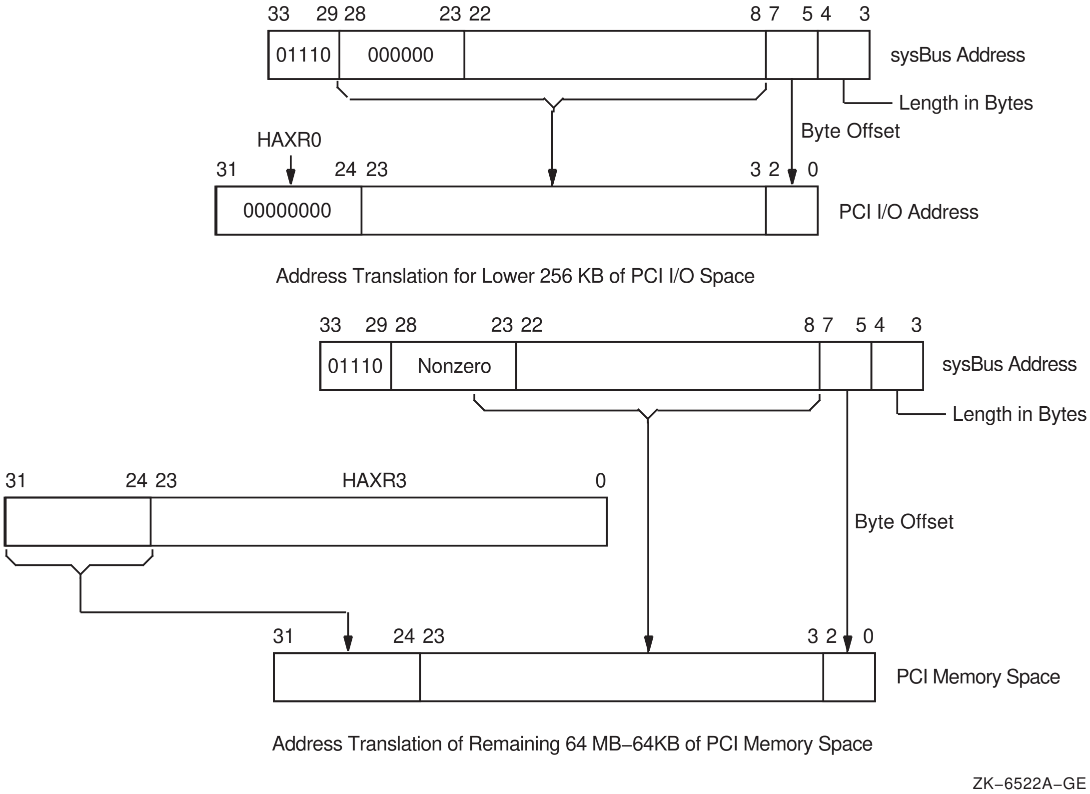


Table 4–4 PCI Sparse I/O Space Byte Enable Generation

Length	CPU Address <6..5>	CPU Address <4..3>	PCI Byte Enable[1]	PCI Address <2..0>
Byte	00	00	1110	CPU Address<7>, 00
	01	00	1101	CPU Address<7>, 01
	10	00	1011	CPU Address<7>, 10
	11	00	0111	CPU Address<7>, 11

[1]Byte enable set to 0 indicates that byte lane carries meaningful data.

(continued on next page)

Table 4–4 (Cont.) PCI Sparse I/O Space Byte Enable Generation

Length	CPU Address <6..5>	CPU Address <4..3>	PCI Byte Enable[1]	PCI Address <2..0>
Word	00	01	1100	CPU Address<7>, 00
	01	01	1001	CPU Address<7>, 01
	10	01	0011	CPU Address<7>, 10
	11	01	Illegal[2]	-
Tribyte	00	10	1000	CPU Address<7>, 00
	01	10	0001	CPU Address<7>, 01
	10	10	Illegal[2]	-
	11	10	Illegal[2]	-
Longword	00	11	0000	CPU Address<7>, 00
Longword	01	11	Illegal[2]	-
Longword	10	11	Illegal[2]	-
Quadword	11	11	0000	000

[1]Byte enable set to 0 indicates that byte lane carries meaningful data.

[2]These combinations are architecturally illegal. If there is an access with this combination of address<6..3>, the 21071-DA responds to the transactions but the results are unpredictable.

Caution

Quadword accesses to this PCI sparse I/O space causes a 2-longword burst on the PCI. PCI devices cannot support bursting in I/O space.

4.1.7 PCI Configuration Space—1 E000 0000 .. 1 FFFF FFFF

A read or write access to this space causes a configuration read or write cycle on the PCI. There are two classes of targets–devices on the primary PCI bus and devices on the secondary PCI buses that are accessed through PCI-to-PCI bridge chips.

During PCI configuration cycles, the meanings of the address fields vary depending on the intended target of the configuration cycle. Bits **ad<1..0>** which are supplied by the HAXR2 register, indicate the target bus:

Bits **ad<1..0>** equal to 00 indicate the primary PCI bus
Bits **ad<1..0>** equal to 01 indicate a secondary PCI bus

Table 4–5 defines the various fields of **ad** during the address phase of a configuration read or write cycle.

Table 4–5 PCI Configuration Space Definition

Target Bus	ad Bits	Definition
Primary PCI Bus		
	<31..11>	Decoded from **sysAdr<20..16>** according to Table 4–6. Can be used for IDSEL# or don't cares. Typically, the IDSEL# pin of each device is connected to a unique **ad** line.
	<10..8>	Function select (1 of 8), from **sysAdr<15..13>**
	<7..2>	Register select, from **sysAdr<12..7>**
	<1..0>	00, from HAXR2<1..0>
Secondary PCI Busses **(Must pass through a PCI-to-PCI bridge)**		
	<31..24>	Forced to 0 by the 21071-DA chip
	<23..16>	Secondary bus number from **sysAdr<28..21>**
	<15..11>	Device number from **sysAdr<20..16>**
	<10..8>	Function select (1 of 8) from **sysAdr<15..13>**
	<7..2>	Register select from **sysAdr<12..7>**
	<1..0>	01 from HAXR2<1..0>

Table 4–6 PCI Address Decoding for Primary Bus Configuration Accesses

Device Number (sysAdr<20..16>)	PCI AD<31..11
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
00100	0000 0000 0000 0000 1000 0

(continued on next page)

Table 4–6 (Cont.) PCI Address Decoding for Primary Bus Configuration Accesses

Device Number (sysAdr<20..16>)	PCI AD<31..11
00101	0000 0000 0000 0001 0000 0
00110	0000 0000 0000 0010 0000 0
00111	0000 0000 0000 0100 0000 0
01000	0000 0000 0000 1000 0000 0
01001	0000 0000 0001 0000 0000 0
01010	0000 0000 0010 0000 0000 0
01011	0000 0000 0100 0000 0000 0
01100	0000 0000 1000 0000 0000 0
01101	0000 0001 0000 0000 0000 0
01110	0000 0010 0000 0000 0000 0
01111	0000 0100 0000 0000 0000 0
10000	0000 1000 0000 0000 0000 0
10001	0001 0000 0000 0000 0000 0
10010	0010 0000 0000 0000 0000 0
10011	0100 0000 0000 0000 0000 0
10100	1000 0000 0000 0000 0000 0
10101	0000 0000 0000 0000 0000 0
10110	0000 0000 0000 0000 0000 0
10111	0000 0000 0000 0000 0000 0
11000	0000 0000 0000 0000 0000 0
11001	0000 0000 0000 0000 0000 0
11010	0000 0000 0000 0000 0000 0
11011	0000 0000 0000 0000 0000 0
11100	0000 0000 0000 0000 0000 0
11101	0000 0000 0000 0000 0000 0
11110	0000 0000 0000 0000 0000 0
11111	0000 0000 0000 0000 0000 0

4.1.7.1 PCI Configuration Cycles to Primary Bus Targets

Primary PCI bus devices are selected during a PCI configuration cycle if their IDSEL# pin is asserted, the PCI bus command indicates a configuration read or write, and **ad<1..0>** are 00. Bits **ad<7..2>**, which are taken from **sysAdr<12..7>**, select a long word register in the device's 256-byte configuration address space. Configuration accesses can use byte masks, which may be derived by following the method shown in Table 4–4.

Peripherals that integrate multiple functional units (for example, SCSI, Ethernet, and so on) can provide configuration spaces for each function. Bits **ad<10..8>**, which are taken from **sysAdr<15..13>**, can be decoded by the peripheral to select one of eight functional units.

Bits **<31..11>** are used to generate the IDSEL signals. Typically, the IDSEL# pin of each PCI peripheral is connected to a unique address line. Bits **ad<31..11>**, are decoded from **sysAdr<20..16>** according to Table 4–6, ensuring that only one bit of **ad<31..11>** is asserted for any given configuration space transaction on the primary PCI bus. **sysAdr<28..21>** are ignored.

4.1.7.2 PCI Configuration Cycles to Secondary Bus Targets

If the PCI cycle is a configuration read or write cycle but **ad<1..0>** are 01, a device on a secondary PCI bus is being selected across a PCI-to-PCI bridge. This cycle will be accepted by a PCI-to-PCI bridge for propagation to its secondary PCI bus. During this cycle **ad<23..16>**, taken from **sysAdr<28..21>**, select a unique bus number; **ad<15..11>**, taken from **sysAdr<20..16>**, select a device on that bus (typically decoded by the target bridge to generate IDSEL# signals); **ad<10..8>**, taken from **sysAdr<15..13>** select one of eight functional units per device; and **ad<7..2>**, taken from **sysAdr<12..7>**, select a longword in the device's configuration register space.

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

If the bus number of the configuration cycle matches the bus number of the bridge chip secondary PCI interface, it will intercept the configuration cycle, decode it, and generate a PCI configuration cycle with **ad<1..0>** equal to 01 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 configuration cycle passes, unmodified (leaving **ad<1..0>** = 01), through the bridge. The configuration cycle will be intercepted and decoded by a downstream bridge.

4.1.8 PCI Sparse Memory Space—2 0000 0000 .. 2 FFFF FFFF

Access to this space can have byte, word, tribyte, longword, or quadword granularity. The Alpha AXP architecture does not provide byte, word, or tribyte granularity which the PCI requires. To provide this granularity, the byte enable and byte length information are encoded in the lower address bits of this space (**ad<7..3>**).

Bits **sysBus<31..8>** generate quadword addresses on the PCI, resulting in a sparse 4 GB space that maps to 128 MB of PCI address space. An access to this space causes a memory read or write access on the PCI.

Bits **sysBus<33..32>** identify the various address spaces on the SysBus. Bits **sysBus<7..3>** generate the length of the PCI transaction in bytes, the byte enables, and **ad<2..0>**. (See Table 4–7). Bits **sysBus<31..8>** correspond to the quadword PCI addresses and are sent out on **ad<26..3>** during the address phase on the PCI.

Bits **ad<31..27>** are obtained from one of two host address extension registers (HAXR0 and HAXR1). HAXR0 (which is hard coded as 0) is used for sysBus addresses between 2 0000 0000 and 2 1FFF FFFF (that is, when **sysBus<31..29>** is 0). The HAXR1 register maps sysBus addresses between 2 2000 0000 and 2 FFFF FFFF (that is, when **sysBus<31..29>** is nonzero anywhere in the PCI address space).

HAXR1 is a CSR in the 21071-DA and is fully programmable. This allows ISA devices that require memory to be mapped in the lower 16 MB to coexist with other devices that do not have that restriction. The lower 16 MB have a fixed mapping (HAXR0) to 0, and the remaining 112 MB can be programmed anywhere in PCI space.

Figure 4–3 shows the sysBus to PCI memory address translation. Table 4–7 shows the generation of the byte enables and PCI address **ad<2..0>** from **sysBus<6..3>**.

Table 4–7 PCI Sparse Memory Space Byte Enable Generation

Length	CPU Address <6..5>	CPU Address <4..3>	PCI Byte Enable[1]	PCI Address<2..0>[2]
Byte	00	00	1110	CPU Address<7>, 00

[1]Byte enable set to 0 indicates that byte lane carries meaningful data.

[2]In PCI sparse memory space, PCI Address <1..0> are always 00.

(continued on next page)

Table 4–7 (Cont.) PCI Sparse Memory Space Byte Enable Generation

Length	CPU Address <6..5>	CPU Address <4..3>	PCI Byte Enable[1]	PCI Address<2..0>[2]
	01	00	1101	CPU Address<7>, 00
	10	00	1011	CPU Address<7>, 00
	11	00	0111	CPU Address<7>, 00
Word	00	01	1100	CPU Address<7>, 00
	01	01	1001	CPU Address<7>, 00
	10	01	0011	CPU Address<7>, 00
	11	01	Illegal[3]	—
Tribyte	00	10	1000	CPU Address<7>, 00
	01	10	0001	CPU Address<7>, 00
	10	10	Illegal[3]	—
	11	10	Illegal[3]	—
Longword	00	11	0000	CPU Address<7>, 00
Longword	01	11	Illegal[3]	—
Longword	10	11	Illegal[3]	—
Quadword	11	11	0000	000

[1]Byte enable set to 0 indicates that byte lane carries meaningful data.

[2]In PCI sparse memory space, PCI Address <1..0> are always 00.

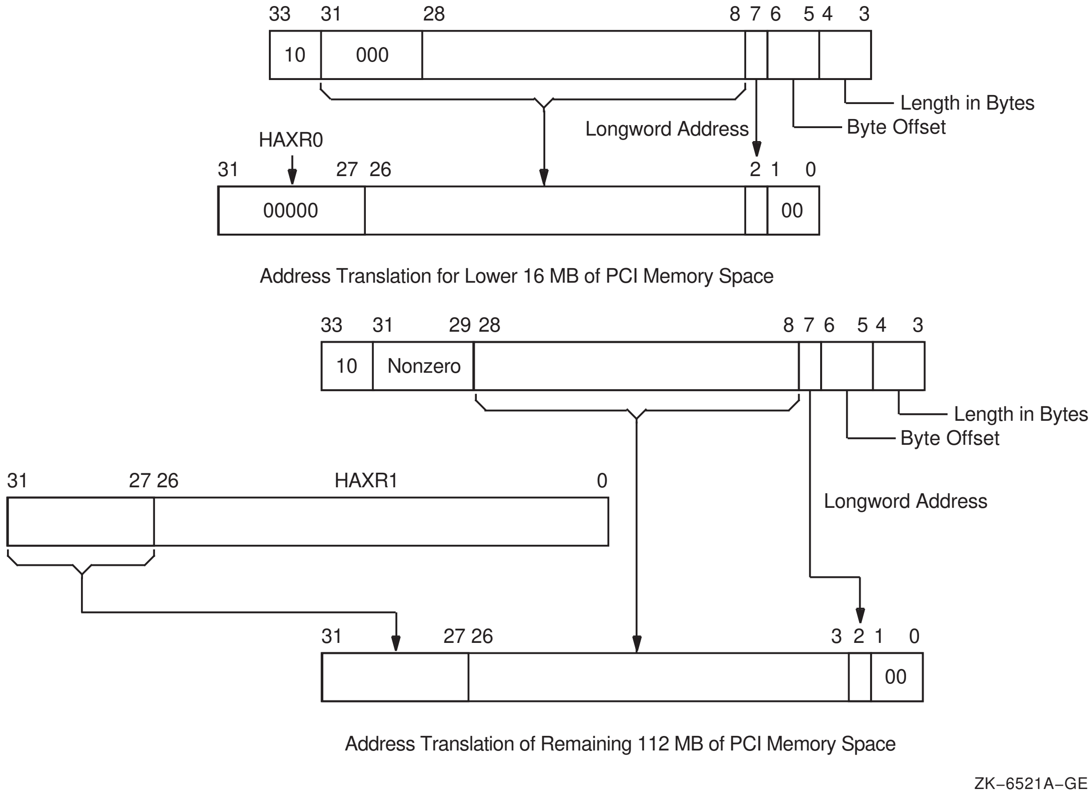
[3]These combinations are architecturally illegal. If there is an access with this combination of address<6..3>, the 21071-DA will respond to the transactions but the results are unpredictable.

Note that **sysBus<33..5>** are directly available from the 21064 or 21064A processor. Bits **sysBus<4..3>** are derived form the longword masks (**cpucwmask<7..0>**). On read transactions, the 21064 or 21064A sends out **sysBus<4..3>** on **cpucwmask<1..0>**.

On write transactions the relationship between **cpucwmask<7..0>** and **sysBus<4..3>** is as follows:

If **cpucwmask<1..0>** is nonzero, **sysBus<4..3>** is 00.
If **cpucwmask<3..2>** is nonzero, **sysBus<4..3>** is 01.
If **cpucwmask<5..4>** is nonzero, **sysBus<4..3>** is 10.

Figure 4–3 PCI Memory Space Address Translation



If **cpucwmask<7..6>** is nonzero, **sysBus<4..3>** is 11.

Accesses in this space are no more than a quadword. Software must ensure that the processor does not merge consecutive writes in its write buffers by using memory barriers after each write. Architecturally, if a byte, word, tribyte or longword is written on the PCI, an STL instruction must be executed to the lower longword in the corresponding quadword address. An STQ or STL instruction to the upper longword is not allowed.

One bit pair of **cpucwmask<1..0>**, **<3..2>**, **<5..4>**, and **<7..6>** must have value of 01 (binary). The other fields must be 00. The location of the 01 field indicates whether the reference is byte, word, tribyte, or longword (respectively) in length.

Similarly, if a quadword is written to the PCI, software must execute an STQ instruction to the corresponding address. The only legal value on **cpucwmask<7..6>** in sparse space is 11000000.

If a byte, word, tribyte, or longword is read from the PCI, an LDL instruction must be executed to the lower longword in the corresponding quadword address. An LDL instruction to the upper longword or LDQ instruction returns the wrong data. If a quadword is read from the PCI, software must use an LDQ instruction. An LDL instruction returns wrong data.

4.1.9 PCI Dense Memory Space—3 0000 0000 .. 3 FFFF FFFF

PCI dense memory space is typically used for data buffers on the PCI and has the following characteristics:

- There is a one-to-one mapping between CPU addresses and PCI addresses. A longword address from the CPU maps to a longword on the PCI. (Thus the name *dense space* as opposed to PCI sparse memory space.)
- Byte or word accesses are not allowed in this space. Minimum access granularity is a longword. The maximum transfer length implemented by the 21072 chipset is a cache line (32 bytes) on writes, and a quadword on reads.
- Read prefetching is allowed in this space; additional reads have no side effects. The 21064 or 21064A does not specify a longword address on read transactions; it only specifies a quadword address. Therefore, reads in this space are always performed as a quadword read with a burst length of two on the PCI.
- Writes to addresses in this space can be buffered in the 21064 or 21064A. The 21072 chipset supports a maximum burst length of 8 on the PCI corresponding to a cache line of data.

The address generation in dense space is as follows:

- Bits **sysBus<31..5>** is sent out on **ad<31..5>**
- On read transactions, **ad<4..3>** is generated from **cpucwmask<1.0>**; **ad<2>** is always 0.
- On write transactions, **ad<4..2>** is generated from **cpucwmask<7..0>**. If the lower longword is to be written, **ad<2>** is 0; if the lower longword is masked out and the upper longword is to be written, **ad<2>** is 1. The number of longwords written on the PCI is directly obtained from **cpucwmask<7..0>**. Any combination of **cpucwmask<7..0>** is allowed by the 21072 chipset.

Note

If the cache line written by the processor has holes, that is, some of the longwords are masked out, the corresponding transfer is still performed on the PCI with disabled byte enables. Downstream bridges must be able to deal with disabled byte enables on the PCI during write transactions.

4.2 PCI to Physical Memory Addressing

Incoming 32-bit memory addresses are mapped to the 34-bit physical memory addresses. The 21071-DA allows two regions in PCI memory space to be mapped to system memory with two programmable address windows. The mapping from the PCI address to the physical address can be direct (physical mapping with an extension register) or scatter gather mapped (virtual). These two address windows are referred to as the PCI target windows.

Each window has three registers associated with it: PCI base register, PCI mask register, and the translated base register. The register descriptions are contained in Appendix A.

The PCI mask register provides a mask corresponding to **ad<31..20>** of an incoming PCI address. The size of each window can be programmed from 1 MB to 4 GB (in powers of 2) by masking bits of the incoming PCI address using the PCI mask register as specified in Table 4–8.

Table 4–8 PCI Target Window Enables

PCI_MASK<31..20>[1]	Window Size	Value of n[2]
0000 0000 0000	1 MB	20
0000 0000 0001	2 MB	21
0000 0000 0011	4 MB	22
0000 0000 0111	8 MB	23
0000 0000 1111	16 MB	24

[1]Combinations of bits not shown in **pci_mask<31..20>** are not supported.

[2]Depending on the target window size, only the incoming address bits **<31..n>** are compared with bits **<31..n>** of the PCI base registers as shown in Figure 4–4. If n = 20 to 32, no comparison is performed; n is also used in Figure 4–6.

(continued on next page)

Table 4–8 (Cont.) PCI Target Window Enables

PCI_MASK<31..20>[1]	Window Size	Value of n[2]
0000 0001 1111	32 MB	25
0000 0011 1111	64 MB	26
0000 0111 1111	128 MB	27
0000 1111 1111	256 MB	28
0001 1111 1111	512 MB	29
0011 1111 1111	1 GB	30
0111 1111 1111	2 GB	31
1111 1111 1111	4 GB[3]	32

[1]Combinations of bits not shown in **pci_mask<31..20>** are not supported.

[2]Depending on the target window size, only the incoming address bits **<31..n>** are compared with bits **<31..n>** of the PCI base registers as shown in Figure 4–4. If n = 20 to 32, no comparison is performed; n is also used in Figure 4–6.

[3]When this combination is chosen, the WENB bit in the other PCI base register must be cleared, otherwise the two windows will overlap.

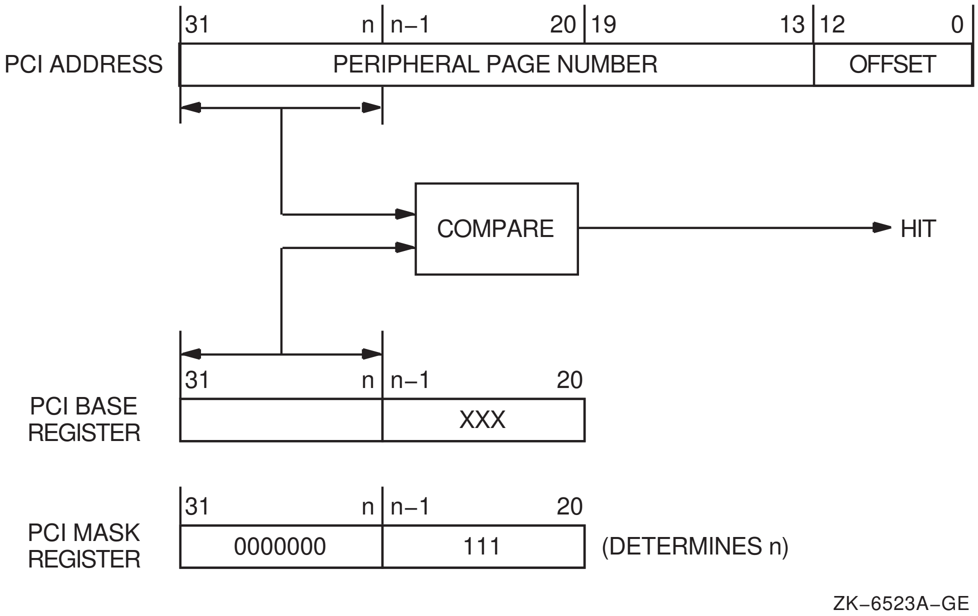
Based on the value of the PCI mask register, the unmasked bits of the incoming PCI address are compared with the corresponding bit of each window's PCI base register. If the base registers and the incoming PCI address match, the incoming PCI address has hit that target window, otherwise it missed that window. A window enable bit (WENB) is provided in the PCI base register of each window to allow them to be independently enabled and disabled.

The PCI target windows must be programmed such that the PCI address ranges do not overlap. The compare scheme between the incoming PCI address and the PCI base register (together with the PCI mask register) is shown in Figure 4–4.

Note

The window base addresses must be on naturally aligned address boundaries depending on the size of the window.

Figure 4–4 PCI Target Window Compare Scheme



When an address match occurs with a PCI target window, the 21071-DA translates the 32-bit PCI address to a 34-bit processor byte address (actually a 29-bit hexaword address). The translated address is generated in one of two ways as determined by the scatter gather enable (SGEN) bit of the PCI base register of the associated window.

If SGEN is cleared, the DMA address is direct mapped, and the translated address is generated by concatenating bits from the matching window translated base address register with bits from the incoming PCI address. The PCI mask register determines which bits of the translated base register and PCI address are used to generate the translated address as shown in Table 4–9

Note that the unused bits of the translated base register must be cleared for correct operation. Because system memory is located in the lower half of the CPU address space, **sysBus<33>** is always zero. Bits **sysBus<32..5>** are obtained from the translated base register.

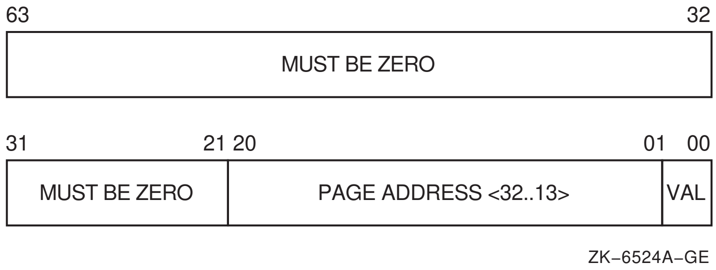
Table 4–9 PCI Target Address Translation—Direct Mapped

PCI_MASK<31..20>	Translated Address<32..5>
0000 0000 0000	T_BASE<32..20>:PCI_ADDRESS<19..5>
0000 0000 0001	T_BASE<32..21>:PCI_ADDRESS<20..5>
0000 0000 0011	T_BASE<32..22>:PCI_ADDRESS<21..5>
0000 0000 0111	T_BASE<32..23>:PCI_ADDRESS<22..5>
0000 0000 1111	T_BASE<32..24>:PCI_ADDRESS<23..5>
0000 0001 1111	T_BASE<32..25>:PCI_ADDRESS<24..5>
0000 0011 1111	T_BASE<32..26>:PCI_ADDRESS<25..5>
0000 0111 1111	T_BASE<32..27>:PCI_ADDRESS<26..5>
0000 1111 1111	T_BASE<32..28>:PCI_ADDRESS<27..5>
0001 1111 1111	T_BASE<32..29>:PCI_ADDRESS<28..5>
0011 1111 1111	T_BASE<32..30>:PCI_ADDRESS<29..5>
0111 1111 1111	T_BASE<32..31>:PCI_ADDRESS<30..5>
1111 1111 1111	T_BASE<32>:PCI_ADDRESS<31..5>

If the SG bit is set, the translated address is generated by a table lookup. The incoming PCI address indexes a table stored in system memory. The table is referred to as a scatter gather map (SG map). The translated address base register specifies the starting address of the SG map. Bits of the incoming PCI address are used as an offset from the base of the map. The map entry provides the physical address of the page.

Each SG map entry maps an 8 KB page of PCI address space into an 8 KB page of processor address space. Each SG map entry is a quadword. Each entry has a valid bit in position 0. Address bit **ad<13>** is at bit position 1 of the map entry. Because the 21072 only implements valid memory addresses up to 6 GB, bits **ad<63..21>** of the SG map entry must be programmed to 0. Bits **ad<21..1>** of the SG entry generate the physical page address. This is appended to bits **ad<12..5>** of the incoming PCI address to generate the memory address placed on the SysBus. Figure 4–5 shows the SG map entry.

Figure 4–5 SG Map Page Table Entry in Memory



The size of the SG map table is determined by the size of the PCI target window as defined by the PCI mask register as specified in Table 4–10. Because the SG map is located in system memory, **sysBus<33>** is always zero. Bits **sysBus<32..2>** are obtained from the translated base register and the PCI address.

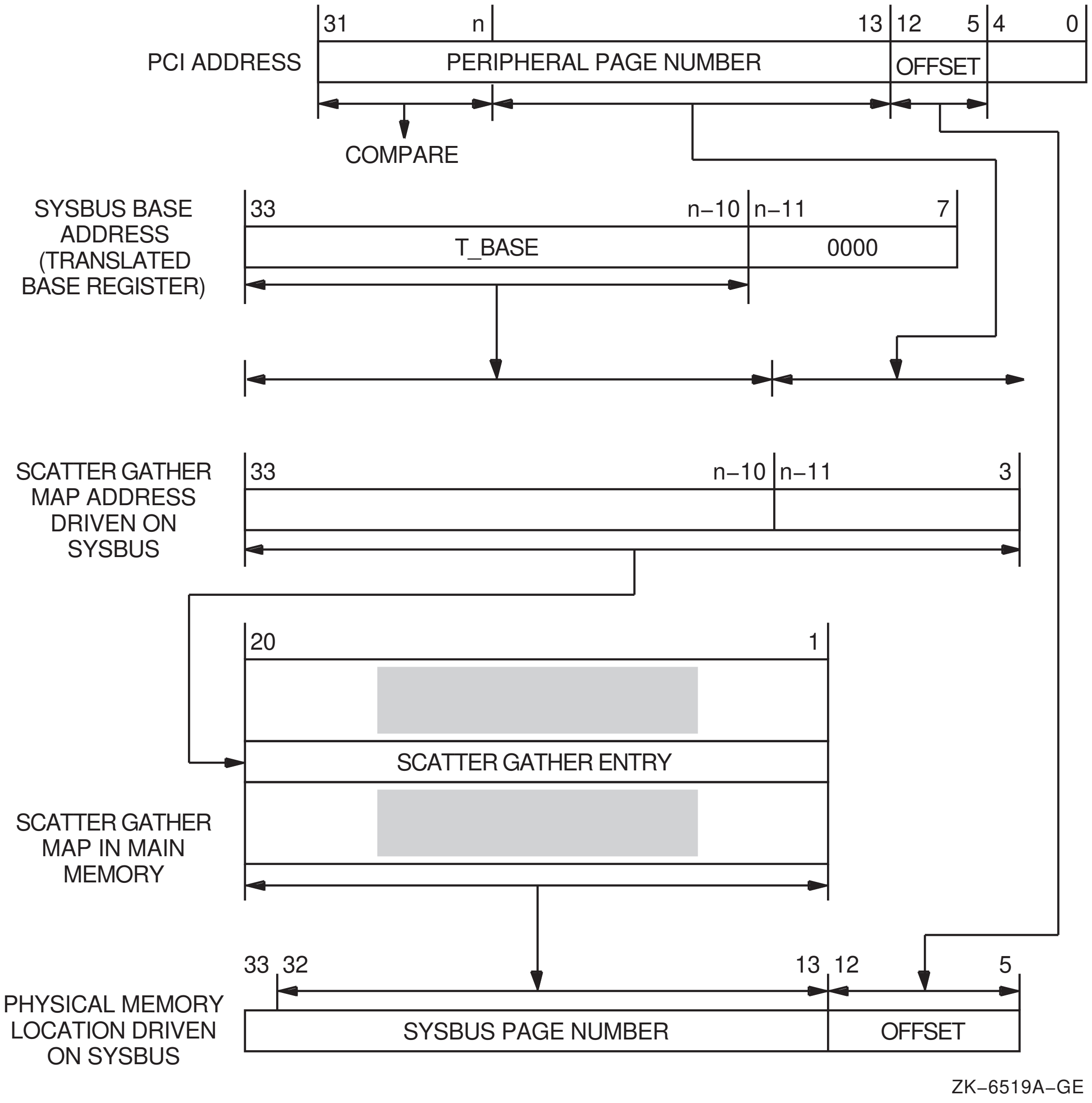
Table 4–10 Scatter Gather Map Address

PCI_MASK<31..20>	SG Map Table Size	SG Map Address<32..3>
0000 0000 0000	1 K byte	T_BASE<32..10>:PCI_ADDRESS<19..13>
0000 0000 0001	2K bytes	T_BASE<32..11>:PCI_ADDRESS<20..13>
0000 0000 0011	4K bytes	T_BASE<32..12>:PCI_ADDRESS<21..13>
0000 0000 0111	8K bytes	T_BASE<32..13>:PCI_ADDRESS<22..13>
0000 0000 1111	16K bytes	T_BASE<32..14>:PCI_ADDRESS<23..13>
0000 0001 1111	32K bytes	T_BASE<32..15>:PCI_ADDRESS<24..13>
0000 0011 1111	64K bytes	T_BASE<32..16>:PCI_ADDRESS<25..13>
0000 0111 1111	128K bytes	T_BASE<32..17>:PCI_ADDRESS<26..13>
0000 1111 1111	256K bytes	T_BASE<32..18>:PCI_ADDRESS<27..13>
0001 1111 1111	512K bytes	T_BASE<32..19>:PCI_ADDRESS<28..13>
0011 1111 1111	1M bytes	T_BASE<32..20>:PCI_ADDRESS<29..13>
0111 1111 1111	2M bytes	T_BASE<32..21>:PCI_ADDRESS<30..1>
1111 1111 1111	4M bytes	T_BASE<32..22>:PCI_ADDRESS<31..13>

Figure 4–6 shows the entire translation process from the PCI address to physical address on a window implementing SG mapping and described in the following procedure:

1. Bits **ad<12..5>** of the PCI address directly generate the page offset.
2. The relevant bits of the PCI address (as specified by the window mask register, depending on the size of the window) to generate the offset into the SG map.
3. The relevant bits of the translated base register indicate the base address of the SG map.
4. The map base is appended to the map offset to generate the address of the corresponding SG entry.
5. Bits **<20..1>** of the map are used to generate the physical page address, which is appended to the page offset to generate the PCI address.
6. Bit **<0>** is the valid bit for the page table entry.

Figure 4–6 SG Map Translation of PCI to SysBus Address



5

Board Requirements and Parameters

This chapter describes the evaluation board power and environmental requirements, and physical board parameters.

5.1 Power Requirements

The EB64+ derives its main dc power from a user-supplied, industry-standard PC power supply. The board has a total power dissipation of 103 Watts excluding PCI and ISA devices. Table 5–1 lists the power requirements of each dc supply voltage.

The power supply must supply a **dcok** signal to the system reset logic. Refer to Section 3.13 and schematic page **eb64+.47** for additional information.

Table 5–1 Power Supply DC Current Requirements

Voltage	Current
+5 Vdc	18 A
-5 Vdc	0 A
+12 Vdc	1 A
-12 Vdc	0.1 A

Caution: Fan sensor required

The 21064 or 21064A cooling fan *must* have a built-in sensor that drives a signal if the airflow stops. The sensor is connected to J54.

When the sensor signal is generated, it places the system into $\overline{}\ dcok$ mode. This protects the 21064 or 21064A under fan-failure conditions because the 21064 and 21064A dissipate less heat in $\overline{}\ dcok$ mode.

5.2 Environmental Characteristics

The board environmental characteristics are:

- Operating temperature range of 15°C (59°F) to 32°C (90°F)
- Storage temperature range of -55°C (-67°F) to 125°C (257°F)

5.3 Physical Board Parameters

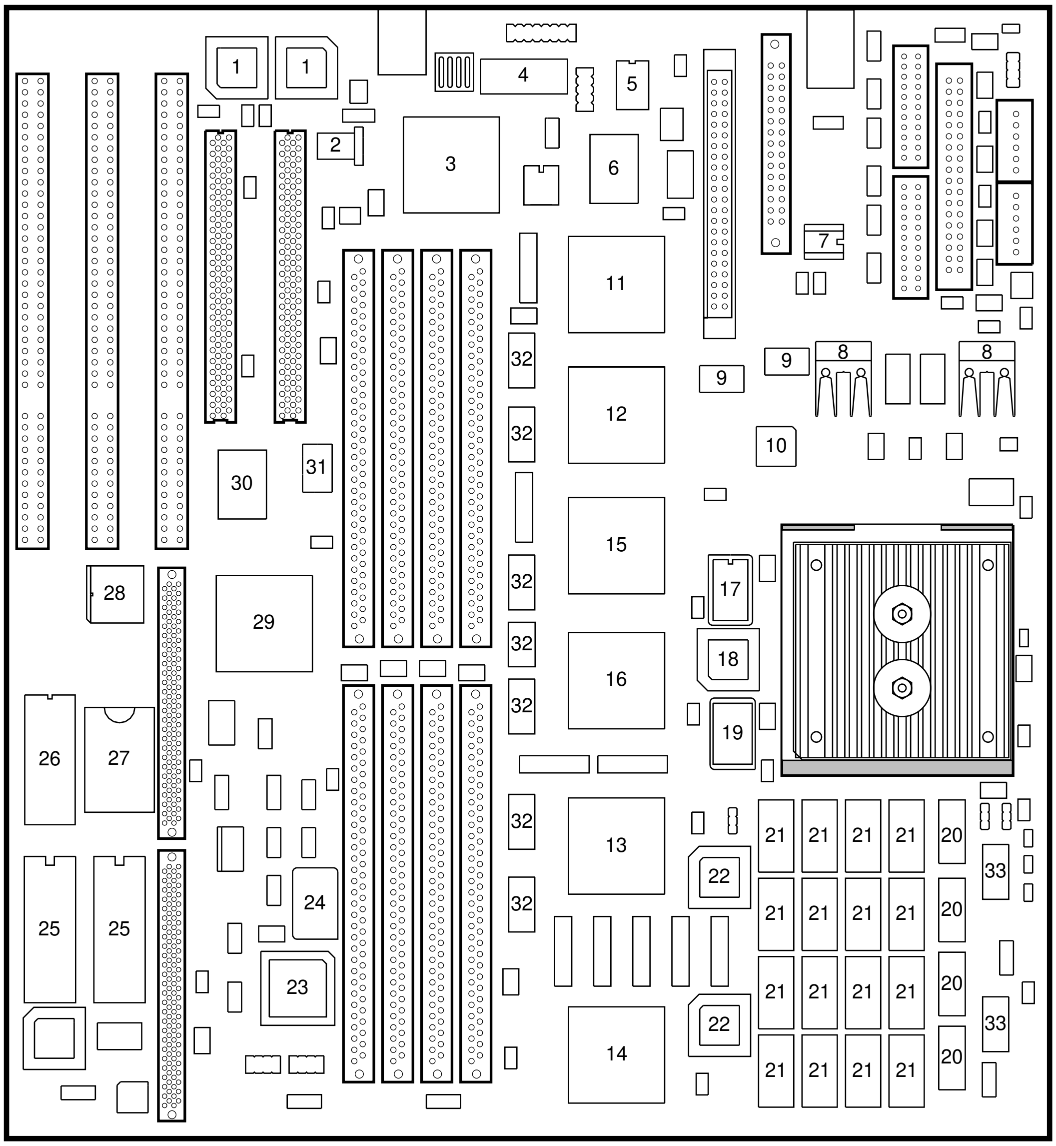
The EB64+ board consists of a 6-layer printed wiring board. The board is populated with integrated circuit packages together with supporting active and passive components. The EB64+ is a standard, full-size PC AT board with dimensions of:

- Width: 30.48 cm (12.0 inches ±.0005 inch)
- Length: 33.02 cm (13.0 inches ±.0005 inch)

The board can be used in certain desktop systems that have adequate clearance for the 21064 or 21064A and regulator heatsinks. All ISA and PCI expansion slots are usable in standard desktop or deskside enclosures.

Figure 5–1 shows the board and component outlines, and identifies the major components. The components are described in Table 5–2. Refer to Chapter 2 for jumper and connector locations.

Figure 5–1 Major Board Component Layout



LJ-03823.AI

Table 5–2 Major Board Component Descriptions

Number	Component Description
1	PCI Arbiter PALs (PCI arbitration)
2	20 MHz Ethernet controller (21040) oscillator crystal
3	DECchip 21040 (PCI to Ethernet interface)
4	Ethernet transformer (10BASE-T)
5	40 MHz SCSI clock
6	NCR 53C810 (PCI to SCSI interface)
7	Serial ROM (initialization code)
8	5 volt to 3 volt regulators
9	PLL clock drivers
10	S4402 PLL clock
11	DECchip 21071-BA3
12	DECchip 21071-BA2
13	DECchip 21071-BA1
14	DECchip 21071-BA0
15	DECchip 21071-DA
16	DECchip 21071-CA
17	400 MHz ECL clock oscillator
18	Triquint PLL clock
19	40 MHz clock oscillator
20	Tag cache
21	Data cache
22	Bcache PALs
23	MACH 210A interrupt controller
24	14 MHz PCI to ISA bridge (SIO) oscillator
25	Boot and diagnostic PROMs
26	DS1225Y 8KB system support RAM
27	Dallas DS1287A TOY
28	Intel 8242 mouse and keyboard controller

(continued on next page)

Table 5–2 (Cont.) Major Board Component Descriptions

Number	Component Description
29	PCI to ISA bridge (Intel 82378IB or 82378ZB)
30	National 87312 combination chip
31	24 MHz National combination chip oscillator
32	DRAM buffers
33	Bcache address drivers

A

System Register Descriptions

This appendix describes the control and status registers (CSRs) of the DECchip 21071-CA (Section A.1 and Section A.2), and DECchip 21071-DA (Section A.3).

A.1 DECchip 21071-CA CSR Descriptions

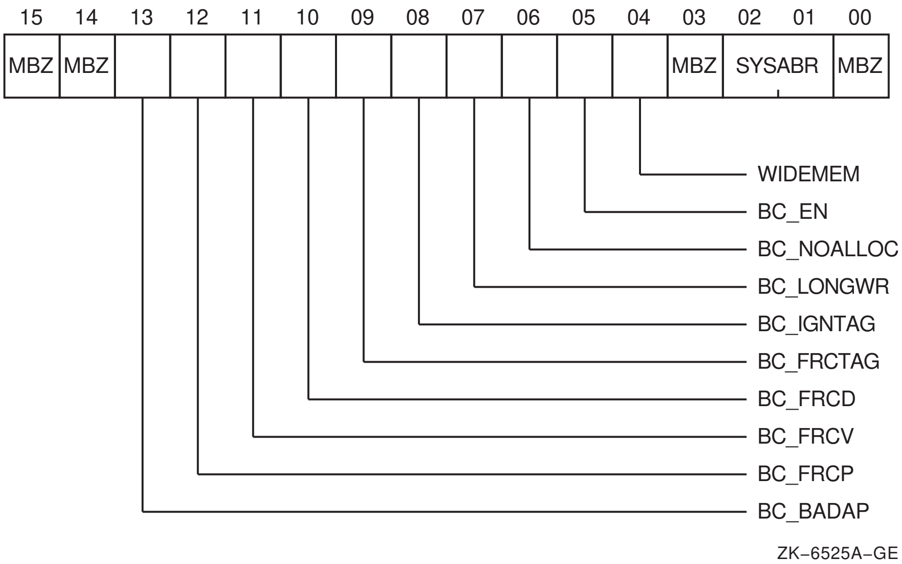
The CSRs are 16 bits wide and addressed on cache-line boundaries. Writes to read-only registers could result in unpredictable behavior; reads are nondestructive. Only zeros should be written to unspecified bits within a CSR. Only bits **<15..0>** of each register are defined. Other bits are undefined. CSRs are initialized as specified in the register descriptions.

Register addresses are specified in Table 4–2.

A.1.1 General Control Register

The general control register contains status information that affects the major operational modes of the 21071-CA. See Figure A–1.

Figure A–1 General Control Register



<15..14>: Reserved—Must be zero.

<13>: BC_BADAP—Bcache Force Bad Address Parity. Type: RW, reset: 0. When set, the tag address parity will be loaded as bad (independent of the BC_FRCTAG bit).

<12>: BC_FRCP—Bcache Force Parity. Type: RW, reset: 0. When set, the parity bit will be set on the next cache fill.

<11>: BC_FRCV—Bcache Force Valid. Type: RW, reset: 0. When set, the valid bit will be set on the next cache fill.

<10>: BC_FRCD—Bcache Force Dirty. Type: RW, reset: 0. When set, the dirty bit will be set on the next cache fill.

<9>: BC_FRCTAG—Bcache Force Tag. Type: RW, reset: 0. When set, the Bcache will be probed for victims, and the line will be invalidated using the values in the BC_FRCD, BC_FRCV, and BC_FRCP. CSRs will be used as the tag controls. Although the line is invalidated (assuming BC_FRCV is reset), the data is loaded into the cache, and will be returned to the CPU as cacheable.

Used for diagnostic testing of the cache RAM, and for flushing the cache by setting this bit, clearing BC_FRCV, and cycling through the address range present in the cache.

<8>: BC_IGNTAG—Bcache Ignore Tag. Type: RW, reset: 0. When set, Bcache probes will act as if the valid bit was invalid. All tag results will be ignored, (and any victims will be lost.) Tag and address parity will be ignored. May be used to fill the cache with valid data.

<7>: BC_LONGWR—Bcache Long Writes. Type: RW, reset: 0. When set, two sysBus cycles are required to write to the cache data RAMs.

<6>: BC_NOALLOC—Bcache No Allocate Mode. Type: RW, reset: 0. When set, CPU writes to cacheable memory space will not be allocated into the cache.

<5>: BC_EN—Bcache Enable. Type: RW, reset: 0. When clear, the Bcache is disabled and the cache state machine will not probe the cache.

<4>: WIDEMEM—Wide memory Size. Type: RO. Reads the status of the wideMEM input pin. Returns 1 for the 128-bit memory interface.

<3>: Reserved Type: Must be zero.

<2..1>: SYSARB - Type: RW, reset: 0. DMA Arbitration mode. Determines arbitration scheme for sysBus transactions.

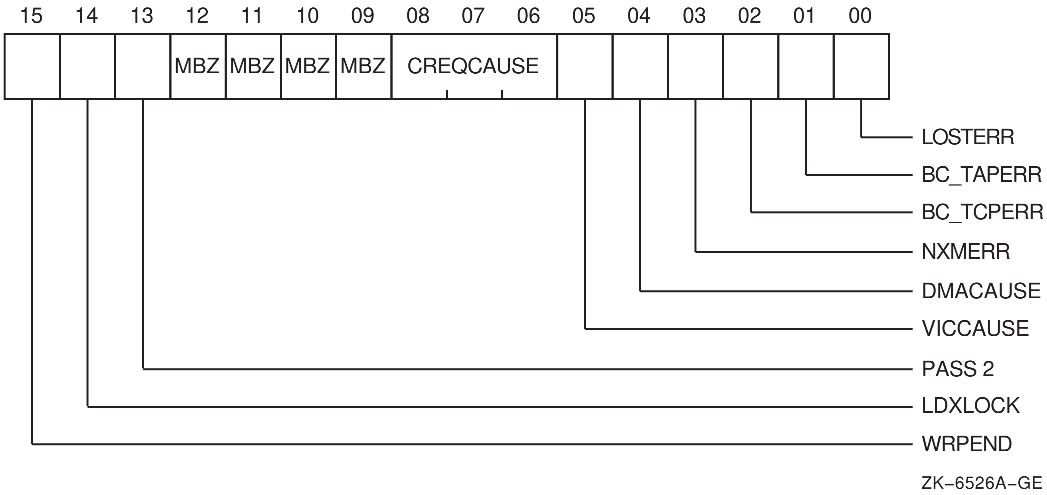
Value	Meaning
0X	CPU priority
10	DMA priority
11	DMA strong priority

<0>: Reserved Type: Must be zero.

A.1.2 Error and Diagnostic Status Register

The error and diagnostic status register contains read-only status information for diagnostics and for error analysis. The occurrence of an error sets one or more error bits (BC_TAPERR, BC_TCPERR, NXMERR) and locks the address of the error. After the address is locked, any additional error will set LOSTERR and will not affect the address or other error bits (BC_TAPERR, BC_TCPERR, NXMERR). Clearing all of the error bits (not the LOSTERR bit) unlocks the address. See Figure A–2.

Figure A–2 Error and Diagnostic Status Register



<15>: WRPEND—Write Pending. Type: RO, reset: 0. When set, indicates that valid write data is stored in the write buffer.

<14>: LDXLLOCK—LDx_L Locked. When set, indicates that the lock bit for LDx_L is set, and that the next STx_C may succeed. Writing to any CSR or I/O space location clears this lock bit.

<13>: PASS 2—Type: RO. Chip version reads low on pass1 and high on pass2.

<12..9>: RESERVED—Must be zero.

<8..6>: CREQCAUSE—Cycle Request that Caused Error. Type: RO. Indicates the DMA or CPU cycle request type that caused the error. Copy of either the cpuCReq or ioCmd lines, depending on the DmaCause CSR. Locked with the error address. Only valid when a error is indicated on BC_TAPERR, BC_ TCPERR, or MEMERR.

<5>: VICCAUSE—Victim Write Caused Error. Type: RO. When set, indicates that a NXM error was caused by a victim write transaction. Undefined for other types of errors. Locked with the error address. Valid only when an error is indicated on BC_TAPERR, BC_TCPERR, or MEMERR.

<4>: DMACAUSE—DMA Transaction Caused Error. Type: RO. When set, indicates that the BC_TAPERR, BC_TCPERR, or NXMERR was caused by a DMA transaction. Locked with the error address.Valid only when an error is indicated on BC_TAPERR, BC_TCPERR, or MEMERR.

<3>: NXMERR—Nonexistent Memory Error. Type: RW1C, reset: 0. When set, indicates that a read or write occurred to an invalid address that does not map to any memory bank, CSR, or I/O quadrant. Set only when address is unlocked.

<2>: BC_TCPERR—Bcache Tag Control Parity. Type: RW1C, reset: 0. When set, indicates that a tag probe encountered bad parity in the tag control RAM. Set only when address is unlocked.

<1>: BC_TAPERR—Bcache Tag Address Parity. Type: RW1C, reset: 0. When set, indicates that a tag probe encountered bad parity in the tag address RAM. Set only when address is unlocked.

<0>: LOSTERR—Lost error, multiple errors. Type: RW1C, reset: 0. When set, indicates that additional errors occurred when an error address was already locked. No address or cause information is latched for the error.

A.1.3 Tag Enable Register

The tag enable register is a read/write register; this register indicates which bits of the cache tag are to be compared with **sysadr<33..5>**. If a bit is 1, the corresponding bits in **sysadr<33..5>** and **systag<31..17>** are compared. If a bit is 0, there is no comparison for those bits, and the **systag** bit is assumed to be tied low on the module (through a resistor). Bits **<15..1>** in the register represent **systag<31..17>**. This register is not initialized.

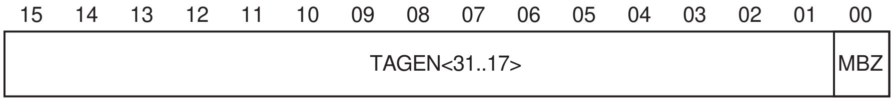
There is no requirement that the upper bits of TagEn be set. An implementation that does not allow the full 4 GB of cacheable memory to be installed may choose to mask off upper bits of TagEn, and save having to store a bit of the tag address in the tag address RAM.

To construct the tagEn bits refer to Figures Table A–1 and Table A–2. The value shown in Table A–1 (based on the cache size) is ANDed with the value in Table A–2 (based on the maximum cacheable system memory.) For example, a system with a 16 MB cache, and a maximum of 1 GB of cacheable memory would program:

1111 1111 0000 000X ANDed with
0011 1111 1111 111X gives
0011 1111 0000 000X which is put into tagEn.

See Figure A–3, Table A–1 and Table A–2.

Figure A–3 Tag Enable Register



ZK–6527A–GE

Table A–1 Cache Size Tag Enable Values

tagEn<15:0>	Compared	Cache Size
0000 0000 0000 000X	None	4 GB cache
1000 0000 0000 000X	<31..31>	2G byte cache
1100 0000 0000 000X	<31..30>	1G byte cache
1110 0000 0000 000X	<31..29>	512M byte cache
1111 0000 0000 000X	<31..28>	256M byte cache
1111 1000 0000 000X	<31..27>	128M byte cache
1111 1100 0000 000X	<31..26>	64M byte cache
1111 1110 0000 000X	<31..25>	32M byte cache
1111 1111 0000 000X	<31..24>	16M byte cache
1111 1111 1000 000X	<31..23>	8M byte cache
1111 1111 1100 000X	<31..22>	4M byte cache
1111 1111 1110 000X	<31..21>	2M byte cache
1111 1111 1111 000X	<31..20>	1M byte cache
1111 1111 1111 100X	<31..19>	512K byte cache
1111 1111 1111 110X	<31..18>	256K bytes cache
1111 1111 1111 111X	<31..17>	128K bytes cache

Table A–2 Maximum Memory Tag Enable Values

tagEn<15:0>	Compared	Memory Size
1111 1111 1111 111X	<31..17>	4G byte memory

(continued on next page)

Table A–2 (Cont.) Maximum Memory Tag Enable Values

tagEn<15:0>	Compared	Memory Size
0111 1111 1111 111X	<30..17>	2G byte memory
0011 1111 1111 111X	<29..17>	1G byte memory
0001 1111 1111 111X	<28..17>	512M byte memory
0000 1111 1111 111X	<27..17>	256M byte memory
0000 0111 1111 111X	<26..17>	128M byte memory
0000 0011 1111 111X	<25..17>	64M byte memory
0000 0001 1111 111X	<24..17>	32M byte memory
0000 0000 1111 111X	<23..17>	16M byte memory
0000 0000 0111 111X	<22..17>	8M byte memory
0000 0000 0011 111X	<21..17>	4M byte memory
0000 0000 0000 111X	<19..17>	1M byte memory
0000 0000 0000 011X	<18..17>	512K byte memory
0000 0000 0000 001X	<17..17>	256K byte memory
0000 0000 0000 000X	None	128 KB memory

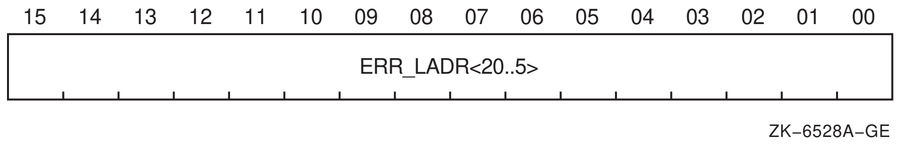
Bit <1> is reserved and must be zero.

A.1.4 Error Low Address Register

The error low address register locks the low order bits of the sysBus address that caused the error that set the BC_TAPERR, BC_TCPERR, or NXMERR bit in the error and diagnostic status register. If a victim read caused the error, the victim address is not latched; rather, the address of the transaction is latched. Bits **<15..0>** represent **sysadr<20..5>**. This register is read only. It is not initialized and is valid only when an error is indicated.

See Figure A–4.

Figure A–4 Error Low Address Register



A.1.5 Error High Address Register

The error high address register locks the high-order bits of the sysBus address. Bits **<12..0>** represent **sysadr<33..21>**. This register is read only. It is not initialized and is only valid when an error is indicated.

Bits **<15..13>** are reserved and must be zero.

See Figure A–5.

Figure A–5 Error High Address Register

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MBZ	MBZ	MBZ	ERR_HADR<33..21>												

ZK−6529A−GE

A.1.6 LDx_L Low Address Register

The LDx_L low address register stores the low-order bits of the last locked address. Bits **<15..0>** in the register represent **sysadr<20..5>**. This register is read only and is not initialized.

See Figure A–6.

Figure A–6 LDx_L Low Address Register

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
LDXL_LARD<20..5>															

ZK−6530A−GE

A.1.7 LDx_L High Address Register

The LDx_L high address register stores the high-order bits of the locked address. Bits **<12..0>** in the register represent **sysadr<33..21>**. This register is read only and is not initialized.

See Figure A–7.

Figure A–7 LDx_L High Address Register

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MBZ	MBZ	MBZ	LDXL_HARD<33..21>												

ZK–6531A–GE

A.2 Memory Control Registers

The following registers on the 21071-CA control memory configuration and timing. Each bankset of memory has one configuration register and two timing registers. The global timing register and refresh timing register apply to all banksets. The video frame pointer is used for video transactions to bankset8.

A.2.1 Video Frame Pointer Register

The video frame pointer register contains address information that points to the beginning of the video frame buffer. The video frame pointer is loaded into the video display pointer at the beginning of each full serial transfer to bankset8. This register is not initialized. See Figure A–8.

Figure A–8 Video Frame Pointer Register

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MBZ	VFP_SUBBANK	VFP_ROWADR									VFP_COLADR				

ZK–6532A–GE

<15>: Reserved—Must be zero.

<14>: VFP_SUBBANK—Video Frame Subbank Pointer. Type: RW. Subbank for the start of the frame buffer. If the subbank is enabled by setting s8_SubEna in the bankset8 configuration register, setting the VFP_SUBBANK bit causes the 21071-CA to assert v0-v1_rasb8_l instead of v0-v1_ras8_l on full serial register loads. VFP_SUBBANK is ignored if S8_SUBENA is cleared.

<13..5>: VFP_ROW<8..0>—Video Frame Row Address Pointer. Type: RW. Row address of the start of the frame buffer.

<4..0>: VFP_COL—Video Frame Column Address Pointer. Type: RW. Used as column address **<6..2>** for all serial register loads.

A.2.2 Presence Detect Low-Data Register

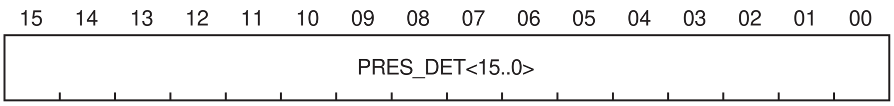
The presence detect low-data register stores the low-order bits of the presence detect information that was shifted in after reset. Bits **<15..0>** in the register represent data bits **<15..0>** that were shifted in.

Note

After deassertion of reset, it takes 148 system clock cycles for this data to become valid.

See Figure A–9.

Figure A–9 Presence Detect Low Data Register



ZK−6533A−GE

A.2.3 Presence Detect High-Data Register

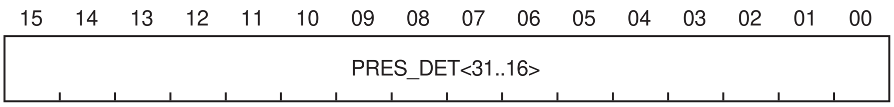
The presence detect high-data register stores the high order bits of the presence detect information that was shifted in after reset. Bits **<15..0>** in the register represent data bits **<31..16>** that were shifted in.

Note

After deassertion of reset, it takes 148 system clock cycles for this data to become valid.

See Figure A–10.

Figure A–10 Presence Detect High-Data Register



ZK−6534A−GE

A.2.4 Base Address Registers

Each memory bankset has a corresponding base address register. The bits in this register are compared with the incoming sysadr to determine the bankset being addressed. The contents of this register are validated by setting the valid bit in the configuration register of that bankset. The number of bits that are compared depends on the size of the corresponding bankset. Banksets 7 to 0 have an 11-bit field, limiting the minimum DRAM bankset size to 8 MB. Bits **<15..5>** in the register correspond to **sysadr<33..23>**.

Bankset8, which can contain video RAMs, and has a minimum size of 1 MB, has the same 11 bit field, where bits **<15..5>** in the register correspond to **sysadr<33..23>** and **sysadr<22..20>** are compared with zero.

The base address of each bankset must begin on a naturally aligned boundary (so for a bankset with 2^n addresses; the n least significant bits must be zero).

Bankset8 must be placed on an aligned 8 MB boundary for bank sizes less than or equal to 8 MB.

If bankset8 has parity checking disabled (**s8_check** clear), then bankset8 must be mapped into noncacheable space (**s8_baseadr<32>** set).

<4..0>: Reserved—Must be zero.

See Figure A–11.

Figure A–11 BankSet0 Base Address Register

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
S0_BASEADR<33..23>											MBZ	MBZ	MBZ	MBZ	MBZ

ZK–6535A–GE

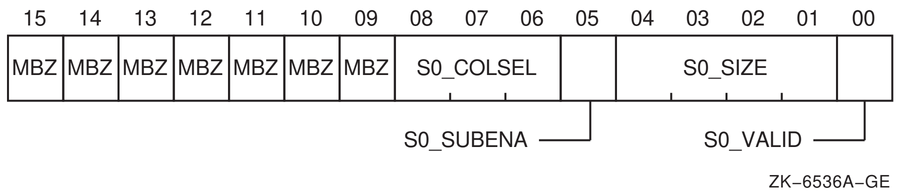
A.2.5 Configuration Registers

Each memory bankset has a corresponding configuration register. This register contains mode bits and bits for memory address generation, and bankset decoding. Bankset0 to 7 have the same limits on bankset size, and type of DRAMS used. The format of the configuration register is the same for all these. Bankset8 is the VRAM bank and supports different minimum DRAM sizes and configurations. Therefore, its configuration register is different.

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

See Figure A–12.

Figure A–12 BankSet 0 to 7 Configuration Register



<15..9>: Reserved—Must be zero.

<8..6>: S0_COLSEL<2..0> - Column Address Selection. Type: RW. Indicates the number of valid column bits expected at the DRAMs. Used together with memory width information to generate row or column addresses. Memory interface width is set at 128 bits. See Table A–3.

Table A–3 S0_ColSel<2..0> Field Codes

S0_ColSel<2..0>	Row, Column Bits
000	12, 12
001	12, 10 or 11, 11
010	Reserved
011	10, 10
1XX	Reserved

<5>: S0_SUBENA—Enable subbanks. Type: RW, reset: 0. When set, subbanks are enabled and determined according to Table A–4. When clear, subbanks are disabled, and the **b0-b3_rasb0_l** pins will be asserted only during refreshes.

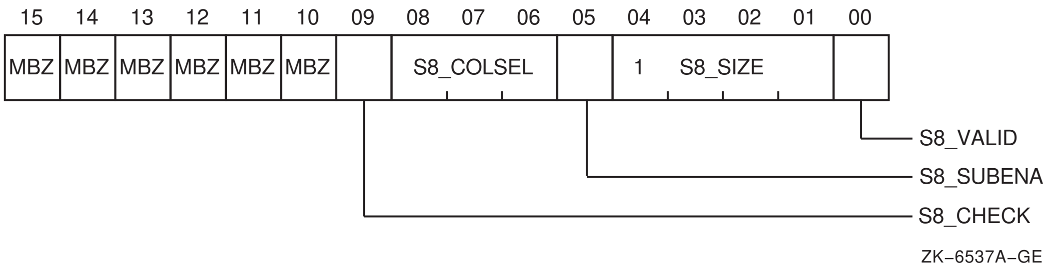
<4..1>: S0_SIZE<3..0>—Bankset8 size in MB. Type: RW. Indicates the size of the bankset to determine which bits are used in comparing the bankset base address with the physical address (PA) and for generating the subset. Corresponds to the total size of the bankset, including subbanks, if present. S0_SIZE<3> must be set to 0. See Table A–4.

Table A–4 S0_Size Field Codes

S0_SIZE<3:0>	Compared	Subset	Set Size
0000	–	–	Reserved
0001	PA<33..29>	PA<28>	512 MB
0010	PA<33..28>	PA<27>	256 MB
0011	PA<33..27>	PA<26>	128 MB
0100	PA<33..26>	PA<25>	64 MB
0101	PA<33..25>	PA<24>	32 MB
0110	PA<33..24>	PA<23>	16 MB
0111	PA<33..23>	PA<22>	8 MB
1XXX	—	—	Reserved

<0>: S0_VALID—Bankset0 Valid. Type: RW, reset: 0. If set, all timing and configuration parameters for bankset0 are valid, and access to bankset0 is allowed. If cleared, access to bankset0 is not allowed. See Figure A–13.

Figure A–13 Bankset8 Configuration Register



<15..10>: Reserved—Must be zero.

<9>: S8_CHECK—Enable Parity Checking. Type: RW, reset: 0. When set, accesses to bankset8 will have their parity checked, as with other banksets. When clear, parity will not be checked. When clear, bankset8 must be mapped into noncacheable space. Only bankset8 has this feature.

<8..6>: S8_COLSEL<2..0>—Column Address Selection. Indicates the number of valid column bits expected at the DRAMs. Used along with memory width information to generate column row or column addresses. Memory width is determined by the **widemem** pin. See Table A–5.

Table A–5 S8_ColSel Field Codes

S8_ColSel	Row, Column Bits
0XX	Reserved
100	9, 9
101	9, 8
11X	Reserved

<5>: S8_SUBENA—Enable Subbanks. Type: RW, reset: 0. When set, subbanks are enabled and determined according to Table A–6. When clear, subbanks are disabled, and the **b0-b1_rasb0_l** pins will only be asserted during refresh.

<4..1>: S8_SIZE<3..0>—Size. Type: RW, reset: 0. Indicates the size of the bankset to determine which bits are used in comparing the base address with the physical address and for selecting the subset (if s8_SubEna is set). Corresponds to the total size of bankset8, including subbanks, if present. See Table A–6.

Table A–6 S8_Size Field Codes

S8_Size<3..0>	Compared	Subbank	Bankset Size
0XXX	—	—	Reserved
1000		PA<23>	Reserved
1001	PA<33..23>	PA<22>	8 MB
1010	PA<33..22>	PA<21>	4 MB
1011	PA<33..21>	PA<20>	2 MB
1100	PA<33..20>	PA<19>	1 MB
1101	—	—	Reserved
1110	—	—	Reserved
1111	—	—	Reserved

<0>: S8_VALID—Valid. Type: RW, reset: 0. If set, all parameters are valid and access to bankset8 is allowed. If cleared, no accesses to bankset8 are allowed.

DMA accesses to this bank should not be performed when error checking is disabled.

A.2.6 Bankset Timing Registers A and B

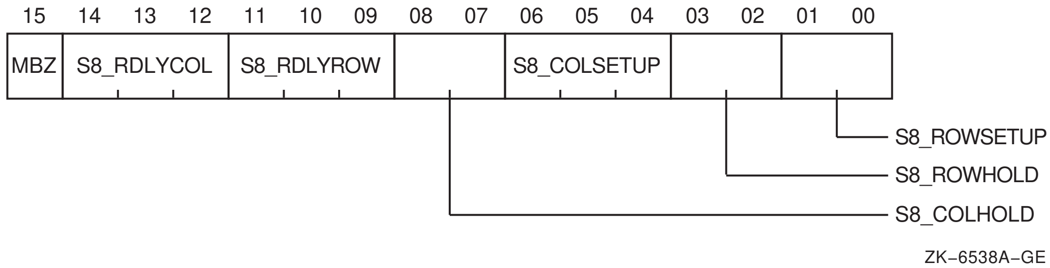
Each bankset has two timing registers associated with it. These registers contain the timing parameters required to perform memory read and write transactions. The format of the timing registers is identical for all banksets.

On reset, all the parameters are set to the maximum value. This may not call for correct operation on the memory interface. The timing registers should be programmed by software before setting the corresponding bankset valid bit in the configuration register.

All the timing parameters are in multiples of **memclk** cycles. Most of the timing parameters in timing registers A and B have a minimum value that is added to the programmed value. The programmer should be careful to subtract this value from the desired value before programming it into the register. The description of the parameters also indicates the corresponding DRAM parameter.

See Figure A–14.

Figure A–14 Bankset Timing Register A



<15>: Reserved.—Must be zero.

<14..12>: S8_RDLYCOL<2..0>— Read Delay from Column Address. Type: RW, reset: 1s. Used only when starting in page mode. Delay from column address to latching first valid read data.

Programmed Value = Desired Value -2.

<11..9>: S8_RDLYROW<2..0> – Read Delay from Row Address. Type: RW, reset: 1s. Delay from Row Address to latching first valid read data.

Programmed Value = Desired Value -4.

<8..7>: S8_COLHOLD<1..0> – Column Hold. Type: RW, reset: 1s. Column Hold (t_{CAH}) from **b0_cas0-1_l** assertion. Used to determine when the current column address can be changed to the next column or row address.

Programmed Value = Desired Value -1.

<6..4>: S8_COLSETUP<2..0>—Column Address Setup. Type: RW, reset: 0. Column address setup (t_{ASC}) to first CAS assertion and write enable setup (t_{CWL}) to CAS assertion. Used to determine first **b0_cas0-1_l** assertion after column address and **b0-b1_cas0-1_l** assertion after **b0_l0-3_we_l**. The maximum of the two setup values should be programmed. A programmed value of 7 is illegal.

Programmed Value = Desired Value -1.

<3..2>: S8_ROWHOLD<1..0>—Row Address Hold. Used to switch **memadr** from row to column after b0-b1_RAS_l assertion.

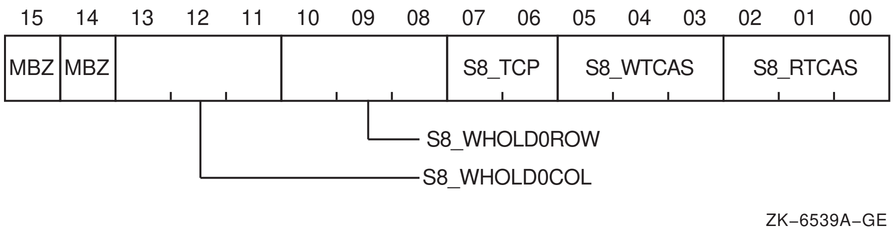
Programmed Value = Desired Value -1.

<1..0>: S8_ROWSETUP<1..0>—Row Address Setup. Type: RW, reset: 1s. Used to generate b0-b1_ras0_l assertion from row address.

Programmed value = Desired Value -1.

See Figure A–15.

Figure A–15 Bankset Timing Register B



<15..14>: Reserved—Must be zero.

<13..11>: S8_WHOLD0COL<2..0>—Write Hold Time from Column Address. Type: RW, reset: 1s. Used only for the first data when starting in page mode. Write data is valid with the column address and is held valid **s8_whold0col** + 2 cycles after the column address.

Programmed Value = Desired Value -2.

<10..8>: S8_WHOLD0ROW<2..0> – Write Hold Time from Row Address. Type: RW, reset: 1s. Hold time of first write data from first row address. The first write data is valid with the row address, and is held valid **s8_whold0row** + 2 cycles after the row address. Used when not starting in page mode. A programmed value of zero is illegal.

Programmed Value = Desired Value -2.

<7..6>: S8_TCP<1..0>—CAS Precharge (t_{CP}). Type: RW, reset: 1s. Delay from **b0_cas0-1_l** deassertion to the next assertion of **b0_cas0-1_l** in page mode.

Programmed Value = Desired Value -1.

<5..3>: S8_WTCAS<2..0>—Write CAS Width (t_{CAS}). Type: RW, reset: 1s. Used on writes to generate the **b0_cas0-1_l** deassertion from the assertion of **b0_cas0-1_l**. Note: WTCas and TCP should be programmed such that their sum is ≤ 5.

Programmed Value = Desired Value -2.

<2..0>: S8_RTCAS<2..0>—Read CAS Width (t_{CAS}). Type: RW, reset: 1s. Used on reads to generate the **b0_cas0-1_l** deassertion from the assertion of **b0_cas0-1_l**. Note: RTCas and TCP should be programmed such that their sum is ≤ 5.

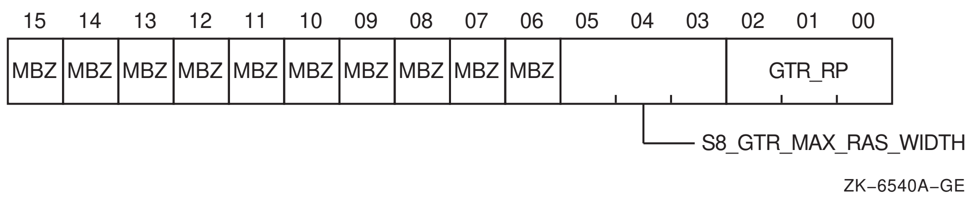
Programmed Value = Desired Value -2.

A.2.7 Global Timing Register

The global timing register contains parameters that are common to all memory banksets. Each parameter counts **memclk** cycles. All pins on the memory interface are referenced to **memclk** rising.

See Figure A–16.

Figure A–16 Global Timing Register



<15..6>: Reserved—Must be be zero.

<5..3>: GTR_MAX_WIDTH<2..0>—Maximum RAS assertion width as a multiple of 128 **memclk** cycles. When this count is reached, the asserted **b0-b3_ras0_l** is deasserted at the end of the ongoing transaction. This value should be programmed with sufficient margin to allow for the timer overflowing during a transaction. Corresponds to DRAM parameter t_{RAS}.

When programmed to a 0, page mode between transactions will be disabled.

<2..0>: GTR_RP<2..0>—Minimum number of RAS precharge cycles. **b0-b3_cas0_l** deassertion to next assertion of the same **b0-b3_cas0_l** pin. Corresponds to DRAM parameter t_{RP}.

Programmed Value = Desired Value -2.

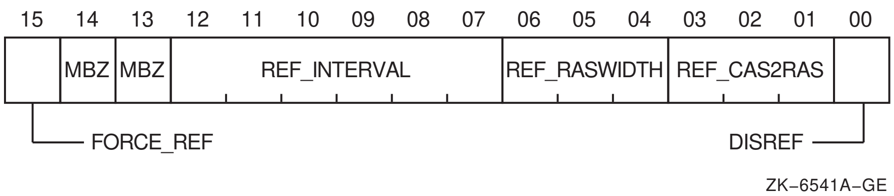
A.2.8 Refresh Timing Register

The refresh timing register contains refresh timing information used to simultaneously refresh all banksets using CAS-RAS refresh. Therefore, these parameters should be programmed to the most conservative value across all sets.

All the timing parameters are in multiples of **memclk** cycles. The parameters have a minimum value which is added to the programmed value. The programmer should be careful to subtract this value from the desired value before programming it to the register.

See Figure A–17.

Figure A–17 Refresh Timing Register



<15>: FORCE_REF—Force refresh. Type: RW, reset: 1s. Writing a 1 to this bit causes a single memory refresh. Reads as 0. Resets the internal Refresh interval counter.

<14..13>: Reserved—Must be zero.

<12..7>: REF_INTERVAL<12..7>— Refresh Interval. Type: RW, reset: 000001. Multiplied by 64 to generate number of **memclk** cycles between refresh requests. A programmed value of zero is illegal.

<6..4>: REF_RASWIDTH<2..0>—Refresh RAS width. Type: RW, reset: 1s. Refresh RAS assertion width from **b0-b3_ras0_l** assertion to **b0-b3_ras0_l** deassertion. **b0-b3_cas0_l** is deasserted with **b0-b3_ras0_l** for refresh. Type: RW, reset: 1s. Corresponds to DRAM parameter t_{RAS}.

Programmed Value = Desired Value -3.

<3..1>: Type: RW, reset: 1s. REF_CAS2RAS<2..0>— Refresh CAS assertion to RAS assertion cycles. Corresponds to DRAM parameter t_{CSR}.

Programmed Value = Desired Value -2.

<0>: DISREF—Disable refresh. Type: RW, reset: 0. Refresh operations will not be performed when DISREF is set.

The other timings in this register should not be changed while this bit is set. Force refresh overrides disable refresh.

A.3 DECchip 21071-DA CSR Descriptions

All CSRs are addressed on cache line boundaries (that is, address bits **<4..2>** must be zero. Register addresses are specified in Table 4–3.

Writes to read-only registers do not cause errors and are acknowledged as normal. Only zeros should be written to unspecified bits within a register. Registers are initialized as specified in the register field descriptions.

In the implementation, address bits **<27..11>** are treated as *don't care*. Therefore, accesses to addresses in 21071-DA CSR space with nonzero address bits **<27..11>** will map to the corresponding CSR address with address bits **<27..11>** equal to zero.

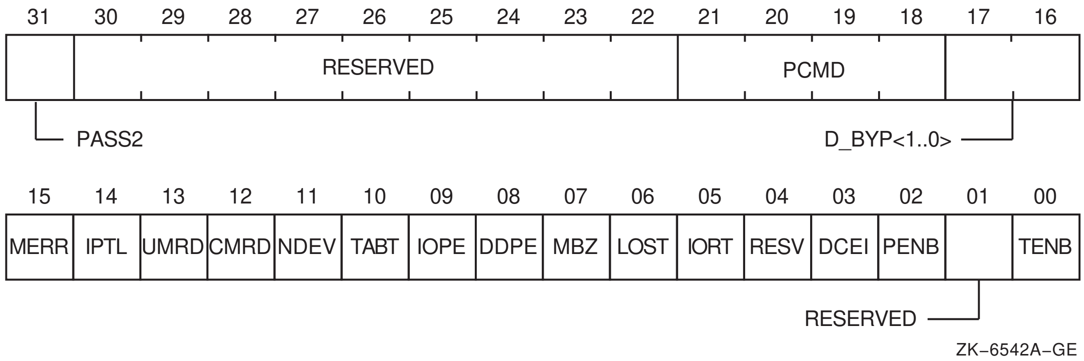
A.3.1 Dummy Registers 1–3

These three registers have no side effects on writes and return zero on reads. Writes to these registers can be used to pack the 21064 write buffers to prevent merging of sparse space I/O writes. Software will not be forced to use MBs between writes if this mechanism is used.

A.3.2 Control and Status Register

The control and status register (DCSR) provides control of operational and diagnostic modes, and reports status and error conditions. See Figure A–18.

Figure A–18 Control and Status Register



<31>: PASS 2—Pass 2. Type: RO. Chip version reads low on pass 1, and high on pass 2.

<30..22>: Reserved—Must be zero.

<21..18>: PCMD—PCI Command. Type: RO. This field indicates the PCI type when a PCI-initiated error is logged in the DCSR. The field is only valid when IPTL, NDEV, TABT, and IOPE are set.

<17..16>: D_BYP<1..0> – Disable Read Bypass. Type: RW, reset: 0. This field is used to control the order of PCI-initiated memory reads with respect to PCI-initiated memory writes. The three modes are described in the following table:

Value	Mode	Description
00	Full bypass	PCI-initiated memory reads will bypass buffered DMA writes if the double hexaword address of the read does not match that of the buffered writes. The address comparison is done across address bits **<31..6>**.
01	NA	Reserved
10	Partial bypass	DMA reads will bypass buffered memory writes, if the address within the page does not match that of the buffered DMA writes. The address comparison is done across bits **<12..6>**.
11	No bypass	DMA read bypassing is disabled. DMA reads will be ordered with respect to DMA writes originating on the PCI.

<15>: MERR—Memory Error. Type: RW, reset: 0. This bit is set when the 21071-DA receives an error code in the **iocack<1..0>** field in response to a memory access. Bits **sysadr<35..5>** for this transaction are logged in sysBus error address register bits **<31..4>**. This bit is not logged if the sysBus error address register is locked by a previous error. In this case, the lost error bit is set.

<14>: IPTL—Invalidate Page Table Lookup. Type: RWC, reset: 0. This bit is set when the longword scatter gather map entry being accessed is invalid. Bits **pci_ad<31..0>** are logged in the PCI error address register, if it is not already locked.

<13>: UMRD—Uncorrectable Memory Read Data. Type: RWC, reset: 0. This bit is set when an uncorrectable error is encountered by the 21071-DA in the data read from the DMA read buffer in the 21071-BA to the 21071-DA on a DMA read or a scatter gather read transaction. Bits **sysadr<33..6>** for this transaction are logged in sysBus error address register bits **<31..4>** if it is not locked.

<12>: CMRD—Correctable Memory Read Data. Type: RWC, reset: 0. Not applicable. Longword parity is implemented on the EB64+.

<11>: NDEV—No Device. Type: RWC, reset: 0. This bit is set when DEVSEL# is not asserted in response to an I/O read or write transaction initiated on the PCI by the 21071-DA. Bits **ad<31..0>** for this transaction are logged in the PCI error address register.

<10>: TABT—Target Abort. Type: RWC, reset: 0. This bit is set when a PCI slave device ends an I/O read or write transaction using the PCI target abort protocol. Bits **ad<31..0>** for this transaction are logged in the PCI error address register.

<9>: IOPE—I/O Parity Error. Type: RWC, reset: 0. This bit is set when a parity error occurs in the data phase of an I/O read or write transaction. Bits **ad<31..0>** for this transaction are logged in the PCI error address register.

<8>: DDPE—DMA Data Parity Error. Type: RWC, reset: 0. This bit is set when a parity error occurs in the data phase of a DMA transaction. Bits **ad<31..0>** for this transaction are logged in the PCI error address register.

<7>: Reserved.—Must be zero.

<6>: LOST—Lost Error. Type: RWC, reset: 0. This bit is set by a 21071-DA error condition when the address register corresponding to that error is locked because of a previous error. In this case error information for the second error is lost. The logged address information in the sysBus error address register or the PCI error address register will remain valid for the initial error condition.

<5>: IORT—I/O Retry Timeout. Type: RWC, reset: 0. This bit is set when a retry timeout error occurs on CPU-initiated read or write transactions on the PCI. Bits **ad<31..0>** are logged in the PCI error address register.

<4>: DPEC—Disable Parity Error Checking. Type: RW, reset: 0. When DPEC is set, parity checking will not be performed on the PCI bus (address and data cycles, DMA and I/O transactions) Parity generation is not affected.

<3>: DCEI—Disable Correctable Error Interrupt. Type: RW, reset: 0. Not applicable. Longword parity is implemented on the EB64+.

<2>: PENB—Prefetch Enable Bit. Type: RWC, reset: 0. If this bit is set, the sysBus master state machine will enable prefetching on DMA read transactions.

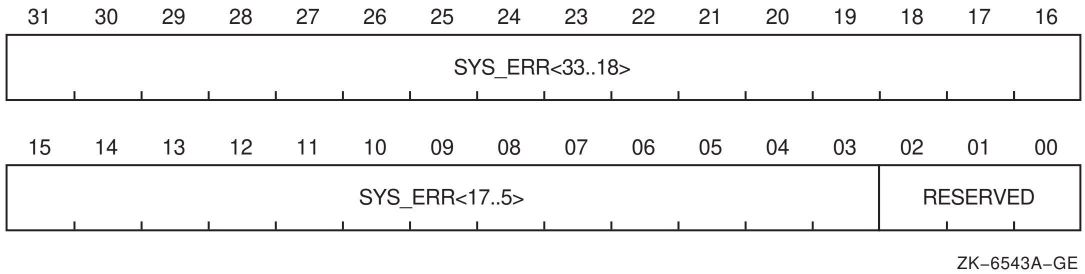
<1>: Reserved—Must be zero.

<0>: TENB—TLB Enable. Type: RW, reset: 0. When this bit is set, the entire TLB is enabled. When the bit is cleared, the TLB will be turned off and subsequent scatter gather reads will not result in allocation of TLB entries. Entries that were valid when the TENB bit was cleared will remain valid. To invalidate entries, software must write to the TBIA register.

A.3.3 SysBus Error Address Register

See Figure A–19.

Figure A–19 SysBus Error Address Register



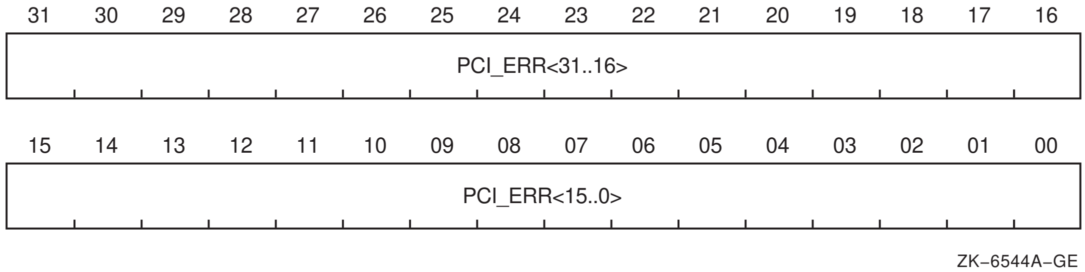
<31..3>: SYS_ERR<33..5>—SysBus Error Address. Type: RO. The address sent on SysBus **sysadr<33..5>** as a result of a DMA transaction is stored in this field. The field logs errors indicated by the MERR, UMRD, or CMRD bits in the DCSR, and is valid only when one of these bits is set. If an error bit is set, a subsequent error of the same type will not update the address logged in this register and the LOST bit is set in the DCSR.

<2..0>: Reserved—Must be zero.

A.3.4 PCI Error Address Register

See Figure A–20.

Figure A–20 PCI Error Address Register

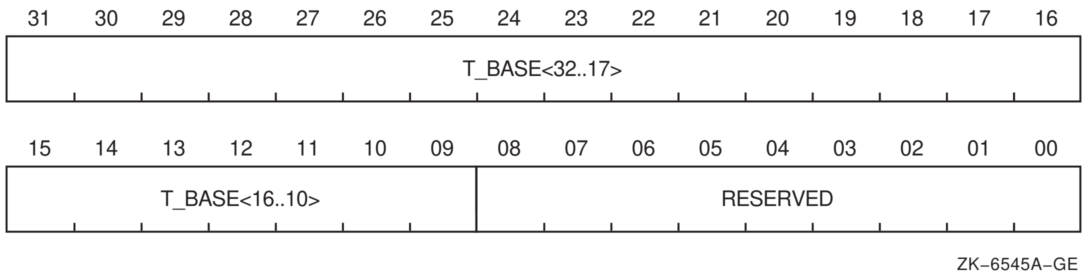


<31..0>: PCI_ERR<31..0>—PCI Error. Type: RO. The address sent out on the PCI Bus (**ad<1..0>**) as a result of an I/O transaction is stored in this register. The field logs the address of the errors indicated by the NDEV, TABT, IOPE, DDPE, IPTL, and IORT bits in the DCSR. The register is valid only when one of these error bits is set. If one of the bits is set, a subsequent error of the same type will not update the address logged in this register and the LOST bit is set in DCSR.

A.3.5 Translated Base Registers 1 and 2

See Figure A–21.

Figure A–21 Translated Base Registers 1–2



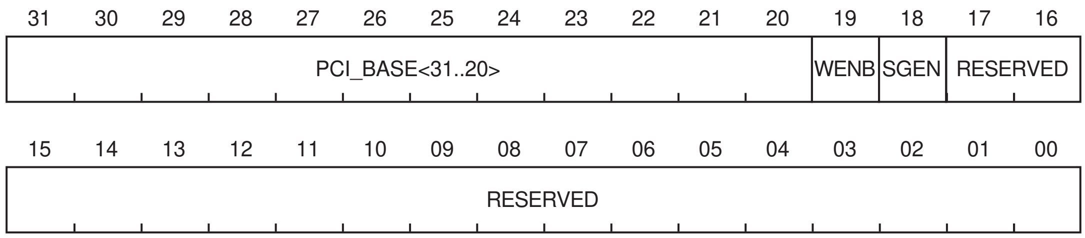
<31..9>: T_BASE<32..10>—Type RW. If scatter gather mapping is disabled, T_BASE specifies the base CPU address of the translated PCI address for the PCI target window. If scatter gather mapping is enabled, T_BASE specifies the base CPU address for the scatter gather map table for the PCI target window.

<8..0>: Reserved—Must be zero.

A.3.6 PCI Base Registers 1 and 2

See Figure A–22.

Figure A–22 PCI Base Registers 1–2



ZK–6546A–GE

<31..20>: PCI_BASE<31..20>—PCI Base. Type: RW. This field specifies the base address of the PCI target window.

<19>: WENB—Window Enable. Type: RW, reset: 0. When this bit is cleared, the PCI target window is disabled and will not respond to PCI-initiated transfers. When WENB is set, the PCI target window is enabled and will respond to PCI-initiated transfers that hit in the address range of the target window. This bit should be disabled by the processor (software) when modifying any of the PCI target window registers (base, mask, or translated base).

<18>: SGEN—Scatter Gather Enable. Type: RW, reset: 0. When this bit is cleared the PCI target window uses direct mapping to translate a PCI address to a CPU address. When the bit is set, the PCI target window uses scatter gather mapping to translate a PCI address to a CPU address.

<17..0>: Reserved—Must be zero.

A.3.7 PCI Mask Registers 1 and 2

See Figure A–23.

Figure A–23 PCI Mask Registers 1–2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PCI_MASK<31..20>												RESERVED			

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RESERVED															

ZK–6547A–GE

<31..20>: PCI_MASK<31..20>—PCI Mask. Type: RW. This field specifies the size of the PCI target window, and is also used in the PCI to CPU address translation.

<19..0>: Reserved—Must be zero.

A.3.8 Host Address Extension Register 0

See Figure A–24. This register is hard coded to zero. A read from this register returns zero; a write has no effect.

Figure A–24 Host Address Extension Register 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HARD CODED TO ZERO															

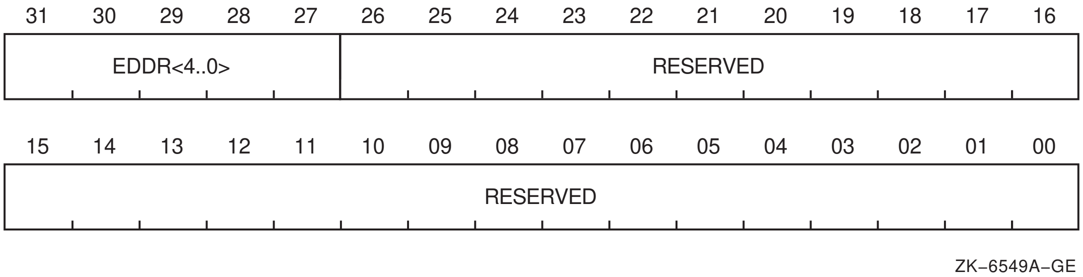
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
HARD CODED TO ZERO															

ZK–6548A–GE

A.3.9 Host Address Extension Register 1

This register generates **pci_ad<31..27>** on CPU-initiated transactions addressing PCI memory space. See Figure A–25.

Figure A–25 Host Address Extension Register 1



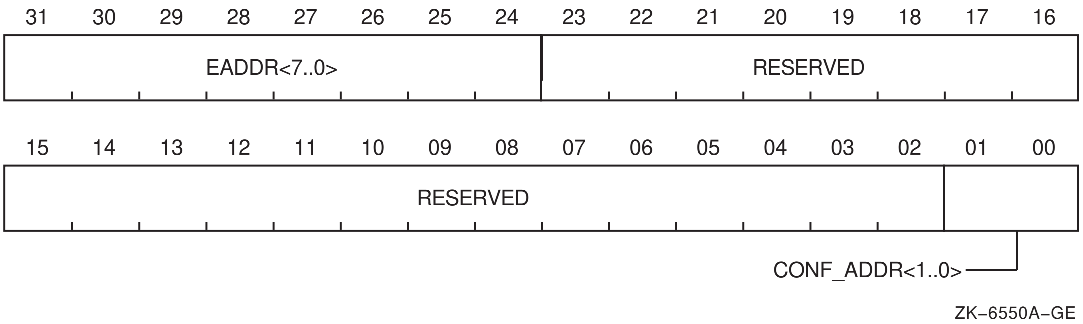
<31..27>: EADDR<4..0>—Type: RW, reset: 0. This field is used as the five high-order PCI address bits **<31..27>** for CPU-initiated transactions to PCI memory.

<26..0>: Reserved—Must be zero.

A.3.10 Host Address Extension Register 2

This register generates **pci_ad<31..24>** on CPU-initiated transactions addressing PCI I/O space. It also generates **pci_ad<1..0>** during PCI configuration read and write transactions. See Figure A–26.

Figure A–26 Host Address Extension Register 2



<31..24>: EADDR<7..0>—Type: RW, reset: 0. This field is used as the eight high-order PCI address bits **pa<31..24>** for CPU-initiated transactions to PCI I/O space..

<23.2>: Reserved—Must be zero.

<1..0>: CONF_ADDR<1..0>—Type: RW, reset: 0. This field is used as the two low-order PCI address bits **ad<1..0>** for CPU-initiated transactions to PCI configuration space.

A.3.11 PCI Master Latency Timer Register

See Figure A–27.

Figure A–27 PCI Master Latency Timer Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MUST BE ZERO															

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MUST BE ZERO								PMLC							

ZK–6827A–GE

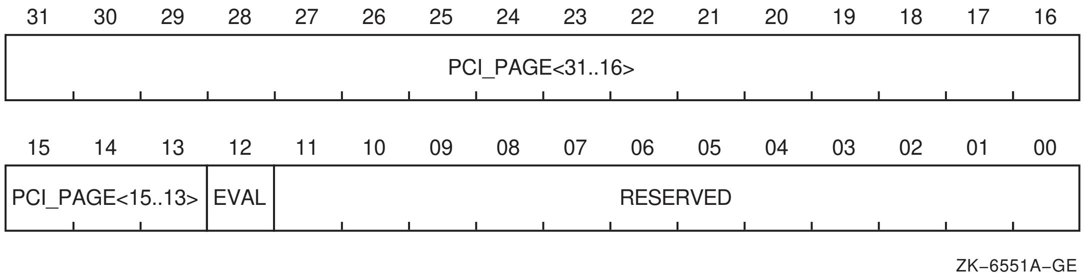
<31..8>: Reserved– Must be zero.

<7..0>: PMLC<7..0>—This field is loaded into the master latency timer register at the start of a PCI master transaction initiated by the 21071-DA. The register resets to zero.

A.3.12 TLB Tag Registers 0–7

See Figure A–28.

Figure A–28 TLB Tag Registers 0–7



<31..13>: PCI_PAGE<31..13>—PCI Page. Type: RO. This field specifies the PCI page address (TAG) corresponding to the translated CPU page address in the associated TLB data register.

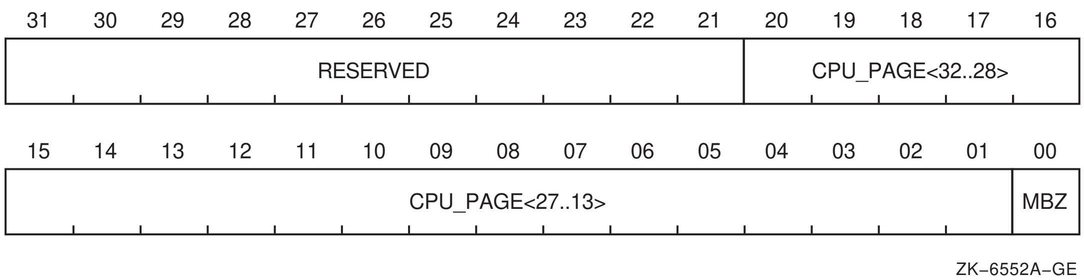
<12>: EVAL—Entry Valid. Type: RO. The entry valid bit can be read and written through this bit. Normally, the invalid bit contains the value read during a page table entry read.

<11..0>:Reserved:—Must be zero.

A.3.13 TLB Data Registers 0–7

See Figure A–29.

Figure A–29 TLB Data Registers 0–7



<31..21>: Reserved—Must be zero.

<20..1>: CPU_PAGE<32..13>—CPU Page. Type: RO. Bits **<32..13>** of the translated CPU address can be read or written through this field.

<0>: Reserved—Must be zero.

A.3.14 Translation Buffer Invalidate All Register (TBIA)

The TBIA register is write only. A write to this register invalidates all valid entries in the scatter gather map TLB.

B

SROM Initialization

B.1 Introduction

The DECchip 21064 family of microprocessors provide a mechanism for loading the initial instruction stream (Istream) from a compact serial ROM (SROM) to start the bootstrap procedure. The SROM executable image is limited to the size of the CPU instruction cache (Icache). Because the image is running only in the Icache, it is relatively difficult to debug. Therefore, it is suggested that the scope and purpose of this code be limited to performing the system initialization necessary to boot the next level of firmware contained in the larger System ROM.

However, tradeoffs between simplicity and convenience were made to support the EB64+ in various configurations. The source code for the EB64+ SROM is available with free licensing for use and modification.

B.1.1 SROM Initialization

After reset the contents of the SROM is loaded into the Icache. After loading the Icache the CPU begins execution at location zero. Execution is performed in the CPU PAL mode environment with privileged access to the computer hardware. The general steps performed by the SROM initialization are:

1. Initialize the CPU's internal processor registers (IPRs).
2. Perform the minimum I/O subsystem initialization necessary to access the Real Time Clock (RTC) and the System ROM.
3. Detect CPU speed by polling the Periodic Interrupt flag (PIF) in the RTC.
4. Set up memory and/or Backup cache (Bcache) parameters based on the speed of the CPU.
5. Wake up the DRAMs.
6. Initialize the Bcache.
7. Copy the contents of the entire memory to itself to insure good memory data parity.

8. Scan System ROM for special header which specifies where and how System ROM firmware should be loaded.
9. Copy the contents of the System ROM to memory and begin code execution.
10. Pass parameters up to the next level of firmware to provide a predictable firmware interface.

B.1.2 Firmware Interface

A firmware interface provides a mechanism for passing critical information about the state of the system and CPU up to the next level of firmware. This interface is achieved through the use of a set of defined SROM output parameters as described in Table B–1

This particular firmware interface serves the DECchip 21064 family of microprocessors. Other Alpha AXP implementations would likely require a different firmware interface.

Table B–1 Output Parameter Descriptions

Output Parameter	Parameter Description
r1 (t0) - AboxCtl value	The AboxCtl value allows the next-level software to preserve any system-specific Dcache configuration information. This register also contains the superpage enables which could be modified by both the next-level firmware and/or operating system PALcodes.
r2 (t1) - BiuCtl value	The BiuCtl value aids in machine check processing and may be used to log and delete errors.
r17 (a1) - Memory size	This value is an unsigned quadword count of the number of contiguous bytes of good memory in the system starting at physical address zero. This simple mechanism will be sufficient for simple systems. Systems that need to communicate more detailed memory configuration may do so through the system context value (see last table entry).
r18 (a2) - Cycle count in picoseconds	This value is the number of picoseconds that elapse for each increment of the processor cycle count (as read by the RPCC instruction). Note that this may be a multiple of the actual internal cycle count of the microprocessor as specified in the Alpha Architecture Reference Manual (a microprocessor will increment the processor cycle count a multiple of the microprocessor clock where the multiple is a power of 2, including $2^0 = 1$).

(continued on next page)

Table B–1 (Cont.) Output Parameter Descriptions

Output Parameter	Parameter Description
r19 (a3) - Signature and System Revision ID	This register includes a signature which specifies that the transfer is following the standard protocol and that the other values may be trusted. In addition, the signature can identify which version of the protocol is being followed. The system revision is a 16-bit field that communicates system revisions that would be significant to operating system software. The register has the following format: <63..32> Don't care <31..16> Signature <15..0> System Revision Valid signatures have the following values: 0xdeca - V1 (previous version of this specification) 0xdecb - V2 (current version of this specification)
r20 (a4) - Active Processor Mask	The processor mask identifies each processor that is present on the current system. Each mask bit corresponds to a processor number associated by the bit number (i.e. bit 0 corresponds to processor 0). A value of 1 in the mask indicates that the processor is present, a value of 0 indicates that the processor is not present. To qualify as present a processor must be: • Physically present • Functioning normally • Capable of sending and receiving interprocessor interrupt requests Uniprocessor systems will pass a value of 1 to this register.
r21 (a5) - System Context Value	The context value is interpreted in a system-specific manner. If the system needs to pass more than one system-specific parameter then it may pass a context value which is a physical address pointer to a data structure of many system-specific values.

B.1.3 Automatic CPU Speed Detection

The EB64+ real-time clock (RTC) detects the speed of the CPU. This allows a somewhat generic SROM to support EB64+ systems configured for different CPU speeds. The CPU speed is a fixed interval while polling the RTC PIF.

B.1.4 CPU Bus Interface Timing

The EB64+ Bcache timing is based on CPU speed in addition to fixed delays associated with the Bcache subsystem. The pertinent Bcache delays used in the calculations result from the logic devices used in the Bcache subsystem, SRAM specifications, and board etch delays. This data is used to calculate the appropriate BIU_CTL register setting which determines the CPU pinbus timing.

Table B–2, Table B–3, and Table B–4 describe the fixed delays for the EB64+.

Table B–2 Cache Loop Delay Characteristics

Function	Minimum/Maximum Delay	Description
Tadr1	1.0 1.6 ns	Delay from CPU to input of address buffer
Tbuf	1.0 4.8 ns	Buffer gate delay
Tadr2	1.0 1.2 ns	Address delay from buffer to SRAM inputs
Tdat	NA[1] 1.9 ns	Data return path from SRAM to CPU input pins
Twe1	1.0 1.0 ns	Delay from CPU to the NOR gate in the WE path
Tnor	1.0 5.0 ns	NOR gate delay
Twe2	1.0 1.0 ns	Delay from the NOR gate to the SRAM inputs

[1]NA = Not applicable

Table B–3 Typical SRAM Specifications

Function	Specification	Description
Tacc	10 15 17 ns	Access from address valid to data valid
Twc	10 15 17 ns	Write cycle time
Twp	9 12 15 ns	Write pulse width

(continued on next page)

Table B–3 (Cont.) Typical SRAM Specifications

Function	Specification	Description
Tdw	5 7 7 ns	Data setup to write pulse deassertion
Tdh	0 0 0 ns	Data hold from write pulse deassertion
Taw	9 12 12 ns	Address setup to write pulse deassertion
Twr	0 0 0 ns	Address hold from write pulse deassertion
Tas	0 0 0 ns	Address setup to write pulse assertion

Table B–4 CPU Specifications

Function	Specification	Description
Tsu	4.5 ns	Internal CPU setup time (21064)
Tstable	2.9 ns	CPU data stable time

B.1.5 Read and Write Calculations

Write Cycle Calculations

WRsetup is the earliest from the beginning of a write cycle that the write pulse can be asserted (see Figure B–1).

Figure B–1 Write Cycle Timing

WRsetup = Tadr1 + Tbuf + Tadr2 + skew

Taddress based on the address path, and Tdata based on the data path determine the earliest from the beginning of a write cycle that the write pulse can be deasserted. The larger of these two determine the more critical path on which the write pulse is determined.

Taddress = WRsetup + Taw
Tdata = Tclock + Tstable + Tdat + Tdw

Since WRsetup is the earliest that the write pulse can be asserted, and Taddress and Tdata determine the earliest that the write pulse can be deasserted, it follows that:

WRpulse = Maximum of the following:

Taddress - WRsetup = Taw
Tdata - WRsetup
Twp + skew

WRhold = Twe1 + Tnor + Twe2 + skew

The WRsetup requirement is offset by the write enable path delay (WRhold) and the WRhold requirement is offset by the address path delay (WRsetup). The WRsetup and WRhold delays are each then discounted by the fastest possible delay through the other path. The minimum parameters estimate the absolute fastest propagation through the address and write enable paths.

Therefore:

WRsetup = Tadr1 + Tbuf + Tadr2 + skew - WRhold (minimum)
WRhold = Twe1 + Tnor + Twe2 + skew - WRsetup (minimum)

Read Cycle Calculation

Read = Tadr1 + Tbuf + Tadr2 + Tacc + Tdat + Tsu

B.1.6 Memory Initialization

The memory banks must be configured such that they are naturally aligned. For example, a bank configured with 32 MB must have a base address of zero or some multiple of 32 MB. Therefore, to insure that both banks are contiguous (no gaps), the larger bank should be set to a base of zero, and the smaller bank should be set to the address immediately following the last location in the larger bank.

If the banks are the same size this is still true. There is no requirement for which bank must be the larger one. Therefore, the following algorithm is used to determine the base addresses of the banks.

If bank_0_size $\geq$ bank_1_size

then

bank_0_base = 0
bank_1_base = bank_0_max_addr + 1

else

bank_1_base = 0
bank_0_base = bank_1_max_addr + 1

Eight consecutive RAS cycles are performed for each memory bank to "wake up" the DRAMs. This is done by reading from each bank eight times. The caches are disabled at this point so the reads go directly to the DRAMs.

Good data parity is insured by writing all memory locations. This is done by rewriting the full contents of memory with the same data. Reading before writing memory lengthens the time to initialize data parity, however it conserves the memory state for debugging purposes.

B.1.7 Bcache Initialization

The Bcache is initialized by the following steps:

1. The BIU_CTL register in the CPU is set to ignore the Bcache.
2. The General Control register in the memory controller is set to enable the Bcache while ignoring Tag parity.
3. The Tag Enable register in the memory controller is cleared.
4. Sweep the Bcache with reads at cache block increments.
5. Reset the Tag Enable register with the proper value based on the Bcache and memory size.
6. Reset the General Control register in the memory controller to disable the Ignore Tag parity bit.
7. Reset the BIU_CTL register in the CPU with the proper value to enable the Bcache based on CPU speed, and Bcache size and speed.

When the system is powered up the Bcache will contain unpredictable data in the Tag RAMs. As the Bcache is swept for initialization the old blocks (referred to as dirty victim blocks) will be written back to main memory. These victim writes will occur based on the tag address (Tag) which stores the upper part of the address location for the dirty blocks of memory.

Because the tags are unpredictable the victim writes could occur to unpredictable addresses. Therefore these writes could be attempted to nonexistent memory. Should this happen, the transaction would not complete and the tag would not be updated in the cache.

By clearing the Tag Enable register, victim writes to nonexistent memory are ignored. When the Tag Enable register is cleared, zero is always stored in the tags. Tags of zero only correspond to the block of memory beginning at zero up to the cache size. Therefore to initialize the cache, only that memory range is swept with reads. Reading beyond that memory range results in an incorrect tag address being stored.

B.1.8 Special ROM Header

The MAKEROM tool is used to place a special header on ROM image files. The SROM allows the System ROM to contain two different ROM images, each with it's own header. The header informs the SROM where to load the image, and whether or not it has been compressed with the MAKEROM tool. The header is optional for System ROMs containing a single image. If the header does not exist, the complete 512-KB System ROM is loaded and executed at physical address zero. Figure B–2 shows the header content.

Figure B–2 Special Header Content

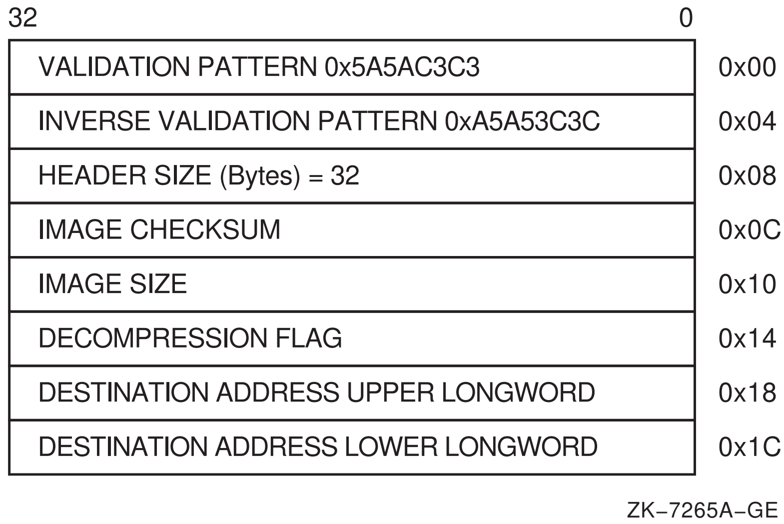


Table B–5 describes each entry in the special header.

Table B–5 Special Header Entry Descriptions

Entry	Description
Validation and Inverse Validation Pattern	This quadword contains a special signature pattern used to validate that the special ROM header has been located. The pattern is 0x5A5AC3C3A5A53C3C.

(continued on next page)

Table B–5 (Cont.) Special Header Entry Descriptions

Entry	Description
Header Block (bytes)	This longword is provided to allow for some backward compatibility in the event that the header is extended in the future. When the the header is located, current versions of SROM code determines where the image begins based on the header size. Additional data added to the header in the future will be ignored by current SROM code.
Image Checksum	This longword is used to verify the integrity of the ROM.
Image Size	The image size is used by the SROM code to determine how much of the System ROM should be loaded.
Decompression Flag	The decompression flag informs the SROM code whether the MAKEROM tool was used to compress the ROM image with a repeating byte algorithm. The SROM code contains routines which execute the decompression algorithm. Other compression and decompression schemes may be employed which work independently from this scheme.
Destination Address	This quadword contains the destination address for the image. The SROM code will load the image at this address and begin execution.

B.1.9 Icache Flush Code

The following code is loaded into memory after the System ROM image. It is then executed to flush the SROM initialization code from the Icache. The SROM initialization code is loaded into the Icache and maps to memory beginning at address zero.

```
77FF0055 mt r31, flushIc
47FF041F bis r31, 31, 31
47FF041F bis r31, 31, 31
47FF041F bis r31, 31, 31
47FF041F bis r31, 31, 31
47FF041F bis r31, 31, 31
47FF041F bis r31, 31, 31
6BEF0000 jmp r31, (r15)
```

In an attempt to transfer execution to the first page in memory, execution would just continue in the SROM initialization code at that address. Therefore, execution must be transferred to some address that does not hit in the Icache where other code can flush the Icache.

The NOPs following the Icache flush allow the instructions that were fetched before the Icache was updated to be cleared from the pipeline. Execution will ultimately continue at the address contained in r15. At this point r15 contains the starting address where the System ROM image was loaded into memory.

B.1.10 EB64+ Configuration Jumpers

The memory controller provides Presence Detect registers that contain the state of the presence detect pins at reset. These pins reflect the SIMM presence detect signals and the software configuration jumpers. Refer to Appendix A.

The software configuration jumpers are completely programmable. The SROM code defines the software configuration jumpers as shown in Figure B–3.

Figure B–3 Software Configuration Jumpers

J1	1	BC_Size<0>	
	2	BC_Size<1>	
	3	BC_Speed	Selects 10ns SRAM timing.
	4	BC_RD_Fast	Drop 1 read cycle.
J2	1	Mini–Debugger	Trap to SROM Mini–Debugger.
	2	Boot_Option	Boot alternate image.
	3	BC_NoAllocate	Disable cache allocation on writes.
	4	No Connection	

ZK–7260A–GE

Each jumper position is described in Table B–6.

Table B–6 Jumper Position Descriptions

<table>
<tr><th>Jumper Position</th><th>Position Description</th></tr>
<tr><td>BC_Size<1..0></td><td>These jumpers force the Bcache to emulate a smaller Bcache as specified in the following table. When both positions are jumpered the Bcache is disabled. These jumpers are changed in conjunction with the appropriate hardware jumpers as specified in Chapter 2, Table 2-1.
<table>
<tr><th>BC_Size<1:0></th><th>BCache</th></tr>
<tr><td>In, In</td><td>Disabled</td></tr>
<tr><td>In, Out</td><td>512KB</td></tr>
<tr><td>Out, In</td><td>1MB</td></tr>
<tr><td>Out, Out</td><td>2MB</td></tr>
</table></td></tr>
<tr><td>BC_Speed</td><td>This jumper selects Bcache timing parameters used to compute the BIU_CTL register value. The BC_Speed jumper (in) selects the appropriate timing for SRAMs with 10ns access time. This defaults to 15ns SRAM timing without a jumper installed.
<table>
<tr><th>BC_Speed</th><th>BCache speed</th></tr>
<tr><td>In</td><td>Selects 10ns SRAM timing</td></tr>
<tr><td>Out</td><td>Selects 15ns SRAM timing</td></tr>
</table></td></tr>
<tr><td>BC_RD_Fast</td><td>This jumper forces a read speed setting of 1 cycle faster than nominal.
<table>
<tr><th>BC_RD_Fast</th><th>BCache speed</th></tr>
<tr><td>In</td><td>Make Read Speed 1 cycle faster</td></tr>
<tr><td>Out</td><td>Nominal Read Speed</td></tr>
</table></td></tr>
<tr><td>Mini-Debugger</td><td>The SROM Mini-Debugger is provided in the SROM. This jumper (In) causes the SROM initialization to trap to the Mini-Debugger after all initialization is complete, but before starting the execution of the System ROM code.</td></tr>
</table>

(continued on next page)

Table B–6 (Cont.) Jumper Position Descriptions

Jumper Position	Position Description
Boot_Option	This jumper (In) selects the second (alternate) image from the System ROM. By default, the first image is loaded. The SROM allows the System ROM to contain two different ROM images each with its own header. The header informs the SROM where to load the image, and if it has been compressed with the MAKEROM tool. For System ROMs which contain a single image, the header is optional. If the header does not exist, the entire 512-KB System ROM is loaded and executed at physical address zero.
BC_NoAllocate	This Jumper (In) disables Bcache allocates on writes to cacheable memory.

C
PCI Address Maps

This appendix provides the EB64+ PCI operating register address space maps.

C.1 PCI Interrupt Acknowledge/Special Cycle Address Space

The PCI interrupt acknowledge/special cycle address space comprises an address range from 1 B000 0000 through 1 BFFF FFFF.

C.2 PCI Sparse I/O Address Space

The PCI Sparse I/O address space comprises an address range from 1 C000 0000 through 1 DFFF FFFF. The PCI operating register set occupies this space.

C.3 SIO PCI to ISA Bridge Operating Register Address Space

Table C–1 is a map of the SIO PCI to ISA bridge operating address space.

Table C–1 SIO PCI to ISA Bridge Operating Register Address Space Map

Offset	Address	Register
000	1 C000 0000	DMA1 CH0 Base and Current Address Register
001	1 C000 0020	DMA1 CH0 Base and Current Count Register
002	1 C000 0040	DMA1 CH1 Base and Current Address Register
003	1 C000 0060	DMA1 CH1 Base and Current Count Register
004	1 C000 0080	DMA1 CH2 Base and Current Address Register
005	1 C000 00A0	DMA1 CH2 Base and Current Count Register
006	1 C000 00C0	DMA1 CH3 Base and Current Address Register
007	1 C000 00E0	DMA1 CH3 Base and Current Count Register

(continued on next page)

Table C–1 (Cont.) SIO PCI to ISA Bridge Operating Register Address Space Map

Offset	Address	Register
008	1 C000 0100	DMA1 Status and Command Register
009	1 C000 0120	DMA1 Write Request Register
00A	1 C000 0140	DMA1 Write Single Mask Bit Register
00B	1 C000 0160	DMA1 Write Mode Register
00C	1 C000 0180	DMA1 Clear Byte Pointer Register
00D	1 C000 01A0	DMA1 Master Clear Register
00E	1 C000 01C0	DMA1 Clear Mask Register
00F	1 C000 01E0	DMA1 Read/Write All Mask Register Bits Register
020	1 C000 0400	INT 1 Control Register
021	1 C000 0420	INT 1 Mask Register
040	1 C000 0800	Timer Counter 1 - Counter 0 Count Register
041	1 C000 0820	Timer Counter 1 - Counter 1 Count Register
042	1 C000 0840	Timer Counter 1 - Counter 2 Count Register
043	1 C000 0860	Timer Counter 1 - Command Mode Register
060	1 C000 0C00	Reset UBus IRQ12 Register
061	1 C000 0C20	NMI Status and Control Register
070	1 C000 0E00	CMOS RAM Address and NMI Mask Register
078–07B	1 C000 0F18	BIOS Timer Register
080	1 C000 1000	DMA Page Register Reserved
081	1 C000 1020	DMA Channel 2 Page Register
082	1 C000 1040	DMA Channel 3 Page Register
083	1 C000 1060	DMA Channel 1 Page Register
084	1 C000 1080	DMA Page Register Reserved
085	1 C000 10A0	DMA Page Register Reserved
086	1 C000 10C0	DMA Page Register Reserved
087	1 C000 10E0	DMA Channel 0 Page Register
088	1 C000 1100	DMA Page Register Reserved
089	1 C000 1120	DMA Channel 6 Page Register

(continued on next page)

Table C–1 (Cont.) SIO PCI to ISA Bridge Operating Register Address Space Map

Offset	Address	Register
08A	1 C000 1140	DMA Channel 7 Page Register
08B	1 C000 1160	DMA Channel 5 Page Register
08C	1 C000 1180	DMA Page Register Reserved
08D	1 C000 11A0	DMA Page Register Reserved
08E	1 C000 11C0	DMA Page Register Reserved
08F	1 C000 11E0	DMA Low Page Register Refresh
090	1 C000 1200	DMA Page Register Reserved
092	1 C000 1240	Port 92 Register
094	1 C000 1280	DMA Page Register Reserved
095	1 C000 12A0	DMA Page Register Reserved
096	1 C000 12C0	DMA Page Register Reserved
098	1 C000 1300	DMA Page Register Reserved
09C	1 C000 1380	DMA Page Register Reserved
09D	1 C000 13A0	DMA Page Register Reserved
09E	1 C000 13C0	DMA Page Register Reserved
09F	1 C000 13E0	DMA Low Page Register Refresh
0A0	1 C000 1400	INT2 Control Register
0A1	1 C000 1420	INT2 Mask Register
0C0	1 C000 1800	DMA2 CH0 Base and Current Address Register
0C2	1 C000 1840	DMA2 CH0 Base and Current Count Register
0C4	1 C000 1880	DMA2 CH1 Base and Current Address Register
0C6	1 C000 18C0	DMA2 CH1 Base and Current Count Register
0C8	1 C000 1900	DMA2 CH2 Base and Current Address Register
0CA	1 C000 1940	DMA2 CH2 Base and Current Count Register
0CC	1 C000 1980	DMA2 CH3 Base and Current Address Register
0CE	1 C000 19C0	DMA2 CH3 Base and Current Count Register
0D0	1 C000 1A00	DMA2 Status(r) and Command(w) Register
0D2	1 C000 1A40	DMA2 Write Request Register

(continued on next page)

Table C–1 (Cont.) SIO PCI to ISA Bridge Operating Register Address Space Map

Offset	Address	Register
0D4	1 C000 1A80	DMA2 Write Single Mask Bit Register
0D6	1 C000 1AC0	DMA2 Write Mode Register
0D8	1 C000 1B00	DMA2 Clear Byte Pointer Register
0DA	1 C000 1B40	DMA2 Master Clear Register
0DC	1 C000 1B80	DMA2 Clear Mask Register
0DE	1 C000 1BC0	DMA2 Read/Write All Mask Register Bits
0F0	1 C000 1E00	Coprocessor Error Register
372	1 C000 6E40	Secondary Floppy Disk Digital Output Register
3F2	1 C000 7E40	Primary Floppy Disk Digital Output Register
40A	1 C000 8140	Scatter/Gather Interrupt Status Register
40B	1 C000 8160	DMA1 Extended Mode Register
410	1 C000 8200	CH0 Scatter/Gather Command Register
411	1 C000 8220	CH1 Scatter/Gather Command Register
412	1 C000 8240	CH2 Scatter/Gather Command Register
413	1 C000 8260	CH3 Scatter/Gather Command Register
415	1 C000 82A0	CH5 Scatter/Gather Command Register
416	1 C000 82C0	CH6 Scatter/Gather Command Register
417	1 C000 82E0	CH7 Scatter/Gather Command Register
418	1 C000 8300	CH0 Scatter/Gather Status Register
419	1 C000 8320	CH1 Scatter/Gather Status Register
41A	1 C000 8340	CH2 Scatter/Gather Status Register
41B	1 C000 8360	CH3 Scatter/Gather Status Register
41D	1 C000 83A0	CH5 Scatter/Gather Status Register
41E	1 C000 83C0	CH6 Scatter/Gather Status Register
41F	1 C000 83E0	CH7 Scatter/Gather Status Register
420–423	1 C000 8418	CH0 Scatter/Gather Descriptor Table Pointer Register
424–427	1 C000 8498	CH1 Scatter/Gather Descriptor Table Pointer Register
428–42B	1 C000 8518	CH2 Scatter/Gather Descriptor Table Pointer Register

(continued on next page)

Table C–1 (Cont.) SIO PCI to ISA Bridge Operating Register Address Space Map

Offset	Address	Register
42C–42F	1 C000 8598	CH3 Scatter/Gather Descriptor Table Pointer Register
434–437	1 C000 8698	CH5 Scatter/Gather Descriptor Table Pointer Register
438–43B	1 C000 8718	CH6 Scatter/Gather Descriptor Table Pointer Register
43C–43F	1 C000 8798	CH7 Scatter/Gather Descriptor Table Pointer Register
481	1 C000 9020	DMA CH2 High Page Register
482	1 C000 9040	DMA CH3 High Page Register
483	1 C000 9060	DMA CH1 High Page Register
487	1 C000 90E0	DMA CH0 High Page Register
489	1 C000 9120	DMA CH6 High Page Register
48A	1 C000 9140	DMA CH7 High Page Register
48B	1 C000 9160	DMA CH5 High Page Register
4D6	1 C000 9AC0	DMA2 Extended Mode Register

C.4 21040 Ethernet Controller Operating Register Address Space

Table C–2 is a map of Ethernet controller operating register address space.

Table C–2 21040 Ethernet Controller Operating Register Address Space Map

Offset	Address[1]	Register
Operating Register Address Space		
000	1 Cxxx 0018	CSR0–Bus Mode Register
008	1 Cxxx 0118	CSR1–Transmit Poll Demand Register
010	1 Cxxx 0218	CSR2–Receive Poll Demand Register
018	1 Cxxx 0318	CSR3–Rx Ring Base Address Register
020	1 Cxxx 0418	CSR4–Tx Ring Base Address Register
028	1 Cxxx 0518	CSR5–Status Register

[1] Software determines the base address.

(continued on next page)

Table C–2 (Cont.) 21040 Ethernet Controller Operating Register Address Space Map

Offset	Address[1]	Register
Operating Register Address Space		
030	1 Cxxx 0618	CSR6–Serial Command Register
038	1 Cxxx 0718	CSR7–Interrupt Mask Register
040	1 Cxxx 0818	CSR8–Missed Frame Counter Register
048	1 Cxxx 0918	CSR9–Address and Mode Diagnostic Register
050	1 Cxxx 0A18	CSR10–Data Diagnostic Register
058	1 Cxxx 0B18	CSR11–Full Duplex Register
060	1 Cxxx 0C18	CSR12–SIA Status Register
068	1 Cxxx 0D18	CSR13–SIA Connectivity Register
070	1 Cxxx 0E18	CSR14–SIA Tx Rx Register
078	1 Cxxx 0F18	CSR15–SIA General Register
Internal Diagnostics I/O Address Space–DMA Block[1]		
088	1 Cxxx 1118	TDES1_ADD
090	1 Cxxx 1218	TDES2_ADD
098	1 Cxxx 1318	TDES3_ADD
0A0	1 Cxxx 1418	TDES_BUF1_ADD
0A8	1 Cxxx 1518	TDES_BUF2_ADD
0B8	1 Cxxx 1718	RDES1_ADD
0C0	1 Cxxx 1818	RDES2_ADD
0C8	1 Cxxx 1918	RDES3_ADD
0D0	1 Cxxx 1A18	RDES_BUF1_ADD
0D8	1 Cxxx 1B18	RDES_BUF2_ADD
0E0	1 Cxxx 1C18	TDES_STAT
0E8	1 Cxxx 1D18	TDES_COMM
0F0	1 Cxxx 1E18	RDES_STAT_OFF
0F8	1 Cxxx 1F18	RDES_COMM_OFF

[1]Software determines the base address.

(continued on next page)

Table C–2 (Cont.) 21040 Ethernet Controller Operating Register Address Space Map

Offset	Address[1]	Register
Internal Diagnostics I/O Address Space–FIFO Block[1]		
100	1 Cxxx 2018	RxFIFO_RD
108	1 Cxxx 2118	RxFIFO_WR
110	1 Cxxx 2218	RxFIFO_STAT
118	1 Cxxx 2318	TxFIFO_WR
120	1 Cxxx 2418	TxFIFO_RD
128	1 Cxxx 2518	TxFIFO_STAT
130	1 Cxxx 2618	TxFIFO_TEST
138	1 Cxxx 2718	RxFIFO_TEST
Internal Diagnostics I/O Address Space–PCI Block[1]		
178	1 Cxxx 2F18	PCI_DIAG
Internal Diagnostics I/O Address Space–RxM Block[1]		
200–378	1 Cxxx 4018– 1 Cxxx 6F18	ADD_REC
380	1 Cxxx 7018	RxM_STAT

[1]Software determines the base address.

C.5 SCSI Controller Operating Register Address Space

Table C–3 is a map of SCSI controller operating register address space.

Table C–3 SCSI Controller Operating Register Address Space Map

Offset	Address[1]	Register
00	1 Cxxx 0000	SCSI Control 0 Register
01	1 Cxxx 0020	SCSI Control 1 Register

[1]Software determines the base address.

(continued on next page)

Table C–3 (Cont.) SCSI Controller Operating Register Address Space Map

Offset	Address[1]	Register
02	1 Cxxx 0040	SCSI Control 2 Register
03	1 Cxxx 0060	SCSI Control 3 Register
04	1 Cxxx 0080	SCSI Chip ID Register
05	1 Cxxx 00A0	SCSI Transfer Register
06	1 Cxxx 00C0	SCSI Destination ID Register
07	1 Cxxx 00E0	General Purpose Bits
08	1 Cxxx 0100	SCSI First Byte Received Register
09	1 Cxxx 0120	SCSI Output Control Latch Register
0A	1 Cxxx 0140	SCSI Selector ID Register
0B	1 Cxxx 0160	SCSI Bus Control Lines
0C	1 Cxxx 0180	DMA Status Register
0D	1 Cxxx 01A0	SCSI Status 0 Register
0E	1 Cxxx 01C0	SCSI Status 1 Register
0F	1 Cxxx 01E0	SCSI Status 2 Register
10–13	1 Cxxx 0218	Data Structure Address Register
14	1 Cxxx 0280	Interrupt Status Register
18	1 Cxxx 0300	Chip Test 0 Register
19	1 Cxxx 0320	Chip Test 1 Register
1A	1 Cxxx 0340	Chip Test 2 Register
1B	1 Cxxx 0360	Chip Test 3 Register
1C–1F	1 Cxxx 0398	Temporary Stack Register
20	1 Cxxx 0400	DMA FIFO Register
21	1 Cxxx 0420	Chip Test 4 Register
22	1 Cxxx 0440	Chip Test 5 Register
23	1 Cxxx 0460	Chip Test 6 Register
24–26	1 Cxxx 0480– 1 Cxxx 04C0	DMA Byte Counter Register
27	1 Cxxx 04E0	DMA Command Register
28–2B	1 Cxxx 0518	DMA Next Address for Data Register

[1]Software determines the base address.

(continued on next page)

Table C–3 (Cont.) SCSI Controller Operating Register Address Space Map

Offset	Address[1]	Register
2C–2F	1 Cxxx 0598	DMA SCRIPTS Pointer Register
30–33	1 Cxxx 0618	DMA SCRIPTS Pointer Save Register
34–37	1 Cxxx 0698	General Purpose Scratch Pad A Register
38	1 Cxxx 0700	DMA Mode Register
39	1 Cxxx 0720	DMA Interrupt Enable Register
3A	1 Cxxx 0740	DMA Watchdog Timer Register
3B	1 Cxxx 0760	DMA Control Register
3C–3F	1 Cxxx 0798	Sum output of internal adder
40	1 Cxxx 0800	SCSI Interrupt Enable 0 Register
41	1 Cxxx 0820	SCSI Interrupt Enable 1 Register
42	1 Cxxx 0840	SCSI Interrupt Status 0 Register
43	1 Cxxx 0860	SCSI Interrupt Status 1 Register
44	1 Cxxx 0880	SCSI Longitudinal Parity Register
46	1 Cxxx 08C0	Memory Access Control Register
47	1 Cxxx 08E0	General Purpose Control Register
48	1 Cxxx 0900	SCSI Timer 0 Register
49	1 Cxxx 0920	SCSI Timer 1 Register
4A	1 Cxxx 0940	Response ID Register
4C	1 Cxxx 0980	SCSI Test 0 Register
4D	1 Cxxx 09A0	SCSI Test 1 Register
4E	1 Cxxx 09C0	SCSI Test 2 Register
4F	1 Cxxx 09E0	SCSI Test 3 Register
50	1 Cxxx 0A00	SCSI Input Data Latch Register
54	1 Cxxx 0A80	SCSI Output Data Latch Register
58	1 Cxxx 0B00	SCSI Bus Data Lines Register
5C	1 Cxxx 0B98	General Purpose Scratch Pad B Register

[1]Software determines the base address.

C.6 PCI Configuration Address Space

The PCI configuration address space comprises an address range from 1 E000 0000 through 1 FFFF FFFF. The PCI configuration register set occupies this space. Table C–4 identifies the PCI devices and the corresponding PCI address bit that drives the device **idsel** pin.

Table C–4 Address Bits and PCI Device idsel Pins

PCI Device	PCI Address Bit Driving idsel Pin	Physical Address
SCSI	**pci_ad<16>**	1 E020 0000
PCI expansion slot 0	**pci_ad<17>**	1 E040 0000
PCI expansion slot 1	**pci_ad<18>**	1 E080 0000
System I/O (SIO)	**pci_ad<19>**	1 E100 0000
Ethernet	**pci_ad<20>**	1 E200 0000

C.7 SCSI Controller Configuration Register Address Space

Table C–5 is a map of SCSI controller configuration register address space. PCI address bit **pci_ad16** drives the **idsel** chip select pin for access to the configuration register space.

Table C–5 SCSI Controller Configuration Register Address Space Map

Offset	Address	Register
00–01	1 E020 0008	Vendor ID Register
02–03	1 E020 0048	Device ID Register
04–05	1 E020 0088	Command Register
06–07	1 E020 00C8	Status Register
08	1 E020 0100	Revision ID Register
09–0B	1 E020 0120—1 E020 0160	Class Code Register
0D	1 E020 01A0	Latency Timer Register
0E	1 E020 01C0	Header Type Register
10–13	1 E020 0208—1 E020 0248	Base Address 0 (I/O) Register

(continued on next page)

Table C–5 (Cont.) SCSI Controller Configuration Register Address Space Map

Offset	Address	Register
14–17	1 E020 0288—1 E020 02C8	Base Address 1 (Memory) Register
3C	1 E020 0780	Interrupt Line Register
3D	1 E020 07A0	Interrupt Pin Register
3E	1 E020 07C0	Minimum Grant Register
3F	1 E020 07E0	Maximum Latency Register

C.8 SIO PCI to ISA Bridge Configuration Address Space

Table C–6 is a map of SIO PCI to ISA bridge configuration address space. PCI address bit **pci_ad19** drives the **idsel** chip select pin for access to the configuration register space.

Table C–6 SIO PCI to ISA Bridge Configuration Address Space Map

Offset	Address	Register
00–01	1 E100 0008	Vendor ID Register
02–03	1 E100 0048	Device ID Register
04–05	1 E100 0088	Command Register
06–07	1 E100 00C8	Device Status Register
08	1 E100 0100	Revision ID Register
40	1 E100 0800	PCI Control Register
41	1 E100 0820	PCI Arbiter Control Register
42	1 E100 0840	PCI Arbiter Priority Control Register
44	1 E100 0880	MEMCS# Control Register
45	1 E100 08A0	MEMCS# Bottom of Hole Register
46	1 E100 08C0	MEMCS# Top of Hole Register
47	1 E100 08E0	MEMCS# Top of Memory Register
48	1 E100 0900	ISA Address Decoder Control Register
49	1 E100 0920	ISA Address Decoder ROM Block Enable Register

(continued on next page)

Table C–6 (Cont.) SIO PCI to ISA Bridge Configuration Address Space Map

Offset	Address	Register
4A	1 E100 0940	ISA Address Decoder Bottom of Hole Register
4B	1 E100 0960	ISA Address Decoder Top of Hole Register
4C	1 E100 0980	ISA Controller Recovery Timer Register
4D	1 E100 09A0	ISA Clock Divisor Register
4E	1 E100 09C0	Utility Bus Chip Select Enable A Register
4F	1 E100 09E0	Utility Bus Chip Select Enable B Register
54	1 E100 0A80	MEMCS# Attribute Register #1
55	1 E100 0AA0	MEMCS# Attribute Register #2
56	1 E100 0AC0	MEMCS# Attribute Register #3
57	1 E100 0AE0	Scatter/Gather Relocation Base Address Register
80–81	1 E100 1008	BIOS Timer Base Address Register

C.9 Ethernet Controller Configuration Register Address Space

Table C–7 is a map of the Ethernet controller configuration register address space. PCI address bit **pci_ad20** drives the **idsel** chip select pin for access to the configuration register space.

Table C–7 21040 Ethernet Controller Configuration Register Address Space Map

Offset	Address	Register
00–03	1 E200 0018	Configuration ID
04–07	1 E200 0098	Configuration Command/Status Register
08–0B	1 E200 0118	Configuration Revision Register
0C–0F	1 E200 0198	Configuration Latency Timer Register
10–13	1 E200 0218	Configuration Base I/O Address Register
14–17	1 E200 0298	Configuration Base Memory Address Register
3C–3F	1 E200 0798	Configuration Interrupt Register

(continued on next page)

Table C–7 (Cont.) 21040 Ethernet Controller Configuration Register Address Space Map

Offset	Address	Register
40–43	1 E200 0818	Configuration Driver Area Register

C.10 PCI Sparse Memory Address Space

The PCI sparse memory address space comprises an address range from 2 0000 0000 through 2 7FFF FFFF.

C.11 PCI Dense Memory Address Space

The PCI dense memory address space comprises an address range from 3 0000 0000 through 3 FFFF FFFF.

C.12 PC87312 Combination Controller Register Address Space

Table C–8 lists the base address values for the PC87312 combination diskette, serial port, and parallel port controller.

The general registers are located at addresses 398 (index address) and 399 (data address). For example, writing an index value of '1' to address 398 selects the Function Address Register. If a read from address 399 follows, the data associated with the Function Address Register is returned. If a write to address 399 follows, the Function Address Register will be updated.

Table C–8 PC87312 Combination Controller Register Address Space Map

Address Offset Read/Write	Physical Address	Register
General Registers		
398	1 C000 7300	Index Address Register
399	1 C000 7320	Data Address Register
	Index	**Register**
	0	Function Enable Register
	1	Function Address Register
	2	Power and Test Register
COM2 Serial Port Registers		
2F8-R 0DLAB=0	1 C000 5F00	COM2 Receiver Buffer Register
2F8-W 0DLAB=0	1 C000 5F00	COM2 Transmitter Holding Register
2F8 0DLAB=1	1 C000 5F00	COM2 Divisor Latch Register (LSB)
2F9 1DLAB=0	1 C000 5F20	COM2 Interrupt Enable Register
2F9 1DLAB=1	1 C000 5F20	COM2 Divisor Latch Register (MSB)
2FA-R	1 C000 5F40	COM2 Interrupt Identification Register
2FA-W	1 C000 5F40	COM2 FIFO Control Register
2FB	1 C000 5F60	COM2 Line Control Register
2FC	1 C000 5F80	COM2 Modem Control Register
2FD	1 C000 5FA0	COM2 Line Status Register
2FE	1 C000 5FC0	COM2 Modem Status Register
2FF	1 C000 5FE0	COM2 Scratch Pad Register

(continued on next page)

Table C–8 (Cont.) PC87312 Combination Controller Register Address Space Map

Address Offset Read/Write	Physical Address	Register
Diskette Registers		
3F0-R	1 C000 7E00	Status A Register
3F1-R	1 C000 7E20	Status B Register
3F2-R/W	1 C000 7E40	Digital Output Register
3F3-R/W	1 C000 7E60	Tape Drive Register
3F4-R	1 C000 7E80	Main Status Register
3F4-W	1 C000 7E80	Data Rate Select Register
3F5-R/W	1 C000 7EA0	Data (FIFO) Register
3F6	1 C000 7EC0	None (tristate bus)
3F7-R	1 C000 7EE0	Digital Input Register
3F7-W	1 C000 7EE0	Configuration Control Register
COM1 Serial Port Registers		
3F8-R 0DLAB=0	1 C000 7F00	COM1 Receiver Buffer Register
3F8-W 0DLAB=0	1 C000 7F00	COM1 Transmitter Holding Register
3F8 0DLAB=1	1 C000 7F00	COM1 Divisor Latch Register (LSB)
3F9 1DLAB=0	1 C000 7F20	COM1 Interrupt Enable Register
3F9 1DLAB=1	1 C000 7F20	COM1 Divisor Latch Register (MSB)
3FA-R	1 C000 7F40	COM1 Interrupt Identification Register
3FA-W	1 C000 7F40	COM1 FIFO Control Register
3FB	1 C000 7F60	COM1 Line Control Register
3FC	1 C000 7F80	COM1 Modem Control Register
3FD	1 C000 7FA0	COM1 Line Status Register
3FE	1 C000 7FC0	COM1 Modem Status Register
3FF	1 C000 7FE0	COM1 Scratch Pad Register

(continued on next page)

Table C–8 (Cont.) PC87312 Combination Controller Register Address Space Map

Address Offset Read/Write	Physical Address	Register
Parallel Port Registers		
3BC-R/W	1 C000 7780	Data Register
3BD-R	1 C000 77A0	Status Register
3BE-R/W	1 C000 77C0	Control Register
3BF	1 C000 77E0	None (tristate)

C.13 Utility Bus Device Address

Table C–9 shows the decoding for Utility Bus devices driven from the SIO bridge.

Table C–9 Utility Bus Device Decode

Device Address Select Bits						
ecsen_l	**ecasaddr_2**	**ecasaddr_1**	**ecasaddr_0**	**Device Select Signal**	**Device Selected**	**Address Decode**
0	0	0	0	**rtcale_l**	TOY address	70, 72, 74, 76
0	0	0	1	**rtccs_l**	TOY data	71, 73, 75, 77
0	0	1	0	**kbcs_l**	Mouse/kbd enable	60, 62, 64, 66
0	0	1	1	**romce_l**	UVPROM enable[1]	–
0	1	0	0	–	Unused	–
0	1	0	1	–	Unused	–
0	1	1	0	–	Unused	–

[1]The encoded chip select signal for **romce_l** will always be generated for accesses to the upper 64 KB at the top of 1 MB (F0000–FFFFF) and its aliases at the top of the 4 GB and 4 GB–1 MB. Access to the lower 64 KB (E0000–EFFFF) and its aliases at the top of the 4 GB and 4 GB–1 MB can be enabled or disabled through the SIO. An additional 384 KB of BIOS memory at the top of 4 GB (FFFD0000–FFFDFFFF) can be enabled for BIOS use.

(continued on next page)

Table C–9 (Cont.) Utility Bus Device Decode

Device Address Select Bits						
ecsen_l	ecasaddr_2	ecasaddr_1	ecasaddr_0	Device Select Signal	Device Selected	Address Decode
0	1	1	1	–	Unused	–
1	0	0	0	–	Unused	–
1	0	0	1	**selpgs_l**	NVRAM address bits <12:8>[2]	0C00
1	0	1	0	**cmoscs_l**	NVRAM enable	0800 - 08FF
1	0	1	1	–	Unused	–
1	1	0	0	–	Unused	–
1	1	0	1	–	Unused	–
1	1	1	0	–	Unused	–
1	1	1	1	–	Unused	–

[2]Signals **ubus_data<4:0>** drive **pga<4:0>**, which drive **nvram_a<12:8>**.

C.14 Interrupt Control PLD Addresses

Table C–10 lists the registers and memory addresses for the interrupt control PLD.

Table C–10 Interrupt Control PLD Addresses

Offset	Physical Address	Register
26	1 C000 04C0	Interrupt status/interrupt mask register 1
27	1 C000 04E0	Interrupt status/interrupt mask register 2

C.15 8242AH Keyboard and Mouse Controller Addresses

Table C–11 lists the register and memory addresses for the keyboard/mouse controller.

Table C–11 Keyboard and Mouse Controller Addresses

Offset	Physical Address	Register
60-R	1 C000 0C00	Auxiliary/keyboard data
60-W	1 C000 0C00	Command data
64-R	1 C000 0C80	Read status
64-W	1 C000 0C80	Command

C.16 Time-Of-Year Clock Device Addresses

Table C–12 lists the register and memory addresses for the TOY clock device. The time-of-year clock register is accessed by writing to address 70 with the latched index. Then, reading from, or writing to, address 71 reads or writes the register. For example, writing an ‘‘8’’ to address 70 followed by a read from address 71 returns the value of the month. Writing a ‘‘4’’ to address 70 followed by a write to address 71 updates the hour register.

Table C–12 Time-Of-Year Clock Device Addresses

Offset	Index Latched	Physical Address	Register
70	0	1 C000 0E00	Seconds
70	1	1 C000 0E00	Seconds alarm
70	2	1 C000 0E00	Minutes
70	3	1 C000 0E00	Minutes alarm
70	4	1 C000 0E00	Hour
70	5	1 C000 0E00	Hour alarm
70	6	1 C000 0E00	Day of week
70	7	1 C000 0E00	Day of month
70	8	1 C000 0E00	Month

(continued on next page)

Table C–12 (Cont.) Time-Of-Year Clock Device Addresses

Offset	Index Latched	Physical Address	Register
70	9	1 C000 0E00	Year
70	A	1 C000 0E00	Register A
70	B	1 C000 0E00	Register B
70	C	1 C000 0E00	Register C
70	D	1 C000 0E00	Register D
71		1 C000 0E20	TOY clock chip select

C.17 SIO Utility Bus Memory Device Addresses

Table C–13 lists the address ranges for the SIO's utility bus memory devices.

Table C–13 Utility Bus Memory Device Addresses

Offset	Address	Register/Data	Capacity
0800–08FF	1 C001 0000 – 1 C001 1FE0	NVRAM	8 kB
FFF8 0000 – FFFE FFFF	3 FFF8 0000– 3 FFFF FFFF	UVPROM	512 kB

D

Technical Support, Ordering, and Associated Literature

This appendix describes how to obtain DECchip information and technical support, and order DECchip products and associated literature.

Calling the DECchip Information Line for Information and Technical Support

Call the DECchip Information Line for information and technical support:

United States and Canada	**1–800–332–2717 (1–800–DEC–2717)**
TTY (United States only)	**1–800–332–2515 (1–800–DEC–2515)**
Outside North America	**+1–508–568–6868**

Ordering DECchip Products

To order the DECchip 21064 and related products, contact your local Digital sales office. When working with your sales representative, you may be able to take advantage of discounts and volume pricing.

To order DECchip samples or Sample Kits call **1–800–DIGITAL**.

You can order the following DECchip products from Digital.

Product	Speed	Order Number
DECchip 21064–150 Alpha AXP Microprocessor	150 MHz	21064–AA
DECchip 21064A–275 Alpha AXP Microprocessor	275 MHz	21064–DB
Evaluation Board Kits		
DECchip 21064–150 Evaluation Board OSF/1 and Windows NT	150 MHz	21A01–03
DECchip 21064–275 PCI Evaluation Board OSF/1 and Windows NT	275 MHz	21A02–02

Ordering Associated DECchip Literature

The following table lists some of the DECchip literature that is available. For a complete list, and for information about ordering, contact the DECchip Information Line.

Title	Order Number
Alpha Architecture Reference Manual *	EY–L520E–DP–YCH
Answers to Common Questions about PALcode for Alpha AXP Systems	EC–N0647–72
DECchip 21064 and DECchip 21064A Alpha AXP PCI Evaluation Board Product Brief	EC–N0639–72
DECchip 21064 and DECchip 21064A Microprocessor Hardware Reference Manual	EC–N0079-72

*To order and purchase the *Alpha Architecture Reference Manual*, call **1–800–DIGITAL** from the U.S. or Canada, or contact your local Digital office, or technical or reference bookstore where Digital Press books are distributed by Prentice Hall.

Ordering Third-Party Documentation

You can order the following documentation directly from the vendor.

Documentation	Order Number
82420/82430 PCIset ISA and EISA Bridges (includes 82378IB SIO)	Intel No 290483
NCR 53C810 PCI-SCSI I/O Processor Data Manual	NCR No T15935I
NCR 53C810/53C820 PCI-SCSI I/O Processor Design In Guide	NCR No E25931I
PC83711 (SuperI/O II/III) Floppy Disk Controller with Dual UARTs, Parallel Port, and IDE Interface	National Semiconductor No 11362
UPI-41AH/42AH Universal Peripheral Interface 8-bit Slave Microcontroller	Intel No 210393
Peripheral Component Interconnect (PCI) Specification	Intel PCI SIG
PCI System Design Guide	Intel PCI SIG

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W