COMPAQ

Tsunami/Typhoon 21272 Chipset
Hardware Reference Manual

Order Number: DS–0025A–TE

Revision/Update Information: Revision 4.0

21 October 1999

Compaq Computer Corporation

Contents

5 Mechanical Specifications

6 Cchip Architecture

7 Dchip Architecture

8 Pchip Architecture

9 System Memory

10   Programmer's Reference

11   Chipset Clock Generation

12   Reset, Initialization, and Power Management

A   Technical Abbreviations

B   Support

Figures

Tables

Preface

Overview

This is a support and reference document for engineers using the 21264 Alpha microprocessor and the 21272 core logic chipset to design dual processor and uniprocessor systems. The document provides information about the architecture, internal design, external interface, and specifications of the 21272 core logic chipset.

Audience

This manual is for system designers, software developers, and hardware engineers who use the 21272 chipset.

Manual Organization

This manual includes the following chapters, an appendix, and an index:

Chapter 1, Introduction, provides an overview of the 21272 chipset features.

Chapter 2, Chipset Configurations, describes the various CPU and memory configurations that are supported by the 21272 chipset.

Chapter 3, Pinouts, lists and describes the signal interface pins for each ASIC in the 21272 chipset.

Chapter 4, Electrical Specifications, lists absolute limits, power requirements, dc characteristics, and ac characteristics.

Chapter 5, Mechanical Specifications, provides package outline dimensions for each ASIC in the 21272 chipset.

Chapter 6, Cchip Architecture, describes the internal architecture for the Cchip, which controls other chips and interfaces with the CPU's command and address buses.

Chapter 7, Dchip Architecture, describes the internal architecture for the Dchip, which provides an interface with the system data bus.

Chapter 8, Pchip Architecture, describes the internal architecture for the Pchip, which interfaces between devices on the PCI bus and the rest of the system.

Chapter 9, System Memory, describes system memory, its organization into arrays, and its control signals.

Chapter 10, Programmer's Reference, provides information about system address mapping and address translation, as well as all internal chipset registers.

Chapter 11, Chipset Clock Generation, provides information about clock generation, clock skew, system interface clock forwarding, PCI bus clocking, and SDRAM clocking.

Chapter 12, Reset, Initialization, and Power Management, provides information about hardware reset for the chipset, SDRAM, and PCI bus, firmware/CSR initialization tasks, and power management.

Appendix A, Technical Abbreviations, contains a list of acronymns and abbreviations that pertain to the 21272.

Appendix B, Support, contains information about the Alpha OEM web page, ordering chips, and obtaining related documentation.

Conventions

This section defines product-specific terminology, abbreviations, and other conventions used throughout this manual.

Abbreviations

- Binary Multiples

  The abbreviations K, M, and G (kilo, mega, and giga) represent binary multiples and have the following values:

K	$=$	2^{10} (1024)
M	$=$	2^{20} (1,048,576)
G	$=$	2^{30} (1,073,741,824)

  For example:

2KB	$=$	2 kilobytes	$=$	2×2^{10} bytes
4MB	$=$	4 megabytes	$=$	4×2^{20} bytes
8GB	$=$	8 gigabytes	$=$	8×2^{30} bytes

- Register Bit and Field Notation

  The abbreviations used to indicate the type of access to register bits and fields have the following definitions:

  MBZ — Must Be Zero

  Software must never place a nonzero value in bits and fields specified as MBZ. Read operations return UNPREDICTABLE values. Such fields are reserved for future use.

RAZ — Read As Zero

Bits and fields specified as RAZ return a zero when read.

RC — Read to Clear

Bits and fields specified as RC are written by hardware and remain unchanged until read by software or microcode, at which point hardware can write a new value into the bit or field.

RES — Reserved

Bits and fields specified as RES are reserved by DIGITAL and should not be used. However, zeros can be written to reserved fields that cannot be masked.

RO — Read Only

Bits and fields specified as RO can be read by software, microcode, or hardware and are ignored (not written) on writes.

RW — Read/Write

Bits and fields specified as RW can be read and written by software, microcode, or hardware.

R/W1C — Read/Write One to Clear

Bits and fields specified as R/W1C can be read. Writing a one clears these bits for the duration of the write; writing a zero has no effect.

SBZ — Should Be Zero

Bits and fields specified as SBZ should be filled by software or microcode with a zero value. Nonzero values in SBZ fields produce UNPREDICTABLE results.

W1C — Write One to Clear

Bits and fields specified as W1C can be cleared by writing a one for the duration of the write; writing a zero has no effect.

WC — Write to Clear

Bits and fields specified as WC can be read by software or microcode. Software or microcode write operations with a one to this bit or field cause the bit to be cleared by hardware. Software or microcode write operations with a zero to this bit or field do not modify the state of the bit.

WO — Write Only

Bits and fields specified as WO can be written by software and microcode but not read. Read operations to this bit or field by software or microcode produce UNPREDICTABLE results.

Addresses

Unless otherwise noted, all addresses and offsets are hexadecimal.

Aligned and Unaligned

The terms *aligned* and *naturally aligned* are interchangeable and refer to data objects that are powers of two in size. An aligned datum of size $2n$ is stored in memory at a byte address that is a multiple of $2n$; that is, one that has n low-order zeros. For example, an aligned 64-byte array has a memory address that is a multiple of 64.

A datum of size $2n$ is *unaligned* if it is stored in a byte address that is not a multiple of $2n$.

Bit Notation

Multiple-bit fields can include contiguous and noncontiguous bits contained in angle brackets (<>). Multiple contiguous bits are indicated by a pair of numbers separated by a colon (:). For example, <9:7,5,2:0> specifies bits 9,8,7,5,2,1, and 0. Similarly, single bits are frequently indicated with angle brackets. For example, <27> specifies bit 27.

Caution

Cautions indicate potential damage to equipment or loss of data.

Data Units

The following data-unit terminology is used throughout this manual.

Term	Words	Bytes	Bits	Other
Byte	½	1	8	—
Word	1	2	16	—
Longword (LW)	2	4	32	Dword.
Quadword	4	8	64	2 longwords.
Octaword	8	16	128	Single read fill; that is, the cache space that can be filled in a single read access. It takes two read accesses to fill one L2 cache block.
Hexword	16	32	256	The space allocated to a single cache block.

External

Unless otherwise stated, external means not contained in the 21272 chipset.

Note

Notes emphasize particularly important information.

Numbering

All numbers are decimal or hexadecimal unless otherwise indicated. The prefix 0*x* indicates a hexadecimal number. For example, 19 is decimal, but 0*x*19 and 0*x*19A are hexadecimal (also see Addresses). Otherwise, the base is indicated by a subscript; for example, 100_2 is a binary number.

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 a colon (:) and are inclusive. Bit fields are often specified as extents. For example, bits <7:3> specifies bits 7, 6, 5, 4, and 3.

Signal Names

Signal names in text and figures are printed in lowercase, boldface type. Active-high signals have no suffix. Active-low signals are indicated by the **_l** suffix or the number sign (#) suffix. For example, **capsel** is an active-high signal, and **cact_l** is a low-asserted signal.

UNPREDICTABLE and UNDEFINED

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. Unprivileged software cannot invoke UNDEFINED operations.

Warning

Warnings provide information to prevent personal injury.

Revision History

Revision	Date	Description
4.0	6/3/99	Preliminary version, including Typhoon specs
3.0	9/3/98	Released for Tsunami
2.1	5/8/98	Added ac specifications, register tables, and reorganized entire book; conditionalized manual for several outputs.
1.1	8/2/96	—
1.0	7/14/95	First External Release
0.1	7/6/95	Preliminary Review

Associated Documents

The following specifications are referenced in this manual.

- *21264 Specifications*, Revision 4.0, August 8, 1998

- *Alpha AXP System Reference Manual*, Revision 6.0, December 1, 1994

- *PCI Local Bus Specification*, Revision 2.1, June 1, 1995

- *Typhoon 21274 Chipset Functional Specification*, October, 1998

1

Introduction

This chapter describes the DIGITAL 21272-AA core logic chipset, a set of application-specific integrated circuits (ASICs) that complement the DIGITAL 21264 family of Alpha microprocessors (hereafter called CPUs). The 21272 chipset provides a solution for low-cost, high-performance, mid to high-end client systems and low-end server systems.

1.1 Chipset Features

The 21272 chipset has the following performance features:

- Support for up to two 21264 CPUs (four for Typhoon variant) without using duplicate cache tags

- Support for up to two 64-bit, 33-MHz PCI buses, each with its own PCI address space

- Support for a large range of main memory capacity using 16MB or 64MB synchronous DRAMs (SDRAMs):

  - 16MB to 1GB using 16Mb SDRAMs

  - 32MB to 4GB using 64Mb SDRAMs

- Low-latency memory access (120-ns CPU access using 83-MHz SDRAMs)

- Support for ECC in main memory

- System clock periods from 12 ns to 15 ns

- Very high bandwidth (2.67-GB/s peak memory bandwidth per processor using SDRAMs and a 83-MHz system clock)

The 21272 chipset has the following technological features:

- Chips are implemented in CMOS5L process

- HSTL-like I/O interfaces between the CPU and the chipset

- HSTL-like I/O interface on the CAPbus interface between the Cchip and the Pchip

- LVTTL interfaces between all 21272 chips (except for the CAPbus)

- LVTTL interfaces between the chipset and memory

1.2 Chipset Overview

This section describes the basic elements and functions of the 21272 chipset. The 21272 chipset consists of the following components:

- 21272–D1 Dchip (data slice chip) – A system can include two, four, or eight Dchips. The Dchips interface with the system data bus and provide the data path between the CPU, DRAM memory, and the Pchip(s).

  21274–D1 Dchip – Functionally equivalent to the 21272–D1 Dchip, except for its memory interface drivers. The 21274–D1 is designed to drive a heavier memory load.

- 21272–C1 Cchip (controller chip) – The Cchip controls the other chips in the chipset, as well as the DRAM memory array in a system. The Cchip interfaces with the CPU's command and address buses.

  21274–C1 Cchip – Used with Typhoon, the 21274–C1 interfaces with four CPUs' command and address buses.

- 21272–P1 Pchip (peripheral interface chip) – A system includes one or two Pchips, the interface to the PCI bus.

For pin counts and signal descriptions for each chip, refer to the pin lists in Chapter 3. For information about Typhoon, refer to the *Typhoon Specification*.

Figure 1–1 shows a typical uniprocessor system using the 21272 chipset in a dual PCI interface configuration. This figure shows that the Cchip can independently control four memory arrays. Although this figure shows all of the data from the DRAMs on one memory data bus, two buses can be used as described in Chapter 2 and Chapter 9.

Figure 1–1 Typical Uniprocessor System with Two PCI Buses

MK272_04.AI4
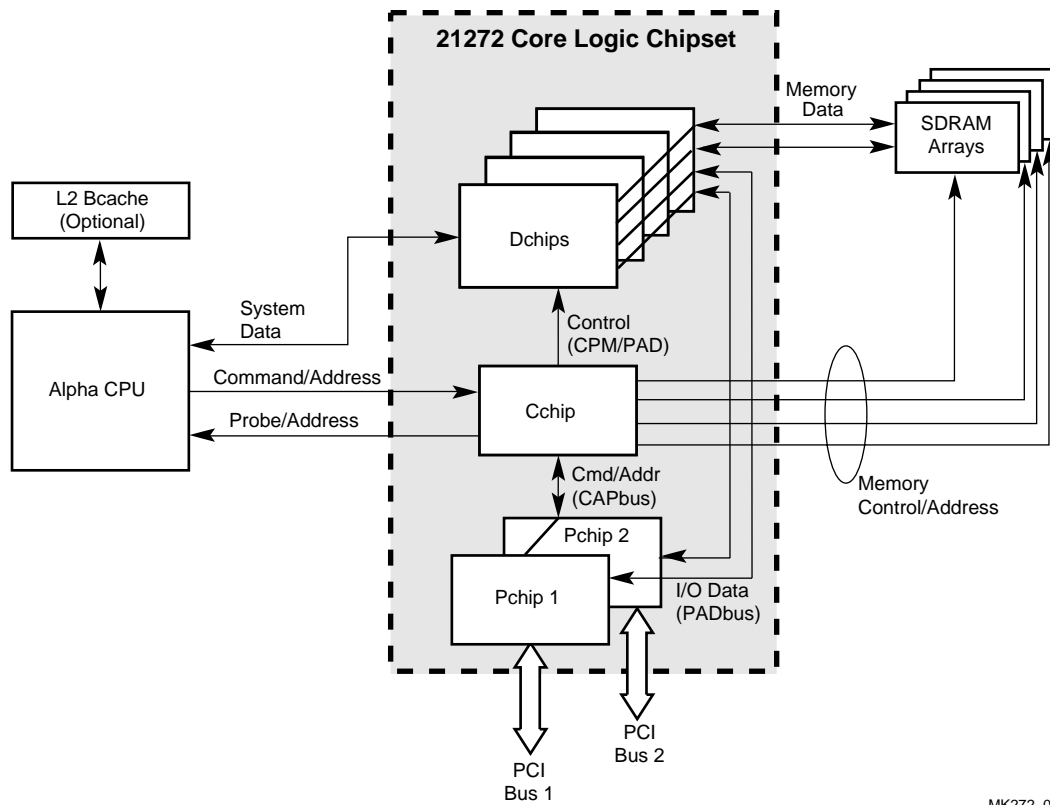
1.2.1 Cchip Overview

The Cchip has the following interfaces:

- System address ports – Two independent ports (four for Typhoon), each of which is a full-duplex, clock-forwarded interface. For details on this interface, refer to the *21264 Specifications*. Each system-address port is clocked at a maximum rate of 3 ns per beat. It takes four beats (12 ns minimum) to transfer a command/address to the Cchip or probe/address from the Cchip.

- DRAM command and address ports – Four independent ports, each of which can supply addresses for one memory array.

- Pchip command and address port (bidirectional) – A single port to the Cchip and Pchip bus (CAPbus). For systems with two Pchips, the port is shared between them. It takes two cycles (20 ns minimum) to transfer a command and address in either direction.

- Dchip control port – A single port with two copies to support eight Dchips.

- TIGbus port – This port handles interrupts, flash ROM, and so forth.

- Miscellaneous test, reset, and clock interfaces.

The Cchip contains the following internal queues and components:

- Three skid buffers – One shared by the Pchips, and one each per CPU, holding requests that have not yet been dispatched to a request queue.

- Four request queues – One per memory array, each of which holds requests for that array (as well as some non-memory requests).

- Wait queues (consisting of pointers to the request queue entries) to enforce ordering requirements among requests across (as well as within) the request queues.

- An interface for each of the CPUs for issuing probes and fills, and for receiving requests and probe results. There are no duplicate tags in the 21272 system, therefore all memory accesses generate probes. The CPU can usually access its Bcache tags in the shadow of a Bcache data transfer, so that this does not impact the Bcache access bandwidth. The system also does not need to have the CPU issue external notification of memory barrier (MB) instructions.

- Pchip interface controller.

- A central bus arbiter that examines and issues requests from the request queues.

- Dchip controllers: one for the PADbus and one for everything else.

- A translator to convert CPU PIO addresses to CSR and PCI addresses.

1.2.2 Dchip Overview

The Dchip has the following interfaces:

- Two memory bus data ports – Each port is 36 bits wide, allowing for 4 bytes of data plus check bits, and can operate at 83 MHz. In some configurations, the two ports operate as a single 72-bit wide port. In other configurations, the ports are half utilized, in which case each operates as an 18-bit wide port.

- Four CPU data ports – Each port is 11 bits wide, allowing for 1 byte of data, plus 1 check bit, plus a pair of forwarded clocks (one in each direction). Each port interfaces with one CPU using a clock-forwarding scheme that allows transfer of data every 3 ns (333 MHz) while compensating for skew. In some configurations, the ports operate as a single 4-byte wide port. In other configurations, the ports operate as two 2-byte wide ports.

- Two Pchip data ports to the Pchip and Dchip bus (PADbus) – For systems with two Pchips, each has its own independent PADbus. Each port is 9 bits wide allowing for 1 byte of data plus 1 check bit and can operate at 83 MHz. In some configurations, only one port is used. In other configurations, each port operates as a single 2-byte wide port. In still other configurations, each port is multiplexed onto five wires with a half-byte transferred to the PADbus every cycle.

- Control from Cchip (CPM/PAD) – The Dchip receives all of its commands from the Cchip. The control from the Cchip consists of setting the switches within the Dchip to move data between ports, or between ports and queues. All connections are possible except from one memory port to the other memory port.

- Test, reset, and clock interfaces.

The Dchip contains the following queues:

- One FromPchipQueue (FPQ), shared by the two Pchips, to hold DMA write data, PIO read data, and peer-to-peer (PTP) data

- One ToPchipQueueMemory (TPQM), shared by the two Pchips, to hold DMA read data

- One ToPchipQueuePIO (TPQP), shared by the two Pchips, to hold PIO write data and PTP data

- Two ToCpuAccumulators (TCA) to allow full bandwidth transfers from a pair of memory buses to a single CPU

- Two ToMemoryAccumulators (TMA) to allow full bandwidth transfers to a pair of memory buses from the CPUs

- One WriteMergeBuffer (WMB) to temporarily hold memory data to be merged with Pchip data for DMA writes

1.2.3 Pchip Overview

The Pchip has a cycle time of 12 ns for the system interface and a cycle time of 30 ns for the PCI interface. It is able to run a 30-ns PCI bus with a 12-ns to 15-ns system interface. It has the following interfaces:

- PCI bus – A single 64-bit PCI implementation running at 33 MHz.

- PCI central resource functions – Arbitration and PCI clock sourcing.

- Dchip port to the PADbus – 40 bits for 4 bytes of data plus check bits. In standard mode, the Pchip receives 4 bytes of data and their 4 associated check bits each cycle (36 pins used). To support a system with eight Dchips, the Pchip has an additional mode where it receives 8 bytes over two cycles, but receives all 8 associated check bits in one cycle (40 pins used). Quadword-based transfers are always used because that is the unit on which the ECC is calculated.

- Cchip command and address port to the CAPbus – It takes two cycles (20 ns minimum) to transfer a command and address in either direction.

- Test, reset, and clock interfaces.

The Pchip contains the following structures:

- Upstream data queue (away from PCI) for DMA and PTP writes.

- Upstream data queue (away from PCI) for PIO and PTP reads.

- Downstream data queue (towards PCI) for DMA and PTP reads.

- Downstream data queue (towards PCI) for PIO and PTP writes.

- Downstream address queues (towards PCI) for PIO and PTP reads and writes.

- Upstream address state machines (away from PCI) for DMA and PTP reads and writes.

- Scatter-gather TLB – The Pchip supports both direct mapped and scatter-gather DMA memory access.

2

Chipset Configurations

This chapter describes the various system configurations that are supported by the 21272 chipset.

2.1 System Building Block Variables

The parameters that may be varied are as follows:

- Number of CPUs (one or two)
- Number of memory data buses (one or two)
- Number of Dchips (two, four, or eight)
- Number of Pchips (one or two)
- Number of main memory DRAM arrays (one, two, three, or four)
- Width of the memory data buses (16 bytes or 32 bytes each)
- Type of DRAM SIMMs (synchronous 16MB or 64MB, with various timing parameters)

The combinations for possible system configurations are listed in Table 2–1.

Table 2–1 System Configurations

Number of Cchips	Number of Dchips	Number of Pchips	Pchip-to-Dchip Bus Width	Number of CPUs	Number of Memory Buses	Memory Bus Width
1	2	1	4 bytes	1	1	16 bytes
1	4	1 or 2	4 bytes	1 or 2	1[1]	32 bytes
1	4	1 or 2	4 bytes	1 or 2	2[2]	16 bytes
1	8	1 or 2	4 bytes	1 or 2	1 or 2[3]	32 bytes
1	8	1 or 2	4 bytes	4	1 or 2[3]	32 bytes

[1] Preferable for uniprocessors.
[2] Preferable for dual processors.
[3] Two memory buses are recommended when using two or four CPUs.

The following notes also apply to Table 2–1.

- A 32-byte memory bus can be half-populated, in which case, it operates as a 16-byte memory bus. The difference is that the maximum number of arrays on the bus is still four.

- Using SDRAMs and a system clock speed of 83 MHz, 16-byte memory buses each deliver 1.35-GB/s and 32-byte memory buses each deliver 2.7-GB/s effective bandwidth.

- The data path from the CPU to the Dchip is always 8 bytes, and can run at 3 ns using clock forwarding for an effective bandwidth of 2.7-GB/s.

- The PADbus (Pchip-to-Dchip) can run at 83 MHz for a raw bandwidth of 400-MB/s, ignoring turnaround cycles.

- In a system with eight Dchips, each Dchip transfers 1 check bit, but only ½ byte per cycle. So the Pchip transfers 8 bytes with check bits every two cycles over a 40-wire interface.

- In a system with eight Dchips, the Dchips support up to four CPUs, but the Cchip only supports one or two CPUs.

- In a system with two memory buses, memory arrays 0 and 2 must be attached to bus 0, while memory arrays 1 and 3 must be attached to bus 1. The memory array number is determined by the set of DRAM control signals from the Cchip. The memory bus number is determined by the set of data signals on the Dchip slices (see Section 7.4).

2.2 Chipset Configurations

The following sections present a variety of typical system configurations.

2.2.1 Systems with Two Dchips

Figure 2–1 shows the minimum system that can be constructed uses two Dchips. The memory bus is 16 bytes wide, and the path between the Pchip and Dchip is 4 bytes wide. A single CPU uses all of the CPU ports on the Dchips. A single Pchip uses all of the Pchip ports on the Dchips.

Figure 2–1 One CPU x One 16-Byte Memory Bus – Two Dchips

LJ-05484.AI4
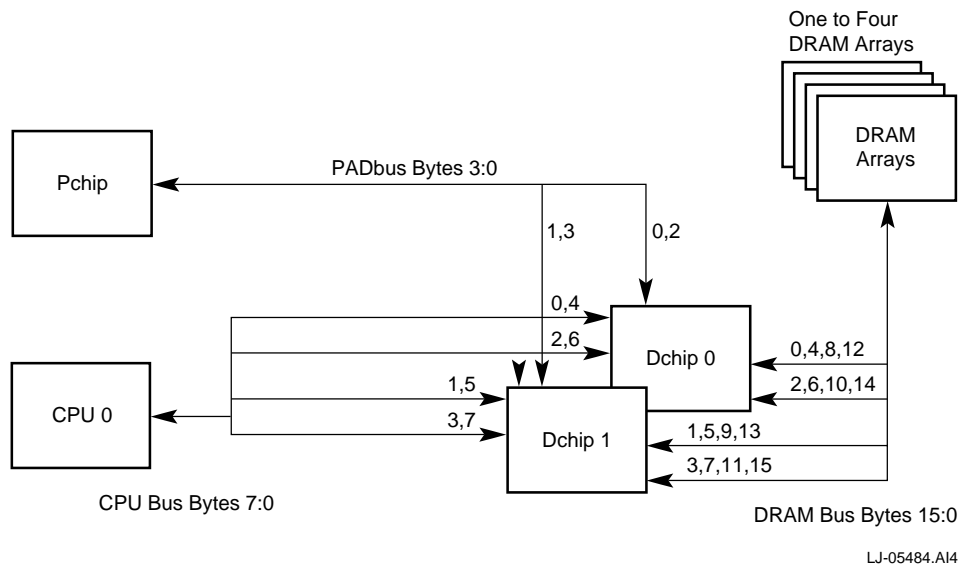
2.2.2 Systems with Four Dchips

Figure 2–2 through Figure 2–4 show how the number and width of memory buses can be varied in systems with four Dchips. Two CPUs are shown but one can be omitted. Two Pchips are shown but one can be omitted.

In Figure 2–2, a system is shown using four Dchips. A single 32-byte bus from the DRAM arrays is split into the two memory ports of the Dchips. A cache block is read or written to memory using two transfers on this bus. Either one or two CPUs may be installed in this system, with each Dchip supplying 2 bytes to each installed CPU. Either one or two Pchips may be installed in this system, with each Dchip supplying 1 byte to each Pchip.

Figure 2–2 One or Two CPU x One 32-Byte Memory Bus – Four Dchips and One or Two Pchips
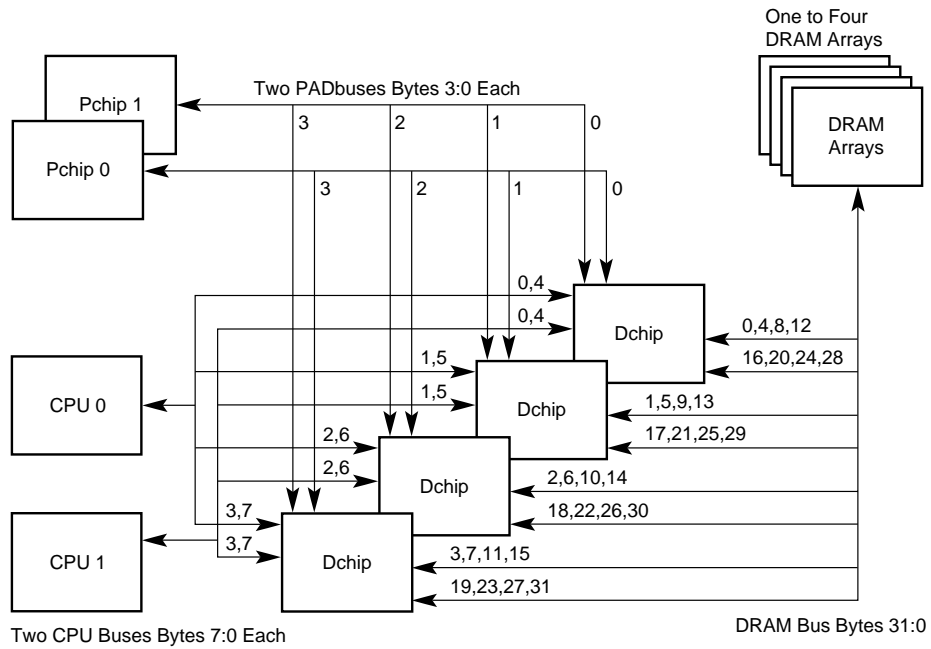
LJ-05485.AI4

Figure 2–3 shows the same system as that in Figure 2–2, except that the memory bus is half-populated so that only a single memory port is used in each Dchip. Four transfers across the memory bus are required for a cache block access.

Figure 2–3 One or Two CPU x One 16-Byte Memory Bus – Four Dchips and One or Two Pchips
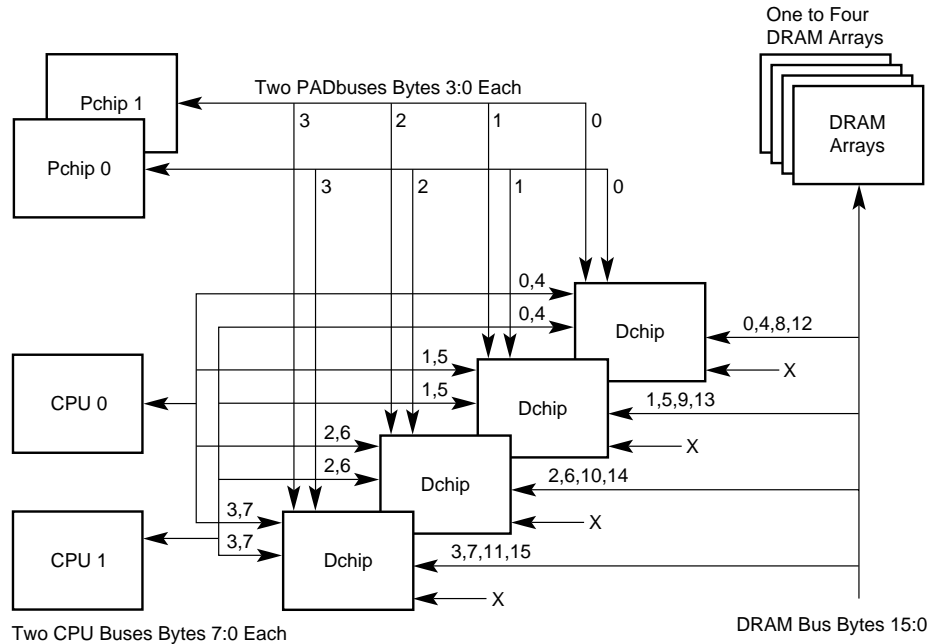
LJ-05486.AI4

In Figure 2–4, another system is shown using four Dchips. In this case, the CPU and Pchip connections are the same as the previous cases, but there are two independent memory buses, each of which is 16 bytes wide so that both memory ports are used in each Dchip. Four transfers across the memory bus are required for a cache block access from either set of DRAM arrays.

Figure 2–4 One or Two CPU x Two 16-Byte Memory Buses – Four Dchips and One or Two Pchips

LJ-05487.AI4
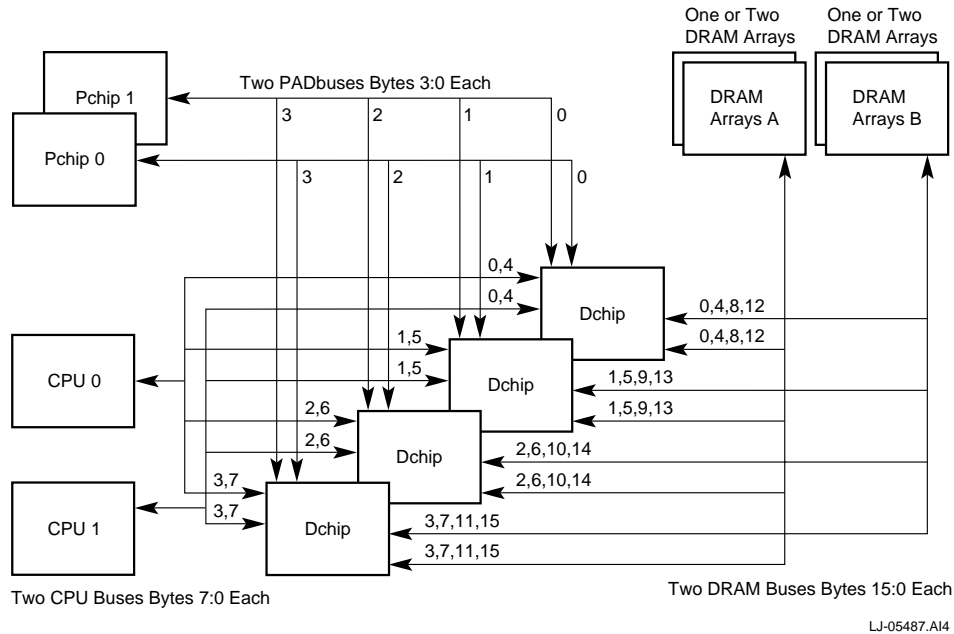
The difference between this system and that shown in Figure 2–2 is that this system has lighter loading on the Dchip memory ports, which may allow easier physical implementation at higher speeds. Also, if two CPUs are installed, simultaneous accesses are possible by means of the two memory buses, which might reduce latency to the critical quadword for any given access. In addition, for a given memory capacity, the system with two small buses has twice as many banks as the system with one large bus. This can improve the throughput on operations such as DMA read-modify-write, which busy a bank for a long period of time. However, the maximum capacity for a given DRAM technology is halved with this organization.

2.2.3 Systems with Eight Dchips

Figure 2–5 and Figure 2–6 show systems using eight Dchips. This enables two full-size 32-byte memory buses. Two Pchips are also enabled in these systems. However, since the data path to each Pchip is only 4 bytes wide, each Dchip transfers ½ byte per cycle as well as 1 check bit (repeated in the two cycles of the associated byte transfer).

Figure 2–5 shows a system that uses eight Dchips and supports two memory buses, each of which is 32 bytes wide. This system provides the highest bandwidth from the DRAM arrays. A cache block from either set of arrays takes two transfers on the appropriate memory bus. Each CPU port on each Dchip is only half utilized. One or two Pchips can be installed, and each has a 4-byte data path to the Dchips with each Dchip transferring ½ byte per cycle.

This "nibble" mode of the PADbus is described in Chapter 7. In the following illustrations, the nomenclature 0H(0L) through 7H(7L) indicates that the lower-order nibble of each byte is transferred in one cycle on the same wires used to carry the high-order nibble of the same byte in the next cycle.

Figure 2–5 One or Two CPU x Two 32-Byte Memory Buses – Eight Dchips and One or Two Pchips
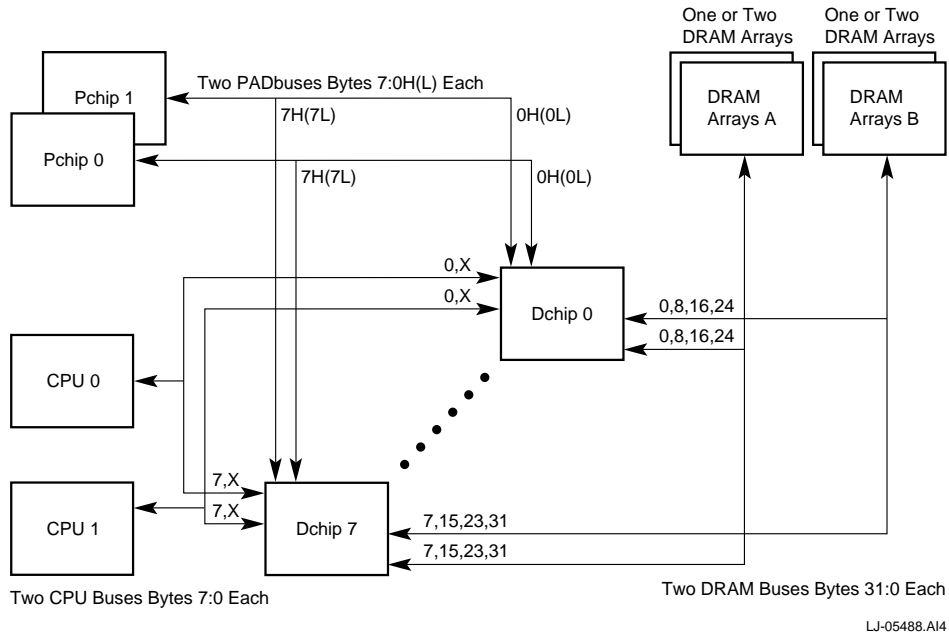
Figure 2–6 shows a system configuration very similar to that shown in Figure 2–5. The only difference is that both of the memory buses are half-populated so that each Dchip uses the data from only one half of each of its memory ports.

Figure 2–6 One or Two CPU x Two 16-Byte Memory Buses – Eight Dchips and One or Two Pchips

LJ-05489.AI4

3
Pinouts
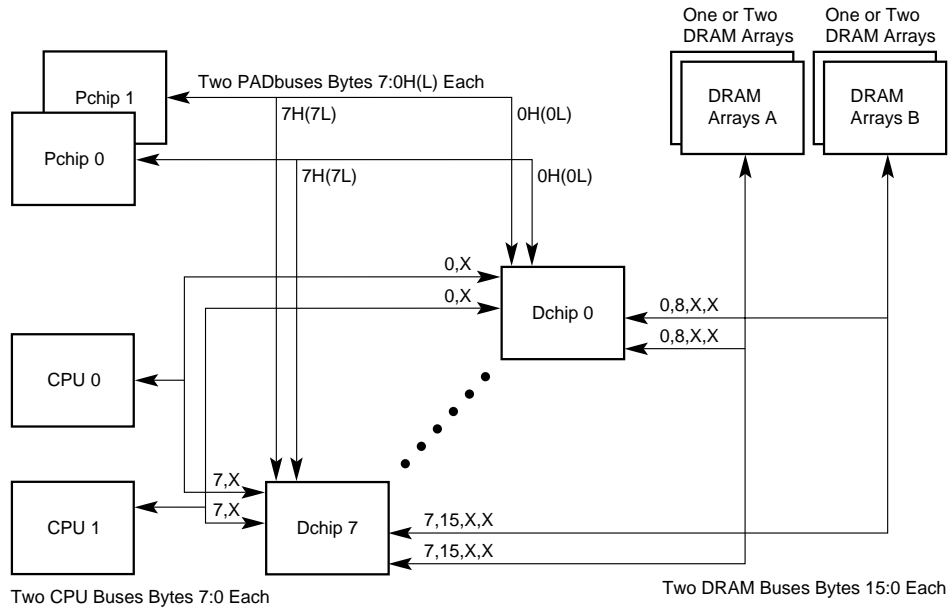
This chapter lists and describes the signal interface pins for each ASIC in the 21272 chipset. The following abbreviations are used in the Type column of the pin list tables:

- B = Bidirectional

- I = Input

- O = Output

- P = Power

3.1 Cchip Pins and Signals

This section provides information about Cchip pins, pin types, pin numbers, and signal definitions.

3.1.1 Cchip Pin List by Function

Table 3–1 lists the pin categories, signal names, types, and signal functions for the Cchip.

Table 3–1 Cchip Pin List by Function

Signal Name	Quantity	Type	Function
CAPbus Interface			
b_cack	1	B	Cchip acknowledge to Pchips
b_cacta_l, **cactb_l**	2	B	Cchip CAPbus active
b_cap<23:0>	24	B	CAPbus command/address
b_capgd<1:0>	2	B	PAD good data sideband signal
b_capsel<1:0>	2	B	Pchip selection
i_creq_l<1:0>	2	I	Pchip CAPbus request
i_pack<1:0>	2	I	Pchip acknowledgment to Cchip
SUBTOTAL	35	—	—

Table 3–1 Cchip Pin List by Function (Continued)

Signal Name	Quantity	Type	Function
CPU Interface			
b_c0clki_l	1	B	CPU 0 clock in
b_c0clko_l	1	B	CPU 0 clock out
b_c2ai_l<14:2>	13	B	CPU 0 command and address
b_c2ao_l<14:2>	13	B	CPU 0 command and address
b_c0div_l	1	B	CPU 0 data in valid
b_c0fv_l	1	B	CPU 0 fill valid
b_c1clki_l	1	B	CPU 1 clock in
b_c1clko_l	1	B	CPU 1 clock out
b_c1ai_l<14:2>	13	B	CPU 1 command and address
b_c1ao_l<14:2>	13	B	CPU 1 command and address
b_c1div_l	1	B	CPU 1 data in valid
b_c1fv_l	1	B	CPU 1 fill valid
SUBTOTAL	60	—	—
Dchip Interface			
b_cpma<7:0>, b_cpmb<7:0>	16	B	CPM command
b_pada<4:0>, b_padb<4:0>	10	B	PAD command
SUBTOTAL	26	—	—
CSALT			
o_nandtr	1	O	NAND tree
i_scanen	1	I	Scan enable
i_scanin	1	I	Scan in
i_trsen_l	1	I	Tristate outputs
SUBTOTAL	4	—	—
Memory Interface			
b_m0a<12:0>	13	B	Array 0 address
b_m0ba<1:0>	2	B	Array 0 bank address
b_m1ba<1:0>	2	B	Array 1 bank address
b_m2ba<1:0>	2	B	Array 2 bank address
b_m3ba<1:0>	2	B	Array 3 bank address
b_m0cs_l<1:0>	2	B	Array 0 chip selects[1]
b_m0dqm<1:0>	2	B	Array 0 OW write control
b_m1a<12:0>	13	B	Array 1 address

Table 3–1 Cchip Pin List by Function (Continued)

Signal Name	Quantity	Type	Function
b_m1cs_l<1:0>	2	B	Array 1 chip selects[1]
b_m1dqm<1:0>	2	B	Array 1 OW write control
b_m2a<12:0>	13	B	Array 2 address
b_m2cs_l<1:0>	2	B	Array 2 chip selects[1]
b_m2dqm<1:0>	2	B	Array 2 OW write control
b_m3a<12:0>	13	B	Array 3 address
b_m3cs_l<1:0>	2	B	Array 3 chip selects[1]
b_m3dqm<1:0>	2	B	Array 3 OW write control
b_mcas_l<3:0>	4	B	CAS[2]
b_mcke_l<3:0>	4	B	CKEs[2]
b_mras_l<3:0>	4	B	RAS[2]
b_mwe_l<3:0>	4	B	Write enables[2]
SUBTOTAL	92	—	—
TIGbus/Interrupt Interface			
b_tas	1	B	TIGbus address strobe
b_tcs_l	1	B	TIGbus read strobe
b_td<7:0>	8	B	TIGbus data
b_tia<2:0>	3	B	TIG interrupt address
b_tioe_l	1	B	Interrupt buffer output enable
b_tis	1	B	TIG interrupt strobe
b_toe_l	1	B	TIG flash ROM output enable
b_twe_l	1	B	TIG flash ROM write enable
SUBTOTAL	17	—	—
Miscellaneous			
b_cfrst<1:0>	2	B	Clock forward reset (per CPU)
i_fwdclk, **i_fwdclk_l**	2	I	Clock forward clock
i_intim_l	1	I	Interval timer
i_modrst_l	1	I	Module reset
b_monitor<7:0>	8	B	Internal signal monitor outputs
b_mpdclk	1	B	Memory presence detect clock
b_mpdd	1	B	Memory presence detect data
b_sromoe_l<1:0>	2	B	SROM output enable (1 per CPU)
i_sysclk, **i_sysclk_l**	2	I	Clock in
b_sysrst{a,b,c}_l	3	B	Reset (deasserts synchronously)

Table 3–1 Cchip Pin List by Function (Continued)

Signal Name	Quantity	Type	Function
i_vref<2:0>	3	I	2-V I/O reference
b_spare<1:0>	2	B	Spare pads
SUBTOTAL	28	—	—
SIGNAL SUBTOTAL	262	—	—
Power Pins			
Vdd	40	P	**Vdd** ring (9000)
Vss	48	P	**Vss** plane (8000)
Vssx	46	P	
SUBTOTAL	133	—	—
SIGNAL/PIN TOTAL	395	—	37 pins not connected

[1] For nonsplit arrays, only bit 0 is used. For split arrays, bit *n* for subarray *n*.

[2] Bit *n* for array *n*.

3.1.2 C4chip Pin List by Function

Table 3–2 lists the pin categories, signal names, types, and signal functions for the C4chip.

Table 3–2 C4chip Pin List by Function

Signal Name	Quantity	Type	Function
CAPbus Interface			
b_cack	1	B	Cchip acknowledge to Pchips
b_cacta_l, **cactb_l**	2	B	Cchip CAPbus active
b_cap<23:0>	24	B	CAPbus command/address
b_capgd<1:0>	2	B	PAD good data sideband signal
b_capsel<1:0>	2	B	Pchip selection
i_creq_l<1:0>	2	I	Pchip CAPbus request
i_pack<1:0>	2	I	Pchip acknowledgment to Cchip
SUBTOTAL	35	—	—

Table 3–2 C4chip Pin List by Function (Continued)

Signal Name	Quantity	Type	Function
CPU Interface			
b_c0clki_l	1	B	CPU 0 clock in
b_c0clko_l	1	B	CPU 0 clock out
b_c0ai_l<14:2>	13	B	CPU 0 command and address
b_c0ao_l<14:2>	13	B	CPU 0 command and address
b_c0div_l	1	B	CPU 0 data in valid
b_c0fv_l	1	B	CPU 0 fill valid
b_c1clki_l	1	B	CPU 1 clock in
b_c1clko_l	1	B	CPU 1 clock out
b_c1ai_l<14:2>	13	B	CPU 1 command and address
b_c1ao_l<14:2>	13	B	CPU 1 command and address
b_c1div_l	1	B	CPU 1 data in valid
b_c1fv_l	1	B	CPU 1 fill valid
b_c2clki_l	1	B	CPU 2 clock in
b_c2clko_l	1	B	CPU 2 clock out
b_c2ai_l<14:2>	13	B	CPU 2 command and address
b_c2ao_l<14:2>	13	B	CPU 2 command and address
b_c2div_l	1	B	CPU 2 data in valid
b_c2fv_l	1	B	CPU 2 fill valid
b_c3clki_l	1	B	CPU 3 clock in
b_c3clko_l	1	B	CPU 3 clock out
b_c3ai_l<14:2>	13	B	CPU 3 command and address
b_c3ao_l<14:2>	13	B	CPU 3 command and address
b_c3div_l	1	B	CPU 3 data in valid
b_c3fv_l	1	B	CPU 3 fill valid
SUBTOTAL	120	—	—
Dchip Interface			
b_cpma<7:0>, **b_cpmb<7:0>**	16	B	CPM command
b_pada<4:0>, **b_padb<4:0>**	10	B	PAD command
SUBTOTAL	26	—	—
CSALT			
o_nandtr	1	O	NAND tree

Table 3–2 C4chip Pin List by Function (Continued)

Signal Name	Quantity	Type	Function
i_scanen	1	I	Scan enable
i_scanin	1	I	Scan in
i_trsen_l	1	I	Tristate outputs
SUBTOTAL	4	—	—
Memory Interface			
b_m0a<12:0>	13	B	Array 0 address
b_m0ba<2:0>	3	B	Array 0 bank address
b_m1ba<2:0>	3	B	Array 1 bank address
b_m2ba<2:0>	3	B	Array 2 bank address
b_m3ba<2:0>	3	B	Array 3 bank address
b_m0cs_l<3:0>	4	B	Array 0 chip selects[1]
b_m0dqm<1:0>	2	B	Array 0 OW write control
b_m1a<12:0>	13	B	Array 1 address
b_m1cs_l<3:0>	4	B	Array 1 chip selects[1]
b_m1dqm<1:0>	2	B	Array 1 OW write control
b_m2a<12:0>	13	B	Array 2 address
b_m2cs_l<3:0>	4	B	Array 2 chip selects[1]
b_m2dqm<1:0>	2	B	Array 2 OW write control
b_m3a<12:0>	13	B	Array 3 address
b_m3cs_l<3:0>	4	B	Array 3 chip selects[1]
b_m3dqm<1:0>	2	B	Array 3 OW write control
b_mcas_l<3:0>	4	B	CAS[2]
b_mcke_l<3:0>	4	B	CKEs[2]
b_mras_l<3:0>	4	B	RAS[2]
b_mwe_l<3:0>	4	B	Write enables[2]
SUBTOTAL	104	—	—
TIGbus/Interrupt Interface			
b_tas	1	B	TIGbus address strobe
b_tcs_l	1	B	TIGbus read strobe
b_td<7:0>	8	B	TIGbus data
b_tia<2:0>	3	B	TIG interrupt address
b_tioe_l	1	B	Interrupt buffer output enable
b_tis, b_tis2	2	B	TIG interrupt strobe
b_toe_l	1	B	TIG flash ROM output enable

Table 3–2 C4chip Pin List by Function (Continued)

Signal Name	Quantity	Type	Function
b_twe_l	1	B	TIG flash ROM write enable
SUBTOTAL	18	—	—
Miscellaneous			
b_cfrst<3:0>	4	B	Clock forward reset (per CPU)
i_fwdclk, i_fwdclk_l	2	I	Clock forward clock
i_intim_l	1	I	Interval timer
i_modrst_l	1	I	Module reset
b_monitor<7:0>	8	B	Internal signal monitor outputs
b_mpdclk	1	B	Memory presence detect clock
b_mpdd	1	B	Memory presence detect data
b_sromoe_l<3:0>	4	B	SROM output enable (1 per CPU)
i_sysclk, i_sysclk_l	2	I	Clock in
b_sysrst{a,b,c}_l	3	B	Reset (deasserts synchronously)
i_vref<4:0>	5	I	2-V I/O reference
SUBTOTAL	32	—	—
Miscellaneous New Signals			
i_fckrep, i_fckrep_l	2	I	?
null	6	—	?
o_fck_fb, o_fck_fb_l	2	O	Feedback
o_fckrep_fb, o_fckrep_fb_l	2	O	Feedback
o_sysck_fb, o_sysck_fb_l	2	O	Feedback
o_sysckrep_fb, o_sysckrep_fb_l	1	B	Memory presence detect clock
SUBTOTAL	15	—	—
SIGNAL SUBTOTAL	353	—	—
Power Pins			
Vdd	58	P	**Vdd** ring (9000)
Vddq	4	P	
Vss	184	P	**Vss** plane (8000)
Vssq	1	P	
SUBTOTAL	246	—	—
SIGNAL/PIN TOTAL	599	—	? pins not connected

[1] For nonsplit arrays, only bit 0 is used. For split arrays, bit n for subarray n.

[2] Bit n for array n.

3.1.3 Cchip Sorted Pin List

Table 3–3 lists the Cchip pins in alphanumeric order by signal name.

Table 3–3 Cchip Pins — Alphanumeric by Signal Name

Signal Name	Pin	Driver	Type
b_c0ai_l<2>	U29	BD16TOD	B
b_c0ai_l<3>	T31	BD16TOD	B
b_c0ai_l<4>	T28	BD16TOD	B
b_c0ai_l<5>	M30	BD16TOD	B
b_c0ai_l<6>	R31	BD16TOD	B
b_c0ai_l<7>	R30	BD16TOD	B
b_c0ai_l<8>	R29	BD16TOD	B
b_c0ai_l<9>	P30	BD16TOD	B
b_c0ai_l<10>	P29	BD16TOD	B
b_c0ai_l<11>	N31	BD16TOD	B
b_c0ai_l<12>	N30	BD16TOD	B
b_c0ai_l<13>	N29	BD16TOD	B
b_c0ai_l<14>	M31	BD16TOD	B
b_c0ao_l<2>	AJ25	BD16TOD	B
b_c0ao_l<3>	AK26	BD16TOD	B
b_c0ao_l<4>	AL27	BD16TOD	B
b_c0ao_l<5>	AH26	BD16TOD	B
b_c0ao_l<6>	AL28	BD16TOD	B
b_c0ao_l<7>	AH31	BD16TOD	B
b_c0ao_l<8>	AF28	BD16TOD	B
b_c0ao_l<9>	AF30	BD16TOD	B
b_c0ao_l<10>	AE29	BD16TOD	B
b_c0ao_l<11>	AE30	BD16TOD	B
b_c0ao_l<12>	AD29	BD16TOD	B
b_c0ao_l<13>	AD30	BD16TOD	B
b_c0ao_l<14>	AC29	BD16TOD	B
b_c0clki_l	R28	BD16TOD	B
b_c0clko_l	AG31	BD16TOD	B
b_c0div_l	AC31	BD16TOD	B
b_c0fv_l	AA28	BD16TOD	B
b_c1ai_l<2>	U28	BD16TOD	B

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_c1ai_l<3>	V30	BD16TOD	B
b_c1ai_l<4>	V29	BD16TOD	B
b_c1ai_l<5>	AA29	BD16TOD	B
b_c1ai_l<6>	W31	BD16TOD	B
b_c1ai_l<7>	W30	BD16TOD	B
b_c1ai_l<8>	W29	BD16TOD	B
b_c1ai_l<9>	Y30	BD16TOD	B
b_c1ai_l<10>	W28	BD16TOD	B
b_c1ai_l<11>	Y29	BD16TOD	B
b_c1ai_l<12>	AA31	BD16TOD	B
b_c1ai_l<13>	AA30	BD16TOD	B
b_c1ai_l<14>	Y28	BD16TOD	B
b_c1ao_l<2>	A27	BD16TOD	B
b_c1ao_l<3>	D26	BD16TOD	B
b_c1ao_l<4>	E29	BD16TOD	B
b_c1ao_l<5>	E30	BD16TOD	B
b_c1ao_l<6>	F29	BD16TOD	B
b_c1ao_l<7>	F31	BD16TOD	B
b_c1ao_l<8>	H28	BD16TOD	B
b_c1ao_l<9>	J28	BD16TOD	B
b_c1ao_l<10>	H31	BD16TOD	B
b_c1ao_l<11>	J30	BD16TOD	B
b_c1ao_l<12>	K29	BD16TOD	B
b_c1ao_l<13>	K30	BD16TOD	B
b_c1ao_l<14>	L29	BD16TOD	B
b_c1clki_l	Y31	BD16TOD	B
b_c1clko_l	G31	BD16TOD	B
b_c1div_l	L30	BD16TOD	B
b_c1fv_l	M29	BD16TOD	B
b_cack	AK11	BD4T	B
b_cacta_l	AJ11	BD4T	B
b_cactb_l	AH12	BD4T	B
b_cap<0>	AH23	BD16TOD	B
b_cap<1>	AK24	BD16TOD	B

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_cap<2>	AL24	BD16TOD	B
b_cap<3>	AK23	BD16TOD	B
b_cap<4>	AL23	BD16TOD	B
b_cap<5>	AJ22	BD16TOD	B
b_cap<6>	AK22	BD16TOD	B
b_cap<7>	AL22	BD16TOD	B
b_cap<8>	AJ21	BD16TOD	B
b_cap<9>	AK21	BD16TOD	B
b_cap<10>	AL21	BD16TOD	B
b_cap<11>	AJ20	BD16TOD	B
b_cap<12>	AJ19	BD16TOD	B
b_cap<13>	AK19	BD16TOD	B
b_cap<14>	AL19	BD16TOD	B
b_cap<15>	AH17	BD16TOD	B
b_cap<16>	AK18	BD16TOD	B
b_cap<17>	AJ17	BD16TOD	B
b_cap<18>	AL17	BD16TOD	B
b_cap<19>	AH16	BD16TOD	B
b_cap<20>	AJ16	BD16TOD	B
b_cap<21>	AL16	BD16TOD	B
b_cap<22>	AJ15	BD16TOD	B
b_cap<23>	AH15	BD16TOD	B
b_capgd<0>	AL11	BD4T	B
b_capgd<1>	AJ12	BD4T	B
b_capsel<0>	AK12	BD4T	B
b_capsel<1>	AL12	BD4T	B
b_cfrst<0>	B15	BD16TOD	B
b_cfrst<1>	C15	BD16TOD	B
b_cpma<0>	P3	BD6T	B
b_cpma<1>	P2	BD6T	B
b_cpma<2>	R4	BD6T	B
b_cpma<3>	R3	BD6T	B
b_cpma<4>	T3	BD6T	B
b_cpma<5>	T4	BD6T	B

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_cpma<6>	U1	BD6T	B
b_cpma<7>	U2	BD6T	B
b_cpmb<0>	U3	BD6T	B
b_cpmb<1>	V2	BD6T	B
b_cpmb<2>	U4	BD6T	B
b_cpmb<3>	V3	BD6T	B
b_cpmb<4>	W1	BD6T	B
b_cpmb<5>	W2	BD6T	B
b_cpmb<6>	W3	BD6T	B
b_cpmb<7>	W4	BD6T	B
b_m0a<0>	A9	BD8T	B
b_m0a<1>	C10	BD8T	B
b_m0a<2>	D11	BD8T	B
b_m0a<3>	B10	BD8T	B
b_m0a<4>	A10	BD8T	B
b_m0a<5>	B11	BD8T	B
b_m0a<6>	A11	BD8T	B
b_m0a<7>	C12	BD8T	B
b_m0a<8>	D13	BD8T	B
b_m0a<9>	B12	BD8T	B
b_m0a<10>	A12	BD8T	B
b_m0a<11>	C13	BD8T	B
b_m0a<12>	B13	BD8T	B
b_m0ba<0>	A13	BD8T	B
b_m0ba<1>	C14	BD8T	B
b_m0cs_l<0>	C9	BD8T	B
b_m0cs_l<1>	B9	BD8T	B
b_m0dqm<0>	A7	BD8T	B
b_m0dqm<1>	C8	BD8T	B
b_m1a<0>	AE1	BD8T	B
b_m1a<1>	AD2	BD8T	B
b_m1a<2>	AD1	BD8T	B
b_m1a<3>	AC3	BD8T	B
b_m1a<4>	AC2	BD8T	B

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_m1a<5>	AC1	BD8T	B
b_m1a<6>	AB3	BD8T	B
b_m1a<7>	AA4	BD8T	B
b_m1a<8>	AB2	BD8T	B
b_m1a<9>	AB1	BD8T	B
b_m1a<10>	AA3	BD8T	B
b_m1a<11>	Y4	BD8T	B
b_m1a<12>	AA2	BD8T	B
b_m1ba<0>	AA1	BD8T	B
b_m1ba<1>	Y3	BD8T	B
b_m1cs_l<0>	AD4	BD8T	B
b_m1cs_l<1>	AE2	BD8T	B
b_m1dqm<0>	AG1	BD8T	B
b_m1dqm<1>	AF3	BD8T	B
b_m2a<0>	H2	BD8T	B
b_m2a<1>	J4	BD8T	B
b_m2a<2>	H3	BD8T	B
b_m2a<3>	G1	BD8T	B
b_m2a<4>	G2	BD8T	B
b_m2a<5>	H4	BD8T	B
b_m2a<6>	G3	BD8T	B
b_m2a<7>	F1	BD8T	B
b_m2a<8>	F2	BD8T	B
b_m2a<9>	F3	BD8T	B
b_m2a<10>	C6	BD8T	B
b_m2a<11>	B6	BD8T	B
b_m2a<12>	A6	BD8T	B
b_m2ba<0>	C7	BD8T	B
b_m2ba<1>	D8	BD8T	B
b_m2cs_l<0>	J3	BD8T	B
b_m2cs_l<1>	H1	BD8T	B
b_m2dqm<0>	L3	BD8T	B
b_m2dqm<1>	K1	BD8T	B
b_m3a<0>	AL8	BD8T	B

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_m3a<1>	AK8	BD8T	B
b_m3a<2>	AL7	BD8T	B
b_m3a<3>	AK7	BD8T	B
b_m3a<4>	AH8	BD8T	B
b_m3a<5>	AJ7	BD8T	B
b_m3a<6>	AL6	BD8T	B
b_m3a<7>	AK6	BD8T	B
b_m3a<8>	AJ6	BD8T	B
b_m3a<9>	AL5	BD8T	B
b_m3a<10>	AK5	BD8T	B
b_m3a<11>	AH6	BD8T	B
b_m3a<12>	AJ5	BD8T	B
b_m3ba<0>	AL4	BD8T	B
b_m3ba<1>	AK4	BD8T	B
b_m3cs_l<0>	AK9	BD8T	B
b_m3cs_l<1>	AJ9	BD8T	B
b_m3dqm<0>	AL10	BD8T	B
b_m3dqm<1>	AK10	BD8T	B
b_mcas_l<0>	B8	BD8T	B
b_mcas_l<1>	AF1	BD8T	B
b_mcas_l<2>	L4	BD8T	B
b_mcas_l<3>	AJ10	BD8T	B
b_mcke_l<0>	A5	BD8T	B
b_mcke_l<1>	AG2	BD8T	B
b_mcke_l<2>	J1	BD8T	B
b_mcke_l<3>	AJ8	BD8T	B
b_monitor<0>	D17	BD4CS	B
b_monitor<1>	B18	BD4CS	B
b_monitor<2>	C18	BD4CS	B
b_monitor<3>	A19	BD4CS	B
b_monitor<4>	A25	BD4CS	B
b_monitor<5>	B7	BD4CS	B
b_monitor<6>	AL25	BD4CS	B
b_monitor<7>	B26	BD4CS	B

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_mpdclk	C24	BD4CS	B
b_mpdd	D23	BD4CS	B
b_mras_l<0>	A8	BD8T	B
b_mras_l<1>	AE3	BD8T	B
b_mras_l<2>	K3	BD8T	B
b_mras_l<3>	AL9	BD8T	B
b_mwe_l<0>	D9	BD8T	B
b_mwe_l<1>	AF2	BD8T	B
b_mwe_l<2>	K2	BD8T	B
b_mwe_l<3>	AH11	BD8T	B
b_pada<0>	M4	BD6T	B
b_pada<1>	L2	BD6T	B
b_pada<2>	L1	BD6T	B
b_pada<3>	M3	BD6T	B
b_pada<4>	N4	BD6T	B
b_padb<0>	M2	BD6T	B
b_padb<1>	M1	BD6T	B
b_padb<2>	N3	BD6T	B
b_padb<3>	N2	BD6T	B
b_padb<4>	N1	BD6T	B
b_spare<0>	B14	BD8CS	B
b_spare<1>	D15	BD8CS	B
b_sromoe_l<0>	A16	BD16TOD	B
b_sromoe_l<1>	B16	BD16TOD	B
b_sysrsta_l	C16	BD6T	B
b_sysrstb_l	D16	BD6T	B
b_sysrstc_l	A15	BD6T	B
b_tas	C20	BD4CS	B
b_tcs_l	B20	BD4CS	B
b_td<0>	A21	BD4CS	B
b_td<1>	B21	BD4CS	B
b_td<2>	D20	BD4CS	B
b_td<3>	C21	BD4CS	B
b_td<4>	A22	BD4CS	B

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_td<5>	B22	BD4CS	B
b_td<6>	D21	BD4CS	B
b_td<7>	C22	BD4CS	B
b_tia<0>	B23	BD4CS	B
b_tia<1>	C23	BD4CS	B
b_tia<2>	A24	BD4CS	B
b_tioe_l	A20	BD4CS	B
b_tis	D19	BD4CS	B
b_toe_l	C19	BD4CS	B
b_twe_l	B24	BD4CS	B
i_creq_l<0>	AL13	IBUF	I
i_creq_l<1>	AJ14	IBUF	I
i_fwdclk	T29	PECLINDIFFA	I
i_fwdclk_l	T30	PECLINDIFFA	I
i_intim_l	B19	IBUF	I
i_modrst_l	C17	IBUF	I
i_pack<0>	AJ13	IBUF	I
i_pack<1>	AK13	IBUF	I
i_scanen	C25	IBUF	I
i_scanin	A26	IBUF	I
i_sysclk	T1	PECLINDIFFA	I
i_sysclk_l	T2	PECLINDIFFA	I
i_trsen_l	B25	IBUF	I
i_vref<0>	N28	DDRV	I
i_vref<1>	AB31	DDRV	I
i_vref<2>	AK20	DDRV	I
o_nandtr	D24	B8	O
vdd	A1	—	P
vdd	A31	—	P
vdd	AB4	—	P
vdd	AB28	—	P
vdd	AE4	—	P
vdd	AE28	—	P
vdd	AH4	—	P

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
vdd	AH7	—	P
vdd	AH10	—	P
vdd	AH14	—	P
vdd	AH18	—	P
vdd	AH22	—	P
vdd	AH25	—	P
vdd	AH28	—	P
vdd	AJ3	—	P
vdd	AJ29	—	P
vdd	AK2	—	P
vdd	AK30	—	P
vdd	AL1	—	P
vdd	AL31	—	P
vdd	B2	—	P
vdd	B30	—	P
vdd	C3	—	P
vdd	C29	—	P
vdd	D10	—	P
vdd	D14	—	P
vdd	D18	—	P
vdd	D22	—	P
vdd	D25	—	P
vdd	D28	—	P
vdd	D4	—	P
vdd	D7	—	P
vdd	G4	—	P
vdd	G28	—	P
vdd	K4	—	P
vdd	K28	—	P
vdd	P4	—	P
vdd	P28	—	P
vdd	V4	—	P
vdd	V28	—	P
vss	A2	—	P

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
vss	A3	—	P
vss	A14	—	P
vss	A17	—	P
vss	A18	—	P
vss	A29	—	P
vss	A30	—	P
vss	AH3	—	P
vss	AH29	—	P
vss	AJ1	—	P
vss	AJ2	—	P
vss	AJ4	—	P
vss	AJ28	—	P
vss	AJ30	—	P
vss	AJ31	—	P
vss	AK1	—	P
vss	AK3	—	P
vss	AK15	—	P
vss	AK29	—	P
vss	AK31	—	P
vss	AL2	—	P
vss	AL3	—	P
vss	AL14	—	P
vss	AL15	—	P
vss	AL18	—	P
vss	AL29	—	P
vss	AL30	—	P
vss	B1	—	P
vss	B3	—	P
vss	B17	—	P
vss	B29	—	P
vss	B31	—	P
vss	C1	—	P
vss	C2	—	P
vss	C4	—	P

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
vss	C28	—	P
vss	C30	—	P
vss	C31	—	P
vss	D3	—	P
vss	D29	—	P
vss	P1	—	P
vss	P31	—	P
vss	R1	—	P
vss	R2	—	P
vss	U30	—	P
vss	U31	—	P
vss	V1	—	P
vss	V31	—	P
vssx	AB29	—	P
vssx	AB30	—	P
vssx	AC28	—	P
vssx	AC30	—	P
vssx	AD28	—	P
vssx	AD31	—	P
vssx	AE31	—	P
vssx	AF29	—	P
vssx	AF31	—	P
vssx	AG29	—	P
vssx	AG30	—	P
vssx	AH19	—	P
vssx	AH20	—	P
vssx	AH21	—	P
vssx	AH24	—	P
vssx	AH30	—	P
vssx	AJ18	—	P
vssx	AJ23	—	P
vssx	AJ24	—	P
vssx	AJ26	—	P
vssx	AJ27	—	P

Table 3–3 Cchip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
vssx	AK14	—	P
vssx	AK16	—	P
vssx	AK17	—	P
vssx	AK27	—	P
vssx	AK28	—	P
vssx	AL20	—	P
vssx	AL26	—	P
vssx	B27	—	P
vssx	C26	—	P
vssx	C27	—	P
vssx	D31	—	P
vssx	E31	—	P
vssx	F28	—	P
vssx	F30	—	P
vssx	G9	—	P
vssx	G29	—	P
vssx	G30	—	P
vssx	H29	—	P
vssx	H30	—	P
vssx	J29	—	P
vssx	J31	—	P
vssx	K31	—	P
vssx	L28	—	P
vssx	L31	—	P
vssx	M28	—	P

3.1.4 Cchip Sorted Pin List

Table 3–4 lists the C4chip pins in alphanumeric order by signal name.

Table 3–4 C4chip Pins — Alphanumeric by Signal Name

Signal Name	Pin	Driver	Type
b_c0ai_l<10>	R04	BODTY	B
b_c0ai_l<11>	R03	BODTY	B
b_c0ai_l<12>	R05	BODTY	B
b_c0ai_l<13>	P02	BODTY	B

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_c0ai_l<14>	R06	BODTY	B
b_c0ai_l<2>	W03	BODTY	B
b_c0ai_l<3>	W04	BODTY	B
b_c0ai_l<4>	W02	BODTY	B
b_c0ai_l<5>	P04	BODTY	B
b_c0ai_l<6>	V06	BODTY	B
b_c0ai_l<7>	V05	BODTY	B
b_c0ai_l<8>	V04	BODTY	B
b_c0ai_l<9>	T04	BODTY	B
b_c0ao_l<10>	AN09	BODTY	B
b_c0ao_l<11>	AH09	BODTY	B
b_c0ao_l<12>	AJ07	BODTY	B
b_c0ao_l<13>	AN07	BODTY	B
b_c0ao_l<14>	AM05	BODTY	B
b_c0ao_l<2>	AH13	BODTY	B
b_c0ao_l<3>	AM10	BODTY	B
b_c0ao_l<4>	AN11	BODTY	B
b_c0ao_l<5>	AJ10	BODTY	B
b_c0ao_l<6>	AM09	BODTY	B
b_c0ao_l<7>	AH11	BODTY	B
b_c0ao_l<8>	AJ09	BODTY	B
b_c0ao_l<9>	AJ08	BODTY	B
b_c0clki_l	R01	BODTY	B
b_c0clko_l	AL10	BODTY	B
b_c0div_l	AL05	BODTY	B
b_c0fv_l	AH07	BODTY	B
b_c1ai_l<10>	P06	BODTY	B
b_c1ai_l<11>	N05	BODTY	B
b_c1ai_l<12>	P05	BODTY	B
b_c1ai_l<13>	P03	BODTY	B
b_c1ai_l<14>	N02	BODTY	B
b_c1ai_l<2>	K02	BODTY	B
b_c1ai_l<3>	N06	BODTY	B
b_c1ai_l<4>	L04	BODTY	B

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_c1ai_l<5>	P01	BODTY	B
b_c1ai_l<6>	M01	BODTY	B
b_c1ai_l<7>	N04	BODTY	B
b_c1ai_l<8>	L02	BODTY	B
b_c1ai_l<9>	N01	BODTY	B
b_c1ao_l<10>	E01	BODTY	B
b_c1ao_l<11>	E03	BODTY	B
b_c1ao_l<12>	F02	BODTY	B
b_c1ao_l<13>	H06	BODTY	B
b_c1ao_l<14>	E02	BODTY	B
b_c1ao_l<2>	A09	BODTY	B
b_c1ao_l<3>	E08	BODTY	B
b_c1ao_l<4>	F09	BODTY	B
b_c1ao_l<5>	C06	BODTY	B
b_c1ao_l<6>	F08	BODTY	B
b_c1ao_l<7>	E06	BODTY	B
b_c1ao_l<8>	B06	BODTY	B
b_c1ao_l<9>	A05	BODTY	B
b_c1clki_l	M04	BODTY	B
b_c1clko_l	E07	BODTY	B
b_c1div_l	G05	BODTY	B
b_c1fv_l	H01	BODTY	B
b_c2ai_l<10>	Y03	BODTY	B
b_c2ai_l<11>	Y04	BODTY	B
b_c2ai_l<12>	Y01	BODTY	B
b_c2ai_l<13>	W06	BODTY	B
b_c2ai_l<14>	Y02	BODTY	B
b_c2ai_l<2>	AC02	BODTY	B
b_c2ai_l<3>	AA04	BODTY	B
b_c2ai_l<4>	AB04	BODTY	B
b_c2ai_l<5>	W05	BODTY	B
b_c2ai_l<6>	AA01	BODTY	B
b_c2ai_l<7>	Y06	BODTY	B
b_c2ai_l<8>	AA05	BODTY	B

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_c2ai_l<9>	AA02	BODTY	B
b_c2ao_l<10>	AD03	BODTY	B
b_c2ao_l<11>	AE04	BODTY	B
b_c2ao_l<12>	AC06	BODTY	B
b_c2ao_l<13>	AD05	BODTY	B
b_c2ao_l<14>	AD04	BODTY	B
b_c2ao_l<2>	AG06	BODTY	B
b_c2ao_l<3>	AH02	BODTY	B
b_c2ao_l<4>	AJ02	BODTY	B
b_c2ao_l<5>	AF06	BODTY	B
b_c2ao_l<6>	AF01	BODTY	B
b_c2ao_l<7>	AG05	BODTY	B
b_c2ao_l<8>	AE03	BODTY	B
b_c2ao_l<9>	AF03	BODTY	B
b_c2clki_l	Y05	BODTY	B
b_c2clko_l	AF05	BODTY	B
b_c2div_l	AA06	BODTY	B
b_c2fv_l	AB03	BODTY	B
b_c3ai_l<10>	L06	BODTY	B
b_c3ai_l<11>	J04	BODTY	B
b_c3ai_l<12>	L05	BODTY	B
b_c3ai_l<13>	L01	BODTY	B
b_c3ai_l<14>	K04	BODTY	B
b_c3ai_l<2>	G04	BODTY	B
b_c3ai_l<3>	J01	BODTY	B
b_c3ai_l<4>	H05	BODTY	B
b_c3ai_l<5>	M03	BODTY	B
b_c3ai_l<6>	K06	BODTY	B
b_c3ai_l<7>	H03	BODTY	B
b_c3ai_l<8>	K03	BODTY	B
b_c3ai_l<9>	J05	BODTY	B
b_c3ao_l<10>	D11	BODTY	B
b_c3ao_l<11>	F13	BODTY	B
b_c3ao_l<12>	A11	BODTY	B

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_c3ao_l<13>	D10	BODTY	B
b_c3ao_l<14>	F11	BODTY	B
b_c3ao_l<2>	C16	BODTY	B
b_c3ao_l<3>	B15	BODTY	B
b_c3ao_l<4>	C15	BODTY	B
b_c3ao_l<5>	B14	BODTY	B
b_c3ao_l<6>	F15	BODTY	B
b_c3ao_l<7>	C14	BODTY	B
b_c3ao_l<8>	B13	BODTY	B
b_c3ao_l<9>	D13	BODTY	B
b_c3clki_l	K01	BODTY	B
b_c3clko_l	D12	BODTY	B
b_c3div_l	E09	BODTY	B
b_c3fv_l	C10	BODTY	B
b_cack	AM23	BDZ20C	B
b_cacta_l	AK22	BDZ50C	B
b_cactb_l	AN21	BDZ50C	B
b_cap<0>	AL19	BODTY	B
b_cap<10>	AM16	BODTY	B
b_cap<11>	AN17	BODTY	B
b_cap<12>	AL16	BODTY	B
b_cap<13>	AM15	BODTY	B
b_cap<14>	AK15	BODTY	B
b_cap<15>	AN15	BODTY	B
b_cap<16>	AL15	BODTY	B
b_cap<17>	AM14	BODTY	B
b_cap<18>	AH15	BODTY	B
b_cap<19>	AN14	BODTY	B
b_cap<1>	AN19	BODTY	B
b_cap<20>	AL14	BODTY	B
b_cap<21>	AM13	BODTY	B
b_cap<22>	AJ13	BODTY	B
b_cap<23>	AK12	BODTY	B
b_cap<2>	AK19	BODTY	B

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_cap<3>	AM19	BODTY	B
b_cap<4>	AL18	BODTY	B
b_cap<5>	AH17	BODTY	B
b_cap<6>	AN18	BODTY	B
b_cap<7>	AM18	BODTY	B
b_cap<8>	AJ17	BODTY	B
b_cap<9>	AM17	BODTY	B
b_capgd<0>	AK21	BDZ50C	B
b_capgd<1>	AN22	BDZ50C	B
b_capsel<0>	AK23	BDZ30C	B
b_capsel<1>	AH21	BDZ30C	B
b_cfrst<0>	B18	BODTY	B
b_cfrst<1>	D17	BODTY	B
b_cfrst<2>	B17	BODTY	B
b_cfrst<3>	E17	BODTY	B
b_cpma<0>	AL26	BDZ20C	B
b_cpma<1>	AH24	BDZ20C	B
b_cpma<2>	AN25	BDZ20C	B
b_cpma<3>	AK27	BDZ20C	B
b_cpma<4>	AL25	BDZ20C	B
b_cpma<5>	AH25	BDZ20C	B
b_cpma<6>	AJ27	BDZ20C	B
b_cpma<7>	AN26	BDZ20C	B
b_cpmb<0>	AN27	BDZ20C	B
b_cpmb<1>	AM29	BDZ20C	B
b_cpmb<2>	AH26	BDZ20C	B
b_cpmb<3>	AN28	BDZ20C	B
b_cpmb<4>	AJ28	BDZ20C	B
b_cpmb<5>	AM28	BDZ20C	B
b_cpmb<6>	AH27	BDZ20C	B
b_cpmb<7>	AN30	BDZ20C	B
b_m0a<0>	AE33	BDZ20C	B
b_m0a<10>	AH29	BDZ20C	B
b_m0a<11>	AH32	BDZ20C	B

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_m0a<12>	AG28	BDZ20C	B
b_m0a<1>	AF29	BDZ20C	B
b_m0a<2>	AE31	BDZ20C	B
b_m0a<3>	AE28	BDZ20C	B
b_m0a<4>	AG29	BDZ20C	B
b_m0a<5>	AF33	BDZ20C	B
b_m0a<6>	AG33	BDZ20C	B
b_m0a<7>	AH31	BDZ20C	B
b_m0a<8>	AF28	BDZ20C	B
b_m0a<9>	AH33	BDZ20C	B
b_m0ba<0>	AJ31	BDZ20C	B
b_m0ba<1>	AJ33	BDZ20C	B
b_m0ba<2>	AN29	BDZ20C	B
b_m0cs_l<0>	AD33	BDZ20C	B
b_m0cs_l<1>	AD31	BDZ20C	B
b_m0cs_l<2>	AF31	BDZ20C	B
b_m0cs_l<3>	AD28	BDZ20C	B
b_m0dqm<0>	AD29	BDZ20C	B
b_m0dqm<1>	AE32	BDZ20C	B
b_m1a<0>	L30	BDZ20C	B
b_m1a<10>	P33	BDZ20C	B
b_m1a<11>	R28	BDZ20C	B
b_m1a<12>	P32	BDZ20C	B
b_m1a<1>	M33	BDZ20C	B
b_m1a<2>	N30	BDZ20C	B
b_m1a<3>	L32	BDZ20C	B
b_m1a<4>	N33	BDZ20C	B
b_m1a<5>	P28	BDZ20C	B
b_m1a<6>	N29	BDZ20C	B
b_m1a<7>	P29	BDZ20C	B
b_m1a<8>	P31	BDZ20C	B
b_m1a<9>	N32	BDZ20C	B
b_m1ba<0>	R29	BDZ20C	B
b_m1ba<1>	R31	BDZ20C	B

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_m1ba<2>	R30	BDZ20C	B
b_m1cs_l<0>	L33	BDZ20C	B
b_m1cs_l<1>	K30	BDZ20C	B
b_m1cs_l<2>	M31	BDZ20C	B
b_m1cs_l<3>	N28	BDZ20C	B
b_m1dqm<0>	J29	BDZ20C	B
b_m1dqm<1>	L28	BDZ20C	B
b_m2a<0>	Y32	BDZ20C	B
b_m2a<10>	AA30	BDZ20C	B
b_m2a<11>	AB33	BDZ20C	B
b_m2a<12>	AC30	BDZ20C	B
b_m2a<1>	W28	BDZ20C	B
b_m2a<2>	Y33	BDZ20C	B
b_m2a<3>	Y30	BDZ20C	B
b_m2a<4>	Y31	BDZ20C	B
b_m2a<5>	Y29	BDZ20C	B
b_m2a<6>	AA29	BDZ20C	B
b_m2a<7>	Y28	BDZ20C	B
b_m2a<8>	AA33	BDZ20C	B
b_m2a<9>	AB30	BDZ20C	B
b_m2ba<0>	AA28	BDZ20C	B
b_m2ba<1>	AB31	BDZ20C	B
b_m2ba<2>	AD32	BDZ20C	B
b_m2cs_l<0>	W32	BDZ20C	B
b_m2cs_l<1>	W30	BDZ20C	B
b_m2cs_l<2>	W31	BDZ20C	B
b_m2cs_l<3>	W29	BDZ20C	B
b_m2dqm<0>	V31	BDZ20C	B
b_m2dqm<1>	T30	BDZ20C	B
b_m3a<0>	G28	BDZ20C	B
b_m3a<10>	J31	BDZ20C	B
b_m3a<11>	G30	BDZ20C	B
b_m3a<12>	J33	BDZ20C	B
b_m3a<1>	F32	BDZ20C	B

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_m3a<2>	F29	BDZ20C	B
b_m3a<3>	F33	BDZ20C	B
b_m3a<4>	H28	BDZ20C	B
b_m3a<5>	E32	BDZ20C	B
b_m3a<6>	G33	BDZ20C	B
b_m3a<7>	H33	BDZ20C	B
b_m3a<8>	G29	BDZ20C	B
b_m3a<9>	J28	BDZ20C	B
b_m3ba<0>	K28	BDZ20C	B
b_m3ba<1>	H31	BDZ20C	B
b_m3ba<2>	K31	BDZ20C	B
b_m3cs_l<0>	C29	BDZ20C	B
b_m3cs_l<1>	A29	BDZ20C	B
b_m3cs_l<2>	E33	BDZ20C	B
b_m3cs_l<3>	D33	BDZ20C	B
b_m3dqm<0>	F26	BDZ20C	B
b_m3dqm<1>	A28	BDZ20C	B
b_mcas_l<0>	AC28	BDZ20C	B
b_mcas_l<1>	J32	BDZ20C	B
b_mcas_l<2>	V30	BDZ20C	B
b_mcas_l<3>	B28	BDZ20C	B
b_mcke<0>	AC33	BDZ20C	B
b_mcke<1>	K33	BDZ20C	B
b_mcke<2>	R33	BDZ20C	B
b_mcke<3>	C28	BDZ20C	B
b_monitor<0>	E21	BDZ50C	B
b_monitor<1>	A21	BDZ50C	B
b_monitor<2>	D22	BDZ50C	B
b_monitor<3>	D21	BDZ50C	B
b_monitor<4>	A22	BDZ50C	B
b_monitor<5>	D23	BDZ50C	B
b_monitor<6>	F21	BDZ50C	B
b_monitor<7>	C22	BDZ50C	B
b_mpdclk	E19	BDZ50C	B

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_mpdd	F19	BDZ50C	B
b_mras_l<0>	AE30	BDZ20C	B
b_mras_l<1>	K29	BDZ20C	B
b_mras_l<2>	V28	BDZ20C	B
b_mras_l<3>	F27	BDZ20C	B
b_mwe_l<0>	AC29	BDZ20C	B
b_mwe_l<1>	L29	BDZ20C	B
b_mwe_l<2>	V29	BDZ20C	B
b_mwe_l<3>	E28	BDZ20C	B
b_pada<0>	AL22	BDZ20C	B
b_pada<1>	AK24	BDZ20C	B
b_pada<2>	AN23	BDZ20C	B
b_pada<3>	AJ24	BDZ20C	B
b_pada<4>	AM25	BDZ20C	B
b_padb<0>	AJ23	BDZ20C	B
b_padb<1>	AH23	BDZ20C	B
b_padb<2>	AJ25	BDZ20C	B
b_padb<3>	AN24	BDZ20C	B
b_padb<4>	AL24	BDZ20C	B
b_sromoe_l<0>	C18	BODTY	B
b_sromoe_l<1>	D18	BODTY	B
b_sromoe_l<2>	A18	BODTY	B
b_sromoe_l<3>	F17	BODTY	B
b_sysrsta_l	B21	BDZ20C	B
b_sysrstb_l	C20	BDZ20C	B
b_sysrstc_l	A20	BDZ20C	B
b_tas	F23	BDZ50C	B
b_tcs_l	E24	BDZ50C	B
b_td<0>	D25	BDZ50C	B
b_td<1>	A24	BDZ50C	B
b_td<2>	C24	BDZ50C	B
b_td<3>	C26	BDZ50C	B
b_td<4>	F24	BDZ50C	B
b_td<5>	A25	BDZ50C	B

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_td<6>	E26	BDZ50C	B
b_td<7>	C25	BDZ50C	B
b_tia<0>	F25	BDZ50C	B
b_tia<1>	E27	BDZ50C	B
b_tia<2>	A26	BDZ50C	B
b_tioe_l	A23	BDZ50C	B
b_tis	E23	BDZ50C	B
b_tis2	B25	BDZ50C	B
b_toe_l	B24	BDZ50C	B
b_twe_l	A27	BDZ50C	B
i_creq_l<0>	AM21	BDZ50C	I
i_creq_l<1>	AJ21	BDZ50C	I
i_fckrep	U01	PECLINTY	I
i_fckrep_l	T02	PECLINTY	I
i_fwdclk	V01	PECLINTY	I
i_fwdclk_l	U02	PECLINTY	I
i_intim_l	D20	BDZ50C	I
i_modrst_l	A19	BDZ50C	I
i_pack<0>	AK20	BDZ50C	I
i_pack<1>	AL20	BDZ50C	I
i_scanen	D19	BDZ50C	I
i_scanin	C19	BDZ50C	I
i_sysckrep	U33	PECLINTY	I
i_sysckrep_l	T32	PECLINTY	I
i_sysclk	V33	PECLINTY	I
i_sysclk_l	U32	PECLINTY	I
i_trsen_l	B19	BDZ50C	I
i_vref<0>	AN06		I
i_vref<1>	K05		I
i_vref<2>	V03		I
i_vref<3>	B09		I
i_vref<4>	AK13		I
null	AM31		
null	AL32		

Table 3–4 C4chip Pins — Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
null	C32		
null	B31		
null	B03		
null	AL02		
o_fck_fb	W01	PECLOUT	O
o_fck_fb_l	V02	PECLOUT	O
o_fckrep_fb	T01	PECLOUT	O
o_fckrep_fb_l	R02	PECLOUT	O
o_nandtr	B20	BDZ50C	O
o_sysck_fb	W33	PECLOUT	O
o_sysck_fb_l	V32	PECLOUT	O
o_sysckrep_fb	R32	PECLOUT	O
o_sysckrep_fb_l	T33	PECLOUT	O
Vdd	AM03		P
Vdd	AJ19		P
Vdd	A04		P
Vdd	C02		P
Vdd	AK01		P
Vddq	U30		P
Vddq	T31		P
Vddq	T03		P
Vddq	U04		P
Vss	AN05		P
Vss	AN04		P
Vss	AM06		P
Vss	AJ06		P
Vss	AH08		P
Vss	AL06		P
Vss	AN08		P

Table 3–5 shows the Cchip pins in alphanumeric order by pin number.

Table 3–5 Cchip Pins — Alphanumeric by Pin Number

Pin Number	Signal Name	Driver	Type
A1	**vdd**	—	P

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
A2	**vss**	—	P
A3	**vss**	—	P
A5	**b_mcke_l<0>**	BD8T	B
A6	**b_m2a<12>**	BD8T	B
A7	**b_m0dqm<0>**	BD8T	B
A8	**b_mras_l<0>**	BD8T	B
A9	**b_m0a<0>**	BD8T	B
A10	**b_m0a<4>**	BD8T	B
A11	**b_m0a<6>**	BD8T	B
A12	**b_m0a<10>**	BD8T	B
A13	**b_m0ba<0>**	BD8T	B
A14	**vss**	—	P
A15	**b_sysrstc_l**	BD6T	B
A16	**b_sromoe_l<0>**	BD16TOD	B
A17	**vss**	—	P
A18	**vss**	—	P
A19	**b_monitor<3>**	BD4CS	B
A20	**b_tioe_l**	BD4CS	B
A21	**b_td<0>**	BD4CS	B
A22	**b_td<4>**	BD4CS	B
A24	**b_tia<2>**	BD4CS	B
A25	**b_monitor<4>**	BD4CS	B
A26	**i_scanin**	IBUF	I
A27	**b_c1ao_l<2>**	BD16TOD	B
A29	**vss**	—	P
A30	**vss**	—	P
A31	**vdd**	—	P
AA1	**b_m1ba<0>**	BD8T	B
AA2	**b_m1a<12>**	BD8T	B
AA3	**b_m1a<10>**	BD8T	B
AA4	**b_m1a<7>**	BD8T	B
AA28	**b_c0fv_l**	BD16TOD	B
AA29	**b_c1ai_l<5>**	BD16TOD	B
AA30	**b_c1ai_l<13>**	BD16TOD	B

Table 3–5 Cchip Pins — Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
AA31	**b_c1ai_l<12>**	BD16TOD	B
AB1	**b_m1a<9>**	BD8T	B
AB2	**b_m1a<8>**	BD8T	B
AB3	**b_m1a<6>**	BD8T	B
AB4	**vdd**	—	P
AB28	**vdd**	—	P
AB29	**vssx**	—	P
AB30	**vssx**	—	P
AB31	**i_vref<1>**	DDRV	I
AC1	**b_m1a<5>**	BD8T	B
AC2	**b_m1a<4>**	BD8T	B
AC3	**b_m1a<3>**	BD8T	B
AC28	**vssx**	—	P
AC29	**b_c0ao_l<14>**	BD16TOD	B
AC30	**vssx**	—	P
AC31	**b_c0div_l**	BD16TOD	B
AD1	**b_m1a<2>**	BD8T	B
AD2	**b_m1a<1>**	BD8T	B
AD4	**b_m1cs_l<0>**	BD8T	B
AD28	**vssx**	—	P
AD29	**b_c0ao_l<12>**	BD16TOD	B
AD30	**b_c0ao_l<13>**	BD16TOD	B
AD31	**vssx**	—	P
AE1	**b_m1a<0>**	BD8T	B
AE2	**b_m1cs_l<1>**	BD8T	B
AE3	**b_mras_l<1>**	BD8T	B
AE4	**vdd**	—	P
AE28	**vdd**	—	P
AE29	**b_c0ao_l<10>**	BD16TOD	B
AE30	**b_c0ao_l<11>**	BD16TOD	B
AE31	**vssx**	—	P
AF1	**b_mcas_l<1>**	BD8T	B
AF2	**b_mwe_l<1>**	BD8T	B
AF3	**b_m1dqm<1>**	BD8T	B

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
AF28	b_c0ao_l<8>	BD16TOD	B
AF29	vssx	—	P
AF30	b_c0ao_l<9>	BD16TOD	B
AF31	vssx	—	P
AG1	b_m1dqm<0>	BD8T	B
AG2	b_mcke_l<1>	BD8T	B
AG29	vssx	—	P
AG30	vssx	—	P
AG31	b_c0clko_l	BD16TOD	B
AH3	vss	—	P
AH4	vdd	—	P
AH6	b_m3a<11>	BD8T	B
AH7	vdd	—	P
AH8	b_m3a<4>	BD8T	B
AH10	vdd		P
AH11	b_mwe_l<3>	BD8T	B
AH12	b_cactb_l	BD4T	B
AH14	vdd	—	P
AH15	b_cap<23>	BD16TOD	B
AH16	b_cap<19>	BD16TOD	B
AH17	b_cap<15>	BD16TOD	B
AH18	vdd	—	P
AH19	vssx	—	P
AH20	vssx	—	P
AH21	vssx	—	P
AH22	vdd	—	P
AH23	b_cap<0>	BD16TOD	B
AH24	vssx	—	P
AH25	vdd	—	P
AH26	b_c0ao_l<5>	BD16TOD	B
AH28	vdd	—	P
AH29	vss	—	P
AH30	vssx	—	P
AH31	b_c0ao_l<7>	BD16TOD	B

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
AJ1	**vss**	—	P
AJ2	**vss**	—	P
AJ3	**vdd**	—	P
AJ4	**vss**	—	P
AJ5	**b_m3a<12>**	BD8T	B
AJ6	**b_m3a<8>**	BD8T	B
AJ7	**b_m3a<5>**	BD8T	B
AJ8	**b_mcke_l<3>**	BD8T	B
AJ9	**b_m3cs_l<1>**	BD8T	B
AJ10	**b_mcas_l<3>**	BD8T	B
AJ11	**b_cacta_l**	BD4T	B
AJ12	**b_capgd<1>**	BD4T	B
AJ13	**i_pack<0>**	IBUF	I
AJ14	**i_creq_l<1>**	IBUF	I
AJ15	**b_cap<22>**	BD16TOD	B
AJ16	**b_cap<20>**	BD16TOD	B
AJ17	**b_cap<17>**	BD16TOD	B
AJ18	**vssx**	—	P
AJ19	**b_cap<12>**	BD16TOD	B
AJ20	**b_cap<11>**	BD16TOD	B
AJ21	**b_cap<8>**	BD16TOD	B
AJ22	**b_cap<5>**	BD16TOD	B
AJ23	**vssx**	—	P
AJ24	**vssx**	—	P
AJ25	**b_c0ao_l<2>**	BD16TOD	B
AJ26	**vssx**	—	P
AJ27	**vssx**	—	P
AJ28	**vss**	—	P
AJ29	**vdd**	—	P
AJ30	**vss**	—	P
AJ31	**vss**	—	P
AK1	**vss**	—	P
AK2	**vdd**	—	P
AK3	**vss**	—	P

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
AK4	b_m3ba<1>	BD8T	B
AK5	b_m3a<10>	BD8T	B
AK6	b_m3a<7>	BD8T	B
AK7	b_m3a<3>	BD8T	B
AK8	b_m3a<1>	BD8T	B
AK9	b_m3cs_l<0>	BD8T	B
AK10	b_m3dqm<1>	BD8T	B
AK11	b_cack	BD4T	B
AK12	b_capsel<0>	BD4T	B
AK13	i_pack<1>	IBUF	I
AK14	vssx	—	P
AK15	vss	—	P
AK16	vssx	—	P
AK17	vssx	—	P
AK18	b_cap<16>	BD16TOD	B
AK19	b_cap<13>	BD16TOD	B
AK20	i_vref<2>	DDRV	I
AK21	b_cap<9>	BD16TOD	B
AK22	b_cap<6>	BD16TOD	B
AK23	b_cap<3>	BD16TOD	B
AK24	b_cap<1>	BD16TOD	B
AK26	b_c0ao_l<3>	BD16TOD	B
AK27	vssx	—	P
AK28	vssx	—	P
AK29	vss	—	P
AK30	vdd	—	P
AK31	vss	—	P
AL1	vdd	—	P
AL2	vss	—	P
AL3	vss	—	P
AL4	b_m3ba<0>	BD8T	B
AL5	b_m3a<9>	BD8T	B
AL6	b_m3a<6>	BD8T	B
AL7	b_m3a<2>	BD8T	B

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
AL8	b_m3a<0>	BD8T	B
AL9	b_mras_l<3>	BD8T	B
AL10	b_m3dqm<0>	BD8T	B
AL11	b_capgd<0>	BD4T	B
AL12	b_capsel<1>	BD4T	B
AL13	i_creq_l<0>	IBUF	I
AL14	vss	—	P
AL15	vss	—	P
AL16	b_cap<21>	BD16TOD	B
AL17	b_cap<18>	BD16TOD	B
AL18	vss	—	P
AL19	b_cap<14>	BD16TOD	B
AL20	vssx	—	P
AL21	b_cap<10>	BD16TOD	B
AL22	b_cap<7>	BD16TOD	B
AL23	b_cap<4>	BD16TOD	B
AL24	b_cap<2>	BD16TOD	B
AL25	b_monitor<6>	BD4CS	B
AL26	vssx	—	P
AL27	b_c0ao_l<4>	BD16TOD	B
AL28	b_c0ao_l<6>	BD16TOD	B
AL29	vss	—	P
AL30	vss	—	P
AL31	vdd	—	P
B1	vss	—	P
B2	vdd	—	P
B3	vss	—	P
B6	b_m2a<11>	BD8T	B
B7	b_monitor<5>	BD4CS	B
B8	b_mcas_l<0>	BD8T	B
B9	b_m0cs_l<1>	BD8T	B
B10	b_m0a<3>	BD8T	B
B11	b_m0a<5>	BD8T	B
B12	b_m0a<9>	BD8T	B

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
B13	b_m0a<12>	BD8T	B
B14	b_spare<0>	BD8CS	B
B15	b_cfrst<0>	BD16TOD	B
B16	b_sromoe_l<1>	BD16TOD	B
B17	vss	—	P
B18	b_monitor<1>	BD4CS	B
B19	i_intim_l	IBUF	I
B20	b_tcs_l	BD4CS	B
B21	b_td<1>	BD4CS	B
B22	b_td<5>	BD4CS	B
B23	b_tia<0>	BD4CS	B
B24	b_twe_l	BD4CS	B
B25	i_trsen_l	IBUF	I
B26	b_monitor<7>	BD4CS	B
B27	vssx	—	P
B29	vss	—	P
B30	vdd	—	P
B31	vss	—	P
C1	vss	—	P
C2	vss	—	P
C3	vdd	—	P
C4	vss	—	P
C6	b_m2a<10>	BD8T	B
C7	b_m2ba<0>	BD8T	B
C8	b_m0dqm<1>	BD8T	B
C9	b_m0cs_l<0>	BD8T	B
C10	b_m0a<1>	BD8T	B
C12	b_m0a<7>	BD8T	B
C13	b_m0a<11>	BD8T	B
C14	b_m0ba<1>	BD8T	B
C15	b_cfrst<1>	BD16TOD	B
C16	b_sysrsta_l	BD6T	B
C17	i_modrst_l	IBUF	I
C18	b_monitor<2>	BD4CS	B

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
C19	b_toe_l	BD4CS	B
C20	b_tas	BD4CS	B
C21	b_td<3>	BD4CS	B
C22	b_td<7>	BD4CS	B
C23	b_tia<1>	BD4CS	B
C24	b_mpdclk	BD4CS	B
C25	i_scanen	IBUF	I
C26	vssx	—	P
C27	vssx	—	P
C28	vss	—	P
C29	vdd	—	P
C30	vss	—	P
C31	vss	—	P
D3	vss	—	P
D4	vdd	—	P
D7	vdd	—	P
D8	b_m2ba<1>	BD8T	B
D9	b_mwe_l<0>	BD8T	B
D10	vdd	—	P
D11	b_m0a<2>	BD8T	B
D13	b_m0a<8>	BD8T	B
D14	vdd	—	P
D15	b_spare<1>	BD8CS	B
D16	b_sysrstb_l	BD6T	B
D17	b_monitor<0>	BD4CS	B
D18	vdd	—	P
D19	b_tis	BD4CS	B
D20	b_td<2>	BD4CS	B
D21	b_td<6>	BD4CS	B
D22	vdd	—	P
D23	b_mpdd	BD4CS	B
D24	o_nandtr	B8	O
D25	vdd	—	P
D26	b_c1ao_l<3>	BD16TOD	B

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
D28	vdd	—	P
D29	vss	—	P
D31	vssx	—	P
E29	b_c1ao_l<4>	BD16TOD	B
E30	b_c1ao_l<5>	BD16TOD	B
E31	vssx	—	P
F1	b_m2a<7>	BD8T	B
F2	b_m2a<8>	BD8T	B
F3	b_m2a<9>	BD8T	B
F28	vssx	—	P
F29	b_c1ao_l<6>	BD16TOD	B
F30	vssx	—	P
F31	b_c1ao_l<7>	BD16TOD	B
G1	b_m2a<3>	BD8T	B
G2	b_m2a<4>	BD8T	B
G3	b_m2a<6>	BD8T	B
G4	vdd	—	P
G9	vssx	—	P
G29	vssx	—	P
G28	vdd	—	P
G30	vssx	—	P
G31	b_c1clko_l	BD16TOD	B
H1	b_m2cs_l<1>	BD8T	B
H2	b_m2a<0>	BD8T	B
H3	b_m2a<2>	BD8T	B
H4	b_m2a<5>	BD8T	B
H28	b_c1ao_l<8>	BD16TOD	B
H29	vssx	—	P
H30	vssx	—	P
H31	b_c1ao_l<10>	BD16TOD	B
J1	b_mcke_l<2>	BD8T	B
J3	b_m2cs_l<0>	BD8T	B
J4	b_m2a<1>	BD8T	B
J28	b_c1ao_l<9>	BD16TOD	B

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
J29	**vssx**	—	P
J30	**b_c1ao_l<11>**	BD16TOD	B
J31	**vssx**	—	P
K1	**b_m2dqm<1>**	BD8T	B
K2	**b_mwe_l<2>**	BD8T	B
K3	**b_mras_l<2>**	BD8T	B
K4	**vdd**	—	P
K28	**vdd**	—	P
K29	**b_c1ao_l<12>**	BD16TOD	B
K30	**b_c1ao_l<13>**	BD16TOD	B
K31	**vssx**	—	P
L1	**b_pada<2>**	BD6T	B
L2	**b_pada<1>**	BD6T	B
L3	**b_m2dqm<0>**	BD8T	B
L4	**b_mcas_l<2>**	BD8T	B
L28	**vssx**	—	P
L29	**b_c1ao_l<14>**	BD16TOD	B
L30	**b_c1div_l**	BD16TOD	B
L31	**vssx**	—	P
M1	**b_padb<1>**	BD6T	B
M2	**b_padb<0>**	BD6T	B
M3	**b_pada<3>**	BD6T	B
M4	**b_pada<0>**	BD6T	B
M28	**vssx**	—	P
M29	**b_c1fv_l**	BD16TOD	B
M30	**b_c0ai_l<5>**	BD16TOD	B
M31	**b_c0ai_l<14>**	BD16TOD	B
N1	**b_padb<4>**	BD6T	B
N2	**b_padb<3>**	BD6T	B
N3	**b_padb<2>**	BD6T	B
N4	**b_pada<4>**	BD6T	B
N28	**i_vref<0>**	DDRV	I
N29	**b_c0ai_l<13>**	BD16TOD	B
N30	**b_c0ai_l<12>**	BD16TOD	B

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
N31	b_c0ai_l<11>	BD16TOD	B
P1	vss	—	P
P2	b_cpma<1>	BD6T	B
P3	b_cpma<0>	BD6T	B
P4	vdd	—	P
P28	vdd	—	P
P29	b_c0ai_l<10>	BD16TOD	B
P30	b_c0ai_l<9>	BD16TOD	B
P31	vss	—	P
R1	vss	—	P
R2	vss	—	P
R3	b_cpma<3>	BD6T	B
R4	b_cpma<2>	BD6T	B
R28	b_c0clki_l	BD16TOD	B
R29	b_c0ai_l<8>	BD16TOD	B
R30	b_c0ai_l<7>	BD16TOD	B
R31	b_c0ai_l<6>	BD16TOD	B
T1	i_sysclk	PECLINDIFFA	I
T2	i_sysclk_l	PECLINDIFFA	I
T3	b_cpma<4>	BD6T	B
T4	b_cpma<5>	BD6T	B
T28	b_c0ai_l<4>	BD16TOD	B
T29	i_fwdclk	PECLINDIFFA	I
T30	i_fwdclk_l	PECLINDIFFA	I
T31	b_c0ai_l<3>	BD16TOD	B
U1	b_cpma<6>	BD6T	B
U2	b_cpma<7>	BD6T	B
U3	b_cpmb<0>	BD6T	B
U4	b_cpmb<2>	BD6T	B
U28	b_c1ai_l<2>	BD16TOD	B
U29	b_c0ai_l<2>	BD16TOD	B
U30	vss	—	P
U31	vss	—	P
V1	vss	—	P

Table 3–5 Cchip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
V2	**b_cpmb<1>**	BD6T	B
V3	**b_cpmb<3>**	BD6T	B
V4	**vdd**	—	P
V28	**vdd**	—	P
V29	**b_c1ai_l<4>**	BD16TOD	B
V30	**b_c1ai_l<3>**	BD16TOD	B
V31	**vss**	—	P
W1	**b_cpmb<4>**	BD6T	B
W2	**b_cpmb<5>**	BD6T	B
W3	**b_cpmb<6>**	BD6T	B
W4	**b_cpmb<7>**	BD6T	B
W28	**b_c1ai_l<10>**	BD16TOD	B
W29	**b_c1ai_l<8>**	BD16TOD	B
W30	**b_c1ai_l<7>**	BD16TOD	B
W31	**b_c1ai_l<6>**	BD16TOD	B
Y3	**b_m1ba<1>**	BD8T	B
Y4	**b_m1a<11>**	BD8T	B
Y28	**b_c1ai_l<14>**	BD16TOD	B
Y29	**b_c1ai_l<11>**	BD16TOD	B
Y30	**b_c1ai_l<9>**	BD16TOD	B
Y31	**b_c1clki_l**	BD16TOD	B

Table 3–6 shows the Cchip pins in alphanumeric order by pin number.

Table 3–6 C4chip Pins — Alphanumeric by Pin Number

Pin Number	Signal Name	Driver	Type
A04	Vdd		
A05	b_c1ao_l<9>	BODTY	
A06	Vss		
A07	Vss		
A08	Vss		
A09	b_c1ao_l<2>	BODTY	
A10	Vss		
A11	b_c3ao_l<12>	BODTY	
A12	Vss		
A13	Vss		
A14	Vss		
A15	Vss		
A16	Vss		
A17	Vss		
A18	b_sromoe_l<2>	BODTY	
A19	i_modrst_l	BDZ50C	
A20	b_sysrstc_l	BDZ20C	
A21	b_monitor<1>	BDZ50C	
A22	b_monitor<4>	BDZ50C	
A23	b_tioe_l	BDZ50C	
A24	b_td<1>	BDZ50C	
A25	b_td<5>	BDZ50C	
A26	b_tia<2>	BDZ50C	
A27	b_twe_l	BDZ50C	
A28	b_m3dqm<1>	BDZ20C	
A29	b_m3cs_l<1>	BDZ20C	
A30	Vss		
AA01	b_c2ai_l<6>	BODTY	
AA02	b_c2ai_l<9>	BODTY	
AA04	b_c2ai_l<3>	BODTY	
AA05	b_c2ai_l<8>	BODTY	
AA06	b_c2div_l	BODTY	
AA28	b_m2ba<0>	BDZ20C	

Table 3–6 C4chip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
AA29	b_m2a<6>	BDZ20C	
AA30	b_m2a<10>	BDZ20C	
AA32	Vss		
AA33	b_m2a<8>	BDZ20C	
AB01	Vss		
AB03	b_c2fv_l	BODTY	
AB04	b_c2ai_l<4>	BODTY	
AB30	b_m2a<9>	BDZ20C	
AB31	b_m2ba<1>	BDZ20C	
AB33	b_m2a<11>	BDZ20C	
AC01	Vss		
AC02	b_c2ai_l<2>	BODTY	
AC04	Vss		
AC05	Vss		
AC06	b_c2ao_l<12>	BODTY	
AC28	b_mcas_l<0>	BDZ20C	
AC29	b_mwe_l<0>	BDZ20C	
AC30	b_m2a<12>	BDZ20C	
AC32	Vss		
AC33	b_mcke<0>	BDZ20C	
AD01	Vss		
AD02	Vss		
AD03	b_c2ao_l<10>	BODTY	
AD04	b_c2ao_l<14>	BODTY	
AD05	b_c2ao_l<13>	BODTY	
AD06	Vss		
AD28	b_m0cs_l<3>	BDZ20C	
AD29	b_m0dqm<0>	BDZ20C	
AD30	Vss		
AD31	b_m0cs_l<1>	BDZ20C	
AD32	b_m2ba<2>	BDZ20C	
AD33	b_m0cs_l<0>	BDZ20C	
AE01	Vss		
AE02	Vss		

Table 3–6 C4chip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
AE03	b_c2ao_l<8>	BODTY	
AE04	b_c2ao_l<11>	BODTY	
AE05	Vss		
AE06	Vss		
AE28	b_m0a<3>	BDZ20C	
AE29	Vss		
AE30	b_mras_l<0>	BDZ20C	
AE31	b_m0a<2>	BDZ20C	
AE32	b_m0dqm<1>	BDZ20C	
AE33	b_m0a<0>	BDZ20C	
AF01	b_c2ao_l<6>	BODTY	
AF03	b_c2ao_l<9>	BODTY	
AF05	b_c2clko_l	BODTY	
AF06	b_c2ao_l<5>	BODTY	
AF28	b_m0a<8>	BDZ20C	
AF29	b_m0a<1>	BDZ20C	
AF31	b_m0cs_l<2>	BDZ20C	
AF33	b_m0a<5>	BDZ20C	
AG01	Vss		
AG04	Vss		
AG05	b_c2ao_l<7>	BODTY	
AG06	b_c2ao_l<2>	BODTY	
AG28	b_m0a<12>	BDZ20C	
AG29	b_m0a<4>	BDZ20C	
AG30	Vss		
AG33	b_m0a<6>	BDZ20C	
AH01	Vss		
AH02	b_c2ao_l<3>	BODTY	
AH03	Vss		
AH05	Vss		
AH07	b_c0fv_l	BODTY	
AH08	Vss		
AH09	b_c0ao_l<11>	BODTY	
AH10	Vss		

Table 3–6 C4chip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
AH11	b_c0ao_l<7>	BODTY	
AH13	b_c0ao_l<2>	BODTY	
AH15	b_cap<18>	BODTY	
AH17	b_cap<5>	BODTY	
AH19	Vss		
AH21	b_capsel<1>	BDZ30C	
AH23	b_padb<1>	BDZ20C	
AH24	b_cpma<1>	BDZ20C	
AH25	b_cpma<5>	BDZ20C	
AH26	b_cpmb<2>	BDZ20C	
AH27	b_cpmb<6>	BDZ20C	
AH29	b_m0a<10>	BDZ20C	
AH31	b_m0a<7>	BDZ20C	
AH32	b_m0a<11>	BDZ20C	
AH33	b_m0a<9>	BDZ20C	
AJ01	Vss		
AJ02	b_c2ao_l<4>	BODTY	
AJ03	Vss		
AJ06	Vss		
AJ07	b_c0ao_l<12>	BODTY	
AJ08	b_c0ao_l<9>	BODTY	
AJ09	b_c0ao_l<8>	BODTY	
AJ10	b_c0ao_l<5>	BODTY	
AJ11	Vss		
AJ13	b_cap<22>	BODTY	
AJ15	Vss		
AJ17	b_cap<8>	BODTY	
AJ19	Vdd		
AJ21	i_creq_l<1>	BDZ50C	
AJ23	b_padb<0>	BDZ20C	
AJ24	b_pada<3>	BDZ20C	
AJ25	b_padb<2>	BDZ20C	
AJ26	Vss		
AJ27	b_cpma<6>	BDZ20C	

Table 3–6 C4chip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
AJ28	b_cpmb<4>	BDZ20C	
AJ31	b_m0ba<0>	BDZ20C	
AJ32	Vss		
AJ33	b_m0ba<1>	BDZ20C	
AK01	Vdd		
AK07	Vss		
AK09	Vss		
AK10	Vss		
AK11	Vss		
AK12	b_cap<23>	BODTY	
AK13	i_vref<4>		
AK14	Vss		
AK15	b_cap<14>	BODTY	
AK16	Vss		
AK17	Vss		
AK18	Vss		
AK19	b_cap<2>	BODTY	
AK20	i_pack<0>	BDZ50C	
AK21	b_capgd<0>	BDZ50C	
AK22	b_cacta_l	BDZ50C	
AK23	b_capsel<0>	BDZ30C	
AK24	b_pada<1>	BDZ20C	
AK25	Vss		
AK27	b_cpma<3>	BDZ20C	
AK33	Vss		
AL02	null		
AL05	b_c0div_l	BODTY	
AL06	Vss		
AL08	Vss		
AL09	Vss		
AL10	b_c0clko_l	BODTY	
AL12	Vss		
AL14	b_cap<20>	BODTY	
AL15	b_cap<16>	BODTY	

Table 3–6 C4chip Pins — Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
AL16	b_cap<12>	BODTY	
AL18	b_cap<4>	BODTY	
AL19	b_cap<0>	BODTY	
AL20	i_pack<1>	BDZ50C	
AL22	b_pada<0>	BDZ20C	
AL24	b_padb<4>	BDZ20C	
AL25	b_cpma<4>	BDZ20C	
AL26	b_cpma<0>	BDZ20C	
AL28	Vss		
AL29	Vss		
AL32	null		
AM03	Vdd		
AM05	b_c0ao_l<14>	BODTY	
AM06	Vss		
AM09	b_c0ao_l<6>	BODTY	
AM10	b_c0ao_l<3>	BODTY	
AM11	Vss		
AM13	b_cap<21>	BODTY	
AM14	b_cap<17>	BODTY	
AM15	b_cap<13>	BODTY	
AM16	b_cap<10>	BODTY	
AM17	b_cap<9>	BODTY	
AM18	b_cap<7>	BODTY	
AM19	b_cap<3>	BODTY	
AM20	Vss		
AM21	i_creq_l<0>	BDZ50C	
AM23	b_cack	BDZ20C	
AM24	Vss		
AM25	b_pada<4>	BDZ20C	
AM28	b_cpmb<5>	BDZ20C	
AM29	b_cpmb<1>	BDZ20C	
AM31	null		
AN04	Vss		
AN05	Vss		

Table 3–6 C4chip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
AN06	i_vref<0>		
AN07	b_c0ao_l<13>	BODTY	
AN08	Vss		
AN09	b_c0ao_l<10>	BODTY	
AN10	Vss		
AN11	b_c0ao_l<4>	BODTY	
AN12	Vss		
AN13	Vss		
AN14	b_cap<19>	BODTY	
AN15	b_cap<15>	BODTY	
AN16	Vss		
AN17	b_cap<11>	BODTY	
AN18	b_cap<6>	BODTY	
AN19	b_cap<1>	BODTY	
AN20	Vss		
AN21	b_cactb_l	BDZ50C	
AN22	b_capgd<1>	BDZ50C	
AN23	b_pada<2>	BDZ20C	
AN24	b_padb<3>	BDZ20C	
AN25	b_cpma<2>	BDZ20C	
AN26	b_cpma<7>	BDZ20C	
AN27	b_cpmb<0>	BDZ20C	
AN28	b_cpmb<3>	BDZ20C	
AN29	b_m0ba<2>	BDZ20C	
AN30	b_cpmb<7>	BDZ20C	
B03	null		
B05	Vss		
B06	b_c1ao_l<8>	BODTY	
B09	i_vref<3>		
B10	Vss		
B11	Vss		
B13	b_c3ao_l<8>	BODTY	
B14	b_c3ao_l<5>	BODTY	
B15	b_c3ao_l<3>	BODTY	

Table 3–6 C4chip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
B16	Vss		
B17	b_cfrst<2>	BODTY	
B18	b_cfrst<0>	BODTY	
B19	i_trsen_l	BDZ50C	
B20	o_nandtr	BDZ50C	
B21	b_sysrsta_l	BDZ20C	
B23	Vss		
B24	b_toe_l	BDZ50C	
B25	b_tis2	BDZ50C	
B28	b_mcas_l<3>	BDZ20C	
B29	Vss		
B31	null		
C02	Vdd		
C05	Vss		
C06	b_c1ao_l<5>	BODTY	
C08	Vss		
C09	Vss		
C10	b_c3fv_l	BODTY	
C12	Vss		
C14	b_c3ao_l<7>	BODTY	
C15	b_c3ao_l<4>	BODTY	
C16	b_c3ao_l<2>	BODTY	
C18	b_sromoe_l<0>	BODTY	
C19	i_scanin	BDZ50C	
C20	b_sysrstb_l	BDZ20C	
C22	b_monitor<7>	BDZ50C	
C24	b_td<2>	BDZ50C	
C25	b_td<7>	BDZ50C	
C26	b_td<3>	BDZ50C	
C28	b_mcke<3>	BDZ20C	
C29	b_m3cs_l<0>	BDZ20C	
C32	null		
D01	Vss		
D07	Vss		

Table 3–6 C4chip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
D09	Vss		
D10	b_c3ao_l<13>	BODTY	
D11	b_c3ao_l<10>	BODTY	
D12	b_c3clko_l	BODTY	
D13	b_c3ao_l<9>	BODTY	
D14	Vss		
D15	Vss		
D16	Vss		
D17	b_cfrst<1>	BODTY	
D18	b_sromoe_l<1>	BODTY	
D19	i_scanen	BDZ50C	
D20	i_intim_l	BDZ50C	
D21	b_monitor<3>	BDZ50C	
D22	b_monitor<2>	BDZ50C	
D23	b_monitor<5>	BDZ50C	
D24	Vss		
D25	b_td<0>	BDZ50C	
D27	Vss		
D33	b_m3cs_l<3>	BDZ20C	
E01	b_c1ao_l<10>	BODTY	
E02	b_c1ao_l<14>	BODTY	
E03	b_c1ao_l<11>	BODTY	
E06	b_c1ao_l<7>	BODTY	
E07	b_c1clko_l	BODTY	
E08	b_c1ao_l<3>	BODTY	
E09	b_c3div_l	BODTY	
E10	Vss		
E11	Vss		
E13	Vss		
E15	Vss		
E17	b_cfrst<3>	BODTY	
E19	b_mpdclk	BDZ50C	
E21	b_monitor<0>	BDZ50C	
E23	b_tis	BDZ50C	

Table 3–6 C4chip Pins — Alphanumeric by Pin Number   (Continued)

Pin Number	Signal Name	Driver	Type
E24	b_tcs_l	BDZ50C	
E25	Vss		
E26	b_td<6>	BDZ50C	
E27	b_tia<1>	BDZ50C	
E28	b_mwe_l<3>	BDZ20C	
E31	Vss		
E32	b_m3a<5>	BDZ20C	
E33	b_m3cs_l<2>	BDZ20C	
F01	Vss		
F02	b_c1ao_l<12>	BODTY	
F03	Vss		
F05	Vss		
F07	Vss		
F08	b_c1ao_l<6>	BODTY	
F09	b_c1ao_l<4>	BODTY	
F10	Vss		
F11	b_c3ao_l<14>	BODTY	
F13	b_c3ao_l<11>	BODTY	
F15	b_c3ao_l<6>	BODTY	
F17	b_sromoe_l<3>	BODTY	
F19	b_mpdd	BDZ50C	
F21	b_monitor<6>	BDZ50C	
F23	b_tas	BDZ50C	
F24	b_td<4>	BDZ50C	
F25	b_tia<0>	BDZ50C	
F26	b_m3dqm<0>	BDZ20C	
F27	b_mras_l<3>	BDZ20C	
F29	b_m3a<2>	BDZ20C	
F31	Vss		
F32	b_m3a<1>	BDZ20C	
F33	b_m3a<3>	BDZ20C	
G01	Vss		
G04	b_c3ai_l<2>	BODTY	
G05	b_c1div_l	BODTY	

Table 3–6 C4chip Pins — Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
G06	Vss		
G28	b_m3a<0>	BDZ20C	
G29	b_m3a<8>	BDZ20C	
G30	b_m3a<11>	BDZ20C	
G33	b_m3a<6>	BDZ20C	
H01	b_c1fv_l	BODTY	
H03	b_c3ai_l<7>	BODTY	
H05	b_c3ai_l<4>	BODTY	
H06	b_c1ao_l<13>	BODTY	
H28	b_m3a<4>	BDZ20C	
H29	Vss		
H31	b_m3ba<1>	BDZ20C	
H33	b_m3a<7>	BDZ20C	
J01	b_c3ai_l<3>	BODTY	
J02	Vss		
J03	Vss		
J04	b_c3ai_l<11>	BODTY	
J05	b_c3ai_l<9>	BODTY	
J06	Vss		
J28	b_m3a<9>	BDZ20C	
J29	b_m1dqm<0>	BDZ20C	
J30	Vss		
J31	b_m3a<10>	BDZ20C	
J32	b_mcas_l<1>	BDZ20C	
J33	b_m3a<12>	BDZ20C	
K01	b_c3clki_l	BODTY	
K02	b_c1ai_l<2>	BODTY	
K03	b_c3ai_l<8>	BODTY	
K04	b_c3ai_l<14>	BODTY	
K05	i_vref<1>		
K06	b_c3ai_l<6>	BODTY	
K28	b_m3ba<0>	BDZ20C	
K29	b_mras_l<1>	BDZ20C	
K30	b_m1cs_l<1>	BDZ20C	

Table 3–6 C4chip Pins — Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
K31	b_m3ba<2>	BDZ20C	
K32	Vss		
K33	b_mcke<1>	BDZ20C	
L01	b_c3ai_l<13>	BODTY	
L02	b_c1ai_l<8>	BODTY	
L04	b_c1ai_l<4>	BODTY	
L05	b_c3ai_l<12>	BODTY	
L06	b_c3ai_l<10>	BODTY	
L28	b_m1dqm<1>	BDZ20C	
L29	b_mwe_l<1>	BDZ20C	
L30	b_m1a<0>	BDZ20C	
L32	b_m1a<3>	BDZ20C	
L33	b_m1cs_l<0>	BDZ20C	
M01	b_c1ai_l<6>	BODTY	
M03	b_c3ai_l<5>	BODTY	
M04	b_c1clki_l	BODTY	
M30	Vss		
M31	b_m1cs_l<2>	BDZ20C	
M33	b_m1a<1>	BDZ20C	
N01	b_c1ai_l<9>	BODTY	
N02	b_c1ai_l<14>	BODTY	
N04	b_c1ai_l<7>	BODTY	
N05	b_c1ai_l<11>	BODTY	

3.2 Dchip Pins and Signals

This section provides information about Dchip pins, pin types, pin numbers, and signal definitions.

3.2.1 Dchip Pin List by Function

Table 3–7 lists the pin categories, signal names, types, and signal functions for the Dchip.

Table 3–7 Dchip Pin List by Function

Signal Name	Quantity	Type	Function
CPU Interface			
b_cc_l<3:0>	4	B	CPU check bits
b_cclki_l<3:0>	4	I	CPU interface clock in
b_cclko_l<3:0>	4	O	CPU interface clock out
b_cd_l<31:0>	32	B	CPU data
SUBTOTAL	44	—	—
PADbus Interface			
b_p0c	1	B	PADbus 0 check
b_p0d<7:0>	8	B	PADbus 0 address and data
b_p1c	1	B	PADbus 1 check
b_p1d<7:0>	8	B	PADbus 1 address and data
SUBTOTAL	18	—	—
Cchip Interface			
i_cpm<7:0>	8	I	CPM command
i_pad<4:0>	5	I	PAD command
SUBTOTAL	13	—	—
Memory Interface			
b_m0c<3:0>	4	B	Memory bus 0 check bits
b_m0d<31:0>	32	B	Memory bus 0 data
b_m1c<3:0>	4	B	Memory bus 1 check bits
b_m1d<31:0>	32	B	Memory bus 1 data
SUBTOTAL	72	—	—
Miscellaneous Signals			
i_fwdclk, **i_fwdclk_l**	2	I	Forward clock in
i_scanrclk	1	I	Scan receive clock
i_scanrclken	1	I	Scan receive clock enable
i_sysclk, **i_sysclk_l**	2	I	System clock in
i_sysrst_l	1	I	Reset

Table 3–7 Dchip Pin List by Function (Continued)

Signal Name	Quantity	Type	Function
i_vref<3:0>	4	I	2-V I/O reference
b_spare <7:2>	6	B	Spare
SUBTOTAL	17	—	—
CSALT			
o_nandtr	1	O	NAND tree
i_scanen	1	I	Scan enable
i_scanin	1	I	Scan in
i_trsen_l	1	I	Tristate outputs
SUBTOTAL	4	—	—
SIGNAL SUBTOTAL	168	—	—
Power Pins			
Vdd	36	P	**Vdd** ring (9000)
Vss	36	P	**Vss** plane (8000)
Vssx	43	P	
SUBTOTAL	115	—	—
SIGNAL/PIN TOTAL	283	—	21 pins not connected

3.2.2 Dchip Sorted Pin List

Table 3–8 lists the Dchip pins in alphanumeric order by signal name.

Table 3–8 Dchip Pins – Alphanumeric by Signal Name

Signal Name	Pin	Driver	Type
b_cc_l<0>	W21	BD16TOD	B
b_cc_l<1>	M21	BD16TOD	B
b_cc_l<2>	F21	BD16TOD	B
b_cc_l<3>	B15	BD16TOD	B
b_cclki_l<0>	AA18	BD16TOD	I
b_cclki_l<1>	R21	—	I
b_cclki_l<2>	H22	BD16TOD	I
b_cclki_l<3>	E21	BD16TOD	I
b_cclko_l<0>	AB19	BD16TOD	O
b_cclko_l<1>	R22	BD16TOD	O
b_cclko_l<2>	G23	BD16TOD	O
b_cclko_l<3>	B18	BD16TOD	O
b_cd_l<0>	AB15	BD16TOD	B

Table 3–8 Dchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_cd_l<1>	AB16	BD16TOD	B
b_cd_l<2>	AA16	BD16TOD	B
b_cd_l<3>	Y16	BD16TOD	B
b_cd_l<4>	AB18	BD16TOD	B
b_cd_l<5>	AA19	BD16TOD	B
b_cd_l<6>	AC21	BD16TOD	B
b_cd_l<7>	AA23	BD16TOD	B
b_cd_l<8>	W22	BD16TOD	B
b_cd_l<9>	U20	BD16TOD	B
b_cd_l<10>	V22	BD16TOD	B
b_cd_l<11>	T20	BD16TOD	B
b_cd_l<12>	U23	BD16TOD	B
b_cd_l<13>	R23	BD16TOD	B
b_cd_l<14>	P22	BD16TOD	B
b_cd_l<15>	N20	BD16TOD	B
b_cd_l<16>	L23	BD16TOD	B
b_cd_l<17>	L21	BD16TOD	B
b_cd_l<18>	K23	BD16TOD	B
b_cd_l<19>	K21	BD16TOD	B
b_cd_l<20>	K20	BD16TOD	B
b_cd_l<21>	G22	BD16TOD	B
b_cd_l<22>	G21	BD16TOD	B
b_cd_l<23>	E23	BD16TOD	B
b_cd_l<24>	D22	BD16TOD	B
b_cd_l<25>	A21	BD16TOD	B
b_cd_l<26>	C19	BD16TOD	B
b_cd_l<27>	B19	BD16TOD	B
b_cd_l<28>	D17	BD16TOD	B
b_cd_l<29>	D16	BD16TOD	B
b_cd_l<30>	A17	BD16TOD	B
b_cd_l<31>	B16	BD16TOD	B
b_m0c<0>	B6	BD8CS	B
b_m0c<1>	D5	BD8CS	B
b_m0c<2>	H3	BD8CS	B

Table 3–8 Dchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_m0c<3>	L4	BD8CS	B
b_m0d<0>	B9	BD8CS	B
b_m0d<1>	C9	BD8CS	B
b_m0d<2>	B8	BD8CS	B
b_m0d<3>	A7	BD8CS	B
b_m0d<4>	C8	BD8CS	B
b_m0d<5>	B7	BD8CS	B
b_m0d<6>	D8	BD8CS	B
b_m0d<7>	C7	BD8CS	B
b_m0d<8>	A5	BD8CS	B
b_m0d<9>	D7	BD8CS	B
b_m0d<10>	C6	BD8CS	B
b_m0d<11>	B5	BD8CS	B
b_m0d<12>	A4	BD8CS	B
b_m0d<13>	C5	BD8CS	B
b_m0d<14>	B4	BD8CS	B
b_m0d<15>	A3	BD8CS	B
b_m0d<16>	E2	BD8CS	B
b_m0d<17>	F3	BD8CS	B
b_m0d<18>	G4	BD8CS	B
b_m0d<19>	F2	BD8CS	B
b_m0d<20>	G3	BD8CS	B
b_m0d<21>	H4	BD8CS	B
b_m0d<22>	G2	BD8CS	B
b_m0d<23>	G1	BD8CS	B
b_m0d<24>	H2	BD8CS	B
b_m0d<25>	J3	BD8CS	B
b_m0d<26>	J2	BD8CS	B
b_m0d<27>	K4	BD8CS	B
b_m0d<28>	J1	BD8CS	B
b_m0d<29>	K3	BD8CS	B
b_m0d<30>	K2	BD8CS	B
b_m0d<31>	K1	BD8CS	B
b_m1c<0>	P3	BD8CS	B

Table 3–8 Dchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_m1c<1>	T4	BD8CS	B
b_m1c<2>	Y2	BD8CS	B
b_m1c<3>	AA7	BD8CS	B
b_m1d<0>	M3	BD8CS	B
b_m1d<1>	M2	BD8CS	B
b_m1d<2>	N1	BD8CS	B
b_m1d<3>	N2	BD8CS	B
b_m1d<4>	N3	BD8CS	B
b_m1d<5>	N4	BD8CS	B
b_m1d<6>	P1	BD8CS	B
b_m1d<7>	P2	BD8CS	B
b_m1d<8>	R1	BD8CS	B
b_m1d<9>	P4	BD8CS	B
b_m1d<10>	R2	BD8CS	B
b_m1d<11>	R3	BD8CS	B
b_m1d<12>	T2	BD8CS	B
b_m1d<13>	U1	BD8CS	B
b_m1d<14>	T3	BD8CS	B
b_m1d<15>	U2	BD8CS	B
b_m1d<16>	U3	BD8CS	B
b_m1d<17>	V2	BD8CS	B
b_m1d<18>	W1	BD8CS	B
b_m1d<19>	U4	BD8CS	B
b_m1d<20>	V3	BD8CS	B
b_m1d<21>	W2	BD8CS	B
b_m1d<22>	Y1	BD8CS	B
b_m1d<23>	W3	BD8CS	B
b_m1d<24>	AB4	BD8CS	B
b_m1d<25>	AA5	BD8CS	B
b_m1d<26>	AC4	BD8CS	B
b_m1d<27>	AB5	BD8CS	B
b_m1d<28>	AA6	BD8CS	B
b_m1d<29>	Y7	BD8CS	B
b_m1d<30>	AC5	BD8CS	B

Table 3–8 Dchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
b_m1d<31>	AB6	BD8CS	B
b_p0c	C12	BD4TS	B
b_p0d<0>	A15	BD4TS	B
b_p0d<1>	C14	BD4TS	B
b_p0d<2>	B14	BD4TS	B
b_p0d<3>	A14	BD4TS	B
b_p0d<4>	D13	BD4TS	B
b_p0d<5>	C13	BD4TS	B
b_p0d<6>	B13	BD4TS	B
b_p0d<7>	A13	BD4TS	B
b_p1c	D10	BD4TS	B
b_p1d<0>	B12	BD4TS	B
b_p1d<1>	A11	BD4TS	B
b_p1d<2>	B11	BD4TS	B
b_p1d<3>	C11	BD4TS	B
b_p1d<4>	D11	BD4TS	B
b_p1d<5>	A10	BD4TS	B
b_p1d<6>	C10	BD4TS	B
b_p1d<7>	A9	BD4TS	B
b_spare<2>	AB9	BD4TS	B
b_spare<3>	Y10	BD4TS	B
b_spare<4>	AC9	BD4TS	B
b_spare<5>	AA10	BD4TS	B
b_spare<6>	AB10	BD4TS	B
b_spare<7>	AC10	BD4TS	B
i_cpm<0>	Y11	IBUF	I
i_cpm<1>	AA11	IBUF	I
i_cpm<2>	AB11	IBUF	I
i_cpm<3>	AC11	IBUF	I
i_cpm<4>	AA12	IBUF	I
i_cpm<5>	AB12	IBUF	I
i_cpm<6>	AC13	IBUF	I
i_cpm<7>	AB13	IBUF	I
i_fwdclk	N22	PECLINDIFFA	I

Table 3–8 Dchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
i_fwdclk_l	N23	PECLINDIFFA	I
i_pad<0>	AA13	IBUF	I
i_pad<1>	Y13	IBUF	I
i_pad<2>	AC14	IBUF	I
i_pad<3>	AB14	IBUF	I
i_pad<4>	AA14	IBUF	I
i_scanen	AC7	IBUF	I
i_scanin	AB7	IBUF	I
i_scanrclk	AB	IBUF	I
i_scanrclken	AA9	IBUF	I
i_sysclk	L1	PECLINDIFFA	I
i_sysclk_l	L2	PECLINDIFFA	I
i_sysrst_l	AC15	IBUF	I
i_trsen_l	AA8	IBUF	I
i_vref<0>	Y17	DDRV	I
i_vref<1>	T22	DDRV	I
i_vref<2>	J21	DDRV	I
i_vref<3>	D23	DDRV	I
o_nandtr	Y8	B8	O
Vdd	A1	—	P
Vdd	A23	—	P
Vdd	AA3	—	P
Vdd	AA21	—	P
Vdd	AB2	—	P
Vdd	AB22	—	P
Vdd	AC1	—	P
Vdd	AC23	—	P
Vdd	B2	—	P
Vdd	B22	—	P
Vdd	C3	—	P
Vdd	C21	—	P
Vdd	D4	—	P
Vdd	D6	—	P
Vdd	D9	—	P

Table 3–8 Dchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
Vdd	D12	—	P
Vdd	D15	—	P
Vdd	D18	—	P
Vdd	D20	—	P
Vdd	F4	—	P
Vdd	F20	—	P
Vdd	J4	—	P
Vdd	J20	—	P
Vdd	M4	—	P
Vdd	M20	—	P
Vdd	R4	—	P
Vdd	R20	—	P
Vdd	V4	—	P
Vdd	V20	—	P
Vdd	Y4	—	P
Vdd	Y6	—	P
Vdd	Y9	—	P
Vdd	Y12	—	P
Vdd	Y15	—	P
Vdd	Y18	—	P
Vdd	Y20	—	P
Vss	A2	—	P
Vss	A6	—	P
Vss	A8	—	P
Vss	A12	—	P
Vss	A16	—	P
Vss	A18	—	P
Vss	A22	—	P
Vss	AA2	—	P
Vss	AA22	—	P
Vss	AB1	—	P
Vss	AB3	—	P
Vss	AB21	—	P
Vss	AB23	—	P

Table 3–8 Dchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
Vss	AC2	—	P
Vss	AC6	—	P
Vss	AC8	—	P
Vss	AC12	—	P
Vss	AC16	—	P
Vss	AC18	—	P
Vss	AC22	—	P
Vss	B1	—	P
Vss	B3	—	P
Vss	B21	—	P
Vss	B23	—	P
Vss	C2	—	P
Vss	C22	—	P
Vss	F1	—	P
Vss	F23	—	P
Vss	H1	—	P
Vss	H23	—	P
Vss	M1	—	P
Vss	M23	—	P
Vss	T1	—	P
Vss	T23	—	P
Vss	V1	—	P
Vss	V23	—	P
Vssx	A19	—	P
Vssx	A20	—	P
Vssx	AA15	—	P
Vssx	AA17	—	P
Vssx	AB17	—	P
Vssx	AB20	—	P
Vssx	AC17	—	P
Vssx	AC19	—	P
Vssx	AC20	—	P
Vssx	B17	—	P
Vssx	B20	—	P

Table 3–8 Dchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin	Driver	Type
Vssx	C15	—	P
Vssx	C16	—	P
Vssx	C17	—	P
Vssx	C18	—	P
Vssx	C23	—	P
Vssx	D14	—	P
Vssx	D19	—	P
Vssx	E22	—	P
Vssx	F22	—	P
Vssx	G20	—	P
Vssx	H20	—	P
Vssx	H21	—	P
Vssx	J22	—	P
Vssx	J23	—	P
Vssx	K22	—	P
Vssx	L20	—	P
Vssx	L22	—	P
Vssx	M22	—	P
Vssx	N21	—	P
Vssx	P20	—	P
Vssx	P21	—	P
Vssx	P23	—	P
Vssx	T21	—	P
Vssx	U21	—	P
Vssx	U22	—	P
Vssx	V21	—	P
Vssx	W20	—	P
Vssx	W23	—	P
Vssx	Y14	—	P
Vssx	Y19	—	P
Vssx	Y22	—	P
Vssx	Y23	—	P

Table 3–9 lists the Dchip pins in alphanumeric order by pin number.

Table 3–9 Dchip Pins – Alphanumeric by Pin Number

Pin Number	Signal Name	Driver	Type
A1	**Vdd**	—	P
A2	**Vss**	—	P
A3	**b_m0d<15>**	BD8CS	B
A4	**b_m0d<12>**	BD8CS	B
A5	**b_m0d<8>**	BD8CS	B
A6	**Vss**	—	P
A7	**b_m0d<3>**	BD8CS	B
A8	**Vss**	—	P
A9	**b_p1d<7>**	BD4TS	B
A10	**b_p1d<5>**	BD4TS	B
A11	**b_p1d<1>**	BD4TS	B
A12	**Vss**	—	P
A13	**b_p0d<7>**	BD4TS	B
A14	**b_p0d<3>**	BD4TS	B
A15	**b_p0d<0>**	BD4TS	B
A16	**Vss**	—	P
A17	**b_cd_l<30>**	BD16TOD	B
A18	**Vss**	—	P
A19	**Vssx**	—	P
A20	**Vssx**	—	P
A21	**b_cd_l<25>**	BD16TOD	B
A22	**Vss**	—	P
A23	**Vdd**	—	P
AA2	**Vss**	—	P
AA3	**Vdd**	—	P
AA5	**b_m1d<25>**	BD8CS	B
AA6	**b_m1d<28>**	BD8CS	B
AA7	**b_m1c<3>**	BD8CS	B
AA8	**i_trsen_l**	IBUF	I
AA9	**i_scanrclken**	IBUF	I
AA10	**b_spare<5>**	BD4TS	B

Table 3–9 Dchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
AA11	i_cpm<1>	IBUF	I
AA12	i_cpm<4>	IBUF	I
AA13	i_pad<0>	IBUF	I
AA14	i_pad<4>	IBUF	I
AA15	Vssx	—	P
AA16	b_cd_l<2>	BD16TOD	B
AA17	Vssx	—	P
AA18	b_cclki_l<0>	BD16TOD	I
AA19	b_cd_l<5>	BD16TOD	B
AA21	Vdd	—	P
AA22	Vss	—	P
AA23	b_cd_l<7>	BD16TOD	B
AB	i_scanrclk	IBUF	I
AB1	Vss	—	P
AB2	Vdd	—	P
AB3	Vss	—	P
AB4	b_m1d<24>	BD8CS	B
AB5	b_m1d<27>	BD8CS	B
AB6	b_m1d<31>	BD8CS	B
AB7	i_scanin	IBUF	I
AB9	b_spare<2>	BD4TS	B
AB10	b_spare<6>	BD4TS	B
AB11	i_cpm<2>	IBUF	I
AB12	i_cpm<5>	IBUF	I
AB13	i_cpm<7>	IBUF	I
AB14	i_pad<3>	IBUF	I
AB15	b_cd_l<0>	BD16TOD	B
AB16	b_cd_l<1>	BD16TOD	B
AB17	Vssx	—	P
AB18	b_cd_l<4>	BD16TOD	B
AB19	b_cclko_l<0>	BD16TOD	O
AB20	Vssx	—	P
AB21	Vss	—	P
AB22	Vdd	—	P

Table 3–9 Dchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
AB23	Vss	—	P
AC1	Vdd	—	P
AC2	Vss	—	P
AC4	b_m1d<26>	BD8CS	B
AC5	b_m1d<30>	BD8CS	B
AC6	Vss	—	P
AC7	i_scanen	IBUF	I
AC8	Vss	—	P
AC9	b_spare<4>	BD4TS	B
AC10	b_spare<7>	BD4TS	B
AC11	i_cpm<3>	IBUF	I
AC12	Vss	—	P
AC13	i_cpm<6>	IBUF	I
AC14	i_pad<2>	IBUF	I
AC15	i_sysrst_l	IBUF	I
AC16	Vss	—	P
AC17	Vssx	—	P
AC18	Vss	—	P
AC19	Vssx	—	P
AC20	Vssx	—	P
AC21	b_cd_l<6>	BD16TOD	B
AC22	Vss	—	P
AC23	Vdd	—	P
B1	Vss	—	P
B2	Vdd	—	P
B3	Vss	—	P
B4	b_m0d<14>	BD8CS	B
B5	b_m0d<11>	BD8CS	B
B6	b_m0c<0>	BD8CS	B
B7	b_m0d<5>	BD8CS	B
B8	b_m0d<2>	BD8CS	B
B9	b_m0d<0>	BD8CS	B
B11	b_p1d<2>	BD4TS	B
B12	b_p1d<0>	BD4TS	B

Table 3–9 Dchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
B13	b_p0d<6>	BD4TS	B
B14	b_p0d<2>	BD4TS	B
B15	b_cc_l<3>	BD16TOD	B
B16	b_cd_l<31>	BD16TOD	B
B17	Vssx	—	P
B18	b_cclko_l<3>	BD16TOD	O
B19	b_cd_l<27>	BD16TOD	B
B20	Vssx	—	P
B21	Vss	—	P
B22	Vdd	—	P
B23	Vss	—	P
C2	Vss	—	P
C3	Vdd	—	P
C5	b_m0d<13>	BD8CS	B
C6	b_m0d<10>	BD8CS	B
C7	b_m0d<7>	BD8CS	B
C8	b_m0d<4>	BD8CS	B
C9	b_m0d<1>	BD8CS	B
C10	b_p1d<6>	BD4TS	B
C11	b_p1d<3>	BD4TS	B
C12	b_p0c	BD4TS	B
C13	b_p0d<5>	BD4TS	B
C14	b_p0d<1>	BD4TS	B
C15	Vssx	—	P
C16	Vssx	—	P
C17	Vssx	—	P
C18	Vssx	—	P
C19	b_cd_l<26>	BD16TOD	B
C21	Vdd	—	P
C22	Vss	—	P
C23	Vssx	—	P
D4	Vdd	—	P
D5	b_m0c<1>	BD8CS	B
D6	Vdd	—	P

Table 3–9 Dchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
D7	b_m0d<9>	BD8CS	B
D8	b_m0d<6>	BD8CS	B
D9	Vdd	—	P
D10	b_p1c	BD4TS	B
D11	b_p1d<4>	BD4TS	B
D12	Vdd	—	P
D13	b_p0d<4>	BD4TS	B
D14	Vssx	—	P
D15	Vdd	—	P
D16	b_cd_l<29>	BD16TOD	B
D17	b_cd_l<28>	BD16TOD	B
D18	Vdd	—	P
D19	Vssx	—	P
D20	Vdd	—	P
D22	b_cd_l<24>	BD16TOD	B
D23	i_vref<3>	DDRV	I
E2	b_m0d<16>	BD8CS	B
E21	b_cclki_l<3>	BD16TOD	I
E22	Vssx	—	P
E23	b_cd_l<23>	BD16TOD	B
F1	Vss	—	P
F2	b_m0d<19>	BD8CS	B
F3	b_m0d<17>	BD8CS	B
F4	Vdd	—	P
F20	Vdd	—	P
F21	b_cc_l<2>	BD16TOD	B
F22	Vssx	—	P
F23	Vss	—	P
G1	b_m0d<23>	BD8CS	B
G2	b_m0d<22>	BD8CS	B
G3	b_m0d<20>	BD8CS	B
G4	b_m0d<18>	BD8CS	B
G20	Vssx	—	P
G21	b_cd_l<22>	BD16TOD	B

Table 3–9 Dchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
G22	b_cd_l<21>	BD16TOD	B
G23	b_cclko_l<2>	BD16TOD	O
H1	Vss	—	P
H2	b_m0d<24>	BD8CS	B
H3	b_m0c<2>	BD8CS	B
H4	b_m0d<21>	BD8CS	B
H20	Vssx	—	P
H21	Vssx	—	P
H22	b_cclki_l<2>	BD16TOD	I
H23	Vss	—	P
J1	b_m0d<28>	BD8CS	B
J2	b_m0d<26>	BD8CS	B
J3	b_m0d<25>	BD8CS	B
J4	Vdd	—	P
J20	Vdd	—	P
J21	i_vref<2>	DDRV	I
J22	Vssx	—	P
J23	Vssx	—	P
K1	b_m0d<31>	BD8CS	B
K2	b_m0d<30>	BD8CS	B
K3	b_m0d<29>	BD8CS	B
K4	b_m0d<27>	BD8CS	B
K20	b_cd_l<20>	BD16TOD	B
K21	b_cd_l<19>	BD16TOD	B
K22	Vssx	—	P
K23	b_cd_l<18>	BD16TOD	B
L1	i_sysclk	PECLINDIFFA	I
L2	i_sysclk_l	PECLINDIFFA	I
L4	b_m0c<3>	BD8CS	B
L20	Vssx	—	P
L21	b_cd_l<17>	BD16TOD	B
L22	Vssx	—	P
L23	b_cd_l<16>	BD16TOD	B
M1	Vss	—	P

Table 3–9 Dchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
M2	**b_m1d<1>**	BD8CS	B
M3	**b_m1d<0>**	BD8CS	B
M4	**Vdd**	—	P
M20	**Vdd**	—	P
M21	**b_cc_l<1>**	BD16TOD	B
M22	**Vssx**	—	P
M23	**Vss**	—	P
N1	**b_m1d<2>**	BD8CS	B
N2	**b_m1d<3>**	BD8CS	B
N3	**b_m1d<4>**	BD8CS	B
N4	**b_m1d<5>**	BD8CS	B
N20	**b_cd_l<15>**	BD16TOD	B
N21	**Vssx**	—	P
N22	**i_fwdclk**	PECLINDIFFA	I
N23	**i_fwdclk_l**	PECLINDIFFA	I
P1	**b_m1d<6>**	BD8CS	B
P2	**b_m1d<7>**	BD8CS	B
P3	**b_m1c<0>**	BD8CS	B
P4	**b_m1d<9>**	BD8CS	B
P20	**Vssx**	—	P
P21	**Vssx**	—	P
P22	**b_cd_l<14>**	BD16TOD	B
P23	**Vssx**	—	P
R1	**b_m1d<8>**	BD8CS	B
R2	**b_m1d<10>**	BD8CS	B
R3	**b_m1d<11>**	BD8CS	B
R4	**Vdd**	—	P
R20	**Vdd**	—	P
R21	**b_cclki_l<1>**	—	I
R22	**b_cclko_l<1>**	BD16TOD	O
R23	**b_cd_l<13>**	BD16TOD	B
T1	**Vss**	—	P
T2	**b_m1d<12>**	BD8CS	B
T3	**b_m1d<14>**	BD8CS	B

Table 3–9 Dchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
T4	b_m1c<1>	BD8CS	B
T20	b_cd_l<11>	BD16TOD	B
T21	Vssx	—	P
T22	i_vref<1>	DDRV	I
T23	Vss	—	P
U1	b_m1d<13>	BD8CS	B
U2	b_m1d<15>	BD8CS	B
U3	b_m1d<16>	BD8CS	B
U4	b_m1d<19>	BD8CS	B
U20	b_cd_l<9>	BD16TOD	B
U21	Vssx	—	P
U22	Vssx	—	P
U23	b_cd_l<12>	BD16TOD	B
V1	Vss	—	P
V2	b_m1d<17>	BD8CS	B
V3	b_m1d<20>	BD8CS	B
V4	Vdd	—	P
V20	Vdd	—	P
V21	Vssx	—	P
V22	b_cd_l<10>	BD16TOD	B
V23	Vss	—	P
W1	b_m1d<18>	BD8CS	B
W2	b_m1d<21>	BD8CS	B
W3	b_m1d<23>	BD8CS	B
W20	Vssx	—	P
W21	b_cc_l<0>	BD16TOD	B
W22	b_cd_l<8>	BD16TOD	B
W23	Vssx	—	P
Y1	b_m1d<22>	BD8CS	B
Y2	b_m1c<2>	BD8CS	B
Y4	Vdd	—	P
Y6	Vdd	—	P
Y7	b_m1d<29>	BD8CS	B
Y8	o_nandtr	B8	O

Table 3–9 Dchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
Y9	**Vdd**	—	P
Y10	**b_spare<3>**	BD4TS	B
Y11	**i_cpm<0>**	IBUF	I
Y12	**Vdd**	—	P
Y13	**i_pad<1>**	IBUF	I
Y14	**Vssx**	—	P
Y15	**Vdd**	—	P
Y16	**b_cd_l<3>**	BD16TOD	B
Y17	**i_vref<0>**	DDRV	I
Y18	**Vdd**	—	P
Y19	**Vssx**	—	P
Y20	**Vdd**	—	P
Y22	**Vssx**	—	P
Y23	**Vssx**	—	P

3.3 Pchip Pins and Signals

This section provides information about Pchip pins, pin types, pin numbers, and signal definitions.

3.3.1 Pchip Pin List by Function

Table 3–10 lists the pin categories, signal names, types, and signal functions for the Pchip.

Table 3–10 Pchip Pin List by Function

Signal Name	Quantity	Type	Function
CAPbus Interface			
i_cack	1	I	Cchip acknowledge to Pchip
i_cact_l	1	I	Cchip CAP command active
b_cap<23:0>	24	B	CAPbus command/address
i_capgd	1	I	PAD good data sideband signal
i_capsel	1	I	Cchip CAP command valid for this Pchip
i_capselrmt	1	I	Cchip CAP command valid for remote Pchip
b_creqa_l, **creqb_l**	2	O	Request CAPbus
i_creqrmt_l	1	I	Other Pchip CAP request
b_pack	1	O	Pchip acknowledge to Cchip

Table 3–10 Pchip Pin List by Function (Continued)

Signal Name	Quantity	Type	Function
SUBTOTAL	33	—	—
PADbus Interface			
b_padc<7:0>	8	B	PAD check bits
b_padd<31:0>	32	B	PAD data bus
SUBTOTAL	40	—	—
PCI Interface			
b_ack64_l	1	B	64-bit transfer acknowledge
b_ad<63:0>	64	B	Address and data
b_cbe_l<7:0>	8	B	Command and byte enables
b_devsel_l	1	B	Device select
b_frame_l	1	B	Frame
b_gntreq_l<0>	1	O	Bus grant
b_gnt_l<6:1>	6	O	Bus grant
b_irdy_l	1	B	Initiator ready
b_par	1	B	Parity
b_par64	1	B	Parity
i_pclkdiv<1:0>	2	I	PCI clock divisor value
i_pclki	1	I	PCI clock input
b_pclko<7:0>	8	O	Master PCI clocks
b_perr_l	1	B	Parity error
b_prst_l	1	O	PCI reset
b_reqgnt_l<0>	1	I	Bus request
b_req_l<6:1>	6	I	Bus request
b_req64_l	1	B	64-bit transfer request
b_serr_l	1	I	System error
b_stop_l	1	B	Stop
b_trdy_l	1	B	Target ready
SUBTOTAL	109	—	—
Miscellaneous			
b_error	1	O	Error detected
i_fwdclk, i_fwdclk_l	2	I	Clock forward clock
b_monitor<1:0>	2	O	Internal signal monitor outputs
i_pid	1	I	Pchip ID number
i_sysclk, i_sysclk_l	2	I	Clock in

Table 3–10 Pchip Pin List by Function (Continued)

Signal Name	Quantity	Type	Function
i_sysrst_l	1	I	Reset
i_vref	1	I	2-V interface reference voltage
i_scanrclk	1	I	Scan receive clock
i_scanrclken	1	I	Scan receive clock enable
b_spare<3:0>	4	I	Unused pads
SUBTOTAL	16	—	—
CSALT			
o_nandtr	1	O	NAND tree
i_scanen	1	I	Scan enable
i_scanin	1	I	Scan in
i_trsen_l	1	I	Tristate outputs
SUBTOTAL	4	—	—
SIGNAL SUBTOTAL	202	—	—
Power Pins			
Vdd	36	P	**Vdd** ring (9000)
Vss	36	P	**Vss** plane (8000)
Vssx	9	P	
SUBTOTAL	81	—	—
SIGNAL/PIN TOTAL	283	—	21 pins not connected

3.3.2 Pchip Sorted Pin List

Table 3–11 lists the Pchip pins in alphanumeric order by signal name.

Table 3–11 Pchip Pins – Alphanumeric by Signal Name

Signal Name	Pin Number	Driver	Type
b_ack64_l	Y21	BD12CPCIU	B
b_ad<0>	AA6	BD12CPCIU	B
b_ad<1>	Y7	BD12CPCIU	B
b_ad<2>	AC5	BD12CPCIU	B
b_ad<3>	AB6	BD12CPCIU	B
b_ad<4>	AA7	BD12CPCIU	B
b_ad<5>	Y8	BD12CPCIU	B
b_ad<6>	AB7	BD12CPCIU	B
b_ad<7>	AC7	BD12CPCIU	B
b_ad<8>	AA8	BD12CPCIU	B

Table 3–11 Pchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin Number	Driver	Type
b_ad<9>	AB9	BD12CPCIU	B
b_ad<10>	Y10	BD12CPCIU	B
b_ad<11>	AC9	BD12CPCIU	B
b_ad<12>	AA10	BD12CPCIU	B
b_ad<13>	AB10	BD12CPCIU	B
b_ad<14>	AC10	BD12CPCIU	B
b_ad<15>	Y11	BD12CPCIU	B
b_ad<16>	AA11	BD12CPCIU	B
b_ad<17>	AB11	BD12CPCIU	B
b_ad<18>	AC11	BD12CPCIU	B
b_ad<19>	AA12	BD12CPCIU	B
b_ad<20>	AB12	BD12CPCIU	B
b_ad<21>	AC13	BD12CPCIU	B
b_ad<22>	AB13	BD12CPCIU	B
b_ad<23>	AA13	BD12CPCIU	B
b_ad<24>	Y13	BD12CPCIU	B
b_ad<25>	AC14	BD12CPCIU	B
b_ad<26>	AB14	BD12CPCIU	B
b_ad<27>	Y14	BD12CPCIU	B
b_ad<28>	AB15	BD12CPCIU	B
b_ad<29>	AA15	BD12CPCIU	B
b_ad<30>	AB16	BD12CPCIU	B
b_ad<31>	AC17	BD12CPCIU	B
b_ad<32>	W22	BD12CPCIU	B
b_ad<33>	V21	BD12CPCIU	B
b_ad<34>	U20	BD12CPCIU	B
b_ad<35>	W23	BD12CPCIU	B
b_ad<36>	V22	BD12CPCIU	B
b_ad<37>	U21	BD12CPCIU	B
b_ad<38>	U23	BD12CPCIU	B
b_ad<39>	T21	BD12CPCIU	B
b_ad<40>	T22	BD12CPCIU	B
b_ad<41>	R21	BD12CPCIU	B
b_ad<42>	R22	BD12CPCIU	B

Table 3–11 Pchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin Number	Driver	Type
b_ad<43>	P20	BD12CPCIU	B
b_ad<44>	R23	BD12CPCIU	B
b_ad<45>	P21	BD12CPCIU	B
b_ad<46>	P22	BD12CPCIU	B
b_ad<47>	P23	BD12CPCIU	B
b_ad<48>	N20	BD12CPCIU	B
b_ad<49>	N23	BD12CPCIU	B
b_ad<50>	M21	BD12CPCIU	B
b_ad<51>	M22	BD12CPCIU	B
b_ad<52>	L23	BD12CPCIU	B
b_ad<53>	L22	BD12CPCIU	B
b_ad<54>	L21	BD12CPCIU	B
b_ad<55>	L20	BD12CPCIU	B
b_ad<56>	K23	BD12CPCIU	B
b_ad<57>	K22	BD12CPCIU	B
b_ad<58>	K21	BD12CPCIU	B
b_ad<59>	J23	BD12CPCIU	B
b_ad<60>	K20	BD12CPCIU	B
b_ad<61>	J22	BD12CPCIU	B
b_ad<62>	J21	BD12CPCIU	B
b_ad<63>	H22	BD12CPCIU	B
b_cap<0>	A21	BD16TOD	B
b_cap<1>	B20	BD16TOD	B
b_cap<2>	C19	BD16TOD	B
b_cap<3>	B19	BD16TOD	B
b_cap<4>	C18	BD16TOD	B
b_cap<5>	D17	BD16TOD	B
b_cap<6>	B18	BD16TOD	B
b_cap<7>	C17	BD16TOD	B
b_cap<8>	D16	BD16TOD	B
b_cap<9>	A17	BD16TOD	B
b_cap<10>	C16	BD16TOD	B
b_cap<11>	B16	BD16TOD	B
b_cap<12>	B15	BD16TOD	B

Table 3–11 Pchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin Number	Driver	Type
b_cap<13>	A15	BD16TOD	B
b_cap<14>	C14	BD16TOD	B
b_cap<15>	A14	BD16TOD	B
b_cap<16>	D13	BD16TOD	B
b_cap<17>	C13	BD16TOD	B
b_cap<18>	A13	BD16TOD	B
b_cap<19>	C12	BD16TOD	B
b_cap<20>	B12	BD16TOD	B
b_cap<21>	B11	BD16TOD	B
b_cap<22>	C11	BD16TOD	B
b_cap<23>	D11	BD16TOD	B
b_cbe_l<0>	AA16	BD12CPCIU	B
b_cbe_l<1>	AB17	BD12CPCIU	B
b_cbe_l<2>	Y16	BD12CPCIU	B
b_cbe_l<3>	AA17	BD12CPCIU	B
b_cbe_l<4>	AA23	BD12CPCIU	B
b_cbe_l<5>	Y22	BD12CPCIU	B
b_cbe_l<6>	W21	BD12CPCIU	B
b_cbe_l<7>	Y23	BD12CPCIU	B
b_creqa_l	A7	BD4T	O
b_creqb_l	C8	BD4T	O
b_devsel_l	Y19	BD12CPCIU	B
b_error	P2	BD12CPCIU	O
b_frame_l	AA19	BD12CPCIU	B
b_gnt_l<1>	F22	BD12CPCIU	O
b_gnt_l<2>	G20	BD12CPCIU	O
b_gnt_l<3>	E22	BD12CPCIU	O
b_gnt_l<4>	E21	BD12CPCIU	O
b_gnt_l<5>	C23	BD12CPCIU	O
b_gnt_l<6>	D21	BD12CPCIU	O
b_gntreq_l<0>	H21	BD12CPCIU	O
b_irdy_l	AB20	BD12CPCIU	B
b_monitor<0>	Y5	BD4CS	O
b_monitor<1>	AC3	BD4CS	O

Table 3–11 Pchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin Number	Driver	Type
b_pack	B7	BD4T	O
b_padc<0>	L4	BD4TS	B
b_padc<1>	L3	BD4TS	B
b_padc<2>	L2	BD4TS	B
b_padc<3>	L1	BD4TS	B
b_padc<4>	N2	BD4TS	B
b_padc<5>	N3	BD4TS	B
b_padc<6>	N4	BD4TS	B
b_padc<7>	P1	BD4TS	B
b_padd<0>	D8	BD4TS	B
b_padd<1>	C7	BD4TS	B
b_padd<2>	B6	BD4TS	B
b_padd<3>	A5	BD4TS	B
b_padd<4>	D7	BD4TS	B
b_padd<5>	C6	BD4TS	B
b_padd<6>	B5	BD4TS	B
b_padd<7>	A4	BD4TS	B
b_padd<8>	C5	BD4TS	B
b_padd<9>	B4	BD4TS	B
b_padd<10>	A3	BD4TS	B
b_padd<11>	D5	BD4TS	B
b_padd<12>	D2	BD4TS	B
b_padd<13>	E3	BD4TS	B
b_padd<14>	D1	BD4TS	B
b_padd<15>	E2	BD4TS	B
b_padd<16>	F3	BD4TS	B
b_padd<17>	G4	BD4TS	B
b_padd<18>	E1	BD4TS	B
b_padd<19>	F2	BD4TS	B
b_padd<20>	G3	BD4TS	B
b_padd<21>	H4	BD4TS	B
b_padd<22>	G2	BD4TS	B
b_padd<23>	H3	BD4TS	B
b_padd<24>	H2	BD4TS	B

Table 3–11 Pchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin Number	Driver	Type
b_padd<25>	J3	BD4TS	B
b_padd<26>	J2	BD4TS	B
b_padd<27>	K4	BD4TS	B
b_padd<28>	J1	BD4TS	B
b_padd<29>	K3	BD4TS	B
b_padd<30>	K2	BD4TS	B
b_padd<31>	K1	BD4TS	B
b_par	AB18	BD12CPCIU	B
b_par64	W20	BD12CPCIU	B
b_pclko<0>	U2	BD12CPCIU	O
b_pclko<1>	T4	BD12CPCIU	O
b_pclko<2>	U3	BD12CPCIU	O
b_pclko<3>	U4	BD12CPCIU	O
b_pclko<4>	V3	BD12CPCIU	O
b_pclko<5>	W2	BD12CPCIU	O
b_pclko<6>	Y1	BD12CPCIU	O
b_pclko<7>	W3	BD12CPCIU	O
b_perr_l	AC19	BD12CPCIU	B
b_prst_l	AC4	BD12CPCIU	O
b_req64_l	AC20	BD12CPCIU	B
b_req_l<1>	G21	BD12CPCIU	I
b_req_l<2>	E23	BD12CPCIU	I
b_req_l<3>	F21	BD12CPCIU	I
b_req_l<4>	D23	BD12CPCIU	I
b_req_l<5>	D22	BD12CPCIU	I
b_req_l<6>	E20	BD12CPCIU	I
b_reqgnt_l<0>	G23	BD12CPCIU	I
b_serr_l	Y17	BD12CPCIU	I
b_spare<0>	Y2	BD4CS	I
b_spare<1>	AA1	BD4CS	I
b_spare<2>	AA4	BD4CS	I
b_spare<3>	AB5	BD4CS	I
b_stop_l	AA20	BD12CPCIU	B
b_trdy_l	AC21	BD12CPCIU	B

Table 3–11 Pchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin Number	Driver	Type
i_cack	B8	IBUF	I
i_cact_l	A9	IBUF	I
i_capgd	D10	IBUF	I
i_capsel	B9	IBUF	I
i_capselrmt	C9	IBUF	I
i_creqrmt_l	C10	IBUF	I
i_fwdclk	V2	PECLINDIFFA	I
i_fwdclk_l	W1	PECLINDIFFA	I
i_pclkdiv<0>	AB4	TLCHT	I
i_pclkdiv<1>	AA5	TLCHT	I
i_pclki	N21	IBUF	I
i_pid	R3	IBUF	I
i_scanen	P4	IBUF	I
i_scanin	T2	IBUF	I
i_scanrclk	R1	IBUF	I
i_scanrclken	P3	IBUF	I
i_sysclk	M2	PECLINDIFFA	I
i_sysclk_l	M3	PECLINDIFFA	I
i_sysrst_l	R2	IBUF	I
i_trsen_l	U1	IBUF	I
i_vref	D14	DDRV	I
o_nandtr	T3	B8	O
Vdd	A1	—	P
Vdd	A23	—	P
Vdd	AA3	—	P
Vdd	AA21	—	P
Vdd	AB2	—	P
Vdd	AB22	—	P
Vdd	AC1	—	P
Vdd	AC23	—	P
Vdd	B2	—	P
Vdd	B22	—	P
Vdd	C3	—	P
Vdd	C21	—	P

Table 3–11 Pchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin Number	Driver	Type
Vdd	D4	—	P
Vdd	D6	—	P
Vdd	D9	—	P
Vdd	D12	—	P
Vdd	D15	—	P
Vdd	D18	—	P
Vdd	D20	—	P
Vdd	F4	—	P
Vdd	F20	—	P
Vdd	J4	—	P
Vdd	J20	—	P
Vdd	M4	—	P
Vdd	M20	—	P
Vdd	R4	—	P
Vdd	R20	—	P
Vdd	V4	—	P
Vdd	V20	—	P
Vdd	Y4	—	P
Vdd	Y6	—	P
Vdd	Y12	—	P
Vdd	Y18	—	P
Vdd	Y9	—	P
Vdd	Y15	—	P
Vdd	Y20	—	P
Vss	A2	—	P
Vss	A6	—	P
Vss	A8	—	P
Vss	A12	—	P
Vss	A16	—	P
Vss	A18	—	P
Vss	A22	—	P
Vss	AA2	—	P
Vss	AA22	—	P
Vss	AB1	—	P

Table 3–11 Pchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin Number	Driver	Type
Vss	AB3	—	P
Vss	AB21	—	P
Vss	AB23	—	P
Vss	AC2	—	P
Vss	AC6	—	P
Vss	AC8	—	P
Vss	AC12	—	P
Vss	AC16	—	P
Vss	AC18	—	P
Vss	AC22	—	P
Vss	B1	—	P
Vss	B3	—	P
Vss	B21	—	P
Vss	B23	—	P
Vss	C2	—	P
Vss	C22	—	P
Vss	F1	—	P
Vss	F23	—	P
Vss	H1	—	P
Vss	H23	—	P
Vss	M1	—	P
Vss	M23	—	P
Vss	T1	—	P
Vss	T23	—	P
Vss	V1	—	P
Vss	V23	—	P
Vssx	A10	—	P
Vssx	A11	—	P
Vssx	A19	—	P
Vssx	A20	—	P
Vssx	B13	—	P
Vssx	B14	—	P
Vssx	B17	—	P
Vssx	C15	—	P

Table 3–11 Pchip Pins – Alphanumeric by Signal Name (Continued)

Signal Name	Pin Number	Driver	Type
Vssx	D19	—	P

Table 3–12 lists the Pchip pins in alphanumeric order by pin number.

Table 3–12 Pchip Pins – Alphanumeric by Pin Number

Pin Number	Signal Name	Driver	Type
A1	**Vdd**	—	P
A2	**Vss**	—	P
A3	**b_padd<10>**	BD4TS	B
A4	**b_padd<7>**	BD4TS	B
A5	**b_padd<3>**	BD4TS	B
A6	**Vss**	—	P
A7	**b_creqa_l**	BD4T	O
A8	**Vss**	—	P
A9	**i_cact_l**	IBUF	I
A10	**Vssx**	—	P
A11	**Vssx**	—	P
A12	**Vss**	—	P
A13	**b_cap<18>**	BD16TOD	B
A14	**b_cap<15>**	BD16TOD	B
A15	**b_cap<13>**	BD16TOD	B
A16	**Vss**	—	P
A17	**b_cap<9>**	BD16TOD	B
A18	**Vss**	—	P
A19	**Vssx**	—	P
A20	**Vssx**	—	P
A21	**b_cap<0>**	BD16TOD	B
A22	**Vss**	—	P
A23	**Vdd**	—	P
AA1	**b_spare<1>**	BD4CS	I
AA2	**Vss**	—	P
AA3	**Vdd**	—	P
AA4	**b_spare<2>**	BD4CS	I
AA5	**i_pclkdiv<1>**	TLCHT	I
AA6	**b_ad<0>**	BD12CPCIU	B

Table 3–12 Pchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
AA7	**b_ad<4>**	BD12CPCIU	B
AA8	**b_ad<8>**	BD12CPCIU	B
AA10	**b_ad<12>**	BD12CPCIU	B
AA11	**b_ad<16>**	BD12CPCIU	B
AA12	**b_ad<19>**	BD12CPCIU	B
AA13	**b_ad<23>**	BD12CPCIU	B
AA15	**b_ad<29>**	BD12CPCIU	B
AA16	**b_cbe_l<0>**	BD12CPCIU	B
AA17	**b_cbe_l<3>**	BD12CPCIU	B
AA19	**b_frame_l**	BD12CPCIU	B
AA20	**b_stop_l**	BD12CPCIU	B
AA21	**Vdd**	—	P
AA22	**Vss**	—	P
AA23	**b_cbe_l<4>**	BD12CPCIU	B
AB1	**Vss**	—	P
AB2	**Vdd**	—	P
AB3	**Vss**	—	P
AB4	**i_pclkdiv<0>**	TLCHT	I
AB5	**b_spare<3>**	BD4CS	I
AB6	**b_ad<3>**	BD12CPCIU	B
AB7	**b_ad<6>**	BD12CPCIU	B
AB9	**b_ad<9>**	BD12CPCIU	B
AB10	**b_ad<13>**	BD12CPCIU	B
AB11	**b_ad<17>**	BD12CPCIU	B
AB12	**b_ad<20>**	BD12CPCIU	B
AB13	**b_ad<22>**	BD12CPCIU	B
AB14	**b_ad<26>**	BD12CPCIU	B
AB15	**b_ad<28>**	BD12CPCIU	B
AB16	**b_ad<30>**	BD12CPCIU	B
AB17	**b_cbe_l<1>**	BD12CPCIU	B
AB18	**b_par**	BD12CPCIU	B
AB20	**b_irdy_l**	BD12CPCIU	B
AB21	**Vss**	—	P
AB22	**Vdd**	—	P

Table 3–12 Pchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
AB23	**Vss**	—	P
AC1	**Vdd**	—	P
AC2	**Vss**	—	P
AC3	**b_monitor<1>**	BD4CS	O
AC4	**b_prst_l**	BD12CPCIU	O
AC5	**b_ad<2>**	BD12CPCIU	B
AC6	**Vss**	—	P
AC7	**b_ad<7>**	BD12CPCIU	B
AC8	**Vss**	—	P
AC9	**b_ad<11>**	BD12CPCIU	B
AC10	**b_ad<14>**	BD12CPCIU	B
AC11	**b_ad<18>**	BD12CPCIU	B
AC12	**Vss**	—	P
AC13	**b_ad<21>**	BD12CPCIU	B
AC14	**b_ad<25>**	BD12CPCIU	B
AC16	**Vss**	—	P
AC17	**b_ad<31>**	BD12CPCIU	B
AC18	**Vss**	—	P
AC19	**b_perr_l**	BD12CPCIU	B
AC20	**b_req64_l**	BD12CPCIU	B
AC21	**b_trdy_l**	BD12CPCIU	B
AC22	**Vss**	—	P
AC23	**Vdd**	—	P
B1	**Vss**	—	P
B2	**Vdd**	—	P
B3	**Vss**	—	P
B4	**b_padd<9>**	BD4TS	B
B5	**b_padd<6>**	BD4TS	B
B6	**b_padd<2>**	BD4TS	B
B7	**b_pack**	BD4T	O
B8	**i_cack**	IBUF	I
B9	**i_capsel**	IBUF	I
B11	**b_cap<21>**	BD16TOD	B
B12	**b_cap<20>**	BD16TOD	B

Table 3–12 Pchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
B13	**Vssx**	—	P
B14	**Vssx**	—	P
B15	**b_cap<12>**	BD16TOD	B
B16	**b_cap<11>**	BD16TOD	B
B17	**Vssx**	—	P
B18	**b_cap<6>**	BD16TOD	B
B19	**b_cap<3>**	BD16TOD	B
B20	**b_cap<1>**	BD16TOD	B
B21	**Vss**	—	P
B22	**Vdd**	—	P
B23	**Vss**	—	P
C2	**Vss**	—	P
C3	**Vdd**	—	P
C5	**b_padd<8>**	BD4TS	B
C6	**b_padd<5>**	BD4TS	B
C7	**b_padd<1>**	BD4TS	B
C8	**b_creqb_l**	BD4T	O
C9	**i_capselrmt**	IBUF	I
C10	**i_creqrmt_l**	IBUF	I
C11	**b_cap<22>**	BD16TOD	B
C12	**b_cap<19>**	BD16TOD	B
C13	**b_cap<17>**	BD16TOD	B
C14	**b_cap<14>**	BD16TOD	B
C15	**Vssx**	—	P
C16	**b_cap<10>**	BD16TOD	B
C17	**b_cap<7>**	BD16TOD	B
C18	**b_cap<4>**	BD16TOD	B
C19	**b_cap<2>**	BD16TOD	B
C21	**Vdd**	—	P
C22	**Vss**	—	P
C23	**b_gnt_l<5>**	BD12CPCIU	O
D1	**b_padd<14>**	BD4TS	B
D2	**b_padd<12>**	BD4TS	B
D4	**Vdd**	—	P

Table 3–12 Pchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
D5	**b_padd<11>**	BD4TS	B
D6	**Vdd**	—	P
D7	**b_padd<4>**	BD4TS	B
D8	**b_padd<0>**	BD4TS	B
D9	**Vdd**	—	P
D10	**i_capgd**	IBUF	I
D11	**b_cap<23>**	BD16TOD	B
D12	**Vdd**	—	P
D13	**b_cap<16>**	BD16TOD	B
D14	**i_vref**	DDRV	I
D15	**Vdd**	—	P
D16	**b_cap<8>**	BD16TOD	B
D17	**b_cap<5>**	BD16TOD	B
D18	**Vdd**	—	P
D19	**Vssx**	—	P
D20	**Vdd**	—	P
D21	**b_gnt_l<6>**	BD12CPCIU	O
D22	**b_req_l<5>**	BD12CPCIU	I
D23	**b_req_l<4>**	BD12CPCIU	I
E1	**b_padd<18>**	BD4TS	B
E2	**b_padd<15>**	BD4TS	B
E3	**b_padd<13>**	BD4TS	B
E20	**b_req_l<6>**	BD12CPCIU	I
E21	**b_gnt_l<4>**	BD12CPCIU	O
E22	**b_gnt_l<3>**	BD12CPCIU	O
E23	**b_req_l<2>**	BD12CPCIU	I
F1	**Vss**	—	P
F2	**b_padd<19>**	BD4TS	B
F3	**b_padd<16>**	BD4TS	B
F4	**Vdd**	—	P
F20	**Vdd**	—	P
F21	**b_req_l<3>**	BD12CPCIU	I
F22	**b_gnt_l<1>**	BD12CPCIU	O
F23	**Vss**	—	P

Table 3–12 Pchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
G2	b_padd<22>	BD4TS	B
G3	b_padd<20>	BD4TS	B
G4	b_padd<17>	BD4TS	B
G20	b_gnt_l<2>	BD12CPCIU	O
G21	b_req_l<1>	BD12CPCIU	I
G23	b_reqgnt_l<0>	BD12CPCIU	I
H1	Vss	—	P
H2	b_padd<24>	BD4TS	B
H3	b_padd<23>	BD4TS	B
H4	b_padd<21>	BD4TS	B
H21	b_gntreq_l<0>	BD12CPCIU	O
H22	b_ad<63>	BD12CPCIU	B
H23	Vss	—	P
J1	b_padd<28>	BD4TS	B
J2	b_padd<26>	BD4TS	B
J3	b_padd<25>	BD4TS	B
J4	Vdd	—	P
J20	Vdd	—	P
J21	b_ad<62>	BD12CPCIU	B
J22	b_ad<61>	BD12CPCIU	B
J23	b_ad<59>	BD12CPCIU	B
K1	b_padd<31>	BD4TS	B
K2	b_padd<30>	BD4TS	B
K3	b_padd<29>	BD4TS	B
K4	b_padd<27>	BD4TS	B
K20	b_ad<60>	BD12CPCIU	B
K21	b_ad<58>	BD12CPCIU	B
K22	b_ad<57>	BD12CPCIU	B
K23	b_ad<56>	BD12CPCIU	B
L1	b_padc<3>	BD4TS	B
L2	b_padc<2>	BD4TS	B
L3	b_padc<1>	BD4TS	B
L4	b_padc<0>	BD4TS	B
L20	b_ad<55>	BD12CPCIU	B

Table 3–12 Pchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
L21	**b_ad<54>**	BD12CPCIU	B
L22	**b_ad<53>**	BD12CPCIU	B
L23	**b_ad<52>**	BD12CPCIU	B
M1	**Vss**	—	P
M2	**i_sysclk**	PECLINDIFFA	I
M3	**i_sysclk_l**	PECLINDIFFA	I
M4	**Vdd**	—	P
M20	**Vdd**	—	P
M21	**b_ad<50>**	BD12CPCIU	B
M22	**b_ad<51>**	BD12CPCIU	B
M23	**Vss**	—	P
N2	**b_padc<4>**	BD4TS	B
N3	**b_padc<5>**	BD4TS	B
N4	**b_padc<6>**	BD4TS	B
N20	**b_ad<48>**	BD12CPCIU	B
N21	**i_pclki**	IBUF	I
N23	**b_ad<49>**	BD12CPCIU	B
P1	**b_padc<7>**	BD4TS	B
P2	**b_error**	BD12CPCIU	O
P3	**i_scanrclken**	IBUF	I
P4	**i_scanen**	IBUF	I
P20	**b_ad<43>**	BD12CPCIU	B
P21	**b_ad<45>**	BD12CPCIU	B
P22	**b_ad<46>**	BD12CPCIU	B
P23	**b_ad<47>**	BD12CPCIU	B
R1	**i_scanrclk**	IBUF	I
R2	**i_sysrst_l**	IBUF	I
R3	**i_pid**	IBUF	I
R4	**Vdd**	—	P
R20	**Vdd**	—	P
R21	**b_ad<41>**	BD12CPCIU	B
R22	**b_ad<42>**	BD12CPCIU	B
R23	**b_ad<44>**	BD12CPCIU	B
T1	**Vss**	—	P

Table 3–12 Pchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
T2	**i_scanin**	IBUF	I
T3	**o_nandtr**	B8	O
T4	**b_pclko<1>**	BD12CPCIU	O
T21	**b_ad<39>**	BD12CPCIU	B
T22	**b_ad<40>**	BD12CPCIU	B
T23	**Vss**	—	P
U1	**i_trsen_l**	IBUF	I
U2	**b_pclko<0>**	BD12CPCIU	O
U3	**b_pclko<2>**	BD12CPCIU	O
U4	**b_pclko<3>**	BD12CPCIU	O
U20	**b_ad<34>**	BD12CPCIU	B
U21	**b_ad<37>**	BD12CPCIU	B
U23	**b_ad<38>**	BD12CPCIU	B
V1	**Vss**	—	P
V2	**i_fwdclk**	PECLINDIFFA	I
V3	**b_pclko<4>**	BD12CPCIU	O
V4	**Vdd**	—	P
V20	**Vdd**	—	P
V21	**b_ad<33>**	BD12CPCIU	B
V22	**b_ad<36>**	BD12CPCIU	B
V23	**Vss**	—	P
W1	**i_fwdclk_l**	PECLINDIFFA	I
W2	**b_pclko<5>**	BD12CPCIU	O
W3	**b_pclko<7>**	BD12CPCIU	O
W20	**b_par64**	BD12CPCIU	B
W21	**b_cbe_l<6>**	BD12CPCIU	B
W22	**b_ad<32>**	BD12CPCIU	B
W23	**b_ad<35>**	BD12CPCIU	B
Y1	**b_pclko<6>**	BD12CPCIU	O
Y2	**b_spare<0>**	BD4CS	I
Y4	**Vdd**	—	P
Y5	**b_monitor<0>**	BD4CS	O
Y6	**Vdd**	—	P
Y7	**b_ad<1>**	BD12CPCIU	B

Table 3–12 Pchip Pins – Alphanumeric by Pin Number (Continued)

Pin Number	Signal Name	Driver	Type
Y8	**b_ad<5>**	BD12CPCIU	B
Y9	**Vdd**	—	P
Y10	**b_ad<10>**	BD12CPCIU	B
Y11	**b_ad<15>**	BD12CPCIU	B
Y12	**Vdd**	—	P
Y13	**b_ad<24>**	BD12CPCIU	B
Y14	**b_ad<27>**	BD12CPCIU	B
Y15	**Vdd**	—	P
Y16	**b_cbe_l<2>**	BD12CPCIU	B
Y17	**b_serr_l**	BD12CPCIU	I
Y18	**Vdd**	—	P
Y19	**b_devsel_l**	BD12CPCIU	B
Y20	**Vdd**	—	P
Y21	**b_ack64_l**	BD12CPCIU	B
Y22	**b_cbe_l<5>**	BD12CPCIU	B
Y23	**b_cbe_l<7>**	BD12CPCIU	B

4
Electrical Specifications

This chapter provides dc and ac electrical data for each ASIC in the 21272 chipset.

4.1 Absolute Limits

Table 4–1 lists the absolute operating conditions for the 21272 chipset elements using the CMOS5L process.

Table 4–1 CMOS5L Absolute Operating Conditions

Parameter	Minimum	Maximum
Operating ambient temperature	0°C (32°F)	40°C (104°F)
Storage temperature	–55°C (–67°F)	125°C (257°F)
Positive dc supply voltage	**Vss** –0.5 V dc	**Vss** +3.6 V dc
Operating junction temperature	0°C (32°F)	100°C (212°F)
Input signal voltage (3.3-V I/O)	**Vss** –0.3 V	3.6 V
Input signal voltage (5.0-V tolerant I/O)	**Vss** –0.3 V	**Vss** +6.3 V (**Vdd** = 3.0 V)

The maximum power dissipation for each 21272 chip is listed in Table 4–2 and is specified at Tj maximum = 125°C (257°F) with **Vdd** maximum = 3.465 V.

Table 4–2 Maximum Power Dissipation

Device	Frequency/Bus Width	Power
Dchip	83-MHz system interface	3.21 W
Cchip	83-MHz system interface	5.40 W
Pchip	64-bit, 33.3-MHz PCI bus 32-bit, 66.7-MHz PCI bus	3.23 W 3.34 W

4.2 DC Characteristics

The 21272 chipset is designed to run in a CMOS/TTL environment. The design uses the CMOS5L, 3.3-V process. All I/O, except for the PCI interface, the CPU interface, and the internal CAPbus interface, assume a 3.3-V signaling environment. The PCI interface utilizes the 3.3-V supply for output drive and is designed to accept an input voltage of up to 5 V. The CPU interface and the CAPbus operate on a custom 2-V interface.

4.2.1 Power Supply

The 21272 chipset operates on a 3.3-V, ±5% supply. The **i_vref** inputs are set by means of a resistor divider network to deliver 0.67 x (2-V supply – 1.4 V nominal).

4.2.2 Input Clocks

Signals **i_sysclk**/**i_sysclk_l** and **i_fwdclk**/**i_fwdclk_l** are expected to be differential PECL signals. Signal **cfin** is defined as a single-ended signal that operates using the custom 2-V interface.

4.2.3 Signal Pins

Signal pins are organized into three classes: open-drain (OD) I/O, 3.3-V I/O, and 5-V compatible I/O. Table 4–3 lists the dc characteristics for the 21272 I/O signal pins.

Table 4–3 CMOS DC Characteristics

Driver Class	Signaling Type	Vih	Vil	Voh	Vol	Overshoot	Undershoot
Open-drain	Custom 2 V	**vref** + 0.05 V	**vref** – 0.05 V	2 V	0.8 V	—	—
CSALT "T"	LVTTL	2.0 V	1.0 V	2.4 V	0.4 V	**Vss** + 4.2 V	–0.8 V
CSALT "TS"	LVTTL	2.0 V	1.0 V	2.4 V	0.4 V	**Vss** + 4.2 V	–0.8 V
CSALT "C"	LVTTL	2.0 V	1.0 V	2.4 V	0.4 V	**Vss** + 6.3 V	–0.8 V
CSALT "CS"	LVTTL	2.0 V	1.0 V	2.4 V	0.4 V	**Vss** + 6.3 V	–0.8 V
CSALT PCI	—	2.0 V	1.0 V	2.4 V	0.4 V	—	—

4.2.3.1 Open-Drain I/O

Signals that use the OD driver are low asserted on the pins of the chip. Input pins use a single-ended differential receiver. The receiver switchpoint is ±50 mV around **vref**. Output pins use the driver to assert an electrical zero on the network. Deassertion of the network is performed by external termination to the 2-V supply (see Figure 4–1).

Figure 4–1 Open-Drain Termination Scheme
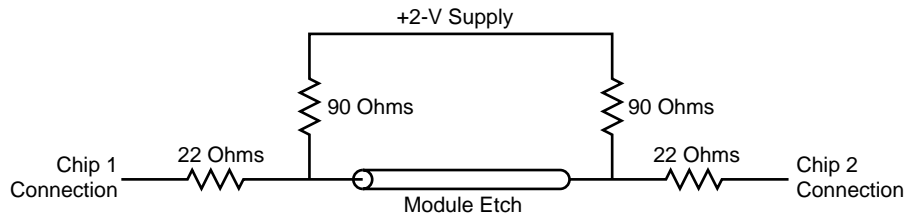
LJ-05490.AI4

4.2.3.2 3.3-V I/O

The 3.3-V I/O pins, when used as an input, receive signals that adhere to TTL signaling conventions. When used as an output, the signals swing from **Vss** to **Vdd** (3.3 V). Output timing is specified to standard TTL levels. These signals assume that there are no devices capable of driving a dc level above the 3.3-V supply.

4.2.3.3 5-V Compatible I/O

The 5-V compatible I/O pins, when used as an input, receive signals that adhere to TTL signaling conventions. When used as an output, the signals swing from **Vss** to **Vdd** (3.3 V). Output timing is specified to standard TTL levels. These signals assume that the network contains devices capable of driving a dc level based upon a 5-V supply.

4.2.4 DC Specifications

Table 4–4 shows the dc specifications for the 21272.

Table 4–4 DC Specifications

Driver	Spec	Lower Level DRV	Upper Level DRV	Units	VDD	Conditions and Limits
BD4CS						
	VIL	—	1.00	V	3.00	0.0 V < Vin < 5.5 V
	VIH	2.00	—	V	3.60	0.0 V < Vin < 5.5 V
	VOL	0.00	0.40	V	3.00	0.0 mA < IOL < 4.0 mA
	VOH	2.40	3.00	V	3.00	-4.0 mA < IOH < 0.0 mA
	IOZL	-10.00	0.00	μA	3.60	Vout = VSS
	IOZH	0.00	10.00	μA	3.60	Vout = VDD
	IOZ5H	0.00	10.00	μA	3.60	Vout = 5.25 V
	DIOL	26.00	50.00	mA	3.00	Vout = 0.5 * VDD
	DIOH	-40.00	-21.00	mA	3.00	Vout = 0.5 * VDD
BD8CS						
	VIL	—	1.00	V	3.00	0.0 V < Vin < 5.5 V
	VIH	2.00	—	V	3.60	0.0 V < Vin < 5.5 V
	VOL	0.00	0.40	V	3.00	0.0 mA < IOL < 8.0 mA

DC Characteristics

Table 4–4 DC Specifications (Continued)

Driver	Spec	Lower Level DRV	Upper Level DRV	Units	VDD	Conditions and Limits
	VOH	2.40	3.00	V	3.00	-8.0 mA < IOH < 0.0 mA
	IOZL	-10.00	0.00	µA	3.60	Vout = VSS
	IOZH	0.00	10.00	µA	3.60	Vout = VDD
	IOZ5H	0.00	10.00	µA	3.60	Vout = 5.25 V
	DIOL	33.00	66.00	mA	3.00	Vout = 0.5 * VDD
	DIOH	-54.00	-28.00	mA	3.00	Vout = 0.5 * VDD
BD4TS						
	VIL	—	1.00	V	3.00	0.0 V < Vin < 5.5 V
	VIH	2.00	—	V	3.60	0.0 V < Vin < 5.5 V
	VOL	0.00	0.40	V	3.00	0.0 mA < IOL < 4.0 mA
	VOH	2.40	3.00	V	3.00	-4.0 mA < IOH < 0.0 mA
	DIOL	24.00	49.00	mA	3.00	Vout = 0.5 * VDD
	DIOH	-45.00	-22.00	mA	3.00	Vout = 0.5 * VDD
BD4T						
	VIL	—	1.00	V	3.00	0.0 V < Vin < 5.5 V
	VIH	2.00	—	V	3.60	0.0 V < Vin < 5.5 V
	VOL	0.00	0.40	V	3.00	0.0 mA < IOL < 4.0 mA
	VOH	2.40	3.00	V	3.00	-4.0 mA < IOH < 0.0 mA
	DIOL	41.00	69.00	mA	3.00	Vout = 0.5 * VDD
	DIOH	-65.00	-37.00	mA	3.00	Vout = 0.5 * VDD
BD6T						
	VIL	—	1.00	V	3.00	0.0 V < Vin < 5.5 V
	VIH	2.00	—	V	3.60	0.0 V < Vin < 5.5 V
	VOL	0.00	0.40	V	3.00	0.0 mA < IOL < 6.0 mA
	VOH	2.40	3.00	V	3.00	-6.0 mA < IOH < 0.0 mA
	DIOL	58.00	96.00	mA	3.00	Vout = 0.5 * VDD
	DIOH	-97.00	-55.00	mA	3.00	Vout = 0.5 * VDD

Table 4–4 DC Specifications (Continued)

Driver	Spec	Lower Level DRV	Upper Level DRV	Units	VDD	Conditions and Limits
BD8T						
	VIL	—	1.00	V	3.00	0.0 V < Vin < 5.5 V
	VIH	2.00	—	V	3.60	0.0 V < Vin < 5.5 V
	VOL	0.00	0.40	V	3.00	0.0 mA < IOL < 8.0 mA
	VOH	2.40	3.00	V	3.00	-8.0 mA < IOH < 0.0 mA
	DIOL	77.00	127.00	mA	3.00	Vout = 0.5 ∗ VDD
	DIOH	-124.00	-70.00	mA	3.00	Vout = 0.5 ∗ VDD
BD12CPCIU						
	VIL	—	1.00	V	3.00	0.0 V < Vin < 5.5 V
	VIH	2.00	—	V	3.60	0.0 V < Vin < 5.5 V
	VOL	0.00	0.40	V	3.00	0.0 mA < IOL < 8.0 mA
	VOH	2.40	3.00	V	3.00	-8.0 mA < IOH < 0.0 mA
	IOZL	-3.00	-1.20	mA	3.60	Vout = VSS
	IOZH	20.00	100.00	µA	3.60	Vout = VDD
	IOZ5H		1.30	mA	3.60	Vout = 5.25 V
	DIOL	110.00	181.00	mA	3.00	Vout = 0.5 ∗ VDD
	DIOH	-79.00	-46.00	mA	3.00	Vout = 0.5 ∗ VDD
	VCLAMP	5.35	6.60	V	3.00	Force +25 mA
IBUF						
	VIL	—	1.00	V	3.00	0.0 V < Vin < 5.5 V
	VIH	2.00	—	V	3.60	0.0 V < Vin < 5.5 V
	IIL	-10.00	0.00	µA	3.60	Vin = VSS
	IIH	0.00	10.00	µA	3.60	Vin = 5.25 V
TLCHT						
	VIL	—	1.00	V	3.00	0.0 V < Vin < 5.5 V
	VIH	2.00	—	V	3.60	0.0 V < Vin < 5.5 V
	IIL	-10.00	0.00	µA	3.60	Vin = VSS
	IIH	0.00	10.00	µA	3.60	Vin = VDD
B8						
	VOL	—	0.4	V	3	0.0 mA < IOL < 8.0 mA
	VOH	2.4	3	V	3	-8.0 mA < IOH < 0.0 mA
	DIOL	63	106	mA	3	Vout = 0.5 ∗ VDD
	DIOH	-85	-47	mA	3	Vout = 0.5 ∗ VDD

Table 4–4 DC Specifications (Continued)

Driver	Spec	Lower Level DRV	Upper Level DRV	Units	VDD	Conditions and Limits
BD16TOD						
	VIL	—	Vref - 0.1V	V	3.00	0.0 V < Vin < VDD + 0.5, 0.7 < Vref < VDD/2
	VIH	Vref + 0.1V	—	V	3.60	0.0 V < Vin < VDD + 0.5, 0.7 < Vref < VDD/2
	VOL	0.00	0.19	V	3.00	0.0 mA < IOL < 20.0 mA
	IOZL	-10.00	10.00	µA	3.60	0 < Vin < VDD
PECLINDIFF						
	VIL	—	1.46	V	3.00	0.0 V < Vin < VDD + 0.5
	VIL	—	1.75	V	3.60	0.0 V < Vin < VDD + 0.5
	VIH	1.56	—	V	3.00	0.0 V < Vin < VDD + 0.5
	VIH	1.85	—	V	3.60	0.0 V < Vin < VDD + 0.5
	IIL	-500.00	500.00	µA	3.60	0 < Vin < VDD

4.3 AC Specifications

This section contains ac specifications for the Cchip, Dchip, and Pchip.

4.3.1 Cchip Specification

Table 4–5 contains ac specifications for the Cchip. All outputs include a 700-ps simultaneous switching adder (maximum case). A standard load of 60 pf is used on all outputs.

Table 4–5 Cchip AC Specification

Signal Name	TT Min Clk Rise	TT Max Clk Rise	TT Min Clk Fall	TT Max Clk Fall	FF Min Clk Rise	FF Max Clk Rise	FF Min Clk Fall	FF Max Clk Fall
b_c0ao_l<14:2>	6.502	7.472	6.164	7.466	3.636	5.449	3.403	5.397
b_c0clko_l	7.425	7.425	6.167	6.167	5.422	5.422	3.41	3.41
b_c0div_l	6.502	7.472	6.164	7.466	3.636	5.449	3.403	5.397
b_c0fv_l	6.502	7.472	6.164	7.466	3.636	5.449	3.403	5.397
b_c1ao_l<14:2>	6.407	7.389	6.064	7.379	3.584	5.391	3.345	5.335
b_c1clko_l	7.383	7.383	6.094	6.094	5.382	5.382	3.355	3.355
b_c1div_l	6.407	7.389	6.064	7.379	3.584	5.391	3.345	5.335
b_c1fv_l	6.407	7.389	6.064	7.379	3.584	5.391	3.345	5.335

	Setup, Clk Rise	Setup, Clk Fall	Hold, Clk Rise	Hold, Clk Fall	Reference Clk
b_c0ai_l<14:2>	0.64	0.825	0.465	0.406	B_CLKI_L
b_c1ai_l<14:2>	0.625	0.815	0.436	0.322	B_CLKI_L

	Min	Max	Setup	Hold
b_cack	3.7	8.10		
b_cacta_l	3.8	8.20		

Table 4–5 Cchip AC Specification (Continued)

	Min	Max	Setup	Hold	
b_cactb_l	3.8	8.20			
b_capgd	3.8	8.20			
b_capsel	3.8	8.30			
b_cfrst_l	2.6	7.70			no SSO
b_cpma	3.3	7.10			
b_cpmb	3.3	7.10			
b_m0a	3.1	7.50			
b_m0ba<2:0>	3.5	7.50			
b_m0cs_l<3:0>	3.2	6.90			
b_m0dqm	3.3	7.20			
b_m1a	3.1	6.80			
b_m1ba<2:0>	3.1	6.80			
b_m1cs_l<3:0>	3.1	6.70			
b_m1dqm	3.1	6.70			
b_m2a	3.1	7.30			
b_m2ba<2:0>	3.3	7.00			
b_m2cs_l<3:0>	3.1	6.70			
b_m2dqm	3.1	6.70			
b_m3a	3.2	7.30			
b_m3ba<2:0>	3.2	6.90			
b_m3cs_l<3:0>	3.1	6.70			
b_m3dqm	3.1	6.70			
b_mcas_l<0>	3.2	6.90			
b_mcas_l<1>	3.1	6.70			
b_mcas_l<2>	3.1	6.70			
b_mcas_l<3>	3.1	6.70			
b_mcke<0>	3.1	6.90			
b_mcke<1>	3.1	6.70			
b_mcke<2>	3.1	6.70			
b_mcke<3>	3.1	6.70			
b_monitor	4.30	10.70			
b_mpdclk	4.60	12.60	0.26	2.01	
b_mpdd	4.60	12.50	0.29	1.98	
b_mras_l<0>	3.2	6.90			
b_mras_l<1>	3.1	6.70			
b_mras_l<2>	3.1	6.70			
b_mras_l<3>	3.1	6.70			
b_mwe_l<0>	3.2	6.90			
b_mwe_l<1>	3.1	6.70			
b_mwe_l<2>	3.1	6.70			
b_mwe_l<3>	3.1	6.70			
b_pada	3.2	7.10			
b_padb	3.3	7.20			
b_sromoe_l			-0.324	1.919	
b_sysrsta_l	3.1	7.20			
b_sysrstb_l	3.1	7.20			
b_sysrstc_l	3.1	7.20			

Table 4–5 Cchip AC Specification (Continued)

	Min	Max	Setup	Hold
b_tas	4.80	10.70		
b_tcs_l	4.70	11.70		
b_td<7:0>	4.00	13.00	0.32	2.07
b_tia	4.30	10.70		
b_tioe_l	4.70	12.00		
b_tis	4.30	9.90		
b_toe_l	4.40	11.50		
b_twe_l	4.30	10.70		
i_creq_l			0.684	2.179
i_intim_l			-0.226	2.299
i_modrst_l				2.925
i_pack			-0.15	2.204

	TT Min Clk Rise	TT Max Clk Rise	TT Min Clk Fall	TT Max Clk Fall	FF Min Clk Rise	FF Max Clk Rise	FF Min Clk Fall	FF Max Clk Fall
b_cap<23:0>	5.8	7.2	5.7	7.2	4	5	2.9	3.8

	Setup, Clk Rise	Setup, Clk Fall	Hold, Clk Rise	Hold, Clk Fall	Reference Clk
b_cap<23:0>	0.511	0.536	1.795	2.31	SYSCLK

4.3.2 C4chip Specification

Table 4–5 contains ac specifications for the C4chip. All outputs include a 700-ps simultaneous switching adder (maximum case). A standard load of 60 pf is used on all outputs.

Table 4–6 C4chip AC Specifications

		TT Process/SS Environment				FF Process/FF Environment			
		Clock Rise		Clock Fall		Clock Rise		Clock Fall	
		DR	DF	DR	DF	DR	DF	DR	DF
b_c0ao_l<14:02>	Min	6.90	7.11	6.99	7.07	3.72	3.65	3.77	3.62
	Max	6.99	7.20	7.09	7.17	3.82	3.74	3.88	3.73
b_c0fv_l	Min	6.90	7.11	6.99	7.07	3.72	3.65	3.77	3.62
	Max	6.99	7.20	7.09	7.17	3.82	3.74	3.88	3.73
b_c0div_l	Min	6.90	7.11	6.99	7.07	3.72	3.65	3.77	3.62
	Max	6.99	7.20	7.09	7.17	3.82	3.74	3.88	3.73
b_c0clko_l	Min	6.95	6.95	7.13	7.13	3.78	3.78	3.69	3.69
	Max	6.95	6.95	7.13	7.13	3.78	3.78	3.69	3.69
b_c1ao_l<14:02>	Min	6.58	6.79	6.66	6.75	3.44	3.36	3.48	3.33
	Max	6.66	6.87	6.74	6.82	3.49	3.41	3.53	3.38

Table 4–6 C4chip AC Specifications

		TT Process/SS Environment				FF Process/FF Environment			
		Clock Rise		Clock Fall		Clock Rise		Clock Fall	
		DR	DF	DR	DF	DR	DF	DR	DF
b_c1fv_l	**Min**	6.58	6.79	6.66	6.75	3.44	3.36	3.48	3.33
	Max	6.66	6.87	6.74	6.82	3.49	3.41	3.53	3.38
b_c1div_l	**Min**	6.58	6.79	6.66	6.75	3.44	3.36	3.48	3.33
	Max	6.66	6.87	6.74	6.82	3.49	3.41	3.53	3.38
b_c1clko_l	**Min**	6.62	6.62	6.78	6.78	3.45	3.45	3.34	3.34
	Max	6.62	6.62	6.78	6.78	3.45	3.45	3.34	3.34
b_c2ao_l<14:02>	**Min**	6.60	6.83	6.68	6.78	3.40	3.33	3.44	3.30
	Max	6.69	6.91	6.77	6.86	3.50	3.41	3.53	3.39
b_c2fv_l	**Min**	6.60	6.83	6.68	6.78	3.40	3.33	3.44	3.30
	Max	6.69	6.91	6.77	6.86	3.50	3.41	3.53	3.39
b_c2div_l	**Min**	6.60	6.83	6.68	6.78	3.40	3.33	3.44	3.30
	Max	6.69	6.91	6.77	6.86	3.50	3.41	3.53	3.39
b_c2clko_l	**Min**	6.62	6.62	6.80	6.80	3.43	3.43	3.32	3.32
	Max	6.62	6.62	6.80	6.80	3.43	3.43	3.32	3.32
b_c3ao_l<14:02>	**Min**	6.90	7.12	7.01	7.09	3.76	3.67	3.81	3.66
	Max	7.01	7.21	7.12	7.19	3.83	3.74	3.88	3.73
b_c3fv_l	**Min**	6.90	7.12	7.01	7.09	3.76	3.67	3.81	3.66
	Max	7.01	7.21	7.12	7.19	3.83	3.74	3.88	3.73
b_c3div_l	**Min**	6.90	7.12	7.01	7.09	3.76	3.67	3.81	3.66
	Max	7.01	7.21	7.12	7.19	3.83	3.74	3.88	3.73
b_c3clko_l	**Min**	7.01	7.01	7.19	7.19	3.83	3.83	3.73	3.73
	Max	7.01	7.01	7.19	7.19	3.83	3.83	3.73	3.73

AC Specifications

Table 4–6 C4chip AC Specifications

	Setup		Hold		
	Clk Rise	Clk Fall	Clk Rise	Clk Fall	Reference Clk
b_c0ai_l<14:02>	0.9	0.7	0.3	0.5	**b_clki_l**
b_c1ai_l<14:02>	0.9	0.8	0.3	0.4	**b_clki_l**
b_c2ai_l<14:02>	0.9	0.9	0.3	0.4	**b_clki_l**
b_c3ai_l<14:02>	0.9	0.7	0.3	0.5	**b_clki_l**

	Min	Max	Setup	Hold	
b_monitor	3.9	8.6			
b_mpdclk	4.1	9.2	–0.1	3.5	
b_mpdd	4.1	9.2	–0.6	3.5	
b_twe_l	4.0	8.9			
b_tia	4.0	8.8			
b_td<07:00>	4.0	9.7	–0.6	2.8	
b_tas	4.0	8.9			
b_tis	4.0	8.9			
b_tcs_l	4.0	8.9			
b_tioe_l	4.0	8.8			
b_toe_l	4.0	8.8			
i_intim_l			–0.8	3.2	
i_modrst_l			–0.8	3.2	
b_sromoe_l			–0.3	2.5	
b_sysrsta_l	3.0	7.9			
b_sysrstb_l	3.0	7.9			
b_sysrstc_l	3.0	7.9			
b_cfrst_l	2.3	8.2			
b_m0ba_l<02:00>	2.2	7.2			
b_m0a	2.8	7.2			
b_0cs_l<03:00>	2.2	7.2			
b_m0dqm	2.8	7.2			
b_mras_l<0>	2.8	7.2			
b_mcas_l<0>	2.8	7.2			
b_mwe_l<0>	2.8	7.2			
b_mcke_l<0>	2.8	7.2			
b_m1ba_l<02:00>	2.2	7.2	7.2		

Table 4–6 C4chip AC Specifications

	Min	Max	Setup	Hold
b_m1a	2.8	7.2	7.2	
b_1cs_l<03:00>	2.2	7.2	7.2	
b_m1dqm	2.8	7.2	7.2	
b_mras_l<1>	2.8	7.2	7.2	
b_mcas_l<1>	2.8	7.2	7.2	
b_mwe_l<1>	2.8	7.2		
b_mcke_l<1>	2.8	7.2		
b_m2ba_l<02:00>	2.2	7.2		
b_m2a	2.8	7.2		
b_2cs_l<03:00>	2.2	7.2		
b_m2dqm	2.8	7.2		
b_mras_l<2>	2.8	7.2		
b_mcas_l<2>	2.8	7.2		
b_mwe_l<2>	2.8	7.2		
b_mcke_l<2>	2.8	7.2		
b_m3ba_l<02:00>	2.2	7.2		
b_m3a	2.8	7.2		
b_3cs_l<03:00>	2.2	7.2		
b_m3dqm	2.8	7.2		
b_mras_l<3>	2.8	7.2		
b_mcas_l<3>	2.8	7.2		
b_mwe_l<3>	2.8	7.2		
b_mcke_l<3>	2.8	7.2		
b_pada	3.0	7.9		
b_padb	3.0	8.0		
b_cpma	3.0	7.9		
b_cpmb	3.0	7.9		
b_cacta_l	3.9	8.8		
b_cactb_l	3.9	8.8		
b_cack	3.2	9.8		
b_capgd	4.1	10.1		
b_capsel	3.2	9.9		
i_pack			0.4	2.1
i_creq_l			–0.1	2.5

Table 4–6 C4chip AC Specifications

	Min	Max	Setup	Hold
o_sysclk_fb	2.1	4.8		

	TT Data				FF Data			
	Min Rise	Max Rise	Min Fall	Max Fall				
b_cap<23:00>	7.0	7.5	7.1	7.6	2.7	2.7	2.5	2.5

	Setup		Hold		
	Data Rise	Data Fall	Data Rise	Data Fall	Reference Clk
b_cap<23:00>	0.4	0.4	2.9	2.9	**sysclk**

4.3.3 Dchip Specification

Table 4–7 contains ac specifications for the Dchip. All outputs include a 700-ps simultaneous switching adder (maximum case). A standard load of 60 pf is used on all outputs.

Table 4–7 Dchip AC Specification

Signal Name	TT Min Clk Rise	TT Max Clk Rise	TT Min Clk Fall	TT Max Clk Fall	FF Min Clk Rise	FF Max Clk Rise	FF Min Clk Fall	FF Max Clk Fall
b_cc_l<0>	6.63	7.59	6.30	7.59	3.75	5.55	3.52	5.51
b_cc_l<1>	6.64	7.58	6.32	7.59	3.74	5.55	3.51	5.49
b_cc_l<2>	6.66	7.63	6.34	7.64	3.77	5.58	3.53	5.53
b_cc_l<3>	6.66	7.61	6.32	7.61	3.77	5.59	3.54	5.53
b_cd_l<15:8>	6.64	7.58	6.32	7.59	3.74	5.55	3.51	5.49
b_cd_l<23:16>	6.66	7.63	6.34	7.64	3.77	5.58	3.53	5.53
b_cd_l<31:24>	6.66	7.61	6.32	7.61	3.77	5.59	3.54	5.53
b_cd_l<7:0>	6.63	7.59	6.30	7.59	3.75	5.55	3.52	5.51
b_clko_l<0>	7.55	7.55	6.33	6.33	5.52	5.52	3.53	3.53
b_clko_l<1>	7.57	7.57	6.35	6.35	5.52	5.52	3.53	3.53
b_clko_l<2>	7.60	7.60	6.38	6.38	5.55	5.55	3.56	3.56
b_clko_l<3>	7.58	7.58	6.35	6.35	5.55	5.55	3.55	3.55

	Setup, Clk Rise	Setup, Clk Fall	Hold, Clk Rise	Hold, Clk Fall	Reference Clk
b_cc_l<3:0>	0.64	0.73	0.35	0.28	B_CLKI_L
b_cd_l<31:0>	0.64	0.73	0.35	0.28	B_CLKI_L

	Min	Max	Setup	Hold
b_m0c	3.50	8.80	0.20	2.00
b_m0d	3.50	8.80	0.20	2.00
b_m1c	3.50	8.80	0.20	2.00
b_m1d	3.50	8.80	0.20	2.00
b_p0c	3.90	9.00	-0.20	2.30
b_p0d	3.90	9.00	-0.20	2.30
b_p1c	3.90	9.00	-0.20	2.30

Table 4–7 Dchip AC Specification (Continued)

Signal Name	TT Min Clk Rise	TT Max Clk Rise	TT Min Clk Fall	TT Max Clk Fall	FF Min Clk Rise	FF Max Clk Rise	FF Min Clk Fall	FF Max Clk Fall
b_p1d	3.90	9.00	-0.20	2.30				
i_cpm			0.00	2.10				
i_pad			0.30	2.00				
i_sysrst				2.70				

4.3.4 D4chip Specification

Table 4–7 contains ac specifications for the D4chip. All outputs include a 700-ps simultaneous switching adder (maximum case). A standard load of 60 pf is used on all outputs.

Table 4–8 D4chip AC Specifications

		TT Process/SS Environment				FF Process/FF Environment			
		Clock Rise		Clock Fall		Clock Rise		Clock Fall	
		DR	DF	DR	DF	DR	DF	DR	DF
b_cd_l<7:0>	Min	7.54	6.63	7.53	6.30	5.51	3.75	5.47	3.52
	Max	7.59	6.69	7.59	6.37	5.55	3.79	5.51	3.56
b_cc_l<0>	Min	7.54	6.63	7.53	6.30	5.51	3.75	5.47	3.52
	Max	7.59	6.69	7.59	6.37	5.55	3.79	5.51	3.56
b_clko_l<0>	Min	7.55	7.55	6.33	6.33	5.52	5.52	3.53	3.53
	Max	7.55	7.55	6.33	6.33	5.52	5.52	3.53	3.53
b_cd_l<15:8>	Min	7.55	6.64	7.55	6.32	5.51	3.74	5.46	3.51
	Max	7.58	6.68	7.59	6.36	5.55	3.78	5.49	3.55
b_cc_l<1>	Min	7.55	6.64	7.55	6.32	5.51	3.74	5.46	3.51
	Max	7.58	6.68	7.59	6.36	5.55	3.78	5.49	3.55
b_clko_l<1>	Min	7.57	7.57	6.35	6.35	5.52	5.52	3.53	3.53
	Max	7.57	7.57	6.35	6.35	5.52	5.52	3.53	3.53
b_cd_l<23:16>	Min	7.58	6.66	7.57	6.34	5.54	3.77	5.48	3.53
	Max	7.63	6.73	7.64	6.41	5.58	3.82	5.53	3.59
b_cc_l<2>	Min	7.58	6.66	7.57	6.34	5.54	3.77	5.48	3.53
	Max	7.63	6.73	7.64	6.41	5.58	3.82	5.53	3.59
b_clko_l<2>	Min	7.60	7.60	6.38	6.38	5.55	5.55	3.56	3.56
	Max	7.60	7.60	6.38	6.38	5.55	5.55	3.56	3.56

Table 4–8 D4chip AC Specifications

b_cd_l<31:24>	Min	7.56	6.66	7.54	6.32	5.54	3.77	5.48	3.54
	Max	7.61	6.71	7.61	6.38	5.59	3.82	5.53	3.59
b_cc_l<3>	Min	7.56	6.66	7.54	6.32	5.54	3.77	5.48	3.54
	Max	7.61	6.71	7.61	6.38	5.59	3.82	5.53	3.59
b_clko_l<3>	Min	7.58	7.58	6.35	6.35	5.55	5.55	3.55	3.55
	Max	7.58	7.58	6.35	6.35	5.55	5.55	3.55	3.55

	Setup		Hold		
	Clk Rise	**Clk Fall**	**Clk Rise**	**Clk Fall**	**Reference Clk**
b_cd_l<31:00>	0.6	0.7	0.4	0.3	b_clki_l
b_cc_l<03:00>	0.6	0.7	0.4	0.3	b_clki_l

	Min	**Max**	**Setup**	**Hold**
b_p0d	3.9	9.00	–0.2	2.3
b_p0c	3.9	9.00	–0.2	2.2
b_p1d	3.9	9.00	–0.2	2.3
b_p1c	3.9	9.00	–0.2	2.3
b_m0d	3.4	8.0	–0.2	2.2
b_m0c	3.4	8.0	–0.2	2.2
b_m1d	3.4	8.0	–0.2	2.2
b_m1c	3.4	8.0	–0.2	2.2
i_cpm			0.0	2.1
i_pad			0.2	1.9
i_sysreset				2.7

4.3.5 Pchip Specification

Table 4–9 contains ac specifications for the Pchip. All outputs include a 700 ps simultaneous switching adder (maximum case). A standard of 60 pf is used on all outputs.

Table 4–9 Pchip AC Specification

Signal Name	Min	Max	Setup	Hold	Reference
b_ack64_l	3.20	8.30	6.30	0.10	PCLKI
b_ad	3.20	8.60	6.60	0.30	PCLKI
b_cbe_l	3.10	8.20	6.40	0.30	PCLKI
b_creqa_l	3.70	8.00			SYSCLK
b_creqb_l	3.70	8.00			SYSCLK
b_devsel_l	3.20	8.00	6.30	0.30	PCLKI
b_error	4.20	9.60			SYSCLK

Table 4–9 Pchip AC Specification (Continued)

Signal Name	Min	Max	Setup	Hold	Reference
b_frame_l	3.10	8.20	6.60	0.20	PCLKI
b_gnt_l	3.20	9.00			PCLKI
b_gntreq_l<0>	3.10	8.00			PCLKI
b_irdy_l	3.10	8.00	6.80	0.30	PCLKI
b_monitor	4.40	10.10			SYSCLK
b_pack	3.70	8.00			SYSCLK
b_padc	3.90	9.30	-0.30	2.40	SYSCLK
b_padd	3.80	9.30	0.40	2.40	SYSCLK
b_par	3.10	8.30	6.60	0.10	PCLKI
b_par64	3.10	8.10	5.90	0.30	PCLKI
b_pclko	3.40	7.70			FWDCLK
	3.40	7.70			FWDCLK_L
b_perr_l	3.20	8.10	6.50	-0.90	PCLKI
b_prst_l	3.70	8.40			PCLKI
b_req_l			5.70	-0.20	PCLKI
b_req64_l	3.40	8.10	4.70	0.30	PCLKI
b_reqgnt_l<0>			6.60	-0.40	PCLKI
b_serr_l			5.30	-0.70	PCLKI
b_stop_l	3.20	8.00	6.40	0.20	PCLKI
b_trdy_l	3.20	8.00	6.60	0.10	PCLKI
i_cack			-0.30	2.20	SYSCLK
i_cact_l			0.70	2.20	SYSCLK
i_capgd			-0.30	2.20	SYSCLK
i_capsel			-0.30	2.20	SYSCLK

Table 4–9 Pchip AC Specification (Continued)

Signal Name	Min	Max	Setup	Hold		Reference
i_capselrmt			-0.30	2.20		SYSCLK
i_creqrmt_l			1.10	2.20		SYSCLK
i_pid			7.90	1.00		SYSCLK
			7.80	-1.20		SYSCLK_L
i_sysrst_l				2.90		SYSCLK

	TT Min Clk Rise	TT Max Clk Rise	TT Min Clk Fall	TT Max Clk Fall	FF Min Clk Rise	FF Max Clk Rise	FF Min Clk Fall	FF Max Clk Fall
b_cap<23:0>	6.50	7.40	5.70	6.50	4.30	5.10	2.70	3.50

	Setup, Clk Rise	Setup, Clk Fall	Hold, Clk Rise	Hold, Clk Fall	Reference Clk
b_cap<23:0>	0.35	0.24	1.80	2.30	SYSCLK

4.4 AC Test Specifications

This section contains the ac test specifications for the Cchip, Dchip, and Pchip. These SPICE test specifications are used internally by DIGITAL for testing wafers and packaging parts.

4.4.1 Cchip AC Test Specifications

Table 4–10 shows the Cchip ac test specifications.

Table 4–10 Cchip AC Test Specifications

Signal Name	TT Min Clk Rise	TT Max Clk Rise	TT Min Clk Fall	TT Max Clk Fall
b_c0ao_l<14:2>	5.75	6.00	5.70	5.75
b_c0clko_l	5.75	5.75	5.70	5.70
b_c0div_l	5.75	6.00	5.70	5.75
b_c0fv_l	5.75	6.00	5.70	5.75
b_c1ao_l<14:2>	5.65	6.00	5.60	5.70
b_c1clko_l	5.70	5.70	5.60	5.60
b_c1div_l	5.65	6.00	5.60	5.70
b_c1fv_l	5.65	6.00	5.60	5.70

	Setup, Clk Rise	Setup, Clk Fall	Hold, Clk Rise	Hold, Clk Fall	Reference Clk
b_c0ai_l<14:2>	0.64	0.83	0.47	0.41	B_CLKI_L
b_c1ai_l<14:2>	0.63	0.82	0.44	0.32	B_CLKI_L

	Min	Max	Setup	Hold
b_cack	3.00	5.90		
b_cacta_l	3.10	6.00		
b_cactb_l	3.10	6.10		
b_capgd	3.10	6.10		
b_capsel	3.10	6.10		
b_cfrst_l	2.70	6.10		
b_cpma	2.80	5.40		
b_cpmb	2.80	5.40		
b_m0a	2.80	6.10		
b_m0ba<2:0>	3.20	6.10		
b_m0cs_l<3:0>	2.90	5.30		

Table 4–10 Cchip AC Test Specifications (Continued)

	Min	Max	Setup	Hold
b_m0dqm	2.90	5.70		
b_m1a	2.80	5.40		
b_m1ba<2:0>	2.80	5.40		
b_m1cs_l<3:0>	2.80	5.30		
b_m1dqm	2.80	5.30		
b_m2a	2.80	6.00		
b_m2ba<2:0>	3.00	5.60		
b_m2cs_l<3:0>	2.80	5.30		
b_m2dqm	2.80	5.30		
b_m3a	2.90	6.00		
b_m3ba<2:0>	2.90	5.50		
b_m3cs_l<3:0>	2.80	5.30		
b_m3dqm	2.80	5.30		
b_mcas_l<0>	2.90	5.30		
b_mcas_l<1>	2.80	5.30		
b_mcas_l<2>	2.80	5.30		
b_mcas_l<3>	2.80	5.30		
b_mcke<0>	2.80	5.30		
b_mcke<1>	2.80	5.30		
b_mcke<2>	2.80	5.30		
b_mcke<3>	2.80	5.30		
b_monitor	3.20	7.10		
b_mpdclk	3.50	9.30	0.26	2.01
b_mpdd	3.50	9.20	2.85	1.98
b_mras_l<0>	2.90	5.30		
b_mras_l<1>	2.80	5.30		
b_mras_l<2>	2.80	5.30		
b_mras_l<3>	2.80	5.30		
b_mwe_l<0>	2.90	5.30		
b_mwe_l<1>	2.80	5.30		
b_mwe_l<2>	2.80	5.30		
b_mwe_l<3>	2.80	5.30		
b_pada	2.80	5.40		
b_padb	2.80	5.40		
b_sromoe_l			-0.32	1.92
b_sysrsta_l	2.70	5.50		
b_sysrstb_l	2.70	5.50		
b_sysrstc_l	2.70	5.50		
b_tas	3.60	7.10		
b_tcs_l	3.60	8.00		
b_td<7:0>	3.00	9.50	0.32	2.07
b_tia	3.20	7.00		
b_tioe_l	3.60	8.30		
b_tis	3.30	6.30		
b_toe_l	3.40	7.90		
b_twe_l	3.20	7.10		
i_creq_l			0.68	2.18
i_intim_l			-0.23	2.30
i_modrst_l				2.93
i_pack			-0.15	2.20
	TT Min Clk Rise	**TT Max Clk Rise**	**TT Min Clk Fall**	**TT Max Clk Fall**
b_cap<23:0>	5.50	6.20	5.30	6.10

Table 4–10 Cchip AC Test Specifications (Continued)

	Setup, Clk Rise	Setup, Clk Fall	Hold, Clk Rise	Hold, Clk Fall	Reference Clk
b_cap<23:0>	0.51	0.54	1.80	2.31	SYSCLK

4.4.2 Dchip AC Test Specifications

Table 4–11 shows the Dchip ac test specifications.

Table 4–11 Dchip AC Test Specifications

Signal Name	Driver	TT Min Clk Rise	TT Max Clk Rise	TT Min Clk Fall	TT Max Clk Fall	
b_cc_l<0>	BD16TOD	5.85	6.20	5.85	5.90	
b_cc_l<1>	BD16TOD	5.85	6.20	5.85	5.90	
b_cc_l<2>	BD16TOD	5.90	6.25	5.85	5.95	
b_cc_l<3>	BD16TOD	5.85	6.25	5.85	5.90	
b_cd_l<15:8>	BD16TOD	5.85	6.20	5.85	5.90	
b_cd_l<23:16>	BD16TOD	5.90	6.25	5.85	5.95	
b_cd_l<31:24>	BD16TOD	5.85	6.25	5.85	5.90	
b_cd_l<7:0>	BD16TOD	5.85	6.20	5.85	5.90	
b_clko_l<0>	BD16TOD	5.85	5.90	6.85	5.85	
b_clko_l<1>	BD16TOD	5.90	5.90	6.85	5.85	
b_clko_l<2>	BD16TOD	5.90	5.90	6.90	5.90	
b_clko_l<3>	BD16TOD	5.90	5.90	6.90	5.90	

		Setup, Clk Rise	Setup, Clk Fall	Hold, Clk Rise	Hold, Clk Fall	Reference Clk
b_cc_l<3:0>	BD16TOD	0.64	0.73	0.35	0.28	B_CLKI_L
b_cd_l<31:0>	BD16TOD	0.64	0.73	0.35	0.28	B_CLKI_L

		Min	Max	Setup	Hold	Reference Clk
b_m0c	BD8CS	2.80	6.00	0.20	2.00	
b_m0d	BD8CS	2.80	6.00	0.20	2.00	
b_m1c	BD8CS	2.80	6.00	0.20	2.00	
b_m1d	BD8CS	2.80	6.00	0.20	2.00	
b_p0c	BD4TS	2.90	6.00	-0.20	2.30	
b_p0d	BD4TS	2.90	6.00	-0.20	2.30	
b_p1c	BD4TS	2.90	6.00	-0.20	2.30	
b_p1d	BD4TS	2.90	6.00	-0.20	2.30	
i_cpm	IBUF			0.00	2.10	
i_pad	IBUF			0.30	2.00	
i_sysrst	IBUF				2.70	

4.4.3 Pchip AC Test Specifications

Table 4–12 shows the Pchip ac test specifications.

Table 4–12 Pchip AC Test Specifications

Signal Name	Min	Max	Setup	Hold	Reference
b_ack64_l	2.90	6.30	6.30	0.10	PCLKI
b_ad	2.90	6.60	6.60	0.30	PCLKI
b_cbe_l	2.80	6.10	6.40	0.30	PCLKI
b_creqa_l	2.90	5.80			SYSCLK
b_creqb_l	2.90	5.80			SYSCLK
b_devsel_l	2.90	6.00	6.30	0.30	PCLKI
b_error	3.20	6.00			SYSCLK
b_frame_l	2.80	6.20	6.50	0.20	PCLKI
b_gnt_l	3.00	7.30			PCLKI
b_gntreq_l<0>	2.80	6.00			PCLKI
b_irdy_l	2.80	6.00	6.70	0.20	PCLKI
b_monitor	3.30	6.50			SYSCLK
b_pack	2.90	5.80			SYSCLK
b_padc	2.90	6.20	-0.26	2.40	SYSCLK
b_padd	2.80	6.20	0.35	2.40	SYSCLK
b_par	2.80	6.30	6.50	0.00	PCLKI
b_par64	2.80	6.10	5.90	0.30	PCLKI
b_pclko	3.10	5.70			FWDCLK
	3.10	5.70			FWDCLK_L
b_perr_l	2.90	6.10	6.40	-0.90	PCLKI
b_prst_l	3.40	6.40			PCLKI
b_req_l			5.60	-0.20	PCLKI
b_req64_l	3.10	6.10	4.70	0.30	PCLKI
b_reqgnt_l<0>			6.60	-0.40	PCLKI
b_serr_l			5.30	-0.70	PCLKI
b_stop_l	2.90	6.10	6.30	0.20	PCLKI
b_trdy_l	2.90	6.00	6.60	0.10	PCLKI
i_cack			-0.30	2.20	SYSCLK
i_cact_l			0.70	2.20	SYSCLK
i_capgd			-0.30	2.20	SYSCLK
i_capsel			-0.30	2.20	SYSCLK
i_capselrmt			-0.30	2.20	SYSCLK
i_creqrmt_l			1.10	2.20	SYSCLK
i_pid			7.90	1.00	SYSCLK
			7.70	-1.20	SYSCLK_L
i_sysrst_l				2.90	SYSCLK

	TT Min Clk Rise	TT Max Clk Rise	TT Min Clk Fall	TT Max Clk Fall	
b_cap<23:0>	5.20	5.40	5.30	5.40	

	Setup, Clk Rise	Setup, Clk Fall	Hold, Clk Rise	Hold, Clk Fall	Reference Clk
b_cap<23:0>	0.35	0.24	1.80	2.30	SYSCLK

AC Test Specifications

5

Mechanical Specifications

This chapter provides dimensional information for each of the 21272 package types.

Chips in the 21272-AA chipset are contained in two custom enhanced super ball grid array (ESBGA) packages. The 21272–C1 Cchip is contained in a 432-point ESBGA. The 21272–D1 Dchip and 21272–P1 Pchip are contained in 304-point ESBGAs. Table 5–1 lists the ESBGA package specifications.

Table 5–1 21272 Packaging

Package Type	Number of Layers	Number of Planes	Number of Balls Tied to Vss Plane and Vdd Ring	Number of Balls Available for I/O	Package Body Size
432-point ESBGA	2	1	88	344	40 mm x 40 mm
304-point ESBGA	2	1	72	232	31 mm x 31 mm

Note: The drawings and tables with dimensions in this chapter are for reference only. Examples of board layout, including detailed engineering drawings and plot files, are available from DIGITAL.

Figures 5–1 and 5–2 show the 21272–C1 432-point ESBGA (Cchip) package.

Figure 5–1 432-Point 2-Layer ESBGA Package (Top and Side View)

Top View

Side View
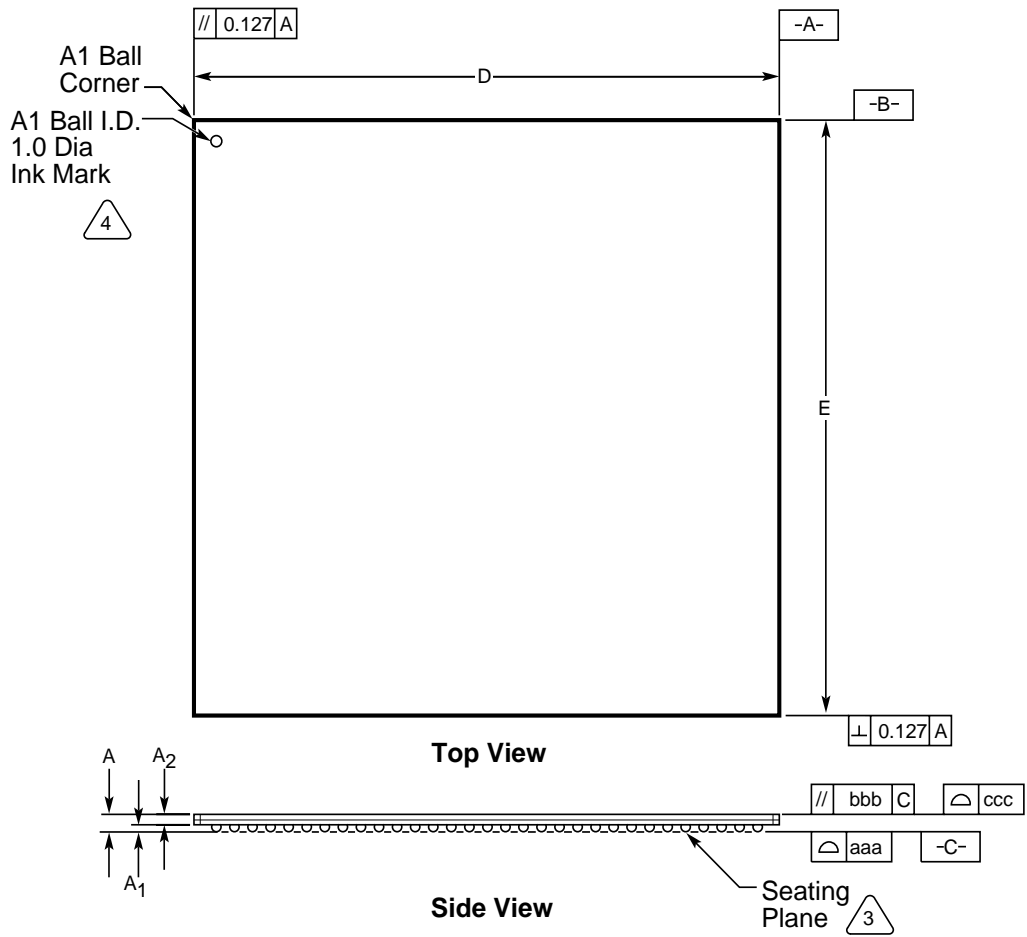
FM-06248.AI4

Figure 5–2 432-Point 2-Layer ESBGA Package (Bottom and Section View)

A-A   Section View

Bottom View
Die Side
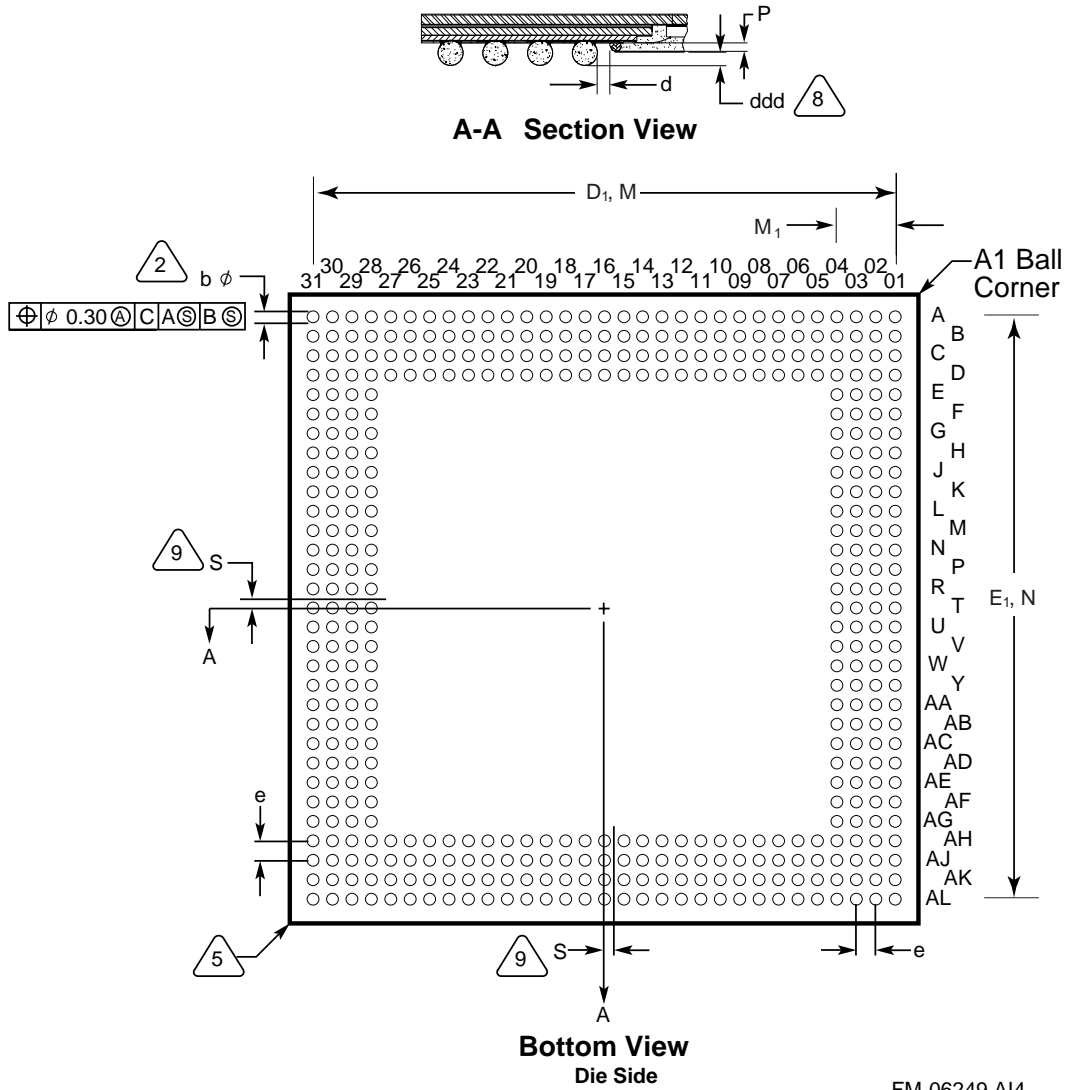
FM-06249.AI4

Table 5–2 lists the 21272–C1 432-point ESBGA (Cchip) package dimensions.

Table 5–2 432-Point 2-Layer ESBGA Package Dimensions

Symbol	Dimension	Value (mm)
A	Package overall thickness	1.41 minimum to 1.67 maximum (1.54 nominal)
A1	Ball height	0.56 minimum to 0.70 maximum (0.63 nominal)
A2	Body thickness	0.85 minimum to 0.97 maximum (0.91 nominal)
D	Package overall width	39.90 minimum to 40.10 maximum (40.00 nominal)
D1	Ball footprint	38.00 minimum to 38.20 maximum (38.10 nominal)
E	Package overall length	39.90 minimum to 40.10 maximum (40.00 nominal)
E1	Ball footprint	38.00 minimum to 38.20 maximum (38.10 nominal)
M,N	Ball matrix	31 X 31
M1	Number of rows deep	4
b	Solder ball diameter	0.60 minimum to 0.90 maximum (0.75 nominal)
d	Corner radius/flat	0.6
e	Solder ball pitch	1.27
aaa	Surface coplanarity	0.20 maximum
bbb	Parallel	0.15 maximum
ccc	Top flatness	0.20 maximum
ddd	Seating plane clearance	0.15 minimum to 0.50 maximum (0.33 nominal)
P	Encapsulation height	0.20 minimum to 0.35 maximum (0.30 nominal)
S	Solder ball placement	0.00 maximum

Figures 5–3 and 5–4 show the 21272–D1 (Dchip) and 21272–P1 (Pchip) 304-point ESBGA package.

Figure 5–3 304-Point 2-Layer ESBGA Package (Top and Side View)
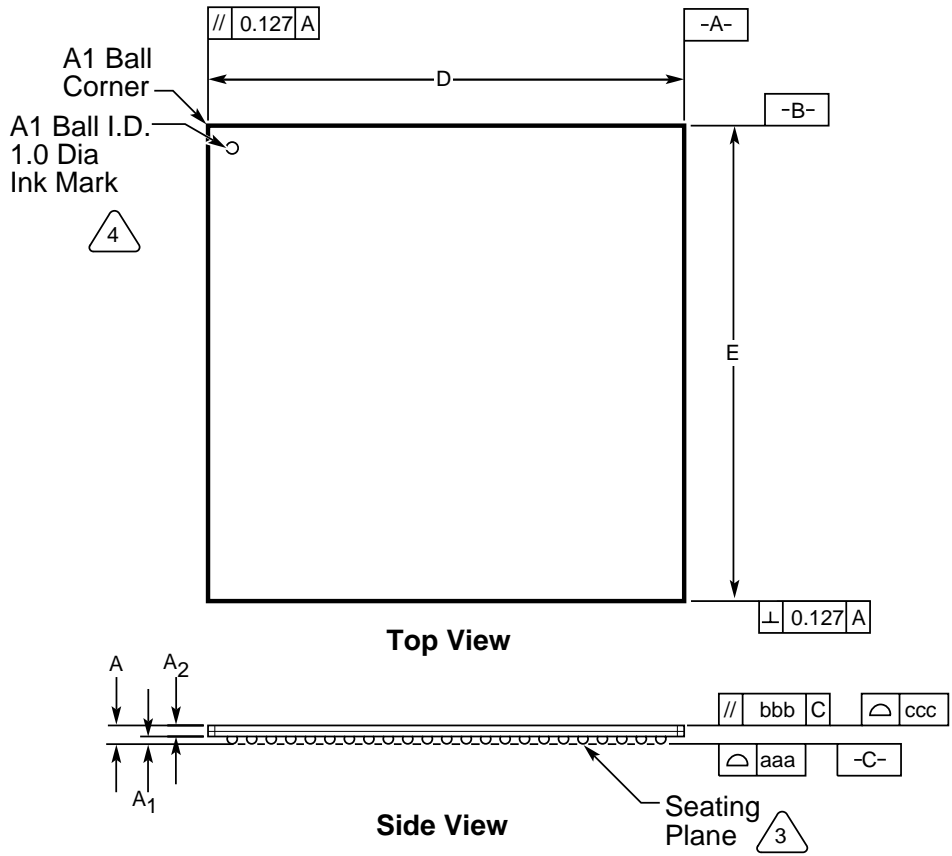
FM-06246.AI4

Figure 5–4 304-Point 2-Layer ESBGA Package (Bottom and Section View)

A-A   Section View

Bottom View
Die Side

FM-06247A.AI4
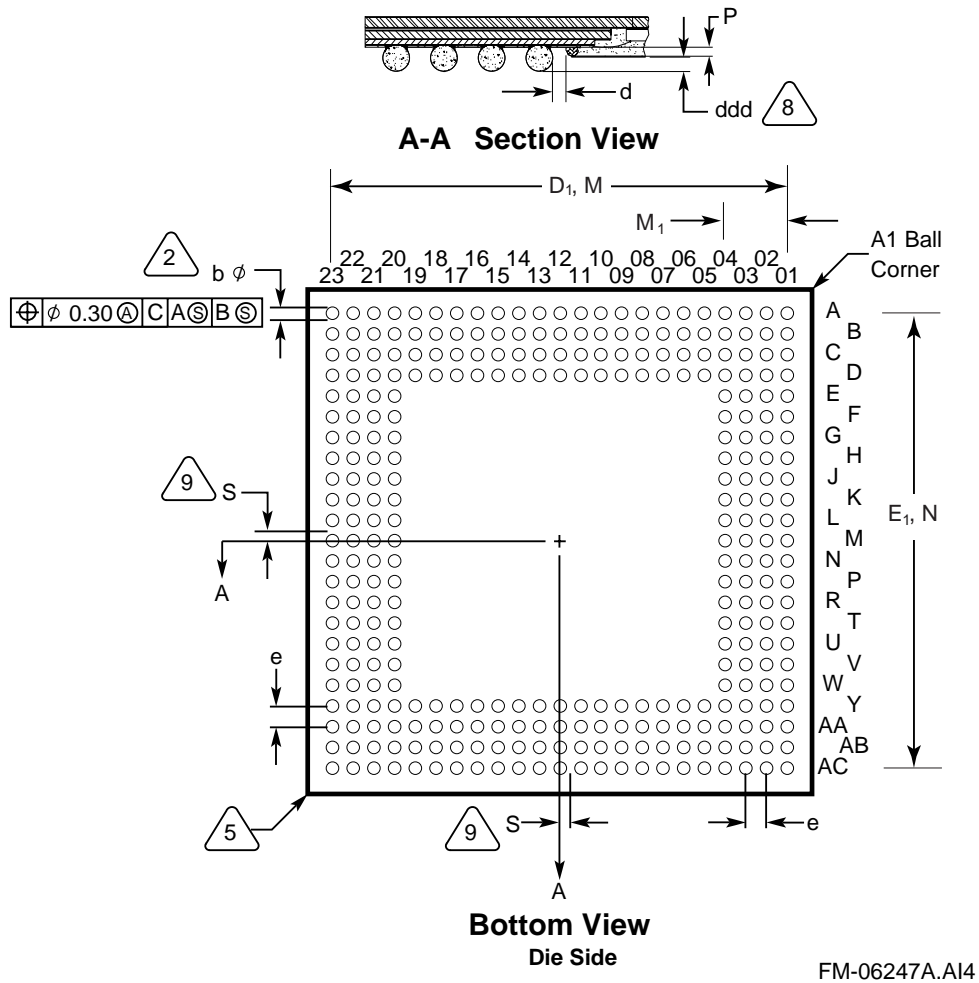
Table 5–3 lists the 21272–D1 (Dchip) and 21272–P1 (Pchip) 304-point ESBGA package dimensions.

Table 5–3 304-Point 2-Layer ESBGA Package Dimensions

Symbol	Dimension	Value (mm)
A	Package overall thickness	1.41 minimum to 1.67 maximum (1.54 nominal)
A1	Ball height	0.56 minimum to 0.70 maximum (0.63 nominal)
A2	Body thickness	0.85 minimum to 0.97 maximum (0.91 nominal)
D	Package overall width	30.90 minimum to 31.10 maximum (31.00 nominal)
D1	Ball footprint	27.84 minimum to 28.04 maximum (27.94 nominal)
E	Package overall length	30.90 minimum to 31.10 maximum (31.00 nominal)
E1	Ball footprint	27.84 minimum to 28.04 maximum (27.94 nominal)
M,N	Ball matrix	23 X 23
M1	Number of rows deep	4
b	Solder ball diameter	0.60 minimum to 0.90 maximum (0.75 nominal)
d	Corner radius/flat	0.6
e	Solder ball pitch	1.27
aaa	Surface coplanarity	0.15 maximum
bbb	Parallel	0.15 maximum
ccc	Top flatness	0.20 maximum
ddd	Seating plane clearance	0.15 minimum to 0.50 maximum (0.33 nominal)
P	Encapsulation height	0.20 minimum to 0.35 maximum (0.30 nominal)
S	Solder ball placement	0.00 maximum

6

Cchip Architecture

This chapter describes the internal architecture for the Cchip.

6.1 Cchip Architecture

The Cchip performs the following functions:

- Accepts requests from the Pchips and the CPUs

- Orders the arriving requests as required

- Selects among the requests to issue controls to the DRAMs

- Issues probes to the CPUs as appropriate to the selected requests

- Translates CPU PIO addresses to PCI and CSR addresses

- Issues commands to the Pchip as appropriate to the selected (PIO or PTP) requests

- Issues responses to the Pchip and CPU as appropriate to the issued requests

- Issues controls to the Dchip as appropriate to the DRAM accesses, and the probe and Pchip responses

- Controls the TIGbus to manage interrupts, and maintains CSRs including those that represent interrupt status

A block diagram of the Cchip is shown in Figure 6–1.

Figure 6–1 Cchip Block Diagram

LJ-05500.AI4
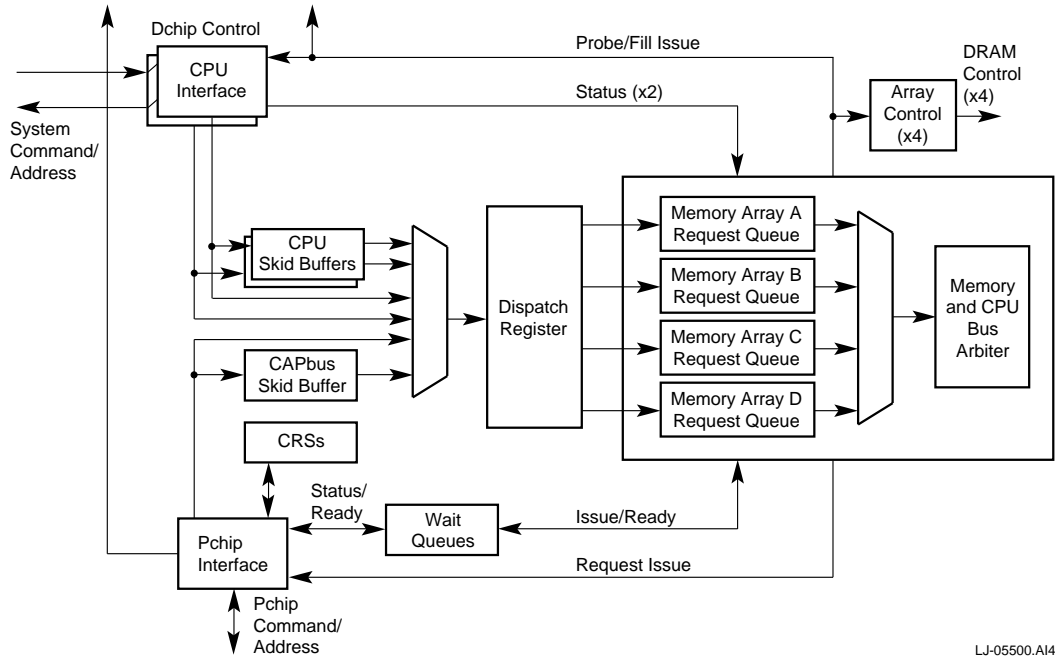
6.1.1 Memory Array Request Queues, Skid Buffers, and Dispatch Register

Each new request that arrives from a CPU or Pchip is eventually dispatched into one of four request queues. Request queues have the following characteristics:

- Each queue corresponds to one of the memory arrays controlled by the Cchip.

- Each queue has six entries.

Even requests that do not require DRAM access are placed in one of these queues. In this case, the selection of the queue is based upon a round-robin algorithm in order to prevent saturation of one array's queue with nonmemory accesses.

A newly arriving request is selected into the dispatch register before it is moved into the appropriate array request queue. In case of simultaneous arrival of requests, or if the dispatch register cannot be unloaded because the appropriate array's request queue is full, requests are temporarily held in skid buffers. Skid buffers have the following characteristics:

- They contain a simple FIFO for arriving requests.

- There are four entries per CPU skid buffer (x 2 CPUs). (x 4 CPUs for Typhoon)

- There are four entries for the CAPbus skid buffer shared by two Pchips. (three entries for Typhoon)

Each 21264 CPU is designed to send, at most, four requests (such as two RdBlk with WrVictimBlk pairs) until it receives an Ack in return. The interface to the Pchips works in a similar fashion; that is, each Pchip sees the other's requests on a common command and address bus.

Although each requestor has a fixed amount of space in its own skid buffer, the requestor receives an Ack as each request is loaded into the dispatch register. Thus, in principle, all of the entries in the array request queues could be filled with requests from a single requestor. Rather, dispatching stops when the dispatch register is destined for an array queue that is already full, even if another request could be dispatched to a non-full array queue.

6.1.2 Request Issuing

To service a request, the Cchip issues it from one to three times. When it issues a request, it initiates various operations that ultimately service the request. To initiate the operations, the issue logic sends various directives (as necessary) to:

- The memory controllers to initiate DRAM accesses

- The CPU interfaces to initiate probes and SysDC command transmissions

- The Pchip interface

- The Dchips to initiate data movement between the CPUs, the Dchip FPQ or TPQ, or memory

- The wait queues to enqueue or dequeue the requests there

For example, a CPU requests a PCI device read operation. It is issued two times. The first issue ultimately causes the read data to be moved from the PCI to the Dchip. The second issue ultimately causes the data to be moved from the Dchip to the CPU.

Typhoon Issue Modes

Because it supports four CPUs, the Typhoon Cchip has a more complex set of issue modes: issue throttling and three CPU alternation modes. The three alternation modes are:

- CPU0 or CPU1 issue alternate with CPU2 or CPU3 issue

- CPU0 or CPU2 issue alternate with CPU1 or CPU3 issue

- CPU0 or CPU3 issue alternate with CPU1 or CPU2 issue

The rules described for the non-Typhoon Cchip that force issue throttling apply also to Typhoon.

If requests exist from only two CPUs, an alternation mode is entered equivalent to the non-Typhoon CPU alternation mode for those two CPUs. Otherwise, the alternation mode used depends on the particular order in which the requests from the CPUs arrived and were retired.

However, if a request exists from one CPU that gets a dirty-probe hit in another CPU, the alternation mode is entered, which allows the simultaneous issue of the probe date extraction and read data fill to the respective CPUs. By contrast, the non-Typhoon Cchip enters throttle mode in this case. If a conflict of required alternation modes occurs – for example, a CPU0 read gets a CPU1 probe-dirty hit while a CPU2 read gets a CPU0 probe-dirty hit – the alternation mode is changed after several cycles to avoid starvation of one of these requests.

6.1.3 Request, Probe, and Data Ordering

The CPU interface logic for each CPU holds a list of requests for which probes have been issued by the request arbiters. The probe queue on each 21264 is eight entries deep. An arbiter cannot issue a request that requires a probe unless there is room in the appropriate probe queue. As the probe results are returned, the count of outstanding probes is decremented, allowing additional requests to issue probes. Using the list of probes issued, the interface updates the corresponding request in the request queue with the probe result. This usually allows the request to proceed to the next phase of its processing.

The 21272 chipset interacts with the 21264 CPU to ensure the ordering rules of Alpha architecture. The Cchip and Pchips interact to ensure the ordering and deadlock avoidance rules of the PCI Specification, Revision 2.1. In this regard, the 21272 performs as a host bridge for CPU PIO and DMA memory accesses. The 21272 also performs as a PCI-to-PCI bridge for PTP operations from one Pchip to the other.

Table 6–1 provides an interpretation of the PCI Specification, Rev 2.1 ordering rules in the context of the 21272 chipset.

Table 6–1 PCI and 21272 Lexicon

PCI Lexicon		21272 Lexicon
PMW	Posted memory write	PIO write request DMA write request PTP write request
DRR	Delayed read request	PIO read request DMA read request PTP read request
DWR	Delayed write request	N/A (all writes are posted)
DRC	Delayed read completion	PIO read data return DMA read data return PTP read data return
DWC	Delayed write completion	N/A (all writes are posted)

Table 6–2 shows the interaction of requests in the request queue. Normally, CPU memory references to different addresses can be issued out of order. The arbiters use this freedom in an attempt to optimize the latency and bandwidth of the system. Memory accesses to the same address are strictly ordered. This is enforced in the array-specific request queue logic because any address matches can occur only within a given array request queue. At the time that a request is loaded, it is compared against all older valid requests, and waits in the event of a match.

DMA requests are ordered with respect to one another. This includes those requests that access different memory addresses, but these requests are pipelined. For example, several DMA read requests can be issued one after another, in arrival order. The data is delivered in order, but the DRAM accesses and issuance of CPU probes can be pipelined.

The rules in Table 6–2 that are not address specific are enforced with the wait queues described in Section 6.1.2. The status of these wait queues is forwarded to the request queues to allow the requests to transition from the Waiting to Ready state.

The one type of access that is not included in this table is scatter-gather table entry (SGTE) fetch requests from the Pchip. As described in Section 6.1.2, these requests can pass all other requests from the Pchip as well as all PIO requests, but not vice versa. These SGTE fetch requests participate in cache coherence. That is, the comments in Table Note #11 (following Table 6–2) with regard to "wait if equal address" apply to the interaction between SGTE fetch requests and all other memory accesses. See the description of scatter-gather translation in Section 10.1.4.3 for more details.

It is necessary to understand the usage of the term *issue*, which refers to different actions depending on the request (in the context of Table 6–2). The following chart helps to interpret the term *issue*:

Request	Issue refers to...
CPU memory	The probes and the DRAM access
DMA memory read	The probes and the DRAM access
DMA memory write	The probe only, but the DRAM location is blocked from other accesses until the data movement takes place
PIO read	Only sending the command to the Pchip
PIO write	Moving the data from the CPU to the TPQ; the data movement takes place when the data is at the head of the TPQ
PTP read	Only sending the command to the target
Pchip PTP write	Moving the data from the FPQ to the TPQ

Table 6–2 Request Wait Conditions

1st request dispatched → 2nd request dispatched ↓	PIO or PTP read	PIO or PTP write	CPU memory (read/write/tag)	DMA memory read	DMA memory write
PIO or PTP read	Wait until issued data return in Pchip order[1]	Wait until issued to Pchip[2]	No wait[3]	Wait until issued data in order[4]	Wait until issued data order from Pchip[5]
PIO or PTP write	No wait [6]	Wait until issued to Pchip[7]	No wait[3]	Wait to issue data in order[8]	Wait to issue data in order[9]
CPU memory (read or write)	No wait[3]	No wait[3]	Wait if equal address and other CPU[10]	Wait if equal address[11]	Wait if equal address[11]
DMA memory read	No wait[4]	Wait to issue data can pass[8]	Wait if equal address[11]	Wait to issue data in order[12]	Wait until issued[13]
DMA memory write	No wait[5]	Wait to issue data in order[9]	Wait if equal address[11]	Wait until issued[13]	Wait to issue data in order[14]

[1] PIO and PTP reads are strictly ordered against each other for issue to the Pchips. The Pchips and any PCI bridges should keep them in order. However, two read requests that are sent to different Pchips complete in whichever order that the data returns.

[2] PIO and PTP reads wait to issue until any prior PIO or PTP writes have been issued (with data) to the target Pchip. The Pchip and bridges are responsible for maintaining the order established by the Cchip. This is required by the PCI Specification, Rev 2.1.

[3] CPU memory accesses are not ordered with respect to PIO or PTP operations. This is because there is no reason for the system to do so. The CPU may just as easily have a cached copy of the memory data. If the program requires such ordering for PIO operations, it must use register dependencies, or issue the MB instruction. If the memory operation is first, the MB ensures that the system has delivered any probes for DMA operations to that cache block before the PIO operation can proceed. If the PIO operation is first, a register dependency ensures that the PIO read has completed. For a PIO write, Alpha architecture requires a subsequent PIO read to ensure that the write has been seen at the device. This is necessary because a DMA read return may pass the write (see Table Note 8).

If DMA is not an issue, an MB after a PIO write can be used to ensure that the write has been initiated, because the CPU waits for the data to be moved before proceeding past the MB. This can be useful for ordering PIO operations between processors, threads or processes in the same processor, CSR operations, and PTP operations.

[4] PIO and PTP reads wait until earlier DMA memory reads issue. Also, PTP reads complete after earlier DMA reads complete (return data to the Pchip). On the other hand, DMA memory reads do not wait for PTP reads to issue or to complete. This is allowed by the PCI Specification, Rev 2.1.

[5] The issue of a DMA write includes the invalidating probes and guarantees that the data will be written to memory before the next access to that memory location by any other request (because of the equal-address waits in the table). A PIO or PTP read that arrives after the DMA write will not issue, and therefore, will not return its data until the DMA write has completed. If a DMA write arrives after a PIO or PTP read has been issued, but before the data for the PIO returns, the DMA data must be written to memory (and the associated invalidate probes issued to the CPUs) before the PIO data is returned to the requestor. If the DMA write arrives after the PIO read data, the DMA write waits until the PIO read data is delivered. (Although nominally not allowed by the PCI Specification, Rev 2.1, there is no deadlock possibility because nothing prevents the DMA write from completing.) In summary, the system keeps the ordering of the returning PIO read data and any DMA write data that occurred on the PCI bus as delivered by the Pchip. This is required by the PCI Specification, Rev 2.1 (except as noted).

[6] PIO and PTP writes may pass PIO and PTP reads. This is allowed by the PCI Specification, Rev 2.1.

[7] PIO and PTP writes are strictly ordered against each other. This is required by the PCI Specification, Rev 2.1.

[8] If a DMA read arrives after a PIO or PTP write, the read waits to issue, but the DMA read data can pass the data for the PIO or PTP write. This apparent violation of the PCI Specification, Rev 2.1, (DRC should not pass a PMW) is required to accommodate devices (such as an ISA bridge), which are not PCI Specification, Rev 2.1 compliant, in that they might not accept a write while waiting for their read to complete. Such a device would lead to deadlock if DMA read data could not pass PIO write data.

If a PIO or PTP write arrives after a DMA read, the write data waits until the data for the DMA read has been delivered to the Pchip. The Pchip and bridges need to reorder this data to prevent deadlock according to the PCI Specification, Rev 2.1.

[9] PIO writes and PTP writes are strictly ordered with respect to DMA writes. The ordering of PTP writes and DMA writes from the same Pchip is required by the PCI Specification, Rev 2.1.

[10] All memory accesses to the same address are ordered. Between CPUs, this is important to cover the interaction of nearly simultaneous SharedToDirty requests, as well as InvalToDirty and WrVictimBlk interactions. Since a given CPU will never have two requests to the same address outstanding at the same time, comparisons of addresses from the same CPU are ignored. In any case, the address comparison is inexact; addresses are considered equal if bits <26:12> and bits <7:6> are equal.

[11] DMA versus CPU ordering on equal addresses ensures, among other things, that DMA quadword RMW operations are atomic. It also ensures that WrVictimBlk requests will wait to be suppressed by invalidating probes, when the request and probe pass on the duplex system address buses. The address comparison is inexact; addresses are considered equal if bits <26:12> and bits <7:6> are equal.

[12] All DMA reads are strictly ordered against each other because DMA devices have no internal MB to enforce ordering. This is allowed but not required by the PCI Specification, Rev 2.1.

[13]DMA reads and writes are issued in arrival order, even if the addresses are not equal. This ordering applies to the issuance of probes only. The actual memory accesses may be reordered if the addresses are not equal so that the impact on DMA read latency is minimized. See the following text for a potential failure of Litmus Test 8 if the probe ordering is not enforced. The fact that DMA reads cannot pass DMA writes from the same Pchip is required by the PCI Specification, Rev 2.1. The fact that DMA writes cannot pass DMA reads from the same Pchip is nominally not allowed by the PCI Specification, Rev 2.1 (to avoid deadlock). Since both operations are guaranteed to complete on the Cchip, there is no deadlock issue to avoid.

[14]All DMA writes are strictly ordered against each other because DMA devices have no internal MB to enforce ordering. This is required by the PCI Specification, Rev 2.1.

The need to strictly order probes for DMA accesses, even reads and writes, to distinct addresses can be shown by the following violation of the impossible sequence of Alpha Architecture Litmus Test 8 between a CPU and a DMA device, assuming the rule for ordering of responses and probes is not followed:

CPU A	Device B	System
Rd X,1		;A has an old copy of X,
	Wr X,2	DMA write to X from B arrives first
	Rd Y,?	DMA read to Y from B arrives second
St Y,2		RdBlkMod to Y from A arrives third DMA read to Y issues first
	Rd Y,1	Probe Y misses, DMA gets memory data RdBlkMod to Y issues second, succeeds
MB		CPU A has not seen the probe for X yet and system does not see the MB
Rd X,1		Violates Litmus Test 8 DMA write to X from B issues third Invalidate X probe sent to A (too late)

The rules of Table 6–2 define the serialization point of the system to be the loading of the request queue for those requests that have wait conditions. In addition, for requests that do not have wait conditions, there is an additional point of serialization defined by the time that the requests are issued. In this context *issue* means both CPU probe issued and memory access issued. Because the Cchip does not receive any indication of an MB instruction executed by the 21264, in order to avoid the impossible sequence of Alpha Architecture Litmus Test 8, the Cchip must also invoke the following rule:

Rule: **Ordering of Responses and Probes:** If Request A from CPU A is issued after Request B from any other source, the response to CPU A for Request A must be issued after any probe for Request B has been issued to CPU A.

The converse is not required. That is, it is permissible for a probe for Request A to be issued before the Request B response is issued to CPU B, even if Request A is issued after Request B. In less formal words it can be said, "probes can pass fills, but fills cannot pass probes."

If this rule is not implemented, the following sequence of events leads to a failure of Litmus Test 8:

CPU A	CPU B	System
Rd Y,1	Rd X,1	
Rd X,1	Rd Y,1	;A has a shared copy of X,
		;B has a shared copy of Y
St Y,2	St X,2	SharedToDirty Y from A issues first
		SharedToDirty X from B issues second
		Invalidate Y probe queued to B
		Invalidate X probe queued to A
		Response SharedToDirty Y Success sent to A
		Response SharedToDirty X Success sent to B
MB	MB	CPUs have not seen the probes yet and system does not see the MB
Rd X,1	Rd Y,1	Violates Litmus Test 8
		Invalidate Y probe sent to B (too late)
		Invalidate X probe sent to A

In addition to the ordering rule for request issue, and the ordering rule for probe delivery, it is also necessary to take into account the ordering of data for PIO and DMA requests. In particular the following rule must be obeyed:

Rule: **Ordering of PIO Read Data:** When PIO read data arrives on the Dchip, it cannot be delivered to the CPU until any previously arrived DMA write requests have been completed (or at least had their probes issued).

If this rule is not obeyed, then a DMA write request (that had been sent from the device prior to the arrival of the PIO read at the device) might not be reflected to the CPU issuing the PIO read. This would render useless the "DMA flushing" purpose of a PIO read. In systems with more than one Pchip, only the DMA writes from the same Pchip need be flushed as dictated by the PCI Specification, Rev 2.1. This is implemented with wait conditions.

Similarly, the PCI Specification, Rev 2.1 producer-consumer paradigm appears to require the following rule, which is violated by the Cchip, in favor of the Alpha architecture mandate to verify the completion of a PIO write by using a subsequent PIO read:

Rule: **Ordering of PIO Write Data:** If any DMA reads have been issued after a given PIO write, the data for the PIO write must be delivered to the Pchip before the data for the DMA read.

If this rule were followed, a CPU issuing a PIO write/MB/memory write sequence could never have the memory update arrive at the device before the PIO write. Because the CPU's knowledge of the ordering of the PIO write and the DMA read is based only on the ordering of the data movement and probe commands, the CPU would not complete the MB in this case until the data movement for the PIO write.

As described in the notes to Table 6–2, this rule is violated in order to avoid a deadlock. A device such as an ISA bridge, which is not PCI Specification, Rev 2.1 compliant, may never allow a write to complete until it has its read data returned. If a program requires the PIO write to complete before updating memory, it should follow the advice given in the Alpha Architecture to issue a subsequent PIO read.

6.1.4 Request Queue Maintenance

The request queue is a unified queue of all requests from the CPUs and the Pchips. In an implementation-dependent manner, the relative ages of any set of entries can be determined. Each queue entry contains the following information:

- Command and other information, such as; CPU MAF/VAF id, number of QW for DMA ops, and PIO mask

- Address

- Phase, Valid

- Status (such as probe results)

- Address match wait vector – A bit vector identifying the older requests in this queue with (nearly) the same address, and for which this request must wait

- Page hit vector – A bit vector identifying the older requests in this queue with the same DRAM page address, so that this request can issue after a previous request without waiting for RAS precharge delay

- Older request vector – A bit vector identifying all older requests in this queue (used to arbitrate among otherwise equal ready requests)

Note: Although there are conceptually 3-bit vectors, they can be combined into a 2-bit vector to represent the following four comparisons against each of the other requests in the array queue (address match wait includes and overrides page hit):

1. Not younger

2. Younger and page hit match
   (but not address match wait)

3. Younger and address match wait

4. Younger and no match

At the time that a request is dispatched into the queue, the following operations take place:

- The command and address are loaded.

- The phase, valid, and status indicators are initialized.

- The command and address are matched against all other valid requests in this array queue.

- Based on the results of the address match, the address match, page hit, and older request vectors are initialized.

6.1.4.1 Request Queue and Data Queue Deadlock Avoidance

Two types of requests, PIO reads and PTP reads, are issued from the request queue, but must wait for data to return from the Pchip before completing. However, in order to receive the returned data, it might be necessary for a DMA write request from the same Pchip to complete first. This can lead to a deadlock if there is no free entry in the appropriate request queue. This could happen if the appropriate request queue is full of PIO read requests waiting for their data.

In addition, when there are devices in the system that are not PCI Specification, Rev 2.1 compliant, DMA read requests from the Pchip may need to complete before any subsequent PIO or PTP operations (reads or writes) can complete. These noncompleted operations may block other PIO or PTP operations from being issued to the Pchip.

The Dchip TPQ, associated with such PIO and PTP operations, holds data for a finite number of requests. In fact, the TPQ is split into the TPQM (for returning DMA data, which can always be delivered) and the TPQP (for PIO write and PTP read or write data, which may not be immediately deliverable).

In particular, the Dchips support:

- A TPQP of four entries in a system with two Dchips
- A TPQP of eight entries in a system with four or eight Dchips

In order to avoid these deadlocks, the following limits are imposed:

Rule: **Limiting PIO Requests:** The total number of PIO requests in the array queues from both CPUs at any time is limited to four.

In addition, the dispatcher never allows a request queue to completely fill with PIO requests. For the CPU, if no more PIO requests can be dispatched, the CPU will not receive an Ack for its PIO (and any subsequent requests), and will stall. For the Pchips, this strategy is not sufficient because a DMA request may become stalled by this strategy, and it is the DMA request that must complete. Therefore, the following rule must be imposed on the Pchips:

Rule: **Limiting PTP Requests:** The total number of PTP requests that may be outstanding from both Pchips at any time is limited to four. The Pchip should force retry on the PCI bus for any requests over this limit.

Due to these limits, in a two-Dchip system (only one Pchip, so no PTP is possible), the TPQP can hold the data for all of the PIO writes in the system that can be issued. In a four- or eight-Dchip system, the TPQP can hold the data for all of the PIO reads (or PTP reads or writes) that can be issued. Once all of the PIO and PTP writes have issued, any DMA reads or writes can issue (the WQI allows DMA operations to pass any unissued PIO or PTP reads) and can also complete (the WQT allows DMA operations to pass all others).

In a system that has PCI-to-PCI bridges on both PCI buses, the PTP limit rule on the Pchips can lead to another deadlock. Consider the case where the bridge tries to send a PTP write to the bridge on the other PCI bus, but is inhibited by the Pchip due to the

preceding rule. At the same time, if the Pchip's downstream queue is filled with reads (possibly from a CPU) targeted at the bridge, the bridge may be unable to respond to the read until its write is serviced because its upstream data buffer is full. In a reciprocal situation, the system will deadlock because the PTP writes can never make progress.

The following rule for the Cchip prevents this deadlock:

Rule: **Limiting PCI Read Requests to the Pchip:** The Cchip should never allow the Pchip's request queue to fill with read requests.

In this sense, the Cchip must obey the PCI Specification, Rev 2.1 ordering rules. That is, it must always allow a PCI write to pass PCI reads. By not filling the Pchip's request queue with PCI reads, it guarantees that it can send a PCI write to the Pchip. If that write is a PTP write to the PCI-to-PCI bridge, the latter bridge (being PCI Specification, Rev 2.1 compliant) must accept the write without requiring its own write to complete. Accepting the write decrements the PTP count on the Cchip, thus breaking the deadlock. If the write sent to the Pchip is any other write to a PCI Specification, Rev 2.1 compliant device, it must also complete. If the write is to a noncompliant device, that device can only delay the write until its DMA request completes, and by the rules of Table 6–2, nothing can block the DMA request completion. (The 21272 does not allow noncompliant devices to perform PTP operations.)

6.1.5 Page Hit DRAM Access

The Cchip uses page-hit DRAM accesses under certain circumstances to improve bandwidth. The Cchip considers two addresses to be in the same DRAM page if, and only if, bits <26:12> of the addresses are equal. Typhoon uses bits <27:12> if the DRAM has 3 blank bits.

6.2 CAPbus Interface

The Cchip and Pchips communicate over a 24-pin open-drain bus called the CAPbus. It is used for command transfers and for Cchip CSR data, since there is no data path from the Cchip to the Dchips.

The CAPbus is implemented with open-drain drivers on the Cchip and Pchips, and pull-up resistors on the module. The command encodings are defined such that when the CAPbus is not actively driven by any chip, a No-op (all ones) command is seen by all receivers.

6.2.1 Power-Up/Reset

At power-up/reset, the Cchip drives **b_cap<1:0>** with the PADbus check-bit width information in the form of the base-configuration setting from **b_td<1:0>**. The Pchip latches this information on the deasserting edge of **i_sysrst_l**. See Section 10.2.2.1 for details of the Cchip system configuration CSR.

6.2.2 CAPbus Protocol

This section describes CAPbus arbitration, data validation, flow control, and the use of byte masks for PTP write operations.

6.2.2.1 CAPbus Arbitration — b_cact*x*_l, i_creq_l<1:0>, b_capsel<1:0>

The CAPbus uses a distributed arbitration scheme. Each device samples the arbitration signals on the rising edge of clock to determine who won arbitration. The arbitration protocol requires a minimum of one turnaround cycle (or idle state) between active bus masters, enforced by each device. The idle state is defined as no bus master actively driving the CAPbus (all devices tristated).

The Cchip and the Pchips use **b_cact*x*_l**, **i_creq_l<0>**, and **i_creq_l<1>** to arbitrate for the CAPbus. During an arbitration cycle:

- If the Cchip asserts **b_cact*x*_l**, it may drive the CAPbus the following cycle.

- If **b_cact*x*_l** is not asserted and only one of the Pchips asserts **i_creq_l**, that Pchip drives the CAPbus the following two cycles and is considered the owner of the CAPbus.

- If **b_cact*x*_l** is not asserted and both Pchips assert **i_creq_l**, the Pchip that was not the most recent owner of the CAPbus drives the CAPbus the following two cycles and is considered the owner of the CAPbus.

An arbitration cycle is defined as being any cycle, except for the two cycles after an arbitration cycle, in which one of the Pchips wins arbitration. This ensures the turnaround cycle after any Pchip drives the CAPbus for two cycles. (The turnaround cycle is the next arbitration cycle.)

The Cchip also asserts **b_cact*x*_l** during any cycle in which it drives the CAPbus. This ensures a turnaround cycle before a Pchip can drive the CAPbus, but allows the Cchip itself to drive multiple times in a row.

The Cchip uses the **b_capsel<0>** and **b_capsel<1>** signals to indicate the target of a command that it drives on the CAPbus. That is, when **b_cact*x*_l** and **b_capsel<0>** are simultaneously asserted, the command must be accepted by Pchip0, and similarly for Pchip1 when **b_cact*x*_l** and **b_capsel<1>** are simultaneously asserted. The signal **b_capsel<*n*>** is only asserted for the first cycle of a multicycle command. This allows the Pchips to decode downstream commands if, and only if, **b_capsel<*n*>** is asserted. Furthermore, each Pchip receives a copy of the other Pchip's **b_capsel** signal. This allows a Pchip to monitor the transmission of a PTP write command from the Cchip to the other Pchip, which is necessary for flow control of PTP operations (see Section 6.2.2.4).

One other special use of **b_cact*x*_l** allows a Pchip to drive the CAPbus without winning an arbitration cycle or becoming the owner of the CAPbus. This occurs either for an RMW operation or for a Cchip CSR write operation.

For RMW, the Cchip first asserts **b_cact*x*_l** and **b_capsel**, and then drives the CAPbus with a Downstream LoadP–RMW command. It holds **b_cact*x*_l** asserted until the target Pchip drives the CAPbus with an Upstream LoadP–RMW command, indicating that the Pchip is returning the merged data. The data flows on the PADbus to the Pchip after the downstream LoadP, and flows from the Pchip after the upstream LoadP.

For Cchip CSR write, the Cchip first asserts **b_cact*x*_l** and **b_capsel**, and then drives the CAPbus with a Cchip CSR write command. The data flows on the PADbus to the Pchip after this command. The Cchip holds **b_cact*x*_l** asserted until the target Pchip drives the CAPbus with an Upstream LoadP–CSR write command, followed by the

CSR write data on the CAPbus in four consecutive cycles (least significant word first). At the conclusion of these cycles, the Cchip may deassert **b_cact*x*_l** (or drive another command itself).

Some examples of the CAPbus arbitration signals are shown in the timing diagram in Figure 6–2. This figure indicates which cycles are arbitration cycles, which chip is the owner of the CAPbus, and which Pchip has priority in case they both assert **i_creq_l<*n*>** in the same arbitration cycle. The owner is not always the entity driving the bus. The least recent Pchip owner (arbitration winner) determines the priority between Pchips for the next arbitration cycle.

At power-up/reset, the priority is determined in each Pchip by examining its Pchip_Num input. Pchip0 has the initial priority. Also, at power-up/reset, **b_cact*x*_l**, **i_creq_l<0>**, and **i_creq_l<1>** are all asserted for one cycle. This allows the Cchip to determine which Pchips are present without a Pchip winning arbitration.

Figure 6–2 CAPbus Arbitration
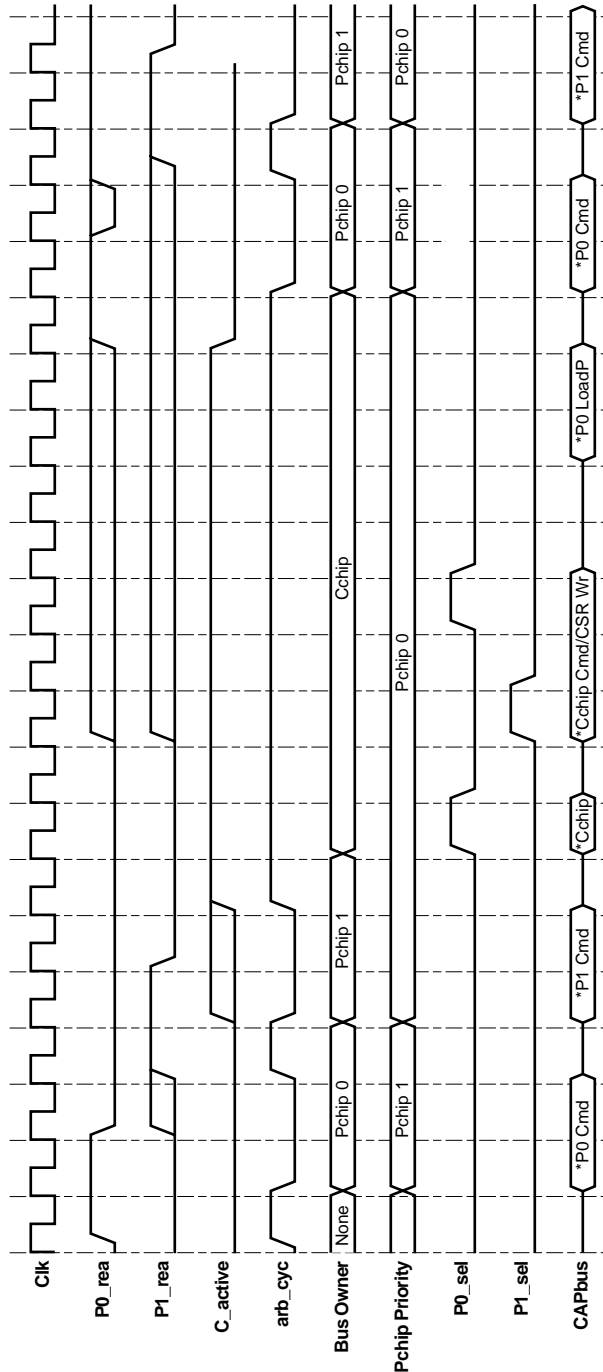
6.2.2.2 Data Validation — b_capgd<1:0>

When the Cchip issues a LoadP command to a Pchip, indicating that it is supplying data in response to an earlier Pchip request, it must validate the data. This is done with the **b_capgd<n>** signal (one per Pchip). This allows the Cchip to speculatively deliver memory data from DRAM while waiting for probe results from the CPUs.

Precisely two cycles after the LoadP command, the Cchip can assert this unidirectional signal to validate the data delivered on the PADbus. If the data is not valid, the Dchips drive the PADbus with one quadword (two PADbus cycles) of junk data. The Cchip may reissue the LoadP command without asserting **b_capgd<*n*>** numerous times. Eventually the Cchip reissues the LoadP command and asserts **b_capgd<*n*>**, and the Dchips supply the correct data. The succession of LoadP commands (without or with **b_capgd<*n*>**) is only constrained by the normal CAPbus and PADbus availability.

If a LoadP command is issued on the CAPbus in cycle *n*, the Good_Data signal is valid in cycle *n*+2. Because the Good_Data signal is needed to determine if the PADbus will continue to be used, the Cchip does not issue another CAPbus command that requires use of the PADbus until cycle *n*+3.

The **b_capgd<*n*>** signal is used for:

- DMA memory read LoadP (speculative memory access)
- SGTE read LoadP (speculative memory access)
- PTP read LoadP (unconditional access to target PCI bus)

The **b_capgd<*n*>** signal is not used for DMA RMW LoadP (no speculative access). If **b_capgd<*n*>** were used with RMW, it would complicate the RMW processing on the Pchip and would also complicate the ownership of the CAPbus. Because RMW operations should be rare, the additional delay, while the Cchip verifies if the data is good before sending the CAPbus command, has negligible performance impact.

6.2.2.3 Flow Control — b_cack, i_pack<1:0>

Signal **b_cack** is a unidirectional signal that informs the Pchips when a Pchip dispatch queue entry has been freed on the Cchip, and when an upstream data buffer has been freed on the Dchip. This signal goes to both Pchips. The signal first synchronizes the Pchips by asserting for one clock cycle on the deasserting edge of **i_sysrst_l**. From then on it transfers the following information on alternating clock cycles:

- Request Queue Ack
- Data Buffer Ack

Signals **i_pack<1:0>** are unidirectional signals (one per Pchip), which inform the Cchip when a downstream data buffer has been freed on the Pchip, and when a Cchip request queue entry has been freed on the Pchip. The signals are first synchronized with the **b_cack** signal, as described in the previous paragraph, and then on alternating clock cycles send Request Queue Ack and Data Buffer Ack.

The Cchip and the Pchips monitor the CAPbus commands, their sources and destinations, and the Ack signals to keep track of the number of data buffers and request queues any given chip has available.

Until the relevant Ack signal is received, the number of requests and data transfers is limited to the following maximum values, although not all transactions count against the limit, as shown in Table 6–3. In the table, the notation +1 means that the transaction increases the count. The notation –1 means that the transaction decreases the count toward the following limits:

- TPR (ToPchipRequests) — Programmable on the Cchip. Set to 4 for the Pchip and applies to each Pchip individually. In addition, to avoid a deadlock that is possible with PTP operations and PCI-to-PCI bridges, the Cchip never allows a Pchip request queue to fill with PCI reads.

- TPD (ToPchipData) — Programmable on the Cchip. Set to 2 for the Pchip. The data transfer can range from one quadword to a full cache block (eight quadwords). The Pchip sends **i_pack<*n*>** signals for multiple small transfers as long as they do not cumulatively consume a full cache block. This applies to each Pchip individually.

- FPR (FromPchipRequests) — Programmable on the Pchip. Set to 4 for the Cchip and applies cumulatively to the requests from both Pchips.

- FPD (FromPchipData) — Programmable on the Pchip. Set to 4 for the Dchip and applies cumulatively to the transfers from both Pchips. For a system with two Dchips (and hence only one Pchip) this is the actual Dchip limit. For a system with four or eight Dchips, there are actually more data buffers available. However, the Pchips are still programmed to the limit of 4 because this allows staggered but overlapping transfers on the two PADbuses. When this limit is reached, the Cchip sends one **b_cack** for each additional nonoverlapped transfer, until the true limit (of 8) is reached.

Table 6–3 lists the flow-control mechanisms used by the Cchip and the Pchips. Typically, data solicited by the Pchip and returned by the Cchip does not count against TPD, but the reverse is not true — data solicited by the Cchip and returned by the Pchip does count against FPD. RMW data does not count in either direction, but for bookkeeping purposes, the initial request is charged with a data transfer (like a DMA write). As for the counts of requests, typically the downstream LoadP commands on the CAPbus do not count against the TPR, but some of the upstream LoadP commands on the CAPbus do count against the FPR.

Table 6–3 Cchip/Pchip Flow Control

Pchip Action	Cchip Action	FPR[1]	FPD[1]	TPR[1]	TPD[1]
Issue DMA N QW Read		+1			
	Signal Req Ack	−1			
	Return LOAD_P_DMA				
	Return Data, Signal Good_Data				
Issue SGTE N QW Read		+1			
	Signal Req Ack	−1			
	Return LOAD_P_SGTE				
	Return Data, Signal Good_Data				
Issue DMA N QW Write		+1			
Send Data			+1		
	Signal Req Ack	−1			
	Signal Data Ack		−1		

Table 6–3 Cchip/Pchip Flow Control (Continued)

Pchip Action	Cchip Action	FPR[1]	FPD[1]	TPR[1]	TPD[1]
Issue DMA RMW QW (no data but increment count)		+1			
			+1		
	Signal Req Ack Return LOAD_P_RMW (down) Return Data	−1			
Return LOAD_P_RMW (up) Return Merged Data	Signal Data Ack				
			−1		
Issue PTP Read		+1			
	Signal Req Ack Return LOAD_P_PTP Return Data, Signal Good_Data	−1			
	Issue PTP or PIO Read			+1	
Return LOAD_P_IO, Status Return Data Signal Req Ack		+1	+1	−1	
	Signal Req Ack Signal Data Ack	−1	−1		
Issue PTP Write Send Data		+1	+1		
	Signal Req Ack Signal Data Ack	−1	−1		
	Issue PTP or PIO Write Send Data			+1	+1
Signal Req Ack Signal Data Ack				−1	−1
	Issue Pchip CSR Read			—[2]	
Return LOAD_P_CSR_Rd Return Data		+1	+1		
	Signal Req Ack Signal Data Ack	−1	−1		
	Issue Pchip CSR Write Send Data			—[2]	
Return LOAD_P_CSR_Rd Return Data (PADbus)	Issue Cchip CSR Read Send Data (CAPbus) Signal Req Ack Signal Data Ack	+1 −1	+1 −1	—[2]	
Return LOADP_CSR_Wr Return Data (CAPbus)	Issue Cchip CSR Write Send Data (PADbus)			—[2]	

[1] The notation +1 means that the transaction increases the count. The notation –1 means that the transaction decreases the count toward the limits stated in the paragraphs preceding this table.

[2] CSR reads and writes do not count against the TPR limit, but be aware of the following:

- Because of limited resources on the Pchip, only one CSR read request may be outstanding to a given Pchip at any time.

- CSR write operations are executed as an atomic CAPbus transaction so that no special restriction is needed. Details of these operations are provided in Section 6.2.2.4 and Section 6.2.2.5.

6.2.2.4 Flow Control — PTP Operations

In addition to the flow control requirements listed in Table 6–3, the Cchip also has a requirement for deadlock avoidance in that the Pchips have no more than four uncompleted PTP operations outstanding in the Cchip at one time. In fact, to avoid overflow of the Dchips TPQP the data must be transferred from the Dchips TPQP for the first request before the fifth request can be made. For PTP read operations, the returning LoadP signifies the completion of the operation. The transfer from TPQP can be assumed to be of maximum length (one cache block) in this case. Each Pchip can snoop the CAPbus for the other Pchip's PTP LoadP.

For PTP write operations, which in the context of the PCI Specification are posted writes, there is no required acknowledgment of completion. Instead, the chipset employs the following strategy. Any PTP write operation originating from a Pchip must eventually result in a PCI memory write operation targeted at the other Pchip. Because both Pchips can observe this request from the Cchip on the CAPbus, they can use it to determine the completion of the PTP write operation (as far as the Cchip is concerned). In fact, this mechanism is the only reason that PCI memory write operations have a separate CAPbus opcode, depending on whether they originate from a CPU or as a PTP (see Table 6–7 for opcodes). The transfer length from TPQP is encoded in the PTP write CAPbus command.

6.2.2.5 Byte Masks — PTP Write Operations

One side effect of the flow control mechanism for PTP operations is that the Pchip must not accept more than one PTP operation from the PCI bus into its request queue, until all previous PTP requests have been sent on the CAPbus. (Otherwise, the Pchip might not be able to send an accepted PTP operation as the PCI ordering rules would block subsequent DMA operations.) However, if a device on the PCI bus turns off some of the byte enables in the middle of a transfer, the Pchip requires one more PCI cycle to stop the transfer. During that extra cycle, still more byte enables may be de-asserted. Combined with the earlier "full" quadwords, the Pchip now has to package the following into one CAPbus request to the Cchip:

- Several full quadwords
- A quadword with some byte enables
- Another quadword with some byte enables

To solve this problem, the chipset implements the following strategy. The CAPbus command to the Cchip for a PTP write operation always specifies a quadword mask that indicates all of the quadwords (full or not) to be transferred on the PADbus. However, before this CAPbus command is sent, a special PTP byte-mask bypass CAPbus operation is sent by the Pchip to the other Pchip. This command contains 16 byte-mask bits corresponding to the byte-enable signals from the PCI bus for the last two quadwords transferred on the PADbus. If only one quadword is sent, only the low-order 8 byte-mask bits are used. The increasing order of byte-mask bits always corresponds to the increasing PCI addressed bytes (see Table 6–8).

The order of these PTP byte-mask bypass commands must match the order of the upstream and downstream PTP memory write commands. This is guaranteed as follows. The Cchip downstream PTP memory write commands follow the order of the upstream PTP memory write commands by design of the Cchip and Dchips (a single

FPQ and TPQ is shared by both Pchips). The upstream PTP memory write commands follow the order of the PTP byte-mask bypass commands because the Pchips maintain a "PTP memory write priority" bit. The priority is assigned to the Pchip that sent the least recent PTP byte-mask bypass without a corresponding PTP memory write. This priority is easy to track because a Pchip *never* sends two PTP byte-mask bypass commands without an intervening PTP memory write command. Even though Pchip0 may lose overall CAPbus priority to Pchip1 between its two commands, Pchip1 cannot send a PTP memory write unless it has already sent a PTP byte-mask bypass. Therefore, the worst case is:

- Pchip0 sends PTP byte-mask bypass for request A and loses CAPbus priority (but gains PTP memory write priority).

- Pchip1 sends PTP byte-mask bypass for request B and loses CAPbus priority.

- Pchip0 sends PTP memory write for request A and loses CAPbus priority (and Pchip1 gains PTP memory write priority).

- Pchip1 sends PTP memory write for request B.

6.2.3 CAPbus Command Encodings

Figure 6–3 shows the format of the 2-cycle CAPbus commands. Table 6–4 lists the encoding of the T field, the number of quadwords for which the PADbus is busy with the transfer, and the address of the first quadword transferred. The mask field denotes which data is valid in the transfer, and is aligned to eight times the data type. That is, for byte transfers, the eight bits represent a total of one quadword. For longword transfers, the eight bits represent four quadwords. For quadword transfers, the eight bits represent up to a full cache block (eight quadwords).

Figure 6–4 shows the format of the 1-cycle CAPbus commands. Table 6–5 lists the C-bit encoding. Table 6–6 lists the encoding of the LDP field. The CSR# is described in Chapter 10.

Figure 6–3 Format of 2-Cycle Commands
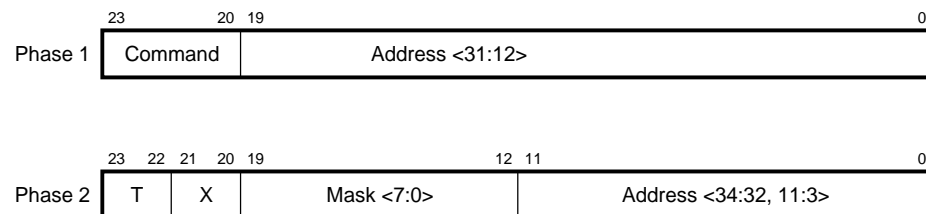
LJ-05508.AI4

Table 6–4 Encoding of T Field T Mask Type PADbus Transfer Characteristics

T	Mask Type	PADbus Transfer Characteristics
00	Byte	One quadword transferred.
01	Longword	Four quadwords are always transferred. The 8-bit mask comprises four longword pairs. The first transferred quadword is specified by **addr<4:3>** and corresponds to the lowest-order nonzero pair of mask bits. If the number of nonzero mask bit pairs is less than four, the trailing quadwords on the PADbus are discarded.
10	Quadword	The number of quadwords transferred is equal to the number of asserted mask bits. The first transferred quadword is specified by **addr<5:3>** and corresponds to the lowest-order asserted mask bit.
11	Illegal	Causes unspecified results.

Figure 6–4 Format of 1-Cycle Commands
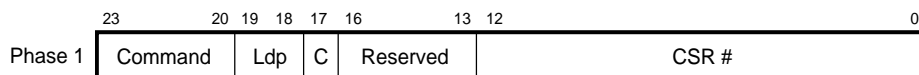
LJ-05509.AI4

Table 6–5 C-Bit Encoding

C-Bit	Meaning
0	Pchip CSR operation
1	Cchip CSR operation

Table 6–6 LDP Encoding

LDP	Meaning for Downstream LoadP	Meaning for Upstream LoadP
00	LoadP DMA read	LoadP PCI read
01	LoadP DMA RMW (to Pchip)	LoadP DMA RMW (from Pchip)
10	LoadP PTP	LoadP CSR read
11	LoadP SGTE read	LoadP CSR write

Table 6–7 lists the commands that the Cchip sends to the Pchips over the CAPbus.

Table 6–7 Cchip-to-Pchip Commands

Code	Command	Cycles	Valid Fields
0000	PCI IACK cycle	2	T, Mask
0001	PCI special cycle	2	T, Mask
0010	PCI IO read	2	T, Mask

Table 6–7 Cchip-to-Pchip Commands (Continued)

Code	Command	Cycles	Valid Fields
0011	PCI IO write	2	T, Mask
0100	Reserved	—	—
0101	PCI memory write, PTP	2	T, Mask
0110	PCI memory read	2	T, Mask
0111	PCI memory write, from CPU	2	T, Mask
1000	CSR read	1 or 5[1]	C-bit, CSR#
1001	CSR write	1 (+1)[1]	C-bit, CSR#
1010	PCI configuration read	2	T, Mask
1011	PCI configuration write	2	T, Mask
1100	Load PADbus data downstream	1	LDP
1101	Reserved (Pchip upstream LoadP)	—	—
1110	Reserved	—	—
1111	No-op	1	—

[1] For details, refer to command descriptions.

Table 6–8 lists the commands that the Pchips send to the Cchip over the CAPbus.

Table 6–8 Pchip-to-Cchip and Pchip-to-Pchip Bypass Commands

Code	Command	Cycles	Valid Fields
0000	DMA read N QW	2	T=10, Mask
0001	Scatter-gather table entry read N QW	2	T=10, Mask
0010	Reserved	—	—
0011	Reserved	—	—
0100	Reserved	—	—
0101	Reserved	—	—
0110	PTP memory read	2	T, Mask
0111	PTP memory write	2	T=10, Mask
1000	DMA RMW QW	2	T=10, only one mask bit set
1001	DMA write N QW	2	T=10, Mask
1010	Reserved	—	—
1011	Reserved	—	—
1100	Reserved (Cchip downstream LoadP)	—	—

Table 6–8 Pchip-to-Cchip and Pchip-to-Pchip Bypass Commands (Continued)

Code	Command	Cycles	Valid Fields
1101	Load PADbus data upstream	2 or 5^1	LDP
1110	PTP write byte-mask bypass	2	See text in Section 6.2.3.2
1111	No-op	1	—

^{1} For details, refer to command descriptions.

6.2.3.1 Cchip-to-Pchip Commands

The following is a detailed description of special-case commands listed in Table 6–7 and Table 6–8.

PIO IACK

The PIO IACK command is issued when the CPU does a PIO read to IACK/special cycle space. It causes an IACK command to be generated on the PCI bus.

PIO Special Cycle

The PIO special cycle command is issued when the CPU does a PIO write to IACK/ special space. It causes a special cycle command to be issued on the PCI bus.

Load PADbus Downstream (To Pchip) — Potential PADbus Conflict

This command is issued when the Cchip has instructed the Dchips to put data on the PADbus for the Pchips. The Cchip always issues this command as a single cycle, asserting **b_cact*x*_l** and one of the **b_capsel** signals simultaneously with the command on the CAPbus. The LDP field indicates whether the data is for a DMA read, an SGTE read, a PTP read, or the first stage of a DMA RMW operation.

In case of DMA RMW, the Cchip holds **b_cact*x*_l** asserted, but disables its drivers on the CAPbus. After at least one turnaround cycle, the target Pchip drives a LoadP RMW upstream command for one cycle when it is ready to return the merged data on the PADbus. Then the Pchip drives a no-op command for one cycle. The cycle after the no-op is an arbitration cycle that serves as a turnaround cycle for the CAPbus.

Because **b_cact*x*_l** is only asserted for one cycle, the target Pchip must resolve the following potential PADbus conflict:

If the Cchip issues a LoadP to Pchip0 on the CAPbus in cycle N, Pchip0 can only decode that LoadP in cycle N+1. In the meantime, Pchip0 may be asserting **i_creq_l** in cycle N+1 for an upstream write operation. If it wins arbitration in cycle N+1 (possible since **b_cact*x*_l** is not asserted in cycle N+1), Pchip0 would normally issue its write command on the CAPbus in cycle N+2. But since the PADbus for Pchip0 will be in use for the LoadP, this would present a conflict. Therefore, if the Pchip decodes a LoadP targeted at itself in cycle N+1, it must convert an upstream write operation to a no-op before driving the CAPbus in cycle N+2. This cannot lead to starvation of a Pchip since the Cchip only sends LoadP operations in response to Pchip requests.

CSR Read

This command is issued when the CPU does a PIO read that the Cchip determines is to a Pchip CSR or to a Cchip CSR (or the TIGbus). If the read is of a Cchip CSR, the Pchip receives the Cchip CSR data (8 bytes, least significant word first) on the next four CAPbus cycles on **b_cap<15:0>**. Otherwise, the Pchip selects its internal CSR data.

In either case, after one (Pchip CSR) or five (Cchip CSR) cycles, normal arbitration resumes. The Cchip may send additional commands to this or the other Pchip, and either Pchip may send commands to the Cchip. However, the Cchip may not send a second CSR read command to the same Pchip until it receives the LoadP CSR upstream command for one cycle from the target Pchip. This happens when that Pchip is ready to drive the CSR data on the PADbus. The Pchip then drives a no-op command for one cycle. The cycle after the no-op is an arbitration cycle, which serves as a turnaround cycle for the CAPbus.

CSR Write — Potential PADbus Conflict

This command is issued when the CPU does a PIO write that the Cchip determines is to a Pchip CSR or to a Cchip CSR (or the TIGbus).

If the CSR write is to a Pchip, the operation completes on the CAPbus after one cycle. However, to avoid a PADbus conflict, the Cchip holds **b_cact*x*_l** for an extra cycle, as denoted by the "+1" in Table 6–7. This extra **b_cact*x*_l** is not a performance issue since Pchip CSR writes are extremely rare.

If the CSR write is to a Cchip, the Cchip holds **b_cact*x*_l** asserted (similar to the downstream LoadP–RMW), but disables its drivers on the CAPbus. When it is ready, the target Pchip drives a LoadP CSR write upstream command for one cycle. In the next four cycles the Pchip drives the data (8 bytes, least significant word first) that it received from the PADbus onto the CAPbus. During those four cycles, it places a no-op in the command field of the CAPbus, and 16 bits of data onto **b_cap<15:0>**.

6.2.3.2 Pchip-to-Cchip Commands (Special Cases)

Scatter Gather Table Entry (SGTE) Read N QW

This command is used when the Pchip gets a TLB miss on a PCI upstream access that requires scatter-gather translation. It accesses memory identical to a DMA read N QW, with its address and mask field specifying the desired quadwords to be read. The difference is that SGTE accesses are not ordered on the Cchip with respect to other accesses, and may need to pass other accesses in order to avoid deadlock. A LoadP command that is specific to SGTE requests is used by the Cchip to return the requested data.

DMA Read/Modify/Write QW

This command is issued when the address of an incoming memory write or memory write and invalidate command lies in either of the following:

* The DMA monster window

* One of the four windows defined by the window base and window mask register pairs that also has any bytes masked out in a quadword

The quadwords that have bytes masked out are issued as different transactions to the Cchip, but are issued in order with the rest of the data phases in that transaction. The T field is set to 10. The mask field has only the appropriate bit set.

LoadP Upstream

As previously described, this command is used for several purposes. For returning PIO read data, or for passing CSR read data, the Pchip arbitrates normally for the CAPbus, and after issuing the LoadP command for one cycle, drives the CAPbus with a no-op for one additional cycle.

For delivering Cchip CSR write data, and for returning merged RMW data, the Pchip does not arbitrate for the CAPbus. Instead it drives the LoadP command in the "shadow" of the **b_cact*x*_l** that is held asserted by the Cchip for that purpose.

PTP Write with Byte-Mask Bypass

As discussed in Section 6.2.2.5, for PTP write operations, the Pchip uses a quadword mask (T=10) to delineate the maximum extent of the data to be transferred on the PAD-bus. (But the T field is not used here.) However, it is necessary to communicate whether the last two quadwords transferred have an associated byte mask, as detected on the originating PCI bus. For this purpose, the Pchip drives the PTP write byte-mask bypass opcode on the CAPbus. The byte mask for the last two quadwords is driven on **b_cap<15:0>** with **b_cap<0>** associated with the lowest addressed byte, and **b_cap<15>** associated with the highest addressed byte. (If only one quadword is transferred, only **b_cap<7:0>** are meaningful.) This occurs in both of the cycles where the Pchip drives the CAPbus. After this CAPbus transaction, which can be ignored by the Cchip, the Pchip rearbitrates for the CAPbus and drives a PTP memory write command to the Cchip.

Because the Cchip maintains PTP memory writes in order, and because a Pchip must always send a bypass before it sends the PTP memory write, the order of the bypass operations is exactly the same as the order of the PTP memory writes arriving from the Cchip. This is true even if the CAPbus priority toggles between Pchips inbetween a bypass and the associated PTP memory write from the same Pchip. See Section 6.2.3 for more details.

6.3 TIGbus and Interrupts

The TIGbus supports miscellaneous system logic such as flash ROM (Figure 6–5) and interrupt inputs (Figure 6–6). The Cchip TIG controller polls interrupts continuously except when a read or write to flash is requested. The 64 possible interrupt inputs are polled eight at a time by selecting a byte with the **b_tia<2:0>** pins, and asserting **b_toe_l** to allow the selected byte to be driven onto **b_td<7:0>**. Using the polled interrupts, the Cchip calculates the **b_irq** values that should be delivered to the CPUs. When any change occurs in these **b_irq** values, the Cchip drives the **b_irq<3:0>** data for both CPUs onto **b_td<7:0>**, and asserts signal **b_tis** to strobe it into a register on the module. If there is no flash read or write outstanding, the polling process is repeated. If there is a flash read or write outstanding, it is serviced between interrupt reads after any pending **b_irq** updates. Thus, the rounds of interrupt polling are not atomic, but the **b_irq** values reflect the most recently polled interrupts. Furthermore, **b_irq<1>** may be artificially suppressed for one full polling loop using the CSR bit MISC<DEVSUP>, as described in Section 6.3.1.

In Typhoon only, the Cchip drives the **b_irq<3:0>** data for CPU3 and CPU2 onto **b_td<7:0>** and asserts b_**tis**<2> to strobe it into a register on the module.

Flash ROM addresses and data move over **b_td<7:0>**. The 24-bit address is sent out least-significant byte first over three address cycles, and captured in a register on the module when signal **b_tas** is asserted. Address bits <23:22> select one of four sets of timing information in the TDR that allows the timing needs of different devices to be met without using a least-common denominator approach. There is only one Cchip select pin for the TIGbus, and it is expected that the high-order address bits will be decoded on the module to determine which device is being addressed. Signals **b_toe_l** and **b_twe_l** control reading and writing of TIGbus devices.

Figure 6–5 TIGbus Flash ROM Control

LJ-05502A-AI4
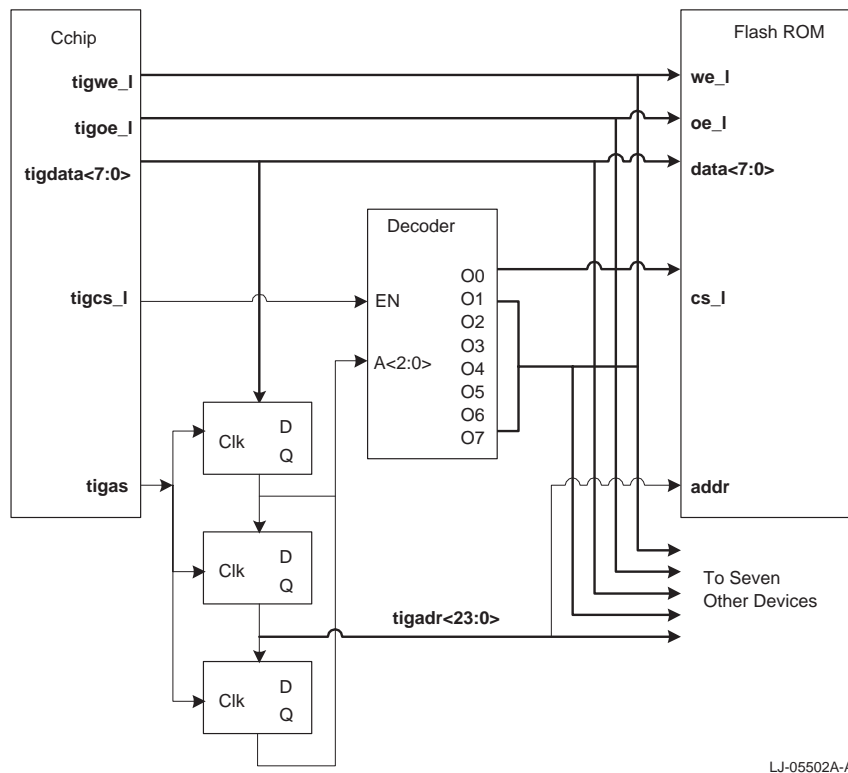
The following CSRs control the timing of interrupt and flash ROM operations:

- TTR – TIGbus timing register (Section 10.2.2.14)
- TDR – TIGbus device timing register (Section 10.2.2.15)

Figure 6–6 TIGbus Interrupt Logic

LJ-05503A-AI4
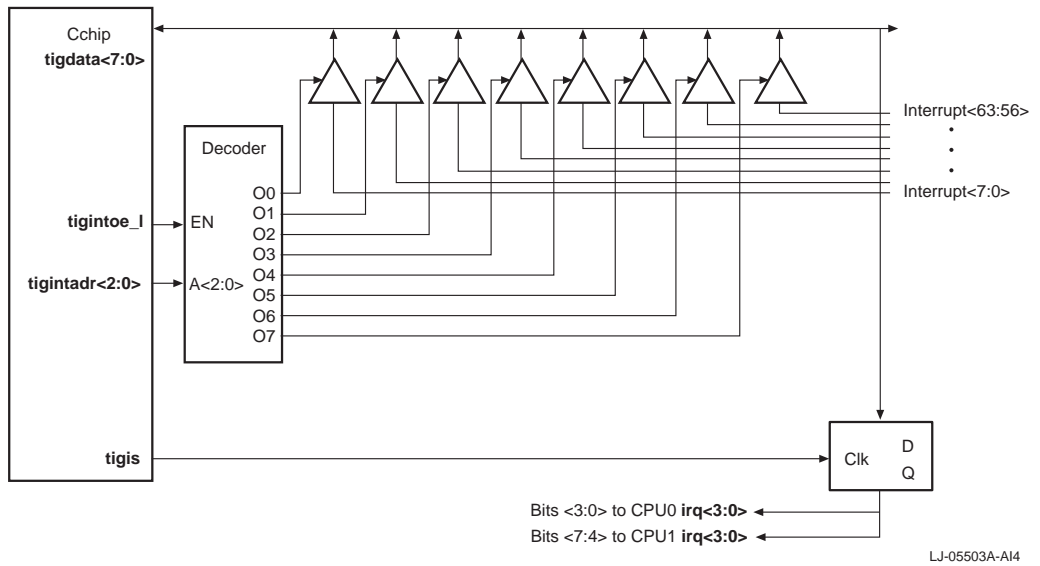
TIGbus timing is shown in Figure 6–7, Figure 6–8, Figure 6–9, and Figure 6–10.

Figure 6–7 Interrupt Timing Parameters
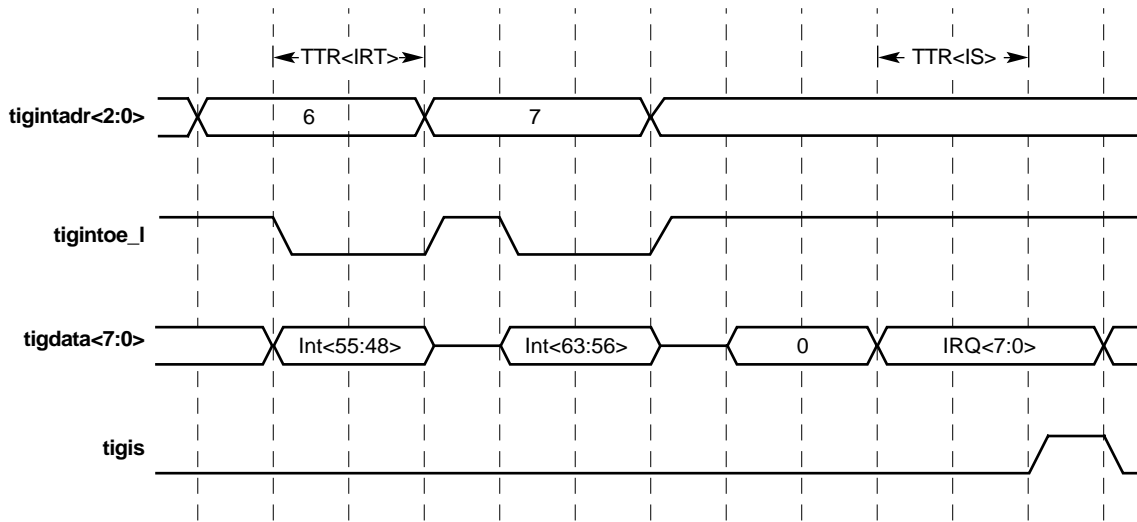
FM-06256.AI4

Figure 6–8 TIG Address Timing Parameters
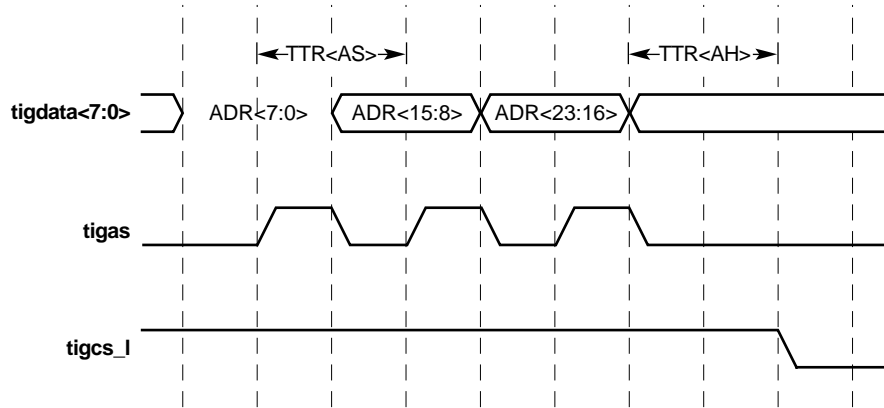
FM-06257.AI4

Figure 6–9 TIG Read Timing Parameters
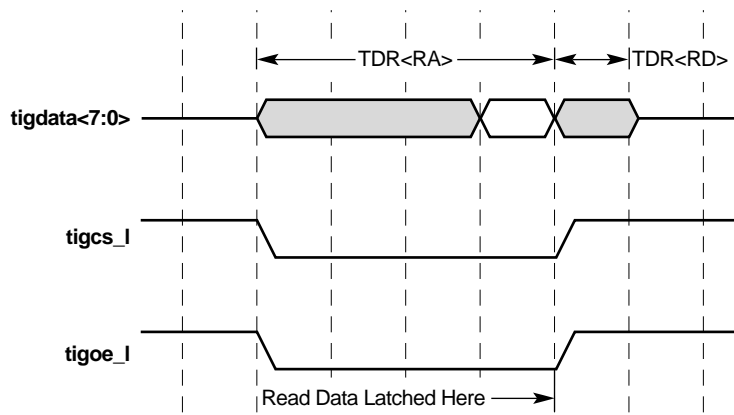
FM-06258.AI4

Figure 6–10 TIG Write Timing Parameters
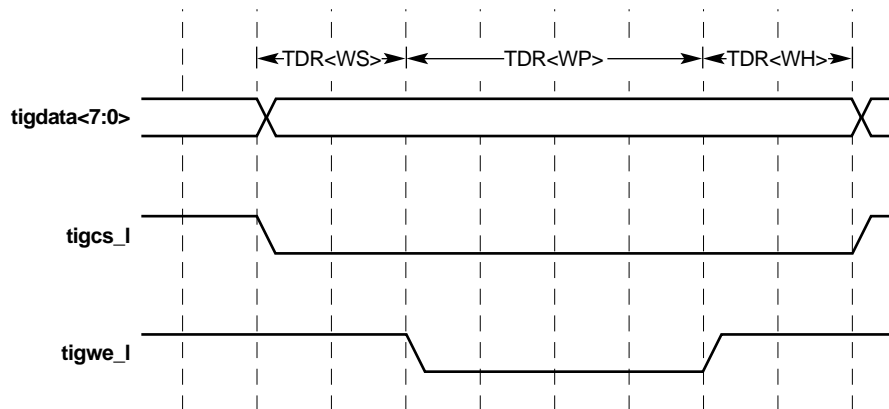
FM-06259.AI4

6.3.1 Device and Error Interrupt Delivery – b_irq<1:0>

As interrupts are read into the Cchip through the TIGbus, the corresponding bits are set in DRIR. These bits are ANDed with the mask bits in DIM*n* and then placed in DIR*n*. If any bits are set in DIR*n*<55:0>, then CPU*n* is interrupted using CPU pin **b_irq<1>**. Interrupt bits <62:58> cause **b_irq<0>** to be asserted and are intended for use as error signals. Assertion of interrupt bits <62:58> causes **b_irq<0>** to be asserted. Interrupt bits <62:61> can be used for Pchip 0 and Pchip 1 errors, respectively. Interrupt bit <63> is special because it is not read from the TIGbus, but is internally generated as the Cchip detected error interrupt (currently used only for NXM requests). Assertion of interrupt bit <63> causes **b_irq<0>** to be asserted. See Chapter 10 for descriptions of the interrupt-related CSRs (DRIR, DIM*n*, DIR*n*, and MISC). A full mask register for each CPU allows software to decide whether to send each of the 64 possible interrupts to either or both CPUs.

After handling all known outstanding interrupts, software may suppress **b_irq<1>** device interrupts to allow the Cchip's polling mechanism to detect the updated (deasserted) value of the interrupt lines from the PCI devices and thereby avoid giving the CPU "stale" interrupts, which require passive release. The field MISC<DEVSUP> is provided for this purpose. When a CPU writes a one to its bit in MISC<DEVSY> the Cchip deasserts **b_irq<1>** to that CPU (regardless of the value in the DIR*n)* until it has completed an entire polling loop. When the Cchip has completed an entire polling loop, **b_irq<1>** will again reflect the value of DIR*n*<55:00>.

Table 6–9 TIG Interrupts and IRQ Lines

TIG Interrupt	Assertion Level	irq<*n*>	Use
63	N/A	**irq<0>**	N/C (internally generated Cchip error) (currently NXM only)
62:58	High	**irq<0>**	Errors (Pchips, and so on) Recommended: • Bit <62> – Pchip0 error • Bit <61> – Pchip1 error
57:56	N/A	N/A	Reserved
55:0	Low	**irq<1>**	PCI devices (level sensitive)

6.3.2 Interval Timer Interrupts – b_irq<2>

The interval timer interrupts the Cchip through a dedicated pin, **i_intim_l**, and is asserted low. When the Cchip sees an asserting (falling) edge of this pin, it asserts MISC<ITINTR> for both CPUs. Pin **b_irq<2>** remains asserted for each CPU <ITINTR>. When the CPU has finished handling the interrupt, it writes a one to its MISC<ITINTR> bit to clear it. Software can suppress interval timer interrupts for *n* cycles by writing *n* into IICn. See Section 6.7 for details about sleep mode (ACPI C3 state).

6.3.3 Interprocessor Interrupts – b_irq<3>

Either CPU can send an interprocessor interrupt to itself, or to the other CPU, by writing MISC<IPREQ> with a mask of the CPUs to be interrupted. Either or both CPUs can be interrupted with a single write to MISC<IPREQ> with the appropriate mask. If an interprocessor interrupt is pending to a CPU, MISC<IPINTR> will be set for that CPU. The interrupt is cleared by writing a 1 to MISC<IPINTR>. The interprocessor interrupts are delivered to the CPUs on **b_irq<3>**. CSC<IPENA> is the enable mask for interprocessor interrupts.

6.4 Monitor Outputs and Counters

The Cchip provides facilities for system monitoring through CSRs, as described in Chapter 10. A CSR on the Cchip allows the selection of two signals from among many chip-internal signals. After a delay of two **i_sysclk** cycles, the selected signals are driven on the chip's external **b_monitor<1:0>** pins.

In addition, the Cchip has a CSR that contains two counters. Each counter is used to count assertions of the associated **b_monitor<*n*>** output signal. For example, Counter0 on the Cchip counts the number of **i_sysclk** cycles during which the Cchip **b_monitor<0>** signal is asserted. Similarly, Counter1 counts the number of **i_sysclk** cycles during which the Cchip **b_monitor<1>** signal is asserted.

For more informatin on monitoring outputs and counters, contact your DIGITAL service representative.

6.5 Cchip Revision

Software can distinguish between the Revision B Cchip and the Revision C Cchip by reading the state of the <PHCW> field of the MTR register after reset. In the Revision B Cchip, this field is initialized to 15; in the Revision C Cchip, this field is initialized to 14.

6.6 Cchip-Detected Errors and Error Reporting

The Cchip reports asynchronous errors to the CPU by asserting **irq<0>** by means of the TIGbus. The sources for asynchronous errors are:

- Interrupt <62:58> asserted active high to the Cchip, and DIM*n*<62:58> asserted
- Nonexistent memory address error, other than for a CPU fill

Module designers should connect the Pchip0 error signal to Interrupt<61> and the Pchip1 error signal to Interrupt<62>. However, the Cchip design does not require this assignment.

6.6.1 Nonexistent Memory Errors

If a CPU requests a fill (using the RdBlk or FetchBlk command) from a nonexistent memory address, the Cchip returns the ReadDataError SysDC command to that CPU, rather than setting an error bit and interrupting both of the CPUs. If either CPU issues any command to a nonexistent memory address other than RdBlk or FetchBlk, it is considered an asynchronous error. If either of the Pchips issues a DMA read or write, or an

SGTE read to a nonexistent memory address, this is also an asynchronous error. For asynchronous errors, the Cchip sets MISC<NXM> and DRIR<63>. The MISC<NXS> field indicates the source of the error. If either CPU has DIMn<63> asserted, the corresponding DIRn<63> will be asserted, and that CPU will be interrupted on **b_irq<0>**. The MISC<NXS> field is locked once MISC<NXM> is set. The interrupt is cleared and the MISC<NXS> field is unlocked by writing a 1 to MISC<NXM>. The failing address is not saved.

6.6.2 Memory Data Errors — CPU Reads and Writes

The 21264 CPU has single-bit ECC error correction and double-bit error detection circuitry. If ECC memory is installed on a system using the 21272 chipset, the ECC bits are passed from main memory to the CPU, and from the CPU to main memory (and from one CPU to the other in case of a probe hit). The ECC bits are treated as data by the Dchips. The Dchips do not detect or correct ECC errors.

6.7 Sleep Mode (ACPI C3 State)

The Cchip supports the 21264 CPU's sleep mode. However, the chipset does not contain any support for reducing its own power consumption. The following sections cover the chipset features that support 21264 sleep mode. For sleep-mode features that affect only the CPU or L2 cache (Bcache), refer to the *DIGITAL 21264 Microprocessor Specification*.

Note: The module must use an edge-triggered interval timer interrupt. That is, interval timer interrupts should be sent as pulses, not levels. While a CPU is "sleeping," it cannot perform a write to clear the interval timer interrupt.

6.7.1 Entering Sleep Mode

The Cchip recognizes that a CPU will enter sleep mode when the CPU reads from the Cchip PRBEN register (Section 10.2.2.9). Prior to reading PRBEN, the CPU must take the following additional steps to ensure proper entry to, and exit from, sleep mode:

1. The CPU makes a decision to enter sleep mode and performs housekeeping activities that require use of the system interface. All dirty cache data is forced out to the system memory and all data in the Bcache is invalidated.

2. The CPU writes the number of interval timer interrupts that it wants to ignore to the Cchip IIC register that corresponds to its CPU ID.

3. The CPU reads the Cchip PRBEN register, signaling that the system interface may be shut down upon completion of this read.

4. The Cchip issue unit informs the sleep mode logic that the read data from the PRBEN has been returned to the requesting CPU and that any probes pending to the CPU, at the time PRBEN was deasserted, have been delivered. The sleep mode logic then requests a toggle clock forward reset for that CPU, which the issue unit places into the next available idle slot. The delay incurred in this handshake is sufficient to guarantee enough clock forward clocks for the CPU to absorb the PRBEN read data.

5. The CPU receives the dummy read data for PRBEN CSR read. The firmware then executes an MB (memory barrier) instruction to ensure that all of the pending probes are processed and responses are sent to the Cchip so that the previous step can complete. The CPU can then slow down its internal clocks to reduce power consumption.

6. When the issue unit sends the command on the CPM bus, it also notifies the CSR section that the command has been sent.

7. One cycle later, the CSR section asserts the **b_cfrst<1:0>** signals that go to the CPU.

8. The Dchips receive and decode the toggle clock forward reset command and assert the clock forward reset signal onto the interface.

9. During this time, the Cchip CSR section waits a number of cycles equivalent to the Dchip decode path, and then asserts the **b_cfrst<1:0>** signals onto the Cchip's clock forward interface.

10. On both the Cchip and the Dchips, the clock forward interfaces are reset, clock transmission to the CPU stops, and the incoming clocks are ignored.

Note: DMA operations and operations from another (nonsleeping) CPU continue normally. The sleeping CPU is not probed for cache coherency, but according to the algorithm, it has no dirty cache blocks.

6.7.2 Exiting Sleep Mode

Once the Cchip IIC register has been programmed, the Cchip begins to decrement the counter for each interval timer interrupt seen on the **i_initim_l** INPUT. When this counter reaches 0, or if a device interrupt must be serviced, the walk-up process begins. The counter continues to decrement and sets the overflow bit to indicate if the counter goes beyond 0.

The Cchip sends either the interval timer interrupt or the device interrupt to the CPU to begin the walk-up process. The Cchip then waits for the CPU to indicate, by asserting **b_sromoe_l**, that it is ready to resume operation. Upon detecting the assertion of **b_sromoe_l**, the Cchip sends a two-SYSCLK cycle pulse to the sleeping CPUs on the CFRST lines.

When the Cchip sends the brief pulse on **b_cfrst<1:0>** to start the CPU SROM load sequence, the Cchip begins counting down the power-up clock forward reset timer value in the WDR (initialized to $2^{18}-1$, or 262,143 **i_sysclk** cycles). This timer value allows 1.5 million cycles for the CPU to perform BiSt, repair, init, and to load the Cbox configuration data from the SROM. This value is based upon the minimum CPU/system clock ratio of 6:1. Higher clock ratios leave spare time at the end of the CPU initialization sequence.

1. When the CPU deasserts **b_sromoe_l**, the Cchip deasserts the clock forward interface reset, following the sequence described in Section 12.1.2. The CPU then writes to the PRBEN CSR indicating that it is ready to participate in the 21272 cache coherence protocol. When the clock forward reset logic detects the deasserting edge of **b_sromoe_l**, it notifies the Cchip issue unit, which issues a toggle clock forward reset command.

2. The Cchip issue unit waits for an idle cycle to insert the toggle clock forward reset command onto the CPM bus to the Dchip. When the issue unit sends the command on the CPM bus, it also notifies the CSR section that the command has been sent.

3. One cycle later, the CSR section deasserts the **b_cfrst<1:0>** signals to the CPU.

4. The Dchip receives and decodes the toggle clock forward reset command, and deasserts the clock forward reset signal onto the clock forward interface.

6.7.3   Sleep Mode in Multiprocessing Systems

The Cchip supports multiple CPUs in independent sleep modes. Each CPU is assigned its own IIC register to define the length of the sleep period, and its own **b_sromoe_l** and **b_cfrst<1:0>** signals to perform handshaking during the wake-up sequence.

Because each CPU has its own interrupt mask register DIMx, no special handling of interrupts is required based on the independent sleep modes of the processors. The Cchip has only one IIC register. If multiple CPUs are asleep when the timer finishes its count, all sleeping CPUs will be awoken at approximately the same time. In a multiprocessor system, a CPU that wishes to go to sleep should determine whether another CPU is already asleep. This task must be managed by system software. In essence, because the Cchip has only one IIC register, writing this CSR while one CPU is already asleep is not advised.

7

Dchip Architecture

This chapter describes the internal architecture for the Dchip.

7.1 Dchip Architecture

The Dchip performs the following functions:

- Implements data flow between the Pchips, CPUs, and memory

- Shifts data to and from the PADbus, as required

- Provides Pchip queue buffering

- Provides memory data buffering

- Implements data merging for quadword write operations to memory and the DMA RMW command

Dchip architecture *does not* implement:

- Flow control

- Error detection

- Error reporting

- Error correction

- Data wrapping

The Dchip uses multiplexers to switch data among its ports and queues. In addition to moving data from one port to another, these multiplexers must support the various system configurations. The system may have two, four, or eight Dchips. This allows for one or two 21264 CPUs, one or two Pchip ports, and one or two 16-byte or 32-byte memory buses. Data may be moved between the CPU, Pchips, or memory ports. Also, data may be transferred between the two CPU ports. PTP transfers are supported between Pchip ports.

Figure 7–1 shows the Dchip internal architecture.

Figure 7–1 Dchip Block Diagram

LJ-05497.AI4
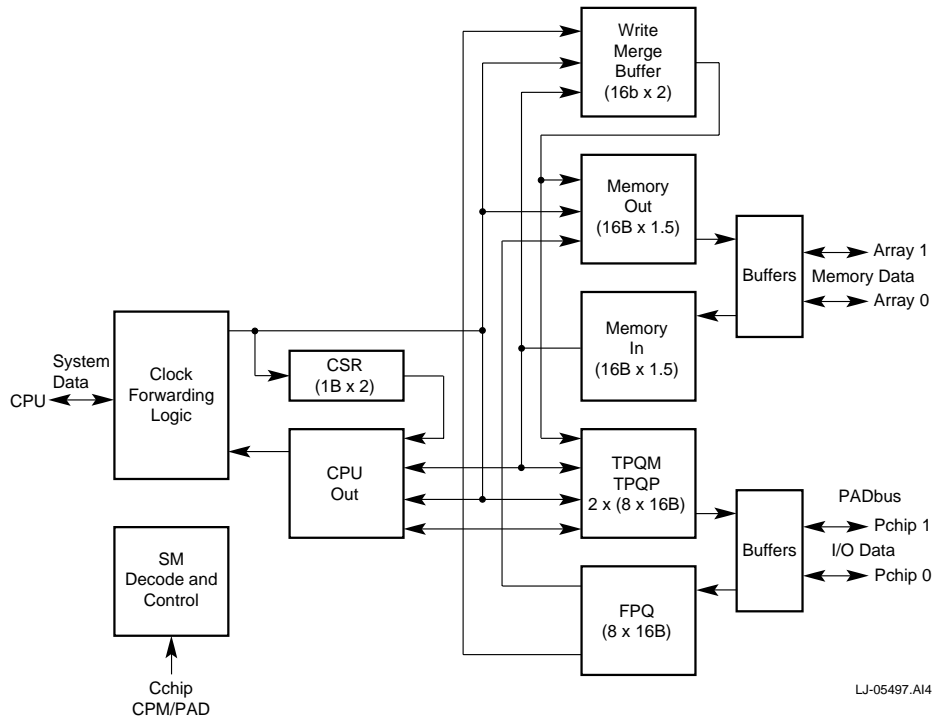
7.2 PADbus Interface

The PADbus is a bidirectional data connection between a Pchip and the Dchips. It comprises a 4-byte data path and 8 check bits. There are two PADbus paths on the Dchips; one for each of two Pchips. Both the Cchip and the Pchip monitor the CAPbus to determine if the operation generates an associated PADbus operation, and if so, for how many cycles. There is a fixed relationship between the start of a CAPbus command and the start of any associated PADbus transfer. If the first cycle of the CAPbus command is driven in cycle N, the first PADbus data is driven in cycle N+4. If a command does not require a PADbus transfer, it can be issued on the CAPbus in the "shadow" of a previous CAPbus command's PADbus transfer.

The PADbus has the following two modes of operation:

- 4-byte mode
- 8-nibble mode

In 4-byte mode, a complete longword (4 bytes) and the 4 associated check bits are transferred each cycle, least significant longword (of the enclosing quadword) first. In 8-nibble mode, 8 nibbles and the 8 check bits associated with the quadword are transferred each cycle, least significant nibble of each byte first, with the 8 check bits repeated during the transfer of the most significant nibbles. These modes arise from the need for having the Pchip-to-Dchip interconnect rearranged depending on the number of Pchips and Dchips in a system. For more information, refer to the configurations in Section 2.2.1 through Section 2.2.3.

During normal system operation, if a Pchip is installed, it is the default driver for its PADbus. However, during reset, while the system is determining the configuration (number of Dchips and whether a second Pchip is installed), the Dchip is the default PADbus driver.

7.3 Dchip Control

The Dchip is controlled by PADbus commands and the CPM commands. The PADbus commands control data movement between the TPQ or the FPQ and the appropriate PADbus. In the special case of the DMA RMW command, the PADbus command controls data movement between the PADbus and the write merge buffer. The PADbus commands are listed in Section 7.3.1.

The CPM commands control all other data transfers. This includes transfers between CPUs, memory, the write merge buffer, the FPQ, and the TPQ. The transfers use the FPQ as a source or the TPQ as a destination only. This interface controls transfers from the FPQ to the TPQ (for PTP operations). These commands are generated by the central bus arbitration logic on the Cchip. The CPM commands are listed in Section 7.3.2.

The Dchip supports several possible system configurations. The various modes that are supported by the Dchip are listed in Chapter 2. The modes are stored in the Dchip system configuration register (DSC) at power-on/reset. The Dchip also supports variable timing parameters for the CPM commands, and 16-byte or 32-byte memory bus widths. This information is stored in the system timing register (STR) during initialization, after system software has determined the parameters for the installed DIMMs or SIMMs. The 32-byte buses can be half-populated (connected to 16-byte arrays in the lower-order bits). Such buses should be programmed as 16-byte buses in the STR CSR (Section 10.2.4.3).

The CPM commands are decoded in the SM block. The SM block creates delayed versions of the commands, and selects one of them for decode according to the values stored in STR. The SM block uses the command decode and the configuration information stored in DSC to generate control signals for the other logical subblocks.

Note: The following bus configurations *are not* supported:

- Mixed-size memory buses
- Half-populated 32-byte buses with components in the upper 16 bytes

The following sections describe the control interface from the Cchip to the Dchips.

7.3.1 Dchip-PADbus Interface Control — PAD Commands

The Cchip issues PADbus commands to the Dchips to control the movement of data between the Pchips and the Dchips. Data from a Pchip is loaded into the FPQ, and data to the Pchips is unloaded from the TPQ. The two-phase command encoding is shown in Table 7–1.

Table 7–1 PADbus Command Format

Cycle 1	V	C	C	T	P
Cycle 2	S1	S0	Length		

The V bit typically indicates that the command is valid. The T field typically indicates that data movement is to the Pchip. The P field indicates whether Pchip0 or Pchip1 is involved in the transaction.

The full VCCT field is interpreted as outlined in Table 7–2.

Table 7–2 PADbus Command Encodings

VCCT	Mnemonic	Command
0xxx	—	No-op
1000	P–FPQ	Move data from the Pchip to the Dchips
1001	TPQM–P	Move data to the Pchip from the Dchip's TPQM
1010	P–WMB	Return data from Pchip to Dchips for RMW
1011	WMB–P	Move data from Dchips to Pchip for RMW
1100	PP–FPQ	Stutter move of data from the Pchip to the Dchips
1101	TPQP–P	Move data to the Pchip from the Dchip's TPQP
111x	—	Reserved

The special "stutter" command is used for PIO read byte and PIO read longword operations from a CPU. In these cases, the transfer to the CPU must have each quadword sent twice in succession. To accomplish this, each quadword from the Pchip is written into two successive locations in the FPQ when the PP–FPQ command is received. Then, a normal CPM command is used to transfer the data from the FPQ to the CPU.

The S<1:0> bits represent a shift amount, in quadwords, from zero to three. The three length bits indicate the length of the transfer in quadwords, modulo 8. Table 7–3 lists the PAD command length field encoding.

Table 7–3 Length Field in PAD Commands

Length	Meaning
000	Eight quadwords
001	One quadword
010	Two quadwords
011	Three quadwords
100	Four quadwords
101	Five quadwords
110	Six quadwords
111	Seven quadwords

The Dchip stores the shift amount and length of valid data for each FPQ queue entry. It uses this knowledge when merging quadwords of data from the FPQ into cache blocks in memory, and when transferring data to a CPU.

For transfers out of the TPQ to the Pchip, the Dchip uses the shift amount and length to transfer only the quadwords containing valid data to the Pchip.

Because the shift amount has a maximum value of three, shifting of four to seven quadwords is accomplished by wrapping the data by four quadwords (one-half cache block) at either the memory or the CPU. Table 7–4 lists the PADbus command shift and length field restrictions. For more details on the shift amount, refer to Section 7.3.3.

Table 7–4 PADbus Command Shift and Length Fields Restrictions

Command	Restriction
P–WMB	Len = 001
WMB–P	Len = 001
PP–FPQ	Len = 001 or 100
Others	Shift + Length ≤ 8
Special case (see the following text)	Shift = 11, Length = 111

There is one special case where the Cchip can send the Dchip an otherwise illegal combination of S<1:0> and length fields. This is used where speculative memory data is sent to the Pchip. If the Cchip determines that it cannot assert **b_capgd<*n*>** for the transfer (see Section 6.2.2.2), it sends the TPQM–P PAD command with a shift amount of 3 and a length of 7. In this case, the Dchip should drive the PADbus for two cycles (one quadword), but should not increment its TPQM pointer. This is because the Cchip will eventually send an h2po CPM command (see Table 7–5) to overwrite the oldest TPQM contents.

7.3.1.1 PAD Command and PADbus Timing

The relationship between the cycle of the PAD command from the Cchip to the Dchip, and the PADbus data transfer between the Pchips and Dchips, depends on the direction of the PADbus transfer. As described in Section 7.2, if the CAPbus command (first cycle) is driven in cycle N, any associated data is driven on the PADbus in cycle N+4.

For upstream transfers (Pchip to Dchip):

- The CAPbus command (first cycle) is driven by the Pchip in cycle N.

- The PAD command (first cycle) is driven by the Cchip in cycle N+2.

- The PADbus data (first cycle) is driven by the Pchip in cycle N+4.

For downstream transfers (Dchip to Pchip):

- The CAPbus command (first cycle) is driven by the Cchip in cycle N.

- The PAD command (first cycle) is driven by the Cchip in cycle N+1.

- The PADbus data (first cycle) is driven by the Dchip in cycle N+4.

7.3.2 CPU Bus, *x*PQ, and Memory Bus Controls — CPM Commands

The central bus arbiter controls transfers between the CPUs, the TPQ and FPQ, and the memory buses. The CPM commands consist of a 5-bit opcode and a 3-bit extension. Table 7–5 lists these commands and gives a brief description of each.

In Table 7–5, the columns labeled "Source Data Timing" and "Destination Data Timing" show the cycle of the first data on the external source and destination buses (that is; the memory bus, CPU bus, or PADbus), relative to the cycle when the command is driven from the Cchip to the Dchip. For example, if the source timing is "3" and the CPM command is on the wires from the Cchip to the Dchip in cycle N, the first source data will be on the wires to the Dchip in cycle N+3.

The notations "iddr" and "iddw" refer to the values (in sysclk cycles) specified in the system configuration CSR (on either the Cchip or Dchip). The notation "iddr–2,4" means that the value iddr–2 is used for 32-byte memory arrays, and the value iddr–4 is used for 16-byte memory arrays. Furthermore, for commands that transfer data to or from the FPQ and TPQ, Table 7–5 indicates the minimum additional time for transfer to or from the PADbus. This information allows proper scheduling of associated CPM and PAD commands as follows:

- For transfers to the PADbus, which require a CPM command *2p and a PAD command TPQ*–P, the sum of the source and destination timings in the table indicates the minimum number of cycles, after the CPM command is driven by the Cchip, that the data can be driven by the Dchip on the PADbus. (For m2p, the minimum delay is always iddr+drtp–2, regardless of whether 16-byte or 32-byte memory arrays are used.)

- For transfers to the PADbus from the WMB, which require a CPM command *2w and a PAD command WMB–P, the sum of the source and destination timings in the table indicates the minimum number of cycles after the CPM command is driven by the Cchip that the data can be driven by the Dchip on the PADbus. (The destination timings are shown in parentheses because the data in the WMB is only sent to the PADbus for RMW operations.)

- For transfers from the PADbus, which require a PAD command P*–FPQ and a CPM command p2*, the sum of the source and destination timings in the table indicates the minimum number of cycles after the final piece of data is driven by the Pchip on the PADbus that the data can be driven by the Dchip on the destination bus.

The notations "dwfp", "dwtp", and "drtp" refer to the values specified in the system configuration CSR on the Cchip (which describe the fixed minimum delays on the Dchip through the TPQ and FPQ).

Table 7–5 CPM Commands and Timing of Data Transfer

Opcode	Extension	Mnemonic	Source Data Timing	Destination Data Timing	Dchip Operation
00000	xxx	nop	—	—	No-op
00001	xxx	rsv	—	—	—
00010	mcc	m2c	iddr–2,4	iddr–0,2	– Mem bus m to CPU bus cc – Pass-through timing
00011	mcc	m2ca	iddr–2,4	iddr	– Mem bus m to CPU bus cc – Accumulate timing
0010x	xxx	rsv	—	—	—
0011x	mcc	c2m	iddw–2	iddw	CPU bus cc to mem bus m
01000	mxx	m2p	iddr–2,4	+drtp+0,2	Mem bus m to TPQ
01001	mxx	pw2m	dwfp+	iddw	FPQ to mem bus m and merge with WMB
0101x	xxx	rsv	—	—	—
0110x	xcc	c2p	iddw–2	+dwtp	CPU bus cc to TPQ
01110	xcc	h2po	iddw–2	+dwtp	– Cache hit on CPU cc to TPQ – Load cache data at top of TPQ and overwrite stale memory data
01111	xcc	h2pb[1]	iddw–2	+dwtp	– Late cache hit on CPU c to TPQ – Decrement top of TPQ pointer, then load late cache data at top of TPQ and overwrite stale memory data
1000d	dcc	c2dp	iddw–2	—	– CPU bus cc to TPQ – CPU bus cc to Dchip csr dd
1001d	dcc	dp2c	—	iddw	Dchip csr dd to CPU bus cc and discard 1 from FPQ
10100	xcc	p2c	dwfp+	iddw	FPQ to CPU bus cc
10101	xxx	rsv	—	—	—
1011x	xxx	p2p	dwfp+	iddw	FPQ to TPQ
110bb	xaa	h2c	iddw–2	iddw	CPU bus aa to CPU bus bb – used for cache hit
11100	xxx	rsv	—	—	—

Table 7–5 CPM Commands and Timing of Data Transfer (Continued)

Opcode	Extension	Mnemonic	Source Data Timing	Destination Data Timing	Dchip Operation
11101	mxx	m2w	iddr–2,4	(+4)	Mem bus m to WMB
11110	mxx	w2m	—	iddw	WMB to mem bus m
11111	xcc	h2w	iddw–2	(+4)	Cache hit on CPU cc to WMB

[1] The Cchip cannot issue this command if the TPQ is full without overwriting the tail of the queue.

The 2-bit CPU designator subfield "cc" (sometimes "aa" or "bb" in Table 7–5) in the CPU designator subfield in the CPM command, has the following meaning depending on the number of Dchips in the configuration:

- Two-Dchip configuration – No meaning; only one CPU supported and must be "00".

- Four-Dchip configuration – Meaning is "xc" and must be "00" or "01", that is, the most significant bit must be zero. The least significant bit indicates one of the two possible CPUs in this configuration.

- Eight-Dchip configuration – The two bits must be fully specified to indicate one of the four possible CPUs in this configuration.

Although the Dchip is capable of overlapping many of the data transfers initiated by these commands, there are some resources within the Dchip that are made busy for more than one cycle. Therefore, it is necessary to restrict the issuance of multiple commands in consecutive cycles. The following rules govern the issuance of CPM commands to the Dchip:

- No commands may be issued that would create a conflict on a memory bus (taking into account the width of the memory bus). Overlapped operations to different memory buses are permitted in any combination.

- No commands may be issued that would create a conflict on a CPU bus. Overlapped operations to different CPU buses are permitted in any combination.

- Consecutive commands, which write the TPQP or TPQM (for instance; m2p, c2p, c2dp, h2po, h2pb, p2p), must be separated by at least one cycle.

- Consecutive commands, which read from the FPQ (for instance; pw2m, p2c, dp2c, p2p), must be separated by at least one cycle.

In addition to these rules, the Cchip is responsible for avoiding overflow and underflow of the FPQ, TPQP, and TPQM. Underflow is governed by the parameters "dwfp", "dwtp", and "drtp" described previously. After loading data in a queue, these parameters specify the earliest that such data can be unloaded. Overflow is governed by observing the following:

- A TPQM or TPQP entry is free (can be written with new data) in the cycle that its last quadword is driven on the PADbus (first cycle of the last quadword on the PADbus).

- An FPQ entry is free (can be written with new data) iddw + 2 cycles after the CPM command is driven by the Cchip.

7.3.3 Data Shifting in the Dchips

As previously described, the PAD command from the Cchip to the Dchip has a 2-bit shift field that controls the shifting of data as it is moved between the PADbus and the FPQ or TPQ on the Dchips. The following sections describe the use of this control for various types of operations.

7.3.3.1 Shifting for Pchip Memory Operations

The Pchip memory operations are:

- Read N quadwords

- Write N quadwords

- Read-modify-write one quadword

Data transfers to and from memory are always one cache block long, either in two 32-byte cycles or four 16-byte cycles. Transfers to and from the Pchips are N quadwords, where $1 \leq N \leq 8$. If the desired quadwords in a Pchip-to-memory transaction are not aligned to the cache block boundaries, the data will be shifted. The Dchips use a combination of shift bits (ss) and length bits (nnn) to select the appropriate quadwords for processing. The shift bits are sourced directly from bits <4:3> of the data's system address.

There are three classifications of quadword alignment that pertain to the Dchips. An example of each is shown in Figure 7–2.

Figure 7–2 DMA Data Alignment: An Example of Each Possible Alignment

Case 1: The transaction data is contained in the first 32 Bytes. ss = 01, nnn =

| QW | | QW | | QW | | QW | | QW | | QW | | QW | | QW |

Case 2: The transaction data spans both halves of the cache block. ss = 10, nnn =

| QW | | QW | | QW | | QW | | QW | | QW | | QW | | QW |

Case 3: The transaction data is contained in the second 32 Bytes. ss = 01, nnn = 010

| QW | | QW | | QW | | QW | | QW | | QW | | QW | | QW |

LJ-05510A.FH8

Cases 1 and 3 look the same to the Dchips, because the Cchip simply reverses the order of the half-cache blocks if the desired data is entirely contained in the last half of the block. This is done by using address bit <5>, which swaps the two half-cache blocks to access either the memory or the CPU cache.

Shifting on a Memory Read N Quadwords

Data for a memory read either comes from the memory (in the m2p command) or from the cache of another CPU (in the h2pb and h2po commands).

The Dchip loads the entry in the TPQM. When the data is retrieved from the TPQM, the Cchip selects only the data that was requested by the Pchip. It does this by using the shift and length bits in the TPQM–P operation. The Dchip uses the shift information to

move the first requested QW to the bottom of the shift register. The Dchip then uses the length information to shift the requested number of quadwords out to the Pchip. Figure 7–3 illustrates this operation.

Figure 7–3 Data Shifting in a DMA Read

Read data in memory or cache.

| QW7 | QW6 | QW5 | QW4 | QW3 | QW2 | QW1 | QW0 |

Data in the TPQ output shift register after shifted by the Dchip. ss = 10, nnn = 100

| XX | XX | XX | XX | QW5 | QW4 | QW3 | QW2 |
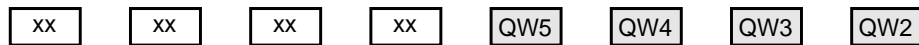
LJ-05511.AI4

Shifting on a Memory Write N Quadwords

A write of N quadwords is merged with a block of data in the WMB, unless N = 2, 4, 6, or 8, and the data is octaword aligned. The Dchips (under the control of the Cchip) first bring the relevant cache block into the WMB, either from memory or cache. The N quadwords of data to be written will be resident in the FPQ. When the pw2m command is issued, a combination of the data in the WMB, and the N quadwords in the FPQ, will be written to memory. The combination is controlled as follows:

1. The first ss quadwords are written from the WMB.

2. The next nnn quadwords are written from the FPQ, where nnn is the length bits of valid FPQ data.

3. The final quadwords, if eight have not yet been written, are loaded from the WMB and aligned so that the last quadword written is the last quadword in the WMB.

In the case where N = 8, there is no need to merge data in the Dchips because an entire cache block is being written with Pchip data, and any CPU cache data will be invalidated. Thus, the preliminary step of loading the WMB is skipped by the Cchip. The pw2m command still functions as described previously.

In the case where N = 2, 4, or 6, and the data is octaword aligned, the Cchip algorithm depends on whether a CPU is found to have the data in Dirty state in its cache. If there is dirty cache data, the algorithm proceeds exactly as for the unaligned case. If there is no dirty cache data, the algorithm is the same as N = 8, and the Cchip only causes the DRAMs to be written for the proper octawords. Figure 7–4 illustrates this operation.

Figure 7–4 Data Shifting in a DMA Write

Data fetched from memory or cache in the WMB.

| QW7 | QW6 | QW5 | QW4 | QW3 | QW2 | QW1 | QW0 |

Data in FPQ, ss = 11, nnn = 100.

| xx | D3 | D2 | D1 | D0 | xx | xx | xx |

Data as it is written to memory shifted and merged by the Dchip.

| QW7 | D3 | D2 | D1 | D0 | QW2 | QW1 | QW0 |
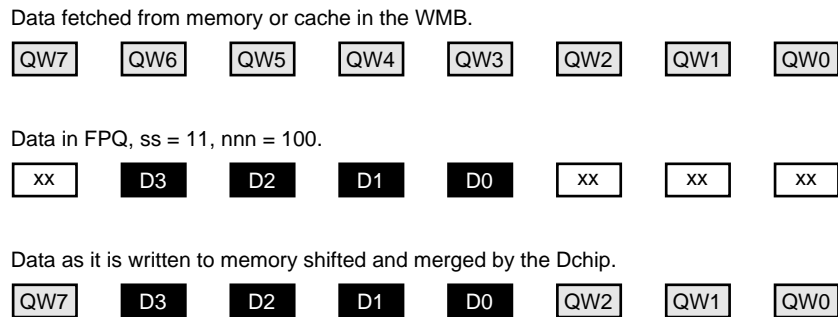
LJ-05512A.AI7

Shifting on a RMW One Quadword

In this case, the entire cache block is transferred to the WMB, either from memory or a CPU's cache (as in Write N Quadwords). Then the shift amount in the WMB–P command is used to select one quadword from the WMB to the PADbus. After the Pchip updates the single quadword, the shift amount in the WMB–P command is used to load the single quadword from the PADbus back into its original location in the WMB. A w2m command is used to transfer the entire cache block from the WMB to the memory.

7.3.3.2 Shifting for CPU Originated PIO Operations

For PIO operations that originate from the CPU, the shift operation depends on the direction of transfer and the granularity of the operation — byte, longword, or quadword. This complication is due to the CPU interface for PIO operations.

As in the case of memory operations, address bit <5> is used to swap the two half-cache blocks on their way to or from the CPU, except in the cases of PIO read bytes or PIO read longwords, where only four quadwords are transferred to the CPU.

For PIO read or write quadwords, the shifting operation to or from the PADbus is the same as for Pchip memory operations. Address bits <4:3> are used as the shift amount. Then, regardless of the value of the length field in the PAD command, eight full quadwords are transferred to the CPU, which then selects the valid quadwords from these eight.

For PIO write bytes or PIO write longwords, the CPU always supplies eight quadwords. Of these, only one quadword (for write bytes), or at most four quadwords (for write longwords), are valid. When the TPQ–P PAD command is issued, address bits <4:3> are used as the shift amount, which causes the transfer to start with the first valid quadword. The length is one quadword for PIO write bytes and four quadwords for PIO write longwords. In the latter case, the Pchip ignores the excess quadwords if there were actually less than four valid quadwords from the CPU.

The exceptional cases are PIO read bytes and PIO read longwords, because the CPU requires each quadword to be repeated during transfer. For these cases, the PP–FPQ "stutter" PAD command is used when the data is loaded from the PADbus to the FPQ. Each quadword on the PADbus is loaded into two locations in the FPQ. This allows the data to be read normally from the FPQ when it must be transferred to the CPU. However, it does complicate the interpretation of the shift amount for the following reason.

For the PP–FPQ command, the shift amount represents a shift of quadword pairs. This is further complicated by the fact that the length can be one quadword (for PIO read bytes) or nominally four quadwords (for PIO read longwords) even with a nonzero shift amount. As for the other operations, address bits <4:3> are used as the shift amount to load the FPQ. When the data is transferred to the CPU, it picks out the valid double-pumped quadwords from the four duplicated quadwords and ignores the invalid data.

Figure 7–5 shows examples of how the shift amount is used to load the FPQ for the PP–FPQ PAD command.

Figure 7–5 Shift Amount for PP–FPQ PAD Command

Data in FPQ, ss = 00, nnn = 100.

| XX | XX | D2 | D2 | D1 | D1 | D0 | D0 |

Data in FPQ, ss = 01, nnn = 100.

| D2 | D2 | D1 | D1 | D0 | D0 | XX | XX |

Data in FPQ, ss = 10, nnn = 100.

| D1 | D1 | D0 | D0 | XX | XX | XX | XX |

Data in FPQ, ss = 11, nnn = 100.

| D0 | D0 | XX | XX | XX | XX | XX | XX |
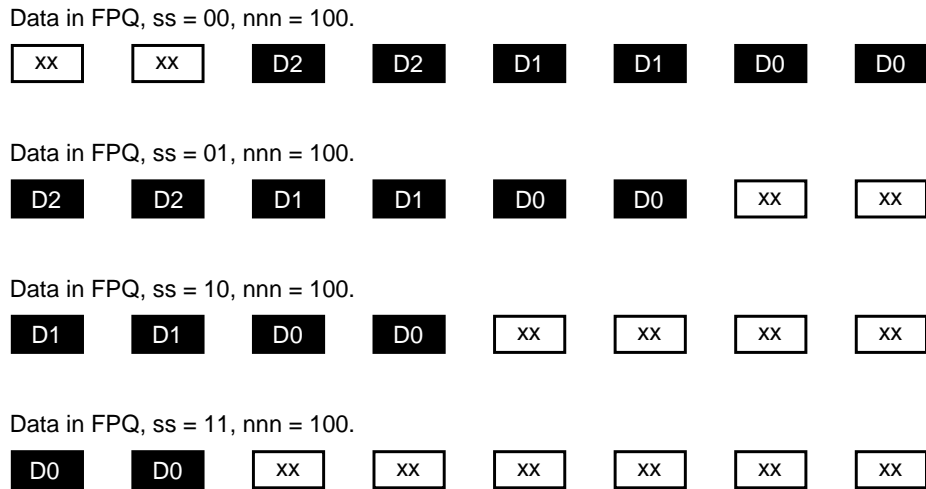
Note: In all cases of PP-FPQ if nnn = 001 only D0 is loaded.

LJ-05513.AI4

7.3.3.3 Shifting for PTP Operations

In the case of PTP operations, the shift amount in the PAD command is redundant because both PADbus transfers begin with the first valid quadword. However, in order to simplify the control logic, address bits <4:3> are used as a shift amount for both incoming and outgoing transfers. Thus, these operations are the same as the CPU PIO operations, except that the "stutter" PAD command is never used.

7.3.3.4 Shift Amount Versus CPU SysDC and Memory Access

Table 7–6 shows how, for each operation that involves the PADbus, the shift amount is related to the SysDC<1:0> field (used for CPU data wrapping) and the access to the memory DRAMs.

Table 7–6 Source of Shift Amount and SysDC Fields

Operation	PAD Command Shift Amount	SysDC<1:0>	Memory Access LSB
Pchip DMA read	**addr<4:3>**	**addr<5>** + 0	**addr<5>**
Pchip DMA write	**addr<4:3>**	**addr<5>** + 0	**addr<5>**
Pchip DMA RMW	**addr<4:3>**	**addr<5>** + 0	**addr<5>**

Table 7–6 Source of Shift Amount and SysDC Fields (Continued)

Operation	PAD Command Shift Amount	SysDC<1:0>	Memory Access LSB
CPU PIO Rd Bytes	**addr<4:3>**	00	N/A
CPU PIO Rd LWs	**addr<4:3>**	00	N/A
CPU PIO Rd QWs	**addr<4:3>**	**addr<5>** + 0	N/A
CPU PIO Wr Bytes	**addr<4:3>**	**addr<5>** + 0	N/A
CPU PIO Wr LWs	**addr<4:3>**	**addr<5>** + 0	N/A
CPU PIO Wr QWs	**addr<4:3>**	**addr<5>** + 0	N/A
PTP	**addr<4:3>**	N/A	N/A

7.3.4 Accumulate Timing

Accumulate timing is used in the case of multiple queued data transfers, where the data arrives on the memory bus over four cycles (16-byte arrays), but is delivered to the CPU in two cycles. The accumulate command causes the Dchips to accumulate the memory data in a storage register, instead of piping it out directly to the CPU. When the Cchip issues the m2ca CPM command, the memory data is delayed by two cycles to the CPU bus. In a series of alternating transfers from two (16-byte) memory buses to the same CPU, the first transfer uses m2c and the subsequent transfers use m2ca. This allows the aggregate bandwidth of the two memory buses to fully utilize the bandwidth of the CPU data bus.

7.3.5 Wrapping

Wrapping is a method of reordering quadwords in a cache block, with the goal of having a particular quadword being moved to the front of the block. This is most commonly done when the CPU is reading in a new cache block. This allows the CPU to use data from any one of the eight quadwords in the block immediately, without waiting for any other quadwords to transfer first.

The Dchips do not support wrapping. However, the Cchip accesses the memory and other CPU's data (in the case of a cache hit) with wrapping dependent on the size of the memory bus. For systems with 32-byte memory arrays, the desired half-cache block is delivered first. For systems with 16-byte memory arrays, the desired quarter-cache block is delivered first.

7.4 Dchip Memory Data Slicing

This section describes the specific quadwords and bytes that are gated to the memory bus for all Dchip configurations. Mapping is listed in detail for Dchip 0. Maps for other Dchips in the system can be translated using the note provided.

2 Dchips

Dchip 0	2 Dchips, 16-Byte Bus							
	Cycle 0		Cycle 1		Cycle 2		Cycle 3	
	QW	Byte	QW	Byte	QW	Byte	QW	Byte
m1d<31:24>	1	6	3	6	5	6	7	6
m1d<23:16>	0	6	2	6	4	6	6	6
m1d<15:8>	1	2	3	2	5	2	7	2
m1d<7:0>	0	2	2	2	4	2	6	2
m0d<31:24>	1	4	3	4	5	4	7	4
m0d<23:16>	0	4	2	4	4	4	6	4
m0d<15:8>	1	0	3	0	5	0	7	0
m0d<7:0>	0	0	2	0	4	0	6	0

Note:
Dchip 1 uses the same pattern with bytes 1, 5, 3, and 7 instead of 0, 4, 2, and 6, respectively.

4 Dchips

Dchip 0	Cycle 0		Cycle 1		Cycle 2		Cycle 3	
	QW	Byte	QW	Byte	QW	Byte	QW	Byte
4 Dchips, 2 16-Byte Buses, Memory 1								
m1d<31:24>	1	4	3	4	5	4	7	4
m1d<23:16>	0	4	2	4	4	4	6	4
m1d<15:8>	1	0	3	0	5	0	7	0
m1d<7:0>	0	0	2	0	4	0	6	0
4 Dchips, 2 16-Byte Buses, Memory 0								
m0d<31:24>	1	4	3	4	5	4	7	4
m0d<23:16>	0	4	2	4	4	4	6	4
m0d<15:8>	1	0	3	0	5	0	7	0
m0d<7:0>	0	0	2	0	4	0	6	0
4 Dchips, 1 32-Byte Bus								
m1d<31:24>	3	4	7	4	–	–	–	–
m1d<23:16>	2	4	6	4	–	–	–	–
m1d<15:8>	1	4	5	4	–	–	–	–
m1d<7:0>	0	4	4	4	–	–	–	–
m0d<31:24>	3	0	7	0	–	–	–	–
m0d<23:16>	2	0	6	0	–	–	–	–
m0d<15:8>	1	0	5	0	–	–	–	–
m0d<7:0>	0	0	4	0	–	–	–	–

Notes:

Dchip 1 uses the same pattern with bytes 1 and 5 instead of 0 and 4, respectively.

Dchip 2 uses the same pattern with bytes 2 and 6 instead of 0 and 4, respectively.

Dchip 3 uses the same pattern with bytes 3 and 7 instead of 0 and 4, respectively.

8 Dchips

Dchip 0	Cycle 0		Cycle 1		Cycle 2		Cycle 3	
	QW	Byte	QW	Byte	QW	Byte	QW	Byte
8 Dchips, 2 16-Byte Buses, Memory 1								
m1d<31:24>	–	–	–	–	–	–	–	–
m1d<23:16>	–	–	–	–	–	–	–	–
m1d<15:8>	1	0	3	0	5	0	7	0
m1d<7:0>	0	0	2	0	4	0	6	0
8 Dchips, 2 16-Byte Buses, Memory 0								
m0d<31:24>	–	–	–	–	–	–	–	–
m0d<23:16>	–	–	–	–	–	–	–	–
m0d<15:8>	1	0	3	0	5	0	7	0
m0d<7:0>	0	0	2	0	4	0	6	0
8 Dchips, 2 32-Byte Buses, Memory 1								
m1d<31:24>	3	0	7	0	–	–	–	–
m1d<23:16>	2	0	6	0	–	–	–	–
m1d<15:8>	1	0	5	0	–	–	–	–
m1d<7:0>	0	0	4	0	–	–	–	–
8 Dchips, 2 32-Byte Buses, Memory 0								
m0d<31:24>	3	0	7	0	–	–	–	–
m0d<23:16>	2	0	6	0	–	–	–	–
m0d<15:8>	1	0	5	0	–	–	–	–
m0d<7:0>	0	0	4	0	–	–	–	–

Note:
Dchips 1 through 7, bytes 1 through 7, use the same pattern as Dchip 0.

7.5 Dchip CPU Data Slicing

This section shows data movement to and from the CPU data pins on the Dchip. Data always moves in 8 cycles, with a quadword moving each cycle. Every four cycles on the CPU port corresponds to a single **i_sysclk** cycle. The following tables show the details of movement for the first four cycles. The second four cycles are identical, except that quadwords 4 through 7 are moved instead of quadwords 0 through 3.

2 Dchips

Dchip 0		2 Dchips							
	CPU	**Cycle 0**		**Cycle 1**		**Cycle 2**		**Cycle 3**	
		QW	**Byte**	**QW**	**Byte**	**QW**	**Byte**	**QW**	**Byte**
cd_l<31:24>	0	0	6	1	6	2	6	3	6
cd_l<23:16>	0	0	2	1	2	2	2	3	2
cd_l<15:8>	0	0	4	1	4	2	4	3	4
cd_l<7:0>	0	0	0	1	0	2	0	3	0

Dchip 1		2 Dchips							
	CPU	**Cycle 0**		**Cycle 1**		**Cycle 2**		**Cycle 3**	
		QW	**Byte**	**QW**	**Byte**	**QW**	**Byte**	**QW**	**Byte**
cd_l<31:24>	0	0	7	1	7	2	7	3	7
cd_l<23:16>	0	0	3	1	3	2	3	3	3
cd_l<15:8>	0	0	5	1	5	2	5	3	5
cd_l<7:0>	0	0	1	1	1	2	1	3	1

Dchip CPU Data Slicing

4 Dchips

Dchip 0	4 Dchips							
	CPU	**Cycle 0**		**Cycle 1**		**Cycle 2**		**Cycle 3**	
		QW	**Byte**	**QW**	**Byte**	**QW**	**Byte**	**QW**	**Byte**
cd_l<31:24>	1	0	4	1	4	2	4	3	4
cd_l<23:16>	1	0	0	1	0	2	0	3	0
cd_l<15:8>	0	0	4	1	4	2	4	3	4
cd_l<7:0>	0	0	0	1	0	2	0	3	0

Dchip 1	4 Dchips							
	CPU	**Cycle 0**		**Cycle 1**		**Cycle 2**		**Cycle 3**	
		QW	**Byte**	**QW**	**Byte**	**QW**	**Byte**	**QW**	**Byte**
cd_l<31:24>	1	0	5	1	5	2	5	3	5
cd_l<23:16>	1	0	1	1	1	2	1	3	1
cd_l<15:8>	0	0	5	1	5	2	5	3	5
cd_l<7:0>	0	0	1	1	1	2	1	3	1

Dchip2	4 Dchips							
	CPU	**Cycle 0**		**Cycle 1**		**Cycle 2**		**Cycle 3**	
		QW	**Byte**	**QW**	**Byte**	**QW**	**Byte**	**QW**	**Byte**
cd_l<31:24>	1	0	6	1	6	2	6	3	6
cd_l<23:16>	1	0	2	1	2	2	2	3	2
cd_l<15:8>	0	0	6	1	6	2	6	3	6
cd_l<7:0>	0	0	2	1	2	2	2	3	2

Dchip 3	4 Dchips							
	CPU	**Cycle 0**		**Cycle 1**		**Cycle 2**		**Cycle 3**	
		QW	**Byte**	**QW**	**Byte**	**QW**	**Byte**	**QW**	**Byte**
cd_l<31:24>	1	0	7	1	7	2	7	3	7
cd_l<23:16>	1	0	3	1	3	2	3	3	3
cd_l<15:8>	0	0	7	1	7	2	7	3	7
cd_l<7:0>	0	0	3	1	3	2	3	3	3

8

Pchip Architecture

This chapter describes the internal architecture for the Pchip.

8.1 Pchip Architecture

The Pchip is the interface chip between devices on the PCI bus and the rest of the system. There can be one or two Pchips, and corresponding single or dual PCI buses, connected to the Cchip and Dchips. The Pchip performs the following functions:

- Accepts requests from the Cchip by means of the CAPbus and enqueues them

- Issues commands to the PCI bus based on these requests

- Accepts requests from the PCI bus and enqueues them

- Issues commands to the Cchip by means of the CAPbus based on these requests

- Transfers data to and from the Dchips based on the above commands and requests

- Buffers the data when necessary

- Reports errors to the Cchip, after recording the nature of the error

Figure 8–1 shows a block diagram of the Pchip.

Figure 8–1 Pchip Block Diagram
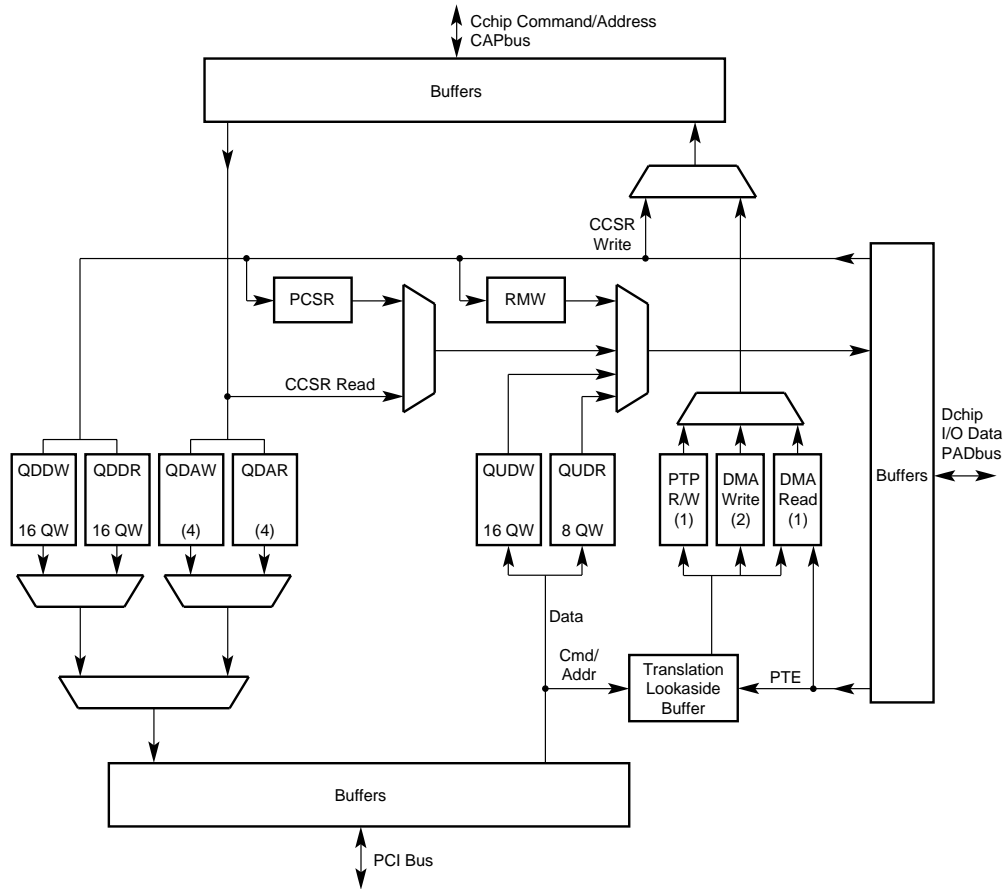
LJ-05504.AI4

Legend:

QDDW: Downstream Write Data Queue

QDDR: Downstream Read Data Queue

QDAW: Downstream Write Address Queue

QDAR: Downstream Read Address Queue

QUDW: Upstream Write Data Queue

QUDR: Upstream Read Data Queue

PCSR: Pchip CSR

CCSR: Cchip CSR

8.1.1 Pchip Interfaces

This section contains descriptions of three Pchip interfaces:

- PCI bus
- CAPbus
- PADbus

8.1.1.1 PCI Bus

The Pchip is compliant with PCI Bus Specification, Revision 2.1 (although as a host bridge it does not comply with all of the ordering rules – see Section 8.1.2.1), and supports the following:

- 33-MHz operation.
- 64-bit wide AD (interoperates with 32-bit devices).
- Dual-address cycle (DAC) is accepted for DMA, with some restrictions.

The Pchip only supports DAC operations on a 64-bit PCI bus when the entire address is provided in the first cycle of the DAC transaction.

Locks are not supported. Further details on precisely which PCI operations are supported by the Pchip, and how they are supported, can be found in Section 8.8.2.

8.1.1.2 CAPbus

The Cchip and Pchips communicate over a bidirectional 24-bit wide multicycle bus called the CAPbus. This communication can take the form of requests for reading or writing data to or from the PCI or system, or as an indication of the data return for read transactions. Because the Cchip has no direct data path connection to the Dchips, Cchip CSR data is also transferred over this bus. The CAPbus also has sideband signals for arbitration of its ownership, and for flow control. See Chapter 6 for details on the CAPbus.

8.1.1.3 PADbus

The PADbus is a bidirectional 32-bit bus with 8 check bits. It is controlled by the Cchip and is used to transfer data between the Pchip and the Dchips. Each Pchip has a separate PADbus. The PADbus will either use the lower 4 check bits on each of the two transfers required for a quadword, or it will present all 8 check bits on the first transfer and repeat them on the second transfer. This mode is static and is determined at power-up/reset. See Chapter 7 for details on the PADbus.

Under control of PCTL<ECCEN>, the Pchip corrects single-bit errors and detects uncorrectable errors using the ECC matrix defined by the 21264 for the following operations only:

- DMA reads

- SGTE reads

- DMA RMW

The Pchip generates check bits using the same ECC matrix for all outgoing data on the PADbus.

8.1.2 Pchip Internals

In the following sections, the terms *upstream* and *downstream* are defined as follows:

- Upstream data – From the Pchip to the Dchips, regardless of the initiator of the request.

- Downstream data – From the Dchips to the Pchip, regardless of the initiator of the request.

- Upstream command – From the Pchip to the Cchip on the CAPbus, regardless of the originator of the request. For example, the Pchip sends an upstream LoadP command on the CAPbus when it is ready to return upstream data in response to an earlier downstream PCI read command on the CAPbus from the Cchip.

- Downstream command – From the Cchip to the Pchip on the CAPbus.

- Upstream operations – These are operations where the request (CAPbus read or write command) is from the Pchip to the Cchip, regardless of the direction of data flow. The Pchip is the PCI target.

- Downstream operations – These are operations where the request (CAPbus read or write command) is from the Cchip to the Pchip, regardless of the direction of data flow. The Pchip is the PCI master.

8.1.2.1 PCI Ordering – Upstream and Downstream Interactions

The PCI Specification, Rev 2.1, has rules governing which transactions can pass other transactions. The Pchip implements the ordering rules using some interactions between the upstream address machines and the two downstream address controllers. The Cchip imposes the ordering rules described in Section 6.1.3, as well as the Pchip rules described in the following paragraphs.

Table 8–1 and the accompanying table notes list the Pchip rules and the manner of implementation. The QDA cannot distinguish between PCI reads initiated by the CPU (PIO read) and those initiated by the other Pchip (PTP read), because the identical CAP-bus command is used for both (PCI read).

Table 8–1 PCI Read and Write Pchip Ordering – Can Second Pass First?

First →	DMA write request	PIO write request	PTP write request	DMA read request	PIO/PTP read request	DMA read completion	PIO/PTP read completion
Second ↓							
DMA write request	No^1	N/A	No^1	Yes	Yes	N/A	Yes
PIO write request	N/A	No^1	No^1	N/A	Yes	Yes	Yes
PTP write request	No^1	No^1	No^1	Yes	Yes	Yes	Yes
DMA read request	No^2	N/A	No^2	No^3	Yes^3	N/A	Yes
PIO/PTP read request	No^2	No^2	No^2	Yes^3	No^4	Yes	Yes
DMA read comp	N/A	Yes^5	Yes^5	N/A	Yes	No^3	Yes
PIO/PTP read comp	No^6	Yes^7	No^8	Yes	Yes	Yes	No^4

[1] The three upstream write machines maintain strict ordering with respect to one another. Also, in QDA, all write requests are handled in order.

[2] The upstream read machines wait to make their CAPbus requests until any earlier upstream write machines have made all of their CAPbus requests. The downstream read controller waits until all earlier downstream writes have completed before making its PCI read request.

[3] There is only one upstream DMA read machine, so DMA read requests or completions cannot pass each other. However, there is no ordering imposed between the upstream DMA read machine and the upstream PTP read machine.

[4] The single upstream PTP read machine implies that upstream PTP read requests or completions cannot pass each other. In addition, in QDA, the downstream PCI read requests and completions (PIO and PTP) are processed in order. (This latter is simply an implementation convenience and is not required.)

[5] In violation of the PCI Specification, Rev 2.1, DMA delayed-read completions are allowed to pass downstream write requests. This is necessary to avoid deadlocks with ISA bridges, because an ISA bus may not be able to complete a write until its DMA read completes. This is also the behavior of the Cchip for DMA reads, as described in Section 6.1.3. This behavior of the Pchip is the same whether the PCI master is retried (delayed completion) or remains connected while the DMA data is fetched from memory.

[6] This is the *pipe-cleaner* function of PIO reads flushing DMA writes in the upstream direction. It is implemented by signals from the upstream machines to QDA. When QDA begins a read, it notes the number of active upstream write machines. QDA does not return its data until all of the active write machines have gone idle at least once.

[7] PIO read completions and PIO write requests travel in opposite directions and have no relationship. A PTP read completion in the upstream PTP read machine does not wait for PIO writes in QDA because they have distinct sources of data (the other Pchip versus the CPU).

[8] This is the *pipe-cleaner* as applied to PTP operations, and is applied in QDA and in the upstream machines. As stated in Table Note #6, QDA waits to complete a read until earlier upstream write machines have gone idle. Conversely, unlike Table Note #7, the upstream PTP read machine, when it receives its completion data, records the number of outstanding PTP writes in QDA. The upstream PTP read machine does not return its data on the PCI bus until all of those outstanding writes have completed.

8.1.2.2 Upstream Address Translation

The Pchip performs the following address translation functions on incoming PCI addresses:

- Determines if the address lies inside an enabled window.

- Determines if the address lies inside the enabled DMA monster window (if a DAC).

- Determines if the address is direct mapped or scatter-gather mapped.

- Determines if the address is for DMA or PTP operation.

- Translates the address to a system address if the address is direct mapped.

- Constructs the system address of the PTE for use by the upstream address machines, if the address is scatter-gather mapped and the scatter-gather PTE TLB does not have the correct valid PTE. Otherwise, it translates the address using the TLB entry.

The scatter-gather TLB is arranged as 168 locations of 4 consecutive quadwords (see Figure 8–2).

Figure 8–2 Scatter-Gather Associative TLB
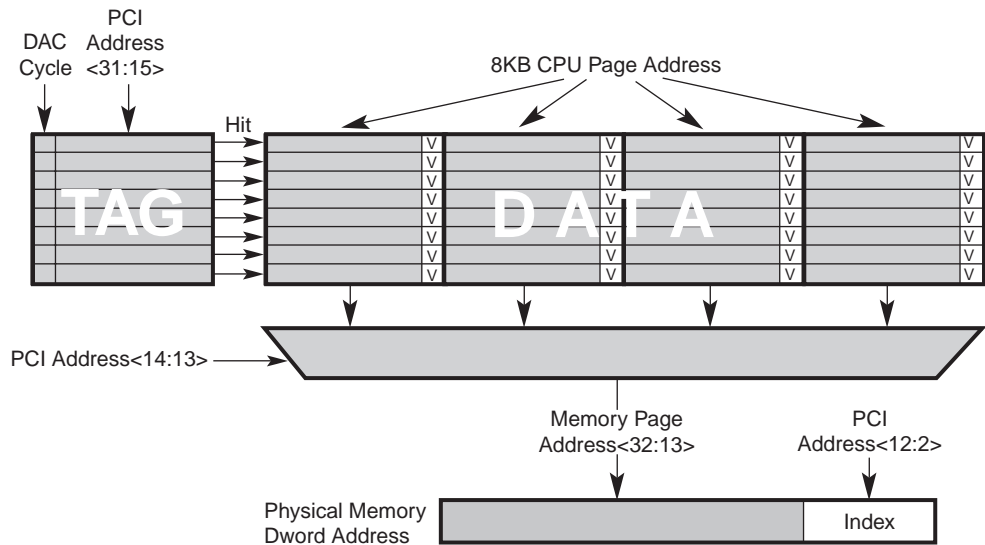
LJ04276A.AI7

8.1.2.3 Clock Control and Generation

The clock control logic generates the PCI clock from one of three possible multiples of the input **i_fwdclk** clock. The multiplier is determined by the values read from two input pins to the Pchip (**i_pclkdiv<1:0>**), which must be tied to the appropriate power and ground values for the desired multiple (see Chapter 11). The PCI clocks are active during system reset.

This logic also generates synchronizing signals that are used to enable transitions to and from state elements (flip-flops), which are clocked on system clocks, to those clocked on PCI clocks. The use of these signals is shown in Figure 8–3. A nominal PCI clock of 30 ns is shown in this figure. Using the multipliers of 6, 5, and 4, this PCI clock would be derived from a system clock with a half-period of 5 ns, 6 ns, or 7.5 ns respectively.

Figure 8–3 PCI Clock to System Clock Transitions
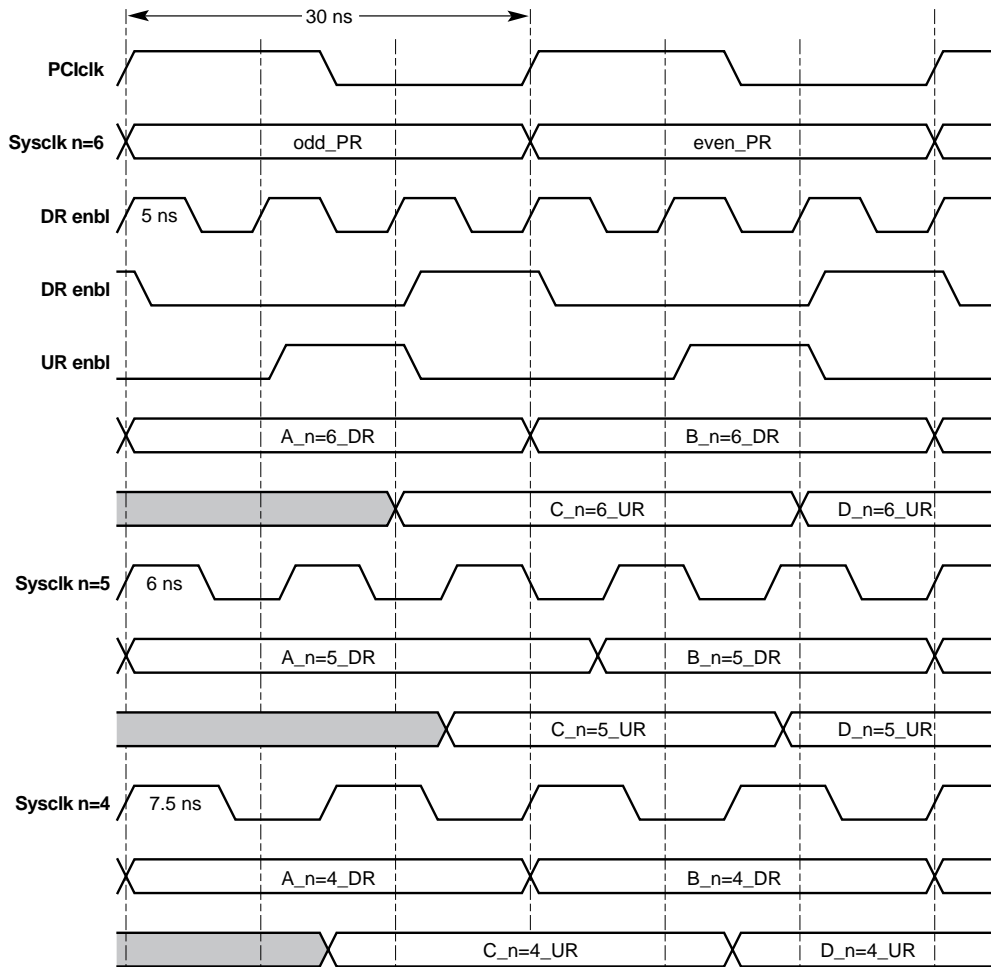
LJ-05505.AI4

The signals from flip-flops triggered by the rising edge of the PCI clock are labeled "signal_PR". The general signals from flip-flops triggered by the rising edge of the system clock are labeled "signal_R". The signals from flip-flops that are used to transition between the PCI clock domain and the system clock domain are labeled as follows:

- signal_UR – Upstream flip-flop; clocked on certain rising edges of the system clock. Such flip-flops can have as inputs signals derived from "_PR" flip-flops.

- signal_DR – Downstream flip-flop; clocked on certain rising edges of the system clock. Such flip-flops can be used to derive signals used as inputs to "_PR" flip-flops.

Figure 8–3 shows (for the multiplier of 6 only) the timing of the signals that enable loading the "_DR" and "_UR" flip-flops ("DR_enbl" and "UR_enbl"). For the multipliers of 5 and 4, the timing of these enabling signals can be inferred from the indicated transitions. This figure shows, for all multipliers, the transitions of the "_DR" and "_UR" flip-flops. The following rules apply:

- There is exactly one "_DR" transition per PCI clock period.

- There is exactly one "_UR" transition per PCI clock period.

- There is always a "_DR" transition at the next rising system clock edge immediately following a "_UR" transition.

- The total delay – From a "_PR" transition to the next "_UR" transition, then to the next "_DR" transition, and then to the next "_PR" transition always consumes exactly two PCI clock periods. Therefore, assuming that some logic function is performed in the system clock domain between the "_UR" and the "_DR" flip-flops, the impact of that logic function will be felt on the second PCI clock after the PCI clock that initiated the function. If there is a logic function that must take effect in a single PCI clock period, it must be performed in the PCI clock domain.

- For nominal static-timing analysis, the minimum path from a "_PR" flip-flop to a "_UR" flip-flop is 15 ns, which occurs in a path from "odd_PR" to "C_n=4_UR". For n=5, the minimum "_PR" to "_UR" path is 18 ns ("even_PR" to "D_n=5_UR"). For n=6, the minimum is 20 ns.

- For nominal static-timing analysis, the minimum path from a "_DR" flip-flop to a "_PR" flip-flop is 24 ns, which occurs in a path from "B_n=5_DR" to "odd_PR". All of the other paths from "_DR" flip-flops to "_PR" flip-flops have a minimum of 30 ns.

8.1.2.4 PCI Corner

The PCI corner comprises the PCI master and the PCI target state machines. These state machines conform to the PCI Specification, Rev 2.1, and support the operations described in Section 8.8.2. The PCI target state machine is a medium speed **b_devsel_l** device. That is, if **b_frame_l** and the address are on the PCI bus in cycle N, the Pchip can assert **b_devsel_l** in cycle N+2.

8.2 Peer-to-Peer PCI Memory Operations

This section explains peer-to-peer PCI memory operations (PCI*x* to System to PCI*y*).

8.2.1 Use of Page Table Entry for Peer-to-Peer Operations

A local PCI memory command that hits a window register whose PTP bit is set is forwarded to the Cchip as a PTP memory access. To support software's use of either system address bits <43> or <40> to indicate PIO space, and because PTE bits <22:1> correspond to page address bits <34:13>, it is convenient to use either bit <31> or bit <28> of the PTE to represent the PTP bit. The Pchip ORs together bits <31> and <28> of the scatter-gather PTE from memory to form the TLB's PTP field. The formats are shown in Figure 8–4.

Figure 8–4 Scatter-Gather Page Table Entry in Memory

LJ-05522A-AI4
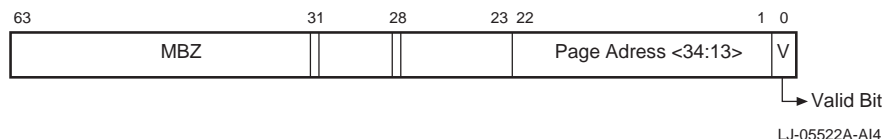
The Pchip checks the PTE if either PTP bit is set. When ECC checking is enabled and an uncorrectable ECC error is detected on a PTE, that entry is marked invalid in the TLB. If a PTP bit is set, the Pchip also invalidates an entry if the Pchip does not detect the presence of a remote Pchip at reset time, or if the PTE bits <22:20> (system address <34:32>) do not correspond to the PCI memory space in the other Pchip, per system address space shown in Table 10–1. Only 4GB of PCI memory space is accessible because the Pchip does not support DAC as a bus master. However, this does not preclude DAC PCI addresses that fall within Window 3 from scatter-gather mapping into the peer's PCI memory space.

8.2.2 General Peer-to-Peer Operations and Deadlock Avoidance

A Pchip responds to a PTP PCI memory operation from the Cchip in exactly the same way that it responds to a PIO PCI memory operation from the CPU. The same command is actually used from the Cchip to the Pchip for both of these cases.

Only the following commands are supported for PTP operations:

- Memory read – A memory read command is forwarded as a PCI memory read with a byte mask corresponding to the byte enables on the originating PCI bus. At most eight bytes are fetched in this case.

- Memory write.

- Memory read line – A memory read line command is forwarded as a PCI memory read with a quadword mask that reads all bytes to the end of the cache block (64-byte boundary) as defined by the CPU.

- Memory read multiple – A memory read multiple command is forwarded as a PCI memory read with a quadword mask that reads all bytes to the end of the cache block (64-byte boundary) as defined by the CPU.

- Memory write and invalidate – A memory write and invalidate command is forwarded as a memory write command.

All PTP read operations use the delayed-completion mechanism and PTP memory writes will be posted. System software is responsible for avoiding deadlocks that can result from simultaneous PTP operations, just as it is if the PTP operations were between devices on the same PCI bus.

For example, a device that does not accept a read or write until its own read completes can lead to a PTP deadlock. An ISA bridge exhibits this behavior (it is not PCI compliant) because it will not accept a write downstream to the ISA bus until a read initiated on that bus completes. The 21272 chipset prevents such deadlocks by allowing DMA reads to system memory to complete before outstanding PIO or PTP writes are completed. However, the 21272 chipset antideadlock mechanism depends on the fact that a PTP write delivered to the target PCI bus will complete unconditionally. Conversely, the 21272 chipset antideadlock mechanism also depends on the fact that the completion of any other operation initiated on a PCI bus is not conditional on the completion of an earlier PTP read initiated on that bus.

For this reason the following rules apply:

Rules: PTP operations should not be targeted to devices on an ISA bus, or any other device that is not PCI Specification, Rev 2.1 compliant.

PTP operations should not be initiated by any devices on an ISA bus, or any other device that is not PCI Specification, Rev 2.1 compliant.

In addition, no prefetching is performed for PTP read operations between Pchips. Therefore, to achieve high-bandwidth transfers between two devices, if the devices are placed on separate PCI buses, only PTP write operations should be used for the transfer. Best performance is obtained if the devices are on the same PCI bus, so that no PTP operations through the Pchip or Cchip are required.

8.3 No Locks

The Pchip does not distinguish lock transactions on the PCI bus because it does not have an input for the PCI signal LOCK#.

8.4 Merging, Splitting, and Chaining Rules

The Pchip attempts to maximize PCI bus utilization by *combining* the transactions listed in the following sections.

8.4.1 Merging Transactions

Merging is defined as using a single write transaction to transfer bytes that were originally in separate requests. The Pchip does not perform merging.

8.4.2 Splitting Transactions

Splitting is defined as taking a single request that has multiple consecutive quadwords masked out (the CPU merges multiple accesses into one request), and generating two back-to-back PCI transactions. The Pchip does not perform splitting.

8.4.3 Chaining Transactions

Chaining is the process of taking two requests to consecutive addresses and continuing the first PCI transaction past the boundary between them without a new PCI address cycle. The Pchip performs chaining only under the following conditions:

• Write requests with quadword masks only.

• The last quadword of the first request is enabled.

• The first two quadwords of the second request are enabled.

8.5 Configuration

Configuration of the PCI buses is conventional for each Pchip individually. The following configuration cycle conversion and forwarding rule applies:

Rule: A Pchip converts a configuration cycle command from the CAPbus to a Type 0 command on its PCI bus if the Bus # is zero. It forwards it as a Type 1 command if the Bus # is nonzero.

8.6 PCI Arbitration

The Pchips have the capability to act as the arbiter for their respective PCI buses. There are seven request inputs (**b_req_l<6:1>** and **b_reqgnt_l<0>**) and seven grant outputs (**b_grant_l<6:1>** and **b_gntreq_l<0>**). This supports eight devices (seven plus the Pchip itself, whose request and grant are internal signals).

The arbiter consists of two round-robin selectors; one high priority and one low priority. The low-priority group as a whole represents one entry in the high-priority group. The PCTL register controls whether a device connected to a request/grant pair is in the low-priority or high-priority group. Each request/grant pair has a corresponding bit in the register, that if set to 1, puts that device in the high-priority group, otherwise, it is in the low-priority group.

Arbitration parking, selectable between the Pchip or the device that last had its grant asserted, is selected by means of a bit in the PCTL CSR.

The previously described internal arbitration scheme can be disabled by means of a bit in the PCTL CSR. In this case, the Pchip sends its request signal out to an external arbiter on **b_gntreq_l<0>** and brings in its grant signal on **b_reqgnt_l<0>**. In this manner, **b_gntreq_l<0>** is always an output pin and **b_reqgnt_l<0>** is always an input pin, regardless of whether the arbiter is internal or external.

The request and grant lines are all pulled up (to the deasserted state) using weak pull-ups in the Pchip. If the Pchip internal arbiter is not enabled, inadvertent grants are prevented. This also prevents spurious requests from PCI slots into which no device has been installed.

8.7 PCI Software Reset

The Pchip supports software writes to reset its PCI bus.

A PIO write to the Pchip's SPRST CSR will cause RST# to be asserted, while another CSR write to the same location will cause RST# to be deasserted. The data written to the Soft PCI Reset Register is ignored. As in the hardware system reset, REQ64# will be driven as necessary. This function will need to be executed independently on each Pchip in the system.

The requirement that there be no traffic requiring Pchip action prior to asserting the software reset is guaranteed by the following procedure, which applies to all cases except PTP. Software must ensure that all PTP traffic is complete before asserting software reset[1]. If a PCI transaction is in progress in which the PChip is the slave or master at the assertion of software reset, the results will be UNPREDICTABLE. This should not happen if the following sequence completes successfully:

1. Disable PTP and wait for all outstanding PTP transactions to complete.

2. Disable all Pchip windows (both Pchips). This will prevent the Pchip target state machines from allowing any new connections or retried connections for delayed read completions.

3. Set Pchip CSR PCTL <FDSC> (both Pchips) for "fast" discard of delayed read completion data. This will allow any prefetched DMA read data or PTP read data to be thrown away and the state machines to go back to idle and free up the queues.

4. Wait at least 2^{14} PCI clock cycles. This will allow any in-progress transactions to complete. Note, however, that some PCI transactions can take longer than this if there are excessive retries or long IRDY slips when the Pchip transfers a lot of data (maximum 8KB). If either of these cases occurs when reset is asserted, the PChip is in the middle of a transaction as either master or slave, and results will be UNPREDICTABLE. Neither case is considered likely with currently known PCI devices. If there is a danger of this in your system, the recommendation is to wait a longer time before asserting software reset.

5. Issue a PIO PCI read (both Pchips). This will ensure that all non-PTP traffic is complete.

6. Disable PChip internal arbiter (both Pchips) if the Pchip was the PCI arbiter. This will ensure proper reset behavior for the arbitration pins.

7. Set PCI reset (both Pchips).

8. Wait for at least 100 µsec. PCI reset will be held during this period. Do not make any new PIO read or write request during this period.

9. Clear PCI reset (both Pchips).

10. Clear the PERROR CSR (both Pchips). There may have been errors logged and the Pchip's error interrupt line pulled during this process.

11. Reenable the Pchip windows and reenable the Pchip internal arbiter as needed.

---

1 A limited window exists in which PTP reads may not have completed by issuing a PIO PCI read as set forth in the reset sequence. But because of this window, software reset is not guaranteed to work if PTP traffic can be active.

8.8 Error Handling

The following sections describe error handling.

8.8.1 Memory Data Errors — DMA Reads and Writes, SGTE Reads

If ECC memory is installed in a system using the 21272 chipset, PCTL<ECCEN> can be set. In this case, the Pchips:

- Generate ECC bits for DMA writes (including DMA RMW operations).

- Check the ECC and correct single-bit errors for DMA reads.

- Check the ECC and correct single-bit errors for DMA RMW operations.

- Check the ECC and correct single-bit errors for SGTE reads.

8.8.1.1 Correctable and Uncorrectable Memory Errors

This section describes how correctable and uncorrectable errors are handled.

If a correctable or uncorrectable ECC error is detected for a cache block prefetch of DMA read data, and that data is not delivered to the PCI bus, it is UNPREDICTABLE whether or not the ECC error will be reported. (This depends on when the PCI operation completed relative to when the ECC error is detected.) If the data is delivered to the PCI bus, any ECC error will be reported (unless the PERRMASK bits are set).

8.8.1.1.1 Correctable Memory Errors

If a single-bit error is corrected, the operation in question completes normally. If PERRMASK<CRE> is enabled, and PERROR<CRE> is set, then the system address (not the PCI address) is captured in PERROR<ADDR>. The PERROR<CMD> field indicates the type of operation (DMA read, DMA RMW, or SGTE read) that detected the error. The Pchip's **b_error** output signal is asserted. This signal should be connected to one of the error interrupt pins routed to the Cchip by means of the TIGbus.

8.8.1.1.2 Uncorrectable Memory Errors

The Pchip can detect an uncorrectable memory error on a DMA read, a DMA RMW, or an SGTE read. In all cases, if PERRMASK<UECC> is enabled, PERROR<UECC> is set, the system address (not the PCI address) is captured in PERROR<ADDR>, and the PERROR<CMD> field indicates the type of operation (DMA read, DMA RMW, or SGTE read) that detected the error. The Pchip's **b_error** output signal is asserted. The rest of the operation depends on the type of transaction.

On DMA reads, the Pchip drives incorrect parity on the PCI bus when the erroneous data is being returned on the PCI.

Note: Incorrect parity will be driven for all subsequent data transfers in that burst. Software is expected to crash the system on the detection of this error.

On the read portion of an RMW, the Pchip forces incorrect ECC into that location during the write portion of the RMW by inverting the upper three checkbits. This ensures that when this location is subsequently read, an uncorrectable memory error will be detected with a known syndrome of E0 (hexadecimal).

Note: As in the case of reads, the uncorrectable error could be forced on subsequent writes to memory, as well. Software is expected to crash the system on the detection of this error.

If an uncorrectable error is detected on an SGTE read operation, the TLB entry is not marked valid, and the PTE is not used to fetch DMA data.

If an uncorrectable error is detected on DMA read data, the Pchip drives incorrect parity on the PCI bus as the data is being returned to the master.

If an uncorrectable error is detected on the read portion of a DMA RMW operation, the Pchip forces incorrect ECC into memory on the write portion of the RMW by inverting the upper three checkbits. This results in a syndrome of E0 (Hexadecimal) when that data is read.

8.8.2 PCI Errors

The PERROR bits that represent PCI errors are defined in Section 10.2.5.6. Each of these bits has a counterpart in the PERRMASK register to enable that error, and a counterpart in the PERRSET register to force a report of that type of error without the error actually occurring. When any of these bits are set in PERROR, the Pchip's **error** signal is asserted to the Cchip, and PERROR is locked. Any further error detection causes PERROR<LOST> to be set.

If PERROR<CRE> or PERROR<UECC> is set, then the PERROR<ADDR> and PERROR<CMD> fields represent the system address and the type of operation that caused the ECC error to be detected. If any of the other (PCI error) bits in PERROR are set, then the PERROR<ADDR> and PERROR<CMD> fields represent the PCI address and PCI command of the operation that caused the error to be detected.

The following sections provide more details on the PCI operations for which errors are detected. Typically, the Pchip only reports errors using the PERROR register when it is the *master* of the PCI transaction. When the Pchip is the *target* of the transaction, it relies on the master to report the error. Exceptions are noted.

8.8.2.1 No devsel_l — PERROR<NDS>

If the Pchip as master does not detect **b_devsel_l** on the PCI bus, it terminates the transaction with master-abort. For a read transaction, the Pchip returns all 1s. For a write transaction, it discards the data. For all transactions, except configuration reads and special cycles, the Pchip sets PERROR<NDS> and asserts its error signal.

8.8.2.2 Target Abort — PERROR<TA>

When the Pchip as master receives a target abort response, it sets PERROR<TA> and asserts its **b_error** signal. For a read transaction, the Pchip returns all 1s. For a write transaction, it discards the data.

8.8.2.3 PCI Read Data Parity Error — PERROR<RDPE>

If the Pchip as master detects a PIO read parity error, it asserts **b_perr_l** on the PCI bus and sets PERROR<RDPE>. It returns the erroneous data as received. As a target of a read operation, the Pchip ignores **b_perr_l**, relying on the PCI master to report the error.

8.8.2.4 PCI Write Data Parity Error — PERROR<PERR>

When the Pchip is the master, if the target device of a PCI write detects a parity error and asserts PERR# on the PCI bus, the Pchip sets PERROR<PERR>.

As the target of a write operation, if the Pchip detects a write data parity error, it asserts **b_perr_l**, sets the PERROR<PERR> bit, and forces uncorrectable memory errors on the erroneous data when the write happens in memory. The uncorrectable memory error is generated by inverting the upper three ECC checkbits. This ensures that a subsequent read of that location results in an uncorrectable memory error with the syndrome E0 (hexadecimal).

Note: This operation has side effects, and could result in subsequent memory writes, forcing uncorrectable ECC errors in memory. Software is expected to crash the system.

8.8.2.5 Invalid Page Table Entry for Scatter-Gather Operation — PERROR<SGE>

If the Pchip is the target of an operation that is not direct mapped and issues an SGTE read, but the target PTE returned from memory is not valid, the Pchip sets PER-ROR<SGE>. The PCI address stored in PERROR<ADDR> is only valid to the granularity of a page (8KB). For a read or write operation, the Pchip continually retries the operation on the PCI bus. (The Pchip always retries operations on the PCI bus that are not direct mapped and that miss in the TLB, or have an invalid PTE in the TLB.)

It is possible to have a valid TLB tag entry with an invalid PTE without setting PERROR<SGE>. This is because four PTEs are fetched at once, and the invalid PTE may have been fetched as a side effect of an earlier operation. In this case, the entire TLB entry is discarded and the four PTEs are refetched. If the target PTE is returned as invalid, PERROR<SGE> is set.

To minimize latency in the case of a DMA read that has a TLB miss, when the SGTE read returns, the DMA memory read is sent to the Cchip without inspecting whether the PTE is valid. Therefore, it is possible that the Cchip will detect an NXM (nonexistent memory) error in this case, in addition to the Pchip's detection of the invalid PTE. However, if the PTE was not valid, the fetched data is discarded and is not delivered to the PCI bus.

For PTP reads, this optimization is not taken to prevent side effects of a PIO-like read to an erroneous location. In fact, the erroneous system address in the invalid PTE may not even translate to the other Pchip's PCI memory space.

8.8.2.6 PCI Address/Command Parity Error — PERROR<APE, SERR>

When the Pchip is not the PCI master, if the Pchip detects a parity error on a command/address cycle on the PCI bus, it will not assert **b_serr_l** on the PCI bus, but will set PERROR<APE>. If the erroneous address hits in a Pchip window, the Pchip asserts **b_devsel_l** normally.

In this case, if the operation is a write, the Pchip will target abort the write and disconnect from the PCI bus. No write will be sent to the Cchip. If the operation is a read, the Pchip will also target abort the operation. However, the Pchip may return one PCI bus transfer's worth of erroneous data before the target abort, if the erroneous address happens to match the address of a pending delayed completion read. As a potential target of the operation, the Pchip does not set PERROR<SERR>.

When the Pchip is the PCI master, if a device detects an address parity error and asserts **b_serr_l**, the Pchip sets PERROR<SERR>. If the device proceeds with the data transfer, that is the only error indication provided and the operation is completed normally. If the device does not proceed with the data transfer, the Pchip reacts as follows:

- If the operation was a read, the Pchip returns all 1s.

- If the operation was a write, the Pchip discards the data.

If the Pchip detects **b_serr_l** asserted at any other time, it also sets PERROR<SERR>.

Note: The address/command information in the PERROR register is undefined when the PERROR<SERR> bit is set.

As a PCI master, the Pchip always supplies good parity on the upper 32 address bits, even though the Pchip never uses a DAC command. As a PCI target, the Pchip only checks the parity on the upper 32 address bits when **b_req64_l** is asserted and the command is DAC.

8.8.2.7 Delayed Completion Retry Timeout — PERROR<DCRTO>

When the Pchip is the target of a read operation, it may retry the request and complete the operation using a delayed completion, as dictated in the PCI Specification. This mechanism is always used for:

- PTP reads

- DMA reads when the data is not returned fast enough from memory (see PCTL<TGTLAT>, Section 10.2.5.4)

- If the address was not direct mapped, and there was a TLB miss on the Pchip

The Pchip holds the delayed completion data for a fixed number of PCI cycles (see PCTL<FDSC>, Section 10.2.5.4) and waits for the PCI master to retry the request. If the master fails to retry the request in the specified time limit, the Pchip sets PERROR<DCRTO> and discards the fetched data. For DMA operations, this has no serious consequences because the data was always prefetchable. For PTP operations, if the initial PCI command was read (not ReadMultiple and not ReadLine), the discarded data may not have been prefetchable, and a side effect may have occurred on the other Pchip's PCI bus as a result of the read.

8.9 Monitor Outputs and Counters

The Pchip provides facilities for system monitoring through CSRs, as described in Chapter 10. On each chip, there is a CSR that allows the selection of two signals from among many chip-internal signals. After a delay of two **i_sysclk** cycles, the selected signals are driven on the chip's external **b_monitor<1:0>** pins.

In addition, each chip has a CSR that contains two counters. Each counter is used to count assertions of the associated **b_monitor<n>** output signal. For example, Counter0 on the Pchip counts the number of **i_sysclk** cycles during which the Pchip **b_monitor<0>** signal is asserted. Similarly, Counter1 counts the number of **i_sysclk** cycles during which the Pchip **b_monitor<1>** signal is asserted.

For more information about the monitor functionality, contact your DIGITAL service representative.

8.10 Pchip Revision

The latest revision of the Pchip can be detected as follows:

- The revision field PCTL<REV> is a 1.

- After the deassertion of reset, the value of CSR PMONCTL is 1. In previous versions, this register defaults to 0 after reset deassertion.

9

System Memory

This chapter describes system memory, its organization into arrays, and its control signals.

9.1 Organization

System memory is implemented in synchronous DRAMs (SDRAMs). SDRAMs are organized into one to four memory arrays as described in Section 2.2.1 through Section 2.2.3. Each cache block of system memory lies within one of the arrays. Each array provides the blocks for a contiguous region of the physical memory address space. Each array is controlled by a separate set of address and control lines driven by the Cchip. Data is moved to and from the arrays by means of one or two memory buses connected to the Dchips. If two memory buses are present, each array is supported by a single bus.

9.2 Memory Arrays

Each memory array can be either split or nonsplit and either 16 bytes or 32 bytes wide. The total number of signal lines for both types of arrays is as follows:

- 16-byte array — 128 data lines plus 16 check bits if ECC is supported

- 32-byte array — 256 data lines plus 32 check bits if ECC is supported

A nonsplit array is a collection of DRAMs that share common address and control lines but have separate data lines as shown in Figure 9–1. In split arrays, the DRAMs are organized into two subarrays as shown in Figure 9–2; twice split arrays used in Typhoon have four subarrays, as shown in Figure 9–3 . Each subarray looks like a nonsplit array of the same width. The subarrays share the same data, address, and control lines except that the subarrays have separate chip selects. If more than one array is present, then all of the arrays must have the same width.

The Cchip has four memory control ports. Each port can drive the address and control lines for a single array. If less than four arrays are present, any subset of the four ports can be used. For identification purposes, each array is assigned an integer from 0 to 3 depending on which port it is connected to.

Figure 9–1 Nonsplit Array Block Diagram
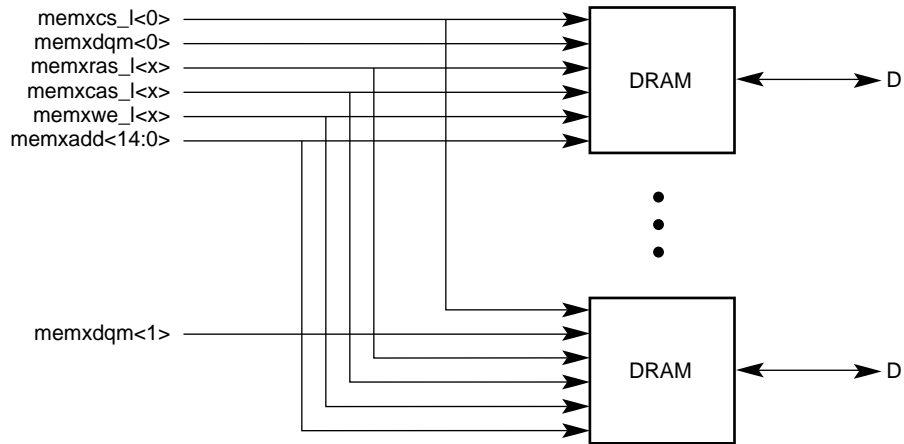
LJ-05514.AI4

Array 0 must be installed. Arrays 1, 2, and 3 are optional. If more than one array is present, then all the arrays must have the same width. All accesses to nonexistent memory locations are handled as accesses to array 0. If 4-byte or 8-byte memory modules (SIMMs or DIMMs) are used, then the number of modules required for a nonsplit array is:

- Two for 16-byte arrays using 8-byte modules

- Four for 16-byte arrays using 4-byte modules, or 32-byte arrays using 8-byte modules

- Eight for 32-byte arrays using 4-byte modules

For split arrays, either the same number of modules or twice as many modules are needed, depending on whether the modules themselves are split.

The D/Q mask (DQM) pins on the SDRAM DIMMs are used to implement partial-block writes. The SDRAMs are always sequenced as if a full 64-byte block is being written, but the DQM pins are asserted to suppress writes to octawords that are not being updated. Partial writes are only used for PCI DMA writes. All other writes update full 64-byte blocks. All reads operate on full blocks. Partial writes of less than octaword granularity are handled by means of RMW operations.

Figure 9–2 Split Array Block Diagram

memxcs<0>
memxdqm<0>
memxras_l<x>
memxcas_l<x>
memxwe_l<x>
memxadd<14:0>

DRAM 0

D

memxcs<1>

DRAM 1

memxdqm<1>

DRAM 0

D

DRAM 1
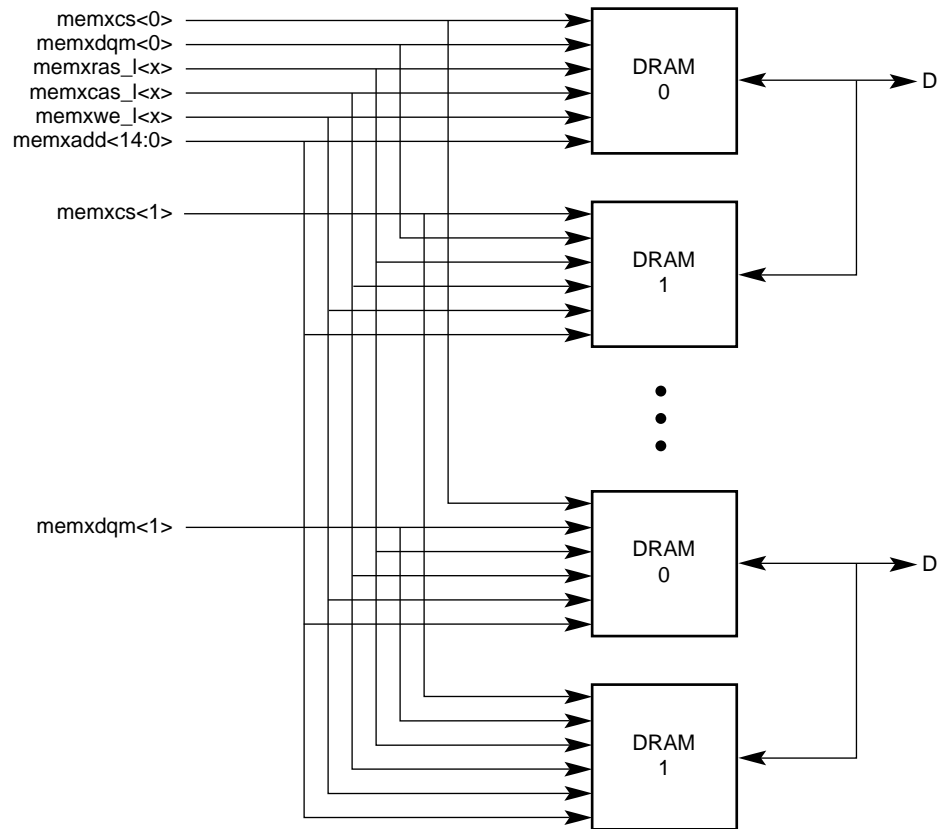
LJ-05515.AI4

Because the DQM signals to nonbuffered DIMMs may have a heavy electrical load, an extra pipeline buffer is allowed on the module for the DQM signals over the other SDRAM control signals. If such a buffer is used, the MTR<MPD> bit must be set to allow the chipset to adjust the timing appropriately. The two DQM pins for each array are driven with the same signal in 16-byte (octaword) buses, and can be used to ease the electrical load on the pins.

9.3 Memory Buses and Sibling Arrays

The memory buses can be 16 bytes or 32 bytes wide. If two buses are present, they must both have the same width. 16-byte buses can only support 16-byte memory arrays. 32-byte buses can support 16-byte (half-populated) arrays or 32-byte (fully-populated) arrays. All arrays must be either fully populated or half populated. Mixing fully-populated arrays with half-populated arrays is not supported. If the arrays are half populated, the DIMMs must be installed in the lower half of the DIMM sockets, not the upper half.

The four memory arrays are divided into two pairs of sibling arrays: arrays 0 and 2 are each others sibling, and the same is true for arrays 1 and 3. However, neither array 0 nor 2 are siblings of either array 1 or 3 and vice versa. In systems with two memory buses,

only siblings can reside on the same memory bus. This restriction exists because in two-bus systems, the Cchip does not check for conflicts on the memory buses between accesses to nonsibling arrays.

Figure 9–3 shows a twice split array.

Figure 9–3 Twice Split Array Block Diagram (Typhoon Only)

FM-06406.AI4
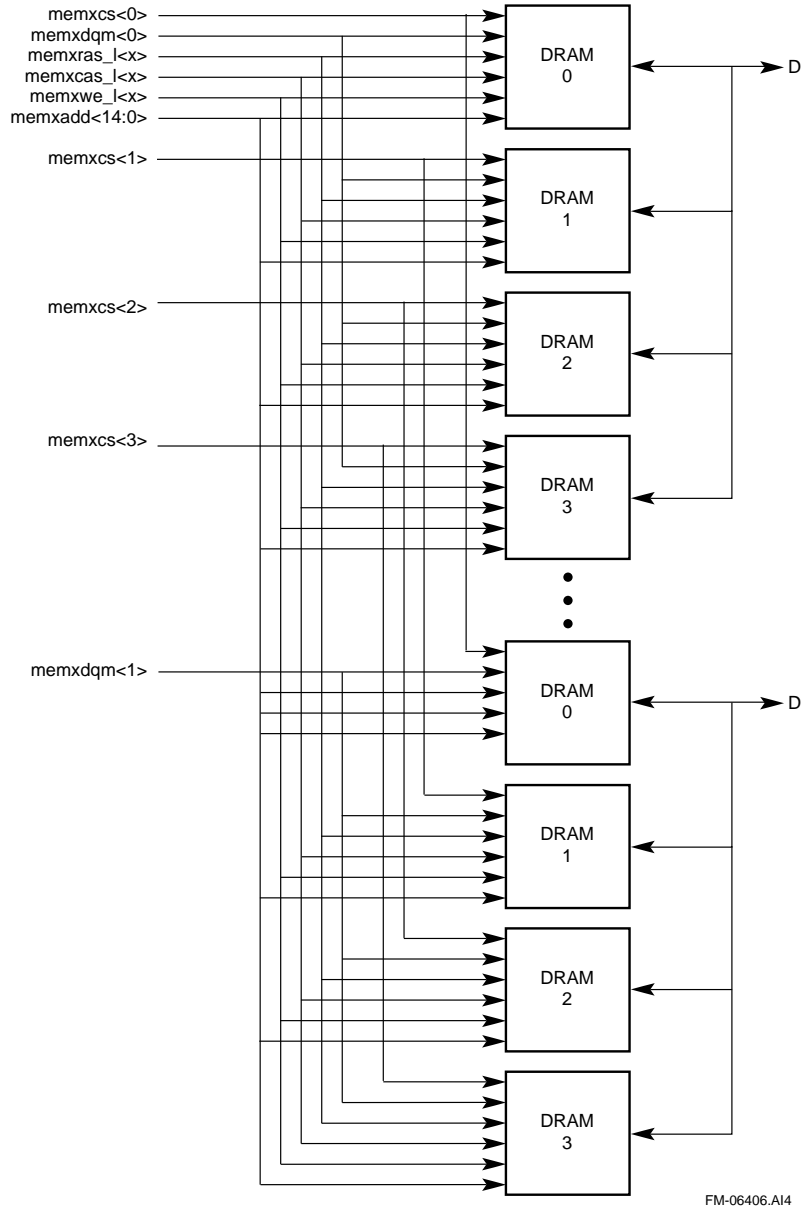
9.4 Supported Array Sizes and DRAM Organizations

The array sizes supported are 16, 32, 64, 128, 256, 512MB, and 1GB. 16MB is only supported in nonsplit arrays. Typhoon also supports 2, 4, and 8GB arrays.

The size S of an array is given by:

Nonsplit array	$S = W \bullet 2^{(B + R + C)}$
Split array	$S = W \bullet 2^{(B + R + C + 1)}$
Twice split array	$S = W \bullet 2^{(B + R + C + 2)}$

where W is the array width, and B, R and C are the number of bank, row address, and column address bits respectively.

Any DRAM organization is supported (irrespective of the DRAM's capacity) provided that the following constraints for B, R, and C are satisfied:

- The array size as calculated from B, R, and C is one of the supported sizes previously described

- $1 \leq B \leq 2$ (Typhoon also supports B=3)

- $11 \leq R \leq 13$

- $8 \leq C \leq 11$

Notes: Many of the supported organizations are not currently manufactured or likely to be manufactured in the future.

Certain organizations might not be available on industry-standard modules even if manufactured.

Certain organizations might not be practical because the chipset can not effectively drive the number of DRAMs that would be required.

Table 9–1 lists Tsunami DRAM organizations supported. Table 9–2 lists Typhoon DRAM organizations supported. The array size for each is a function of:

- The array width

- Whether the array is nonsplit or split or twice-split

Table 9–1 Selected DRAM Organizations Supported (Tsunami Only)

			Array Size			
			NonSplit Array		**Split Array**	
DRAM Organization		$B + R + C$	**16-Byte Array**	**32-Byte Array**	**16-Byte Array**	**32-Byte Array**
16Mb	1M x 16	20	16MB	32MB	32MB	64MB
	2M x 8	21	32MB	64MB	64MB	128MB
	4M x 4	22	64MB	128MB	128MB	256MB
	16M x 1	24	256MB	512MB	512MB	1GB
64Mb	2M x 32	21	32MB	64MB	64MB	128MB
	4M x 16	22	64MB	128MB	128MB	256MB
	8M x 8	23	128MB	256MB	256MB	512MB
	16M x 4	24	256MB	512MB	512MB	1GB
128Mb	4M x 32	22	64MB	128MB	128MB	256MB
	8M x 16	23	128MB	256MB	256MB	512MB
	16M x 8	24	256MB	512MB	512MB	1GB
	32M x 4	25	512MB	1GB	1GB	2GB
256Mb	8M x 32	23	128MB	256MB	256MB	512MB
	16M x 16	24	256MB	512MB	512MB	1GB
	32M x 8	25	512MB	1GB	—	2GB
	64M x 4	26	1GB	—	—	—

Table 9–2 Selected DRAM Organizations Supported (Typhoon Only)

			Array Size					
			NonSplit Array		Split Array		Twice-Split Array	
DRAM Organization		B+R+C	16-Byte Array	32-Byte Array	16-Byte Array	32-Byte Array	16-Byte Array	32-Byte Array
16Mb	1M x 16	20	16MB	32MB	32MB	64MB	64MB[1]	128MB[1]
	2M x 8	21	32MB	64MB	64MB	128MB	128MB[1]	256MB[1]
	4M x 4	22	64MB	128MB	128MB	256MB	256MB[1]	512MB[1]
	16M x 1	24	256MB	512MB	512MB	1GB	1GB[1]	2GB[1]
64Mb	2M x 32	21	32MB	64MB	64MB	128MB	128MB[1]	256MB[1]
	4M x 16	22	64MB	128MB	128MB	256MB	256MB[1]	512MB[1]
	8M x 8	23	128MB	256MB	256MB	512MB	512MB[1]	1GB[1]
	16M x 4	24	256MB	512MB	512MB	1GB	1GB[1]	2GB[1]
128Mb	4M x 32	22	64MB	128MB	128MB	256MB	256MB[1]	512MB[1]
	8M x 16	23	128MB	256MB	256MB	512MB	512MB[1]	1GB[1]
	16M x 8	24	256MB	512MB	512MB	1GB	1GB[1]	2GB[1]
	32M x 4	25	512MB	1GB	1GMB	2GB	2GB[1]	4GB[1]
256Mb	8M x 32	23	128MB	256MB	256MB	512MB	512MB[1]	1GB[1]
	16M x 16	24	256MB	512MB	512MB	1GB	1GB[1]	2GB[1]
	32M x 8	25	512MB	1GB	1GB	2GB[1]	2GB[1]	4GB[1]
	64M x 4	26	1GB	2GB[1]	2GB[1]	4GB[1]	4GB[1]	8GB[1]
	128M x 2	27	2GB[1]	4GB[1]	4GB[1]	8GB[1]	8GB[1]	—

[1] Typhoon only.

9.5 Addressing

Each array supports a contiguous region of the physical memory address space as determined by the base address and array size fields of the array address CSRs. Each region (as programmed) must be naturally aligned, and no two regions may overlap. If arrays are present with different sizes, then firmware can avoid holes in the address space by locating (in the address space) the regions for the smaller arrays above those for the larger arrays.

In Typhoon, all arrays must have at least 256MB aligned addresses, even if they are smaller than 256MB. This means that holes may be unavoidable in a Typhoon system that contains arrays smaller than 256MB.

The following fields are positioned in the physical memory address according to Table 9–3 to Table 9–4.

• Bank bit(s)

- Row address

- Widest supported column address for the array width and values of *B* and *R*

These field extractions are a function of:

- Array width

- *B*

- *R*

The array width is determined by the array width field of the system configuration CSR. *B* and *R* are determined by the DRAM organization fields of the array address CSRs AARx<ROWS> and AARx<BNKS>.

Table 9–3 shows memory array addressing for the Tsunami 21272. Table 9–4 shows memory array addressing for the Typhoon 21274.

Table 9–3 Memory Array Addressing[1] (Tsunami Only)

		Bank Bits			Row Address			Column Address								Minimum Column Bits
B	**R**	2	1	0	12	11	10:0	12	11	10	9	8	7:2	1	0	
16-Byte Array																
1	11			23			22:12	26		25	24	11:6	5	4		8
1	12			23		25	22:12	27		26	24	11:6	5	4		8
1	13			23	26	25	22:12	28		27	24	11:6	5	4		8
2	11		24	23			22:12	27		26	25	11:6	5	4		8
2	12		24	23		25	22:12	28		27	26	11:6	5	4		8
2	13		24	23	26	25	22:12	29		28	27	11:6	5	4		8
32-Byte Array																
1	11			23			22:12	27		26	25	11:6	24	5		8
1	12			23		25	22:12	28		27	26	11:6	24	5		8
1	13			23	26	25	22:12	29		28	27	11:6	24	5		8
2	11		24	23			22:12	28		27	26	11:6	25	5		8
2	12		24	23		25	22:12	29		28	27	11:6	26	5		8
2	13		24	23	26	25	22:12	30		29	28	11:6	27	5		8

[1] Bank and row address bits are transferred one cycle earlier than column address bits and are fixed independent of RAM organization. One cycle is available to select column address bits as a function of CSR settings. Also, the bank bit assignment is selected to keep the width of the minimum column address in the expected range.

Table 9–4 Memory Array Addressing[1] (Typhoon)

	Bank Bits			Row Address			Column Address								Minimum Column Bits
B R	2	1	0	12	11	10:0	12	11	10	9	8	7:2	1	0	
16-Byte Array															
1 11			23			22:12		26	25	24		11:6	5	4	8
1 12			23		25	22:12		27	26	24		11:6	5	4	8
1 13			23	26	25	22:12		28	27	24		11:6	5	4	8
2 11		24	23			22:12		27	26	25		11:6	5	4	8
2 12		24	23		25	22:12		28	27	26		11:6	5	4	8
2 13		24	23	26	25	22:12		29	28	27		11:6	5	4	8
3 11	27	24	23			22:12		28		26	25	11:6	5	4	10 Typhoon only
3 12	27	24	23		25	22:12		29	28		26	11:6	5	4	9 Typhoon only
3 13	27	24	23	26	25	22:12		30	29	28		11:6	5	4	8 Typhoon only
32-Byte Array															
1 11			23			22:12		27	26	25		11:6	24	5	8
1 12			23		25	22:12		28	27	26		11:6	24	5	8
1 13			23	26	25	22:12		29	28	27		11:6	24	5	8
2 11		24	23			22:12		28	27	26		11:6	25	5	8
2 12		24	23		25	22:12		29	28	27		11:6	26	5	8
2 13		24	23	26	25	22:12		30	29	28		11:6	27	5	8
3 11	27	24	23			22:12		29	28		26	11:6	25	5	9 Typhoon only
3 12	27	24	23		25	22:12		30	29	28		11:6	26	5	8 Typhoon only
3 13	27	24	23	26	25	22:12		31	30	29		11:6	28	5	8 Typhoon only

[1] Bank and row address bits are transferred one cycle earlier than column address bits and are fixed independent of RAM organization. One cycle is available to select column address bits as a function of CSR settings. Also, the bank bit assignment is selected to keep the width of the minimum column address in the expected range.

The actual column address bits are the *C* lowest-ordered bits shown for the column address in Table 9–3 in Table 9–4. (The Cchip always drives the widest supported column address to memory.) Due to the mapping of address bits <27:24> it is possible to have a "hole" in the address space if SDRAMs with a small number of column bits are used. The minimum number of column bits supported for each combination of array width, B, and R is noted in Table 9–3 in Table 9–4.

For split arrays, a subarray bit that identifies the subarray is also extracted. This bit is extracted from the bit position shown in Table 9–5 in Table 9–6 (Typhoon) as a function of the array size. This position is one bit higher than the position used for the highest-ordered column address bit (used by memory).

Table 9–5 Position of Subarray Bit (Tsunami Only)

Array Size	Subarray Bit Position CS0 = ~CS1 = CS2 = ~CS3
32MB	24
64MB	25
128MB	26
256MB	27
512MB	28
1GB	29

Table 9–6 Position of Subarray Bits (Typhoon Only)

Array Size	Subarray Bit Position CS0 = ~CS1 = CS2 = ~CS3	Subarray X 2 Bit Position
32MB	24	24:23[1]
64MB	25	25:24[1]
128MB	26	26:25[1]
256MB	27	27:26[1]
512MB	28	28:27[1]
1GB	29	29:28[1]
2GB	30[1]	30:29[1]
4GB	31[1]	31:30[1]
8GB	32[1]	32:31[1]

[1] Typhoon only.

Table 9–7 shows the decode of single-split subarray bit positions into chip select.

Table 9–7 Decode of Single-Split Subarray Bit Position into Chip Select

SA	MemxCS<3>	MemxCS<2>	MemxCS<1>	MemxCS<0>
0	1	0	0	1
1	0	1	1	0

Table 9–8 shows the decode of twice-split subarray bit positions into chip select.

Table 9–8 Decode of Twice-Split Subarray Bit Position into Chip Select (Typhoon)

SA<1>	SA<0>	MemxCS<3>	MemxCS<2>	MemxCS<1>	MemxCS<0>
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

9.6 CPU Address Interface

The Tsunami 21272 Cchip utilizes the "page-mode hit" interface from the 21264. With the address mappings employed, the first two cycles of the CPU address/command transfer determine:

- The array
- The subarray (split arrays only)
- The bank bit(s)
- The row address

The Typhoon 21274 Cchip utilizes the "bank interleave" interface from the 21264. With the address mappings employed, the first two cycles of the CPU address/command transfer determine:

- The array (ignoring the address XORing discussed in Section 9.7)
- The subarray
- The bank bit(s) (ignoring the address XORing discussed in Section 9.7)

Also:

- Each bit of the row and bank address has a fixed position in the physical memory address.
- 64 cache blocks can be accessed at consecutive addresses without leaving page mode (constant row address).

9.7 Address XORing (Typhoon Only)

Address XORing consists of using address bits <7:9> to remap accesses across memory buses, arrays, and banks to avoid resource conflicts and thereby enhance performance. Under certain conditions, these address bits in a request are XORed with the otherwise determined array number and least significant bank bit. In other words, a "virtual array" is first determined by matching against CSRs AARx. Also, the least significant "virtual bank bit" is just address bit <23> that is shown in Table 9–2. The actual array and bank to be accessed are determined using the XOR function of these virtual values with address bits <7:9> of the request. In this manner, a long succession of sequential address accesses are broken into a succession of pairs of cache block accesses (bit <6> toggles once per cache-block pair). Each pair is accessed from a different bus, array, and bank.

This feature is completely disabled if the CSR bit SC<AXD> is asserted. Futhermore, the feature is selectively disabled for address bits <8:7> if the required array sizes are incompatible. (Because all arrays have at least two banks and all banks in an array are of equal size, bit <9> has no such compatibility restriction.) In particular, if all of the arrays to be toggled are of equal size (and thus, have no incompatibilities) and SC<AXD> is off, then:

- Asserted address<7> toggles arrays across memory buses
- Asserted address<8> toggles arrays within buses (sibling arrays)
- Asserted address<9> toggles the least significant bank bit within each array

Table 9–9 indicates exactly which arrays are toggled (between virtual and actual) for each combination of address bits <8:7>. Table 9–9 can also be used to determine which arrays must be of equal size to allow a particular bit to be XORed. For example, if Array0 and Array1 are a different size, then address bit <7> is not used, regardless of SC<AXD>. However, bit <8> may still be used if Array0 = Array2 and Array1 = Array3 in size. If two arrays are both disabled, they are considered to be of equal size for this purpose.

Table 9–9 Array Toggling Due to Address XORing

Address <8:7>	Virtual-to-Actual Bus Toggle	Virtual-to-Actual Array Toggles
00	No	None
01	Yes	Array0 <–> Array1 and Array2 <–> Array3
10	No	Array0 <–> Array2 and Array1 <–> Array3
11	Yes	Array0 <–> Array3 and Array1 <–> Array2

9.8 Bunk and Split Array Addressing

SDRAMs are internally split into two or four banks. Typhoon supports 8 banks. The Cchip groups both banks into two (four for Typhoon) bunks based on the least significant bank bit(s) in the address (bit <23> for Tsunami; bits <24:23> for Typhoon), and can have one row open in each bunk at any time.

For a split array, each bunk spans the two subarrays so that when a row activate command is sent to the DRAMs, the chip select pins to both subarrays are asserted. However, when the read or write command is sent to an open row, only the chip select to the addressed subarray is asserted. If a request is received to a row in a bunk that has a different row already open, the open row is closed and the new row is opened (even if the rows are in different banks in the SDRAMs). Once a row is open, the Cchip considers accesses to that row to be page hits regardless of which subarray the access maps to.

9.9 SDRAM Control Signal Buffering

The control signals to the SDRAMs (**b_mras_l<3:0>**, **b_mcas_l<3:0>**, **b_mwe_l<3:0>**, **b_m***n***dqm<1:0>**, **b_mcke_l<3:0>**, and **b_m***n***a<14:0>**) may be buffered on the module, on the DIMMs, or on both. The system timing CSRs provide for zero, one, or two cycles of buffering on all these signals as a group, as well as a possible additional cycle of buffering on **b_m***n***dqm<1:0>**. The number of these pipeline stages is used to determine the values for STR<IDDW>, STR<IDDR>, and MTR<IRD>. See Section 10.2.4.3 for the formulas to determine these CSR values based on the number of pipeline stages. If the DQM signals have an additional pipeline stage over the other control signals, then MTR<MPD> should be set.

9.10 Serial Presence Detect – CSR MPD

Some SDRAM DIMMs have serial presence detect pins to read data from a serial ROM using the I^2C protocol. The Cchip has two open-drain pins that can be used by software to implement the I^2C protocol and read this information. Access to these pins is provided directly through the Cchip's MPD CSR. The Cchip does not implement an I^2C controller in hardware. The I^2C protocol can address a maximum of eight devices, so it is not possible to connect every DIMM in a system using the 21272 chipset to the bus. Therefore, it is recommended that on a nonsplit-array system, two DIMMs be connected to the I^2C bus in each array; one in the lower 16 bytes of the memory bus and one in the upper 16 bytes of the memory bus. That way, by reading the SROM data firmware, one can determine the size and width of the installed memory DIMMs. In a system design that supports split arrays, one DIMM should be connected to the I^2C bus from each subarray. Firmware should test the width of the installed arrays to determine whether they are 16 bytes or 32 bytes wide. Firmware can use the SROM data to determine which subarrays are installed and how large they are.

9.11 Memory Programming – CSR MPRx

The SDRAMs may need to be programmed after they are powered on. To do so, software can use the Cchip's MPRx CSRs (one per memory array). The programming data supplied by the software is written on the address lines while the Mode Register Set command is delivered to the SDRAMs on the control signals. See Chapter 12 for more information about memory initialization.

9.12 Self Refresh – CSR PWR<SR>

To support power management, the Cchip can be instructed to put the SDRAMs into the self-refresh state by using the CSR bit PWR<SR>. While this bit is set, the Cchip inhibits its standard refresh of the SDRAMs and sends the self-refresh command to all the arrays. This includes deassertion of the **b_mcke_l<3:0>** control signals. SDRAM specifications state that in the cycles after **b_mcke_l<3:0>** is deasserted, other signals to the SDRAMs are ignored. Any SDRAM accesses attempted while PWR<SR> is set will cause UNPREDICTABLE results.

Note: Software must ensure that no DMA operations or CPU memory accesses are in progress whenever PWR<SR> is set. See Chapter 12 for more information about power management.

Self Refresh – CSR PWR<SR>

10

Programmer's Reference

This chapter describes system addressing and the 21272 chipset control and status register (CSR) set.

10.1 System Addressing

This section describes the chipset-implemented translations between several address spaces. System space addresses can be translated to one of the following:

- Pchip0 PCI bus IACK/special operation
- Pchip0 PCI bus memory space
- Pchip0 PCI bus I/O space
- Pchip0 PCI bus configuration space
- Pchip1 PCI bus IACK/special operation
- Pchip1 PCI bus memory space
- Pchip1 PCI bus I/O space
- Pchip1 PCI bus configuration space
- CSR and TIGbus accesses

These translations (from system address to PCI, CSR, or TIG) are collectively referred to as PIO translations and are invoked when the Cchip receives a PIO command from the CPU or a peer-to-peer (PTP) command from a Pchip.

For all other commands (non-PIO from the CPU or DMA from a Pchip), system addresses are used without translation to access system memory.

The CPU determines whether to send a PIO operation by examining bit <43> of its internal address. The Cchip does not see bit <43>; it only sees if the operation from the CPU is PIO or not.

PCI memory space addresses (from a PCI bus on one of the Pchips) are translated into system space addresses using window registers. If the window does not specify PTP, a DMA operation takes place to system memory without further translation. If the window does specify PTP, the system space address is translated back to PCI memory space on the other Pchip's PCI bus so that a PTP operation can take place. Any other PIO translation (from PCI bus to system to PIO) is illegal and results in UNPREDICTABLE results.

10.1.1 System Space and Address Map

The system address space is divided into two parts: system memory and PIO. This division is indicated by physical memory bit <43> = 1 for PIO accesses from the CPU, and by the PTP bit in the window registers for PTP accesses from the Pchip. While the operating system may choose bit <40> instead of bit <43> to represent PIO space, bit <43> is used throughout this chapter. In general, bits <42:35> are don't cares if bit <43> is asserted.

There is 16GB of PIO space available on the 21272 chipset with 8GB assigned to each Pchip. The Pchip supports up to bit <34> (35 bits total) of system address. However, the Version 1 Cchip only supports 4GB of system memory (32 bits total). As described in Chapter 6, the CAPbus protocol between the Pchip and Cchip does support up to bit <34>, as does the Cchip's interface to the CPU. The Typhoon Cchip supports 32GB of system memory (35 bits total).

The system address space is divided as shown in Table 10–1.

Table 10–1 System Address Map

Space	Size	System Address <43:0>	Comments
System memory	4GB	000.0000.0000 – 000.FFFF.FFFF	Cacheable and prefetchable.
Reserved	8188GB	001.0000.0000 – 7FF.FFFF.FFFF	—
Pchip0 PCI memory	4GB	800.0000.0000 – 800.FFFF.FFFF	Linear addressing.
TIGbus	1GB	801.0000.0000 – 801.3FFF.FFFF	**addr<5:0>** = 0. Single byte valid in quadword access. 16MB accessible.
Reserved	1GB	801.4000.0000 – 801.7FFF.FFFF	—
Pchip0 CSRs	256MB	801.8000.0000 – 801.8FFF.FFFF	**addr<5:0>** = 0. Quadword access.
Reserved	256MB	801.9000.0000 – 801.9FFF.FFFF	—
Cchip CSRs	256MB	801.A000.0000 – 801.AFFF.FFFF	**addr<5:0>** = 0. Quadword access.
Dchip CSRs	256MB	801.B000.0000 – 801.BFFF.FFFF	**addr<5:0>** = 0. All eight bytes in quadword access must be identical.
Reserved	768MB	801.C000.0000 – 801.EFFF.FFFF	—
Reserved	128MB	801.F000.0000 – 801.F7FF.FFFF	—
PCI IACK/ special Pchip0	64MB	801.F800.0000 – 801.FBFF.FFFF	Linear addressing.
Pchip0 PCI I/O	32MB	801.FC00.0000 – 801.FDFF.FFFF	Linear addressing.
Pchip0 PCI configuration	16MB	801.FE00.0000 – 801.FEFF.FFFF	Linear addressing.
Reserved	16MB	801.FF00.0000 – 801.FFFF.FFFF	—

Table 10–1 System Address Map   (Continued)

Space	Size	System Address <43:0>	Comments
Pchip1 PCI memory	4GB	802.0000.0000 – 802.FFFF.FFFF	Linear addressing.
Reserved	2GB	803.0000.0000 – 803.7FFF.FFFF	—
Pchip1 CSRs	256MB	803.8000.0000 – 803.8FFF.FFFF	**addr<5:0>** = 0, quadword access.
Reserved	1536MB	803.9000.0000 – 803.EFFF.FFFF	—
Reserved	128MB	803.F000.0000 – 803.F7FF.FFFF	—
PCI IACK/ special Pchip1	64MB	803.F800.0000 – 803.FBFF.FFFF	Linear addressing.
Pchip1 PCI I/O	32MB	803.FC00.0000 – 803.FDFF.FFFF	Linear addressing.
Pchip1 PCI configuration	16MB	803.FE00.0000 – 803.FEFF.FFFF	Linear addressing.
Reserved	16MB	803.FF00.0000 – 803.FFFF.FFFF	—
Reserved	8172GB	804.0000.0000 – FFF.FFFF.FFFF	Bits <42:35> are don't cares if bit <43> is asserted.

10.1.2 PCI Space

PCI space has 4GB of memory space, 32MB of I/O space, and 16MB of configuration space. If two Pchips are present, then each Pchip has its own set of PCI space. PCI memory transactions are directed to the 4GB memory space. PCI I/O transactions are directed to the 32MB I/O space. PCI configuration transactions are directed to the 16MB configuration space.

10.1.2.1 PCI Memory Space

Accesses to a Pchip's PCI memory space, from either the CPU or the other Pchip, are linearly mapped into the 4GB space. Dual-address cycle access to the PCI bus is not supported from the CPU or the other Pchip.

Each Pchip can also have up to five regions in PCI memory space that map to system memory space. This is DMA space, and the regions are referred to as DMA windows. PCI devices can access system memory by means of these windows. There can be four standard DMA windows and one DMA monster window. The DMA monster window is enabled by PCTL<MWIN>. If enabled, this window lies from 100.0000.0000 to 100.FFFF.FFFF, which requires a dual-address cycle (DAC) access from the PCI bus. This window maps to system memory as defined in Section 10.1.4. Because the Cchip's interface to the CPU and the CAPbus protocol between the Pchip and Cchip only support 35 bits of addressing, and because the DMA monster window is direct-mapped, there is not a unique PCI address from this 1Terabyte DMA window that specifies the same 32GB region in system memory. The ordinary DMA windows are enabled and defined by the WSBA*n*, WSM*n*, and TBA*n* CSRs. These windows map to system memory as defined in Section 10.1.4. Window 0, Window 1, and Window 2 are never DAC capable, while Window 3 can be enabled to be DAC capable. If Window 3 is enabled to be DAC capable, it is accessed by PCI addresses that range from 800.0000.0000 to FFF.FFFF.FFFF. Again, the translated system address is limited to a maximum range of 35 bits.

10.1.3 PIO Address Translation (System-to-PCI)

The CPU sends the following commands to the Cchip:

- PIO RdBytes (byte mask)

- PIO RdLWs (longword mask)

- PIO RdQWs (quadword mask)

- PIO WrBytes (byte mask)

- PIO WrLWs (longword mask)

- PIO WrQWs (quadword mask)

The CPU issues the above operations when a read or write operation has physical address bit <43> equal to 1. If the access is to PCI memory space, a PCI memory read or PCI memory write command is issued. If the access is to PCI I/O space, a PCI I/O read or PCI I/O write command is issued. If the access is to PCI configuration space, a PCI configuration read or PCI configuration write command is issued. If the access is a read to PCI IACK/special space, a PCI IACK command is issued. If the access is a write to PCI IACK/special space, a PCI special cycle command is issued. If the read is to CSR space or TIGbus space, the appropriate CSR or TIGbus data is transferred. (The Pchips are not involved in Dchip CSR transfers. The Pchips are involved in Cchip CSR or TIGbus transfers because there is no path for data between the CPU and the Cchip.)

10.1.3.1 Linear Memory Space Translation

A CPU read or write to this space causes a memory read or memory write command, respectively, on the PCI bus. For byte-oriented accesses, the CPU requests only one byte or one word at a time. A longword-oriented access can be up to eight longwords, and a quadword-oriented access can be up to eight quadwords (one cache block).

Figure 10–1 shows how a PCI memory space address is generated from an address in linear memory space. Table 10–2 details the generation of PCI address bits **b_ad<2:0>**, and the PCI command and byte enable bits **b_cbe_l<7:0>**. However, in PCI memory space, PCI **b_ad<1:0>** = 00, and in 64-bit mode PCI **b_ad<2>** = 0 as well.

Figure 10–1 Linear Memory Address Translation

CPU Address

43		32	31		3
1000 0000 0000					

Mask

From Table

PCI Memory Address

31		3	2		0

PCI BE
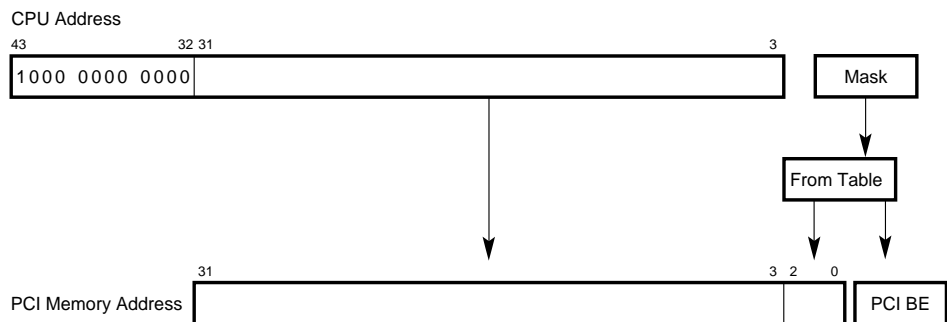
LJ-05516.AI4

Table 10–2 Generation of PCI b_ad<2:0> and PCI b_cbe_l<7:0> from Linear I/O Address

Type	Mask	ad<2:0> (64-Bit)[1]	cbe_l<7:0> (64-Bit)	ad<2:0> (32-Bit)[1]	cbe_l<3:0> (32-Bit)
Byte	0000.0001	000	1111.1110	000	1110
Byte	0000.0010	001	1111.1101	001	1101
Byte	0000.0100	010	1111.1011	010	1011
Byte	0000.1000	011	1111.0111	011	0111
Byte	0001.0000	100	1110.1111	100	1110
Byte	0010.0000	101	1101.1111	101	1101
Byte	0100.0000	110	1011.1111	110	1011
Byte	1000.0000	111	0111.1111	111	0111
Word	0000.0011	000	1111.1100	000	1100
Word	0000.1100	010	1111.0011	010	0011
Word	0011.0000	100	1100.1111	100	1100
Word	1100.0000	110	0011.1111	110	0011
Longword	*xxxx.xxx*1	000	*xxxx*.0000	000	0000
Longword	*xxxx.xx*10	100	0000.1111	100	0000
Longword	*xxxx.x*100	000	*xxxx*.0000	000	0000
Longword	*xxxx*.1000	100	0000.1111	100	0000
Longword	*xxx*1.0000	000	*xxxx*.0000	000	0000
Longword	*xx*10.0000	100	0000.1111	100	0000
Longword	*x*100.0000	000	*xxxx*.0000	000	0000
Longword	1000.0000	100	0000.1111	100	0000
Quadword	*xxxx.xxxx*	000	0000.0000	000	0000

[1] The difference between memory space and I/O space is that the lower address bits are not used in memory space (<1:0> in 32-bit mode, <2:0> in 64-bit mode).

10.1.3.2 Linear I/O Space Translation

A CPU read or write to linear I/O address space causes an I/O read or I/O write command, respectively, on the PCI bus. For byte-oriented accesses, the CPU requests only one byte or one word at a time. A longword-oriented access can be up to eight longwords, and a quadword-oriented access can be up to eight quadwords (one cache block).

Figure 10–2 shows how a PCI address is generated from an address in linear I/O space. Table 10–2 details the generation of PCI address bits **b_ad<2:0>**, and the PCI command and byte enable bits **b_cbe_l<7:0>**.

Figure 10–2 Linear I/O Address Translation

CPU Address

43	25 24		3		
1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0					Mask

From Table

31	25 24		3 2 0		
PCI Memory Address	0 0 0 0 0 0 0				PCI BE

LJ-05517.AI4
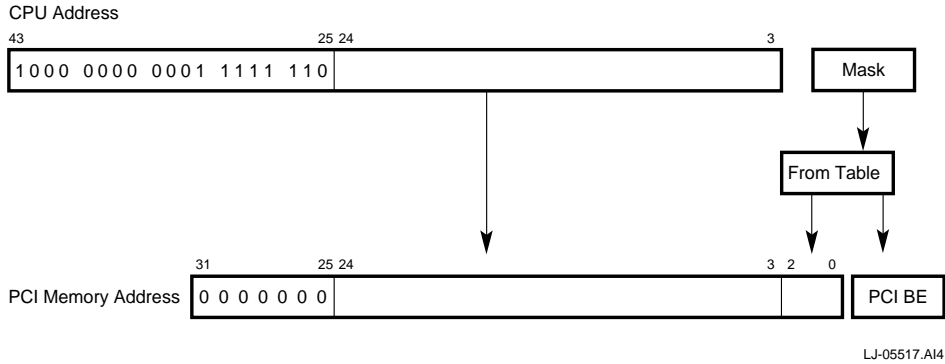
10.1.3.3 Linear Configuration Space Translation

A CPU read or write to linear configuration address space causes a configuration read or configuration write command, respectively, on the PCI bus. There are two types of configuration cycles: Type 0 and Type 1. Refer to Section 8.5 for details on when each is generated.

If the CPU requests more than one longword transfer, the Pchip uses a burst transaction on the PCI bus. Configuration read cycles that do not receive a **b_devsel_l** return all 1s data to the CPU and do not flag an error. Figure 10–3 shows how a Type 0 PCI configuration command is generated from an address in linear configuration space.

Figure 10–3 Converting Linear Configuration Address to Type 0 PCI Configuration Cycle

CPU Address

43	24 23	16 15	11 10	8 7	3		
1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0	Bus # not=0	Device #	Func.	Register #		Mask	

Decode   From Table

31		11 10	8 7	3 2 1 0		
PCI Memory Address	IDSEL: See Table	Func.	Register #	# 0 0	PCI BE

LJ-05518.AI4
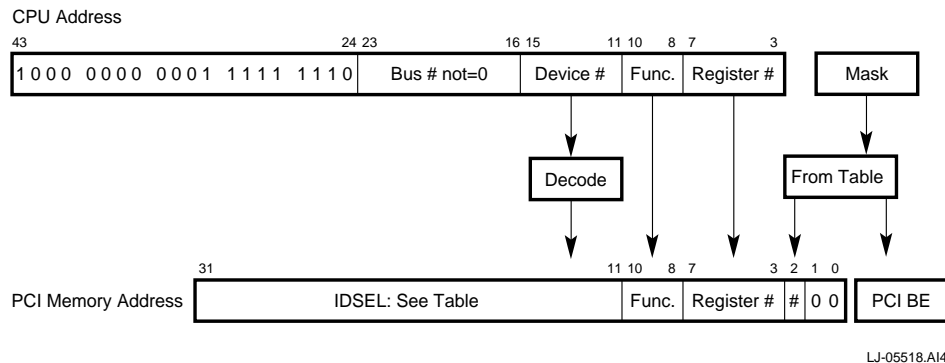
Table 10–3 lists the generation of the individual device IDSEL bits from the PCI device number. Figure 10–4 shows how a Type 1 PCI configuration command is generated from an address in linear configuration space. Table 10–4 shows how the LSB of the register # field and the PCI byte enables from the mask field are generated.

Table 10–3 Decode of Device # to Generate IDSEL

Device# <15:11>	IDSEL <31:11>
0.0000	0.0000.0000.0000.0000.0001
0.0001	0.0000.0000.0000.0000.0010
0.0010	0.0000.0000.0000.0000.0100
0.0011	0.0000.0000.0000.0000.1000

Table 10–3 Decode of Device # to Generate IDSEL (Continued)

Device# <15:11>	IDSEL <31:11>
0.0100	0.0000.0000.0000.0001.0000
0.0101	0.0000.0000.0000.0010.0000
0.0110	0.0000.0000.0000.0100.0000
0.0111	0.0000.0000.0000.1000.0000
0.1000	0.0000.0000.0001.0000.0000
0.1001	0.0000.0000.0010.0000.0000
0.1010	0.0000.0000.0100.0000.0000
0.1011	0.0000.0000.1000.0000.0000
0.1100	0.0000.0001.0000.0000.0000
0.1101	0.0000.0010.0000.0000.0000
0.1110	0.0000.0100.0000.0000.0000
0.1111	0.0000.1000.0000.0000.0000
1.0000	0.0001.0000.0000.0000.0000
1.0001	0.0010.0000.0000.0000.0000
1.0010	0.0100.0000.0000.0000.0000
1.0011	0.1000.0000.0000.0000.0000
1.0100	1.0000.0000.0000.0000.0000
1.0110 – 1.1111	0.0000.0000.0000.0000.0000

Figure 10–4 Converting Linear Configuration Address to Type 1 PCI Configuration Cycle

LJ-05519.AI4
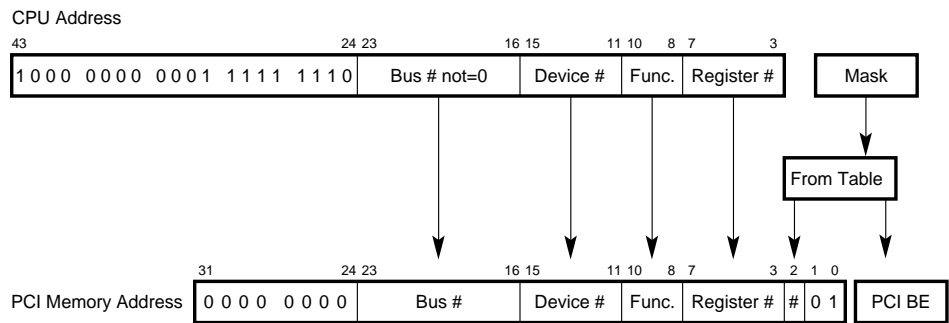
Table 10–4 Generating Configuration Register # LSB and CBE from Mask and Data Type

Type	Mask	PCA ad<2> (Register # LSB)	PCI cbe_l<3:0>
Byte	0000.0001	0	1110
Byte	0000.0010	0	1101
Byte	0000.0100	0	1011
Byte	0000.1000	0	0111
Byte	0001.0000	1	1110
Byte	0010.0000	1	1101
Byte	0100.0000	1	1011
Byte	1000.0000	1	0111
Word	0000.0011	0	1100
Word	0000.1100	0	0011
Word	0011.0000	1	1100
Word	1100.0000	1	0011
Longword	*xxxx.xxx*1	0	0000
Longword	*xxxx.xx*10	1	0000
Longword	*xxxx.x*100	0	0000
Longword	*xxxx.*1000	1	0000
Longword	*xxx*1.0000	0	0000
Longword	*xx*10.0000	1	0000
Longword	*x*100.0000	0	0000
Longword	1000.0000	1	0000
Quadword	*xxxx.xxxx*	0	0000

10.1.3.4 Linear IACK/Special Cycle Space Translation

A CPU read to linear IACK/special cycle space causes an interrupt acknowledge cycle on the PCI bus. The PCI address is a "don't care". A CPU write to this space causes a special cycle on the PCI bus. The PCI address and byte enables are "don't cares". The data will be the same as the data if this were a normal write.

10.1.3.5 CSR Space Translation

A CPU read or write to CSR space causes CSR data to be moved to or from the CPU, respectively. Accesses to this space are to a single quadword aligned to a cache block only. Figure 10–5 shows the CSR address space translation.

Note: Firmware accessing 21272 CSR space must use only the STQ and LDQ instruction so that the CPU uses only the PIO WrQws and PIO RdQws on the interface. Other instructions will cause UNPREDICTABLE results.

Figure 10–5 CSR Space Address Translation
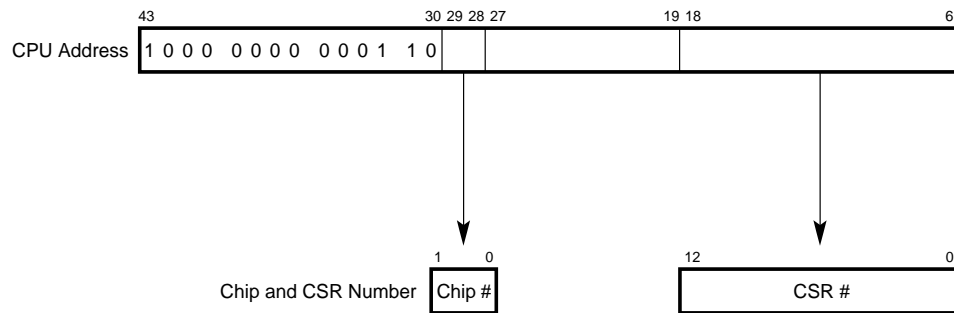
LJ-05520.AI4

10.1.3.6 TIGbus Space Translation

A CPU read or write to TIGbus space causes a read or write, respectively, of a device on the TIGbus. An access to this space is only to a byte aligned to a cache block. **addr<29:6>** are sent out as **tigadr<23:0>**.

Note: Firmware accessing 21272 TIGbus space must use only the STQ and LDQ instruction so that the CPU uses only the PIO WrQws and PIO RdQws on the interface. Other instructions will cause UNPREDICTABLE results.

10.1.4 DMA Address Translation (PCI-to-System)

The 21272 chipset supports some PCI commands as a target and does not support (ignores) others as a target. The Pchip does not respond as a target when it acts as a PCI master.

The Pchip ignores all of the following commands as a target:

- Interrupt acknowledge
- Special cycle
- I/O read
- I/O write
- Configuration read
- Configuration write

The Pchips may respond to the following commands as a target:

- Memory read

- Memory read line

- Memory write

- Memory write and invalidate – The Pchip requests a DMA N QW read or write, respectively, when an incoming PCI address lies inside one of the four DMA windows or in the DMA monster window, and the matching window does not specify PTP. Otherwise, a PTP read or write request will be made. In either case, for a memory read line, the request is made for data until the end of the cache block boundary (64 bytes).

- Memory read multiple – The Pchip requests a DMA N QW read when the PCI command hits in one of the four DMA windows or in the DMA monster window, and the matching window does not specify PTP. The Pchip prefetches data by continuing to issue DMA N QW read commands until the transaction ends. If the matching window does specify PTP, a PTP read request will be made to the end of the cache block boundary (64 bytes).

- Dual-address cycle – This command is accepted by the Pchip when the address lies inside the DMA monster window.

There are two kinds of DMA address translation: direct mapped and scatter-gather mapped. Each type starts by comparing the incoming PCI address with the monster window (if it is enabled and if it is a DAC), and with the four window base and window mask registers (the window base registers also have an enable window bit and a scatter-gather enable bit). This process is shown in Figure 10–6.

Figure 10–6 Determining if PCI Address Is Valid DMA Address (One of Four Windows)
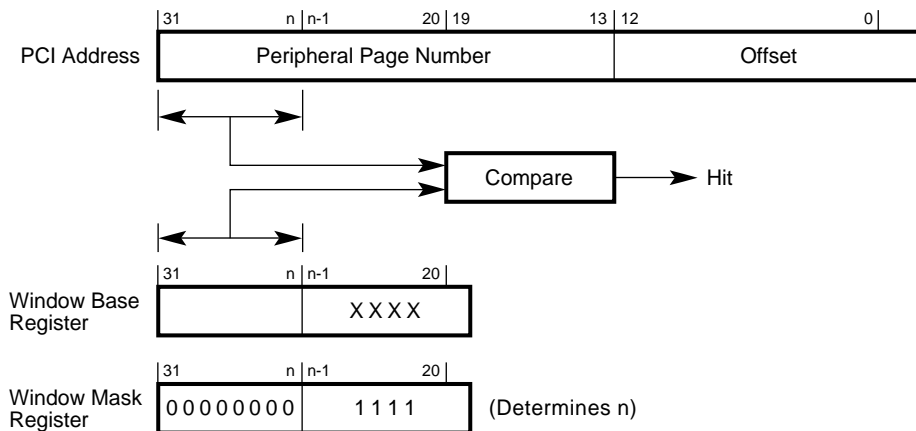
LJ-05521.AI4

If the address resides in one of the windows, and the window is enabled, then if the scatter-gather enable bit is set, the translation is as described in Section 10.1.4.3. Otherwise, the translation described in Section 10.1.4.2 is used.

In addition, if the matching window has the PTP bit set, then the result of the address translation is treated as if it had bit <43> set. That is, it is treated like a PIO address from the CPU. Otherwise, the address is a system memory address.

10.1.4.1 Window Hole

All window registers are simultaneously subject to a hole that inhibits matching, under the control of the PCTL<HOLE> CSR bit described in Section 10.2.5.4. If that bit is set, the hole is enabled in all windows and has the following extent:

- From PCI address base 512K (address<31:0> = 0008.0000)

- To PCI address limit 1M–1 (address<31:0> = 000F.FFFF)

If enabled, the hole applies whether or not the PTP bit is set for the window.

10.1.4.2 Direct-Mapped DMA Address Translation

Direct-mapped addressing uses a base address register, a translated base address (TBA) register, and a mask register. The block of PCI addresses at base address, of a size as determined by the mask register, is translated to a block of addresses at translated base address (see Table 10–5). Values in the WSM*n* field other than those shown produce unspecified results.

Table 10–5 PCI DMA Address to System Address Via Direct Mapping

Window Size	WSM*n*<31:20>	Translated Address <34:2>
1MB	0000.0000.0000	TBA<34:20>:**ad<19:2>**
2MB	0000.0000.0001	TBA<34:21>:**ad<20:2>**
4MB	0000.0000.0011	TBA<34:22>:**ad<21:2>**
8MB	0000.0000.0111	TBA<34:23>:**ad<22:2>**
16MB	0000.0000.1111	TBA<34:24>:**ad<23:2>**
32MB	0000.0001.1111	TBA<34:25>:**ad<24:2>**
64MB	0000.0011.1111	TBA<34:26>:**ad<25:2>**
128MB	0000.0111.1111	TBA<34:27>:**ad<26:2>**
256MB	0000.1111.1111	TBA<34:28>:**ad<27:2>**
512MB	0001.1111.1111	TBA<34:29>:**ad<28:2>**
1GB	0011.1111.1111	TBA<34:30>:**ad<29:2>**
2GB	0111.1111.1111	TBA<34:31>:**ad<30:2>**
4GB	N/A	000:**ad<34:2>** (monster window only)

10.1.4.3 Scatter-Gather DMA Address Translation

Scatter-gather addressing uses a base address register, a mask register, a translated base address register, and a page table entry (PTE) in system memory. An 8KB page of PCI addresses at base address is translated to an 8KB page of system addresses through one level of indirection. The PTE contains the address of the 8KB page.

To improve performance, fetches of PTEs by the Pchip are given priority on the Cchip, and are allowed to pass earlier DMA write operations to system memory. Therefore, the following rule applies:

Rule: System software *must not* depend on the order of completion of DMA writes to a page table with respect to fetches of PTEs from that table for the purpose of scatter-gather translation.

At TBA is a region (of size SG PTE AREA) of PTEs, each of which is eight bytes. Bits <22:1> of the PTE become bits <34:13> (the 8KB page) of the system address, and bits <12:0> of the PCI address become bits <12:0> (the page offset) of the system address.

Table 10–6 shows how the address of the page table entry (to be used as part of the final system address) is generated. Values in the WSM field other than those shown produce unspecified results.

Table 10–6 Generating PTE Address from PCI DMA Address Via Scatter-Gather Mapping

Window Size	SG PTE AREA	WSM*n*<31:20>	PTE Address <34:3>
1MB	1KB	0000.0000.0000	TBA<34:10>:**ad<19:13>**
2MB	2KB	0000.0000.0001	TBA<34:11>:**ad<20:13>**
4MB	4KB	0000.0000.0011	TBA<34:12>:**ad<21:13>**
8MB	8KB	0000.0000.0111	TBA<34:13>:**ad<22:13>**
16MB	16KB	0000.0000.1111	TBA<34:14>:**ad<23:13>**
32MB	32KB	0000.0001.1111	TBA<34:15>:**ad<24:13>**
64MB	64KB	0000.0011.1111	TBA<34:16>:**ad<25:13>**
128MB	128KB	0000.0111.1111	TBA<34:17>:**ad<26:13>**
256MB	256KB	0000.1111.1111	TBA<34:18>:**ad<27:13>**
512MB	512KB	0001.1111.1111	TBA<34:19>:**ad<28:13>**
1GB	1MB	0011.1111.1111	TBA<34:20>:**ad<29:13>**
2GB	2MB	0111.1111.1111	TBA<34:21>:**ad<30:13>**
4GB	4MB	N/A	TBA<34:22>:**ad<31:13>** (Window 3 in DAC mode only)

Figure 10–7 shows the structure of a page table entry in memory. If either bit <31> or bit <28> is set, the address is interpreted as being a peer-to-peer address.

Figure 10–7 Scatter-Gather Page Table Entry in Memory

LJ-05522A-AI4
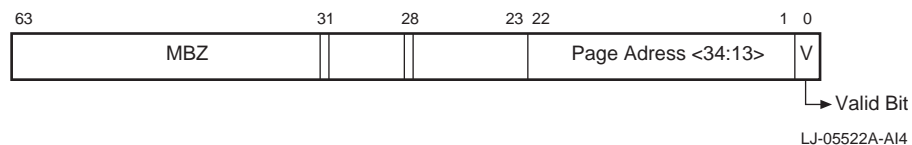
Figure 10–8 shows how a page table entry is used in conjunction with an incoming PCI address to generate a system address.

Figure 10–8 Generating System Address from Scatter-Gather PTE
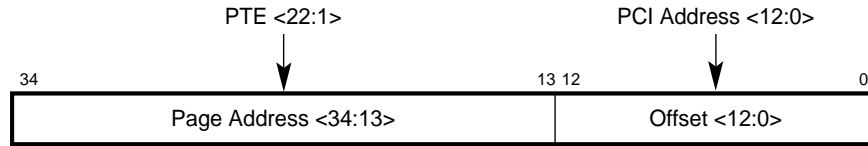
LJ-05523.AI4

10.1.4.4 Monster Window DMA Address Translation

In case of a PCI dual-address cycle command, the high-order PCI address bits <63:40> are compared to the constant value $0x0000\_01$ (that is, bit <40> = 1; all other bits = 0). If these bits match, a monster window hit has occurred and the low-order PCI address bits <34:0> are used unchanged as the system address bits <34:0>. PCI address bits <39:35> are ignored. The high-order 32 PCI address bits are available on **b_ad<31:0>** in the second cycle of a DAC, and also on **b_ad<63:32>** in the first cycle of a DAC if **b_req64_l** is asserted.

10.2 Chipset Registers

The following sections describe the control and status (CSR) register set. All 21272 CSRs are accessed 64 bits wide, although in many cases fewer than 64 bits are implemented. The registers are addressed on 64-byte boundaries. For definitions of abbreviations, refer to the Preface.

Note: Programmers must write only zeros to register fields identified as RES.

10.2.1 Register Addresses

Table 10–7 lists all Tsunami internal chipset registers with their system address.

Table 10–7 Chipset Register Addresses (Tsunami Only)

Register	Address	Type	Register Number
Cchip Registers			
CSC	801.A000.0000	RW	00
MTR	801.A000.0040	RW	01
MISC	801.A000.0080	RW	02
MPD	801.A000.00C0	RW	03
AAR0	801.A000.0100	RW	04
AAR1	801.A000.0140	RW	05
AAR2	801.A000.0180	RW	06
AAR3	801.A000.01C0	RW	07
DIM0	801.A000.0200	RW	08
DIM1	801.A000.0240	RW	09
DIR0	801.A000.0280	RO	0A
DIR1	801.A000.02C0	RO	0B

Table 10–7 Chipset Register Addresses (Tsunami Only) (Continued)

Register	Address	Type	Register Number
DRIR	801.A000.0300	RO	0C
PRBEN	801.A000.0340	Special	0D
IIC0	801.A000.0380	RW	0E
IIC1	801.A000.03C0	RW	0F
MPR0	801.A000.0400	WO	10
MPR1	801.A000.0440	WO	11
MPR2	801.A000.0480	WO	12
MPR3	801.A000.04C0	WO	13
Reserved	801.A000.0500	RW	14
TTR	801.A000.0580	RW	16
TDR	801.A000.05C0	RW	17
Dchip Registers			
DSC	801.B000.0800	RO	20
STR	801.B000.0840	RW	21
DREV	801.B000.0880	RW	22
DSC2	801.B000.008C0	RO	23 (Reserved for future use)
Pchip Registers			
P0–WSBA0	801.8000.0000	RW	00
P0–WSBA1	801.8000.0040	RW	01
P0–WSBA2	801.8000.0080	RW	02
P0–WSBA3	801.8000.00C0	RW	03
P0–WSM0	801.8000.0100	RW	04
P0–WSM1	801.8000.0140	RW	05
P0–WSM2	801.8000.0180	RW	06
P0–WSM3	801.8000.01C0	RW	07
P0–TBA0	801.8000.0200	RW	08
P0–TBA1	801.8000.0240	RW	09
P0–TBA2	801.8000.0280	RW	0A
P0–TBA3	801.8000.02C0	RW	0B
P0–PCTL	801.8000.0300	RW	0C
P0–PLAT	801.8000.0340	RW	0D
P0–RES	801.8000.0380	RW	0E
P0–PERROR	801.8000.03C0	RW	0F
P0–PERRMASK	801.8000.0400	RW	10

Table 10–7 Chipset Register Addresses (Tsunami Only) (Continued)

Register	Address	Type	Register Number
P0–PERRSET	801.8000.0440	WO	11
P0–TLBIV	801.8000.0480	WO	12
P0–TLBIA	801.8000.04C0	WO	13
P0–PMONCTL	801.8000.0500	RW	14
P0–PMONCNT	801.8000.0540	RO	15
P0-SPRST	801.8000.0800	WO	20
P1–WSBA0	803.8000.0000	RW	00
P1–WSBA1	803.8000.0040	RW	01
P1–WSBA2	803.8000.0080	RW	02
P1–WSBA3	803.8000.00C0	RW	03
P1–WSM0	803.8000.0100	RW	04
P1–WSM1	803.8000.0140	RW	05
P1–WSM2	803.8000.0180	RW	06
P1–WSM3	803.8000.01C0	RW	07
P1–TBA0	803.8000.0200	RW	08
P1–TBA1	803.8000.0240	RW	09
P1–TBA2	803.8000.0280	RW	0A
P1–TBA3	803.8000.02C0	RW	0B
P1–PCTL	803.8000.0300	RW	0C
P1–PLAT	803.8000.0340	RW	0D
P1–RES	803.8000.0380	RW	0E
P1–PERROR	803.8000.03C0	RW	0F
P1–PERRMASK	803.8000.0400	RW	10
P1–PERRSET	803.8000.0440	WO	11
P1–TLBIV	803.8000.0480	WO	12
P1–TLBIA	803.8000.04C0	WO	13
P1–PMONCTL	803.8000.0500	RW	14
P1–PMONCNT	803.8000.0540	RO	15
P1-SPRTS	803.8000.0800	WO	20

Table 10–8 lists all Typhoon internal chipset registers with their system address.

Table 10–8 Chipset Register Addresses (Typhoon Only)

Register	Address	Type	Register Number
Cchip Registers			
CSC	801.A000.0000	RW	00
MTR	801.A000.0040	RW	01
MISC	801.A000.0080	RW	02
MPD	801.A000.00C0	RW	03
AAR0	801.A000.0100	RW	04
AAR1	801.A000.0140	RW	05
AAR2	801.A000.0180	RW	06
AAR3	801.A000.01C0	RW	07
DIM0	801.A000.0200	RW	08
DIM1	801.A000.0240	RW	09
DIR0	801.A000.0280	RO	0A
DIR1	801.A000.02C0	RO	0B
DRIR	801.A000.0300	RO	0C
PRBEN	801.A000.0340	Special	0D
IIC0	801.A000.0380	RW	0E
IIC1	801.A000.03C0	RW	0F
MPR0	801.A000.0400	WO	10
MPR1	801.A000.0440	WO	11
MPR2	801.A000.0480	WO	12
MPR3	801.A000.04C0	WO	13
Reserved	801.A000.0500	RW	14
TTR	801.A000.0580	RW	16
TDR	801.A000.05C0	RW	17
DIM2	801.A000.0600	RW	18 (Typhoon only)
DIM3	801.A000.0640	RW	19 (Typhoon only)
DIR2	801.A000.0680	RO	1A (Typhoon only)
DIR3	801.A000.06C0	RO	1B (Typhoon only)
IIC2	801.A000.0700	RW	1C (Typhoon only)
IIC3	801.A000.0740	RW	1D (Typhoon only)
PWR	801.A000.0780	RW	1E (Typhoon only)
Cchip (reserved)	801.A000.07C0–0BC0		1F–2F
CMONCTLA	801.A000.0C00	RW	30 (Typhoon only)

Table 10–8 Chipset Register Addresses (Typhoon Only) (Continued)

Register	Address	Type	Register Number
CMONCTLB	801.A000.0C40	RW	31 (Typhoon only)
CMONCNT01	801.A000.0C80	RO	32(Typhoon only)
CMONCNT23	801.A000.0CC0	RO	33 (Typhoon only)
Dchip Registers			
DSC	801.B000.0800	RO	20
STR	801.B000.0840	RW	21
DREV	801.B000.0880	RW	22
DSC2	801.B000.008C0	RO	23 (Reserved for future use)
Pchip Registers			
P0–WSBA0	801.8000.0000	RW	00
P0–WSBA1	801.8000.0040	RW	01
P0–WSBA2	801.8000.0080	RW	02
P0–WSBA3	801.8000.00C0	RW	03
P0–WSM0	801.8000.0100	RW	04
P0–WSM1	801.8000.0140	RW	05
P0–WSM2	801.8000.0180	RW	06
P0–WSM3	801.8000.01C0	RW	07
P0–TBA0	801.8000.0200	RW	08
P0–TBA1	801.8000.0240	RW	09
P0–TBA2	801.8000.0280	RW	0A
P0–TBA3	801.8000.02C0	RW	0B
P0–PCTL	801.8000.0300	RW	0C
P0–PLAT	801.8000.0340	RW	0D
P0–RES	801.8000.0380	RW	0E
P0–PERROR	801.8000.03C0	RW	0F
P0–PERRMASK	801.8000.0400	RW	10
P0–PERRSET	801.8000.0440	WO	11
P0–TLBIV	801.8000.0480	WO	12
P0–TLBIA	801.8000.04C0	WO	13
P0–PMONCTL	801.8000.0500	RW	14
P0–PMONCNT	801.8000.0540	RO	15
P0-SPRST	801.8000.0800	WO	20
P1–WSBA0	803.8000.0000	RW	00
P1–WSBA1	803.8000.0040	RW	01

Table 10–8 Chipset Register Addresses (Typhoon Only) (Continued)

Register	Address	Type	Register Number
P1–WSBA2	803.8000.0080	RW	02
P1–WSBA3	803.8000.00C0	RW	03
P1–WSM0	803.8000.0100	RW	04
P1–WSM1	803.8000.0140	RW	05
P1–WSM2	803.8000.0180	RW	06
P1–WSM3	803.8000.01C0	RW	07
P1–TBA0	803.8000.0200	RW	08
P1–TBA1	803.8000.0240	RW	09
P1–TBA2	803.8000.0280	RW	0A
P1–TBA3	803.8000.02C0	RW	0B
P1–PCTL	803.8000.0300	RW	0C
P1–PLAT	803.8000.0340	RW	0D
P1–RES	803.8000.0380	RW	0E
P1–PERROR	803.8000.03C0	RW	0F
P1–PERRMASK	803.8000.0400	RW	10
P1–PERRSET	803.8000.0440	WO	11
P1–TLBIV	803.8000.0480	WO	12
P1–TLBIA	803.8000.04C0	WO	13
P1–PMONCTL	803.8000.0500	RW	14
P1–PMONCNT	803.8000.0540	RO	15
P1-SPRTS	803.8000.0800	WO	20

10.2.2 Cchip CSRs

Section 10.2.2.1 through Section 10.2.2.16 describe the Cchip register set.

10.2.2.1 Cchip System Configuration Register (CSC – RW)

Note: Follow the rules listed in Chapter 12 when writing to this CSR. After writing to this register, a delay is required to ensure that subsequent accesses to the 21272 will succeed.

All fields in CSC are read/write except for those in the two low-order bytes, which are read-only. Bits <7:0> are initialized from the pins of the Cchip on power-up. Bits <13:8> are written whenever the Dchip register STR is written. This ensures that the Cchip and Dchip versions of these fields are always synchronized, because the system will not function correctly if they are not the same. Table 10–9 describes the Tsunami Cchip configuration register (CSC).

Table 10–9 Cchip System Configuration Register (CSC) (Tsunami Only)

Field	Bits	Type	Init	Description
RES	<63>	MBZ,RAZ	0	Reserved.
RES	<62>	MBZ,RAZ	0	Reserved.
RES	<61>	MBZ,RAZ		Reserved.
RES	<60>	MBZ,RAZ		Reserved.
RES	<59>	MBZ,RAZ	0	Reserved.
PBQMAX[1]	<58:56>	RW	1	CPU probe queue maximum – 0 indicates 8 entries.
RES	<55>	MBZ,RAZ	0	Reserved.
PRQMAX	<54:52>	RW	2	Maximum requests to one Pchip until ACK, modulo 8 – the value 1 is illegal. Pchip Rev. 0 allows up to 4.
RES	<51>	MBZ,RAZ	0	Reserved.
PDTMAX	<50:48>	RW	1	Maximum data transfers to one Pchip until Ack, modulo 8. Pchip Rev. 0 allows up to 2.
RES	<47>	MBZ,RAZ	0	Reserved.
FPQPMAX	<46:44>	RW	1	Maximum entries in FPQ on Dchips known to Pchips, modulo 8. Dchip Rev. 0 allows up to 4. Must be the same as Pchip CSR PCTL <CDQMAX>.
RES	<43>	MBZ,RAZ	0	Reserved.
FPQCMAX	<42:40>	RW	1	True maximum entries in Dchip FPQ, modulo 8. Must be the same as Pchip CSR PCTL<CDQMAX>.
RES	39	MBZ,RAZ	0	Reserved.
TPQMMAX	<38:36>	RW	1	Maximum entries in TPQM on Dchips, modulo 8 (2 Dchips = 4, 4 or 8 Dchips = 8).
B3D	<35>	RW	0	Bypass 3 issue path disable
B2D	<34>	RW	0	Bypass 2 issue path disable
B1D	<33>	RW	0	Bypass 1 issue path disable

Table 10–9 Cchip System Configuration Register (CSC) (Tsunami Only) (Continued)

Field	Bits	Type	Init	Description
FTI[1]	<32>	RW	0	Force throttle issue
EFT	<31>	RW	1	Extract to fill turnaround cycles.

Value	Cycles
0	0 cycles
1	1 cycle

Field	Bits	Type	Init	Description
QDI	<30:28>	RW	0	Queue drain interval

Value	Cycles
0	Disable draining
1	1024 cycles
2	256 cycles
3	64 cycles
4	16 cycles
5	1 cycles
6	Reserved
7	Reserved

Field	Bits	Type	Init	Description
FET	<27:26>	RW	2	Fill to extract turnaround cycles.

Value	Cycles
0	1 cycle – SED must be 2 or 3 cycles.
1	2 cycles – SED must be 3, 4, or 5 cycles.
2	3 cycles – SED must be 3, 4, or 5 cycles.
3	Reserved.

Field	Bits	Type	Init	Description
QPM	<25>	RW	0	Queue priority mode

Value	Description
0	Round robin
1	Modified round robin

Field	Bits	Type	Init	Description
PME	<24>	RW	0	Page mode enable.
RES	<23:22>	RO	0	Reserved.
DRTP	<21:20>	RW	3	Minimum delay through Dchip from memory bus to PADbus.

Value	Cycles
0	2 cycles (rev 0 Dchip)
1	3 cycles
2	4 cycles
3	5 cycles

Field	Bits	Type	Init	Description
DWFP	<19:18>	RW	3	Minimum delay through Dchip from PADbus to CPU or memory bus.

Table 10–9 Cchip System Configuration Register (CSC) (Tsunami Only) (Continued)

Field	Bits	Type	Init	Description
				Value **Cycles**
				0 2 cycles
				1 3 cycles (rev 0 Dchip)
				2 4 cycles
				3 5 cycles
DWTP	<17:16>	RW	3	Minimum delay through Dchip from CPU bus to PADbus.
				Value **Cycles**
				0 2 cycles
				1 3 cycles
				2 4 cycles (rev 0 Dchip)
				3 5 cycles
RES	<15>	MBZ,RAZ	0	Reserved
P1P[2]	<14>	RO		Pchip 1 present.
IDDW	<13:12>	RO[3]	3	Issue to data delay for all transactions except memory reads (see Table 7–5).
				Value **Cycles**
				0 3 cycles
				1 4 cycles
				2 5 cycles
				3 6 cycles
IDDR	<11:9>	RO[3]	4	Issue to data delay for memory reads (see Table 7–5).
				Value **Cycles**
				0 5 cycles
				1 6 cycles
				2 7 cycles
				3 8 cycles
				4 9 cycles
				5 10 cycles
				6 11 cycles
				7 Reserved
AW	<8>	RO[3]	0	Array width.
				Value **Description**
				0 16 bytes
				32 bytes

Chipset Registers

Table 10–9 Cchip System Configuration Register (CSC) (Tsunami Only) (Continued)

Field	Bits	Type	Init	Description
FW[4]	<7>	RO		Available for firmware.
SFD[4]	<6>	RO		SysDC fill delay. The number of cycles from the SysDC cycle to the first CPU data cycle when moving data into the CPU.

Value	Description
0	2 cycles
1	3 cycles

Field	Bits	Type	Init	Description
SED[4]	<5:4>	RO		SysDC extract delay – The number of cycles from the SysDC cycle to the first CPU data cycle when moving data out of the CPU.

Value	Cycles
0	2 cycles
1	3 cycles
2	4 cycles
3	5 cycles

Field	Bits	Type	Init	Description
C1CFP[4]	<3>	RO		CPU1 clock forward preset (see Chapter 11).
C0CFP[4]	<2>	RO		CPU0 clock forward preset (see Chapter 11).
BC[4]	<1:0>	RO		Base configuration.

Value	Configuration
0	2 Dchips, 1 memory bus
1	4 Dchips, 1 memory bus
2	4 Dchips, 2 memory buses
3	8 Dchips, 2 memory buses

[1] The combination of PBQMAX = 1 and FTI = 0 is not allowed.
[2] Powers up to the value present on the CAPREQ<1> pin.
[3] These fields are updated when the Dchip STR register is written.
[4] Byte 0 powers up to the value present on bits 7:0 of the TIGbus.

Table 10–10 describes the Typhoon Cchip system configuration register.

Table 10–10 Cchip System Configuration Register (CSC) (Typhoon Only)

Field	Bits	Type	Init	Description
RES	<63>	MBZ,RAZ	0	Reserved.
RES	<62>	MBZ,RAZ	0	Reserved.
P1W[1]	<61>	RO	1	= Wide PADbus 1. (Typhoon only)
P0W[1]	<60>	RO	1	= Wide PADbus 0. (Typhoon only)
RES	<59>	MBZ,RAZ	0	Reserved.
PBQMAX[2]	<58:56>	RW	1	CPU probe queue maximum – 0 indicates 8 entries.
RES	<55>	MBZ,RAZ	0	Reserved.
PRQMAX	<54:52>	RW	2	Maximum requests to one Pchip until ACK, modulo 8 – the value 1 is illegal. Pchip Rev. 0 allows up to 4.

Table 10–10 Cchip System Configuration Register (CSC) (Typhoon Only) (Continued)

Field	Bits	Type	Init	Description
RES	<51>	MBZ,RAZ	0	Reserved.
PDTMAX	<50:48>	RW	1	Maximum data transfers to one Pchip until Ack, modulo 8. Pchip Rev. 0 allows up to 2.
RES	<47>	MBZ,RAZ	0	Reserved.
FPQPMAX	<46:44>	RW	1	Maximum entries in FPQ on Dchips known to Pchips, modulo 8. Dchip Rev. 0 allows up to 4. Must be the same as Pchip CSR PCTL <CDQMAX>.
RES	<43>	MBZ,RAZ	0	Reserved.
FPQCMAX	<42:40>	RW	1	True maximum entries in Dchip FPQ, modulo 8. Must be same as Pchip CSR PCTL<CDQMAX>.
AXD	39	RW	0	Disable memory XOR. (Typhoon only)
TPQMMAX	<38:36>	RW	1	Maximum entries in TPQM on Dchips, modulo 8 (2 Dchips = 4, 4 or 8 Dchips = 8).
B3D	<35>	RW	0	Bypass 3 issue path disable
B2D	<34>	RW	0	Bypass 2 issue path disable
B1D	<33>	RW	0	Bypass 1 issue path disable
FTI[2]	<32>	RW	0	Force throttle issue
EFT	<31>	RW	1	Extract to fill turnaround cycles.

Value	Cycles
0	0 cycles
1	1 cycle

Field	Bits	Type	Init	Description
QDI	<30:28>	RW	0	Queue drain interval

Value	Cycles
0	Disable draining
1	1024 cycles
2	256 cycles
3	64 cycles
4	16 cycles
5	1 cycles
6	Reserved
7	Reserved

Field	Bits	Type	Init	Description
FET	<27:26>	RW	2	Fill to extract turnaround cycles.

Value	Cycles
0	1 cycle – SED must be 2 or 3 cycles.
1	2 cycles – SED must be 3, 4, or 5 cycles.
2	3 cycles – SED must be 3, 4, or 5 cycles.
3	Reserved.

Table 10–10 Cchip System Configuration Register (CSC) (Typhoon Only) (Continued)

Field	Bits	Type	Init	Description
QPM	<25>	RW	0	Queue priority mode

	Value	Description
	0	Round robin
	1	Modified round robin

Field	Bits	Type	Init	Description
PME	<24>	RW	0	Page mode enable.
RES	<23:22>	RO	0	Reserved.
DRTP	<21:20>	RW	3	Minimum delay through Dchip from memory bus to PADbus.

	Value	Cycles
	0	2 cycles (rev 0 Dchip)
	1	3 cycles
	2	4 cycles
	3	5 cycles

Field	Bits	Type	Init	Description
DWFP	<19:18>	RW	3	Minimum delay through Dchip from PADbus to CPU or memory bus.

	Value	Cycles
	0	2 cycles
	1	3 cycles (rev 0 Dchip)
	2	4 cycles
	3	5 cycles

Field	Bits	Type	Init	Description
DWTP	<17:16>	RW	3	Minimum delay through Dchip from CPU bus to PADbus.

	Value	Cycles
	0	2 cycles
	1	3 cycles
	2	4 cycles (rev 0 Dchip)
	3	5 cycles

Field	Bits	Type	Init	Description
RES	<15>	MBZ,RAZ	0	Reserved.
P1P[3]	<14>	RO		Pchip 1 present.
IDDW	<13:12>	RO[4]	3	Issue to data delay for all transactions except memory reads (see Table 7–5).

	Value	Cycles
	0	3 cycles
	1	4 cycles

Table 10–10 Cchip System Configuration Register (CSC) (Typhoon Only) (Continued)

Field	Bits	Type	Init	Description
			2	5 cycles
			3	6 cycles
IDDR	<11:9>	RO[4]	4	Issue to data delay for memory reads (see Table 7–5).

Value	Cycles
0	5 cycles
1	6 cycles
2	7 cycles
3	8 cycles
4	9 cycles
5	10 cycles
6	11 cycles
7	Reserved

Field	Bits	Type	Init	Description
AW	<8>	RO[4]	0	Array width.

Value	Description
0	16 bytes
	32 bytes

Field	Bits	Type	Init	Description
FW[5]	<7>	RO		Available for firmware.
SFD[5]	<6>	RO		SysDC fill delay. The number of cycles from the SysDC cycle to the first CPU data cycle when moving data into the CPU.

Value	Description
0	2 cycles
1	3 cycles

Field	Bits	Type	Init	Description
SED[5]	<5:4>	RO		SysDC extract delay – The number of cycles from the SysDC cycle to the first CPU data cycle when moving data out of the CPU.

Value	Cycles
0	2 cycles
1	3 cycles
2	4 cycles
3	5 cycles

Field	Bits	Type	Init	Description
C1CFP[5]	<3>	RO		CPU1 clock forward preset (see Chapter 11).
C0CFP[5]	<2>	RO		CPU0 clock forward preset (see Chapter 11).
BC[5]	<1:0>	RO		Base configuration.

Value	Configuration

Table 10–10 Cchip System Configuration Register (CSC) (Typhoon Only) (Continued)

Field	Bits	Type	Init	Description
			0	2 Dchips, 1 memory bus
			1	4 Dchips, 1 memory bus
			2	4 Dchips, 2 memory buses
			3	8 Dchips, 2 memory buses

[1] Loaded from CAPbus<13:12> during reset.
[2] The combination of PBQMAX = 1 and FTI = 0 is illegal.
[3] Powers up to the value present on the CAPREQ<1> pin.
[4] These fields are updated when the Dchip STR register is written.
[5] Byte 0 powers up to the value present on bits <7:0> of the TIGbus.

10.2.2.2 Memory Timing Register (MTR – RW)

Table 10–11 describes the memory timing register (MTR).

Table 10–11 Memory Timing Register (MTR)

Field	Bits	Type	Init	Description
RES	<63:46>	MBZ,RAZ	0	Reserved.
MPH	<45:40>	RW	0	Maximum page hits – The most page hits the memory controller allows before forcing a page to be closed. The issue unit can, under some circumstances, sneak in one more page hit than this parameter allows.
PHCW	<39:36>	RW	14	Page hit cycles for writes – The number of cycles that the memory controller must wait after a write is issued until it attempts to close the page. **Note:** Must be greater than or equal to the greater of $(IRD + RPW - 2)$ and $(IRD + RCD + bl + tRWL - 3)$, where bl is 4 for 16-byte buses and 2 for 32-byte buses. **Note:** Initializes as 14 in the Rev. C Cchip; initializes as 15 in the Rev. B Cchip.
PHCR	<35:32>	RW	15	Page hit cycles for reads – The number of cycles that the memory controller must wait after a read is issued until it attempts to close the page. **Note:** Must be greater than or equal to the greater of $(RPW - 2)$ and $(RCD + bl - 2)$ where bl is 4 for 16-byte buses and 2 for 32-byte buses.
RES	<31:30>	MBZ,RAZ	0	Reserved.
RI	<29:24>	RW	0	Refresh interval – The number of cycles per refresh interval divided by 64. Each DRAM is refreshed once per refresh interval. A value of 0 disables refreshing. The values 1, 2, and 3 are illegal.
RES	<23:21>	MBZ,RAZ	0	Reserved.
MPD	<20>	RW	0	Mask pipeline delay – The **b_m*n*dqm<1:0>** signals to the SDRAMs may need to be buffered. Setting this bit causes the memory controllers to signal the DQM masks one cycle earlier to compensate.

Table 10–11 Memory Timing Register (MTR) (Continued)

Field	Bits	Type	Init	Description	
				Value	**Delay**
				0	No delay
				1	One pipeline stage
RES	<19:17>	MBZ,RAZ	0	Reserved.	

Table 10–11 Memory Timing Register (MTR) (Continued)

Field	Bits	Type	Init	Description
RRD	<16>	RW	0	Minimum same-array different-bank RAS-to-RAS delay – The minimum number of cycles from the Row Activate or Refresh command to the next Row Activate or Refresh command to the other bank of the same array.

Value	Delay
0	2
1	3

Field	Bits	Type	Init	Description
RES	<15:14>	MBZ,RAZ	0	Reserved.
RPT	<13:12>	RW	0	Minimum RAS precharge time – The minimum number of cycles from a Precharge command to the next Row Activate or Refresh command to the same bank.

Value	Cycles
0	2
1	3
2	4
3	Reserved

Field	Bits	Type	Init	Description
RES	<11:10>	MBZ,RAZ	0	Reserved.
RPW	<9:8>	RW	0	Minimum RAS pulse width (tRAS) – The minimum number of cycles from a Row Activate or Refresh command to the next Precharge command to the same bank. Used by the refresh logic only. PHCW and PHCR are used by the page-hit logic.

Value	Cycles
0	4
1	5
2	6
3	7

Field	Bits	Type	Init	Description
RES	<7>	MBZ,RAZ	0	Reserved.
IRD	<6:4>	RW	0	Issue to RAS delay – For memory writes this is the number of cycles from when the arbitrator issues the request, to when the DRAM row is activated. See Section 10.2.4.3 for the correct value to use.

Value	Cycles
0	0
1	1
2	2
3	3
4	4

Table 10–11 Memory Timing Register (MTR) (Continued)

Field	Bits	Type	Init	Description
				5 5
				6 Reserved
				7 Reserved
RES	<3>	MBZ,RAZ	0	Reserved.
CAT	<2>	RW	0	CAS access time – The number of cycles from the CAS command to when data appears on the DRAM outputs. Must be greater than or equal to RCD.

	Value	Cycles
	0	2
	1	3

Field	Bits	Type	Init	Description
RES	<1>	MBZ,RAZ	0	Reserved.
RCD	<0>	RW	0	RAS-to-CAS delay – The number of cycles from the Row Activate command to the next Read or Write command to the same bank. Must be less than or equal to CAT.

	Value	Cycles
	0	2
	1	3

10.2.2.3 Miscellaneous Register (MISC – RW)

This register is designed so that there are no read side effects, and that writing a 0 to any bit has no effect. Therefore, when software wants to write a 1 to any bit in the register, it need not be concerned with read-modify-write or the status of any other bits in the register. Once NXM is set, the NXS field is locked so that initial NXM error information is not overwritten by subsequent errors. It is unlocked when software clears the NXM field. The ABW (arbitration won) field is locked if either ABW bit is set, so the first CPU to write it locks out the other CPU. Writing a 1 to ACL (arbitration clear) clears both ABW bits and both ABT (arbitration try) bits and unlocks the ABW field. Table 10–12 describes the miscellaneous register (MISC).

Table 10–12 Miscellaneous Register (MISC)

Field	Bits	Type	Init	Description
RES	<63:44>	MBZ,RAZ	0	Reserved.
DEVSUP	<43:40>	WO	0	Suppress IRQ1 (device) interrupts to the CPU corresponding to a 1 in this field until the interrupt polling machine has completed a poll of all PCI devices. (<43:42> are used in Typhoon only)
REV	<39:32>	RO	—	Latest revision of Cchip:
			1	21272 (Tsunami)
			8	21274 (Typhoon)

Table 10–12 Miscellaneous Register (MISC) (Continued)

Field	Bits	Type	Init	Description
NXS	<31:29>	RO	0	NXM source – Device that caused the NXM – UNPREDICTABLE if NXM is not set.

	Value	Source
	0	CPU 0
	1	CPU 1
	2	Reserved CPU2 – Typhoon only
	3	Reserved CPU3 – Typhoon only
	4	Pchip 0
	5	Pchip 1
	6, 7	Reserved

Field	Bits	Type	Init	Description
NXM	<28>	R,W1C	0	Nonexistent memory address detected. Sets DRIR<63> and locks the NXS field until it is cleared.
RES	<27:25>	MBZ,RAZ	0	Reserved.
ACL	<24>	WO	0	Arbitration clear – writing a 1 to this bit clears the ABT and ABW fields.
ABT	<23:20>	R,W1S	0	Arbitration try – writing a 1 to these bits sets them. (<23:22> are used in Typhoon only)
ABW	<19:16>	R,W1S	0	Arbitration won – writing a 1 to these bits sets them unless one is already set, in which case the write is ignored. (<19:18> are used in Typhoon only)
IPREQ	<15:12>	WO	0	Interprocessor interrupt request – write a 1 to the bit corresponding to the CPU you want to interrupt. Writing a 1 here sets the corresponding bit in the IPINTR. (<15:14> are used in Typhoon only)
IPINTR	<11:8>	R,W1C	0	Interprocessor interrupt pending – one bit per CPU. Pin **irq<3>** is asserted to the CPU corresponding to a 1 in this field. (<11:10> are used in Typhoon only)
ITINTR	<7:4>	R,W1C	0	Interval timer interrupt pending – one bit per CPU. Pin **irq<2>** is asserted to the CPU corresponding to a 1 in this field. (<7:6> are used in Typhoon only)
RES	<3:2>	MBZ,RAZ	0	Reserved.
CPUID	<1:0>	RO	—	ID of the CPU performing the read. (<1> are used in Typhoon only)

10.2.2.4 Memory Presence Detect Register (MPD – RW)

The memory presence detect register is connected to two open-drain pins on the Cchip. These pins can be used by software to implement the I^{2}C protocol to read the serial presence detect pins on the SDRAM DIMMs. Table 10–13 describes the memory presence detect register (MPD).

Table 10–13 Memory Presence Detect Register (MPD)

Field	Bits	Type	Init	Description
RES	<63:4>	MBZ,RAZ	0	Reserved
DR	<3>	RO	1	Data receive
CKR	<2>	RO	1	Clock receive
DS	<1>	WO	1	Data send – Must be a 1 to receive
CKS	<0>	WO	1	Clock send

10.2.2.5 Array Address Register (AAR0, AAR1, AAR2, AAR3 – RW)

Table 10–14 describes the Tsunami array address registers.

Table 10–14 Array Address Register (AAR0, AAR1, AAR2, AAR3) (Tsunami Only)

Field	Bits	Type	Init	Description
RES	<63:35>	MBZ,RAZ	0	Reserved.
ADDR	<34:24>	RW	0	Base address – Bits <34:24> of the physical byte address of the first byte in the array. (<27:24> are used in Tsunami only; <31:24> are valid.)
RES	<23:17>	MBZ,RAZ	0	Reserved.
DBG	<16>	RW	0	Enables this memory port to be used as a debug interface.
ASIZ	<15:12>	RW	0	Array size. {table below}

Value	Size
0000	0 (bank disabled)
0001	16MB
0010	32MB
0011	64MB
0100	128MB
0101	256MB
0110	512MB
0111	1GB
1011–1111	Reserved

Field	Bits	Type	Init	Description
RES	<11:9>	MBZ,RAZ	0	Reserved.
SA	<8>	RW	0	Split array.

Table 10–14 Array Address Register (AAR0, AAR1, AAR2, AAR3) (Tsunami Only) (Continued)

Field	Bits	Type	Init	Description
RES	<7:4>	MBZ,RAZ	0	Reserved.
ROWS	<3:2>	RW	0	Number of row bits in the SDRAMs.

Value	Number of Bits
0	11
1	12
2	13
3	Reserved

Field	Bits	Type	Init	Description
BNKS	<1:0>	RW	0	Number of bank bits in the SDRAMs.

Value	Number of Bits
0	1
1	2
2	Reserved
3	Reserved

Table 10–15 describes the Typhoon array address registers.

Table 10–15 Array Address Register (AAR0, AAR1, AAR2, AAR3) (Typhoon Only)

Field	Bits	Type	Init	Description
RES	<63:35>	MBZ,RAZ	0	Reserved.
ADDR	<34:24>	RW	0	Base address – Bits <34:24> of the physical byte address of the first byte in the array. (<34:32> are used in Typhoon only; <34:28> are valid)
RES	<23:17>	MBZ,RAZ	0	Reserved.
DBG	16	RW	0	Enables this memory port to be used as a debug interface.
ASIZ	<15:12>	RW	0	Array size (<15> is used in Typhoon only).

Value	Size
0000	0 (bank disabled)
0001	16MB
0010	32MB
0011	64MB
0100	128MB
0101	256MB
0110	512MB
0111	1GB
1000	2GB (Typhoon only)

Table 10–15 Array Address Register (AAR0, AAR1, AAR2, AAR3) (Typhoon Only) (Continued)

Field	Bits	Type	Init	Description	
				1001	4GB (Typhoon only)
				1010	8GB (Typhoon only)
				1011–1111	Reserved.
RES	<11:10>	MBZ,RAZ	0	Reserved.	
TSA	<9>	RW	0	Twice-split array (Typhoon only)	
SA	<8>	RW	0	Split array.	
RES	<7:4>	MBZ,RAZ	0	Reserved.	
ROWS	<3:2>	RW	0	Number of row bits in the SDRAMs.	

	Value	Number of Bits
	0	11
	1	12
	2	13
	3	Reserved

Field	Bits	Type	Init	Description
BNKS	<1:0>	RW	0	Number of bank bits in the SDRAMs

	Value	Number of Bits
	0	1
	1	2
	2	3 (Typhoon only)
	3	Reserved

10.2.2.6 Device Interrupt Mask Register (DIM*n*, *n*=0,3 – RW)

Register *n* applies to CPU*n*. (Typhoon only: n=2,3.)

These two mask registers control which interrupts are allowed to go through to the CPUs. No interrupt in DRIR will get through to the masked interrupt registers (and on to interrupt the CPUs) unless the corresponding mask bit is set in DIM*n*. All bits are initialized to 0 at reset. Table 10–16 describes the device interrupt mask registers.

Table 10–16 Device Interrupt Mask Register (DIM*n*)

Field	Bits	Type	Init	Description
DIM	<63:0>	RW	0	Interrupts allowed through to the CPU

10.2.2.7 Device Interrupt Request Register (DIR*n*, *n*=0,3 – RO)

Register *n* applies to CPU*n*. (Typhoon only: n=2,3.)

These two registers indicate which interrupts are pending to the CPUs. If a raw request bit is set and the corresponding mask bit is set, then the corresponding bit in this register will be set and the appropriate CPU will be interrupted. Table 10–17 describes the device interrupt request registers.

Table 10–17 Device Interrupt Request Register (DIR*n*)

Field	Bits	Type	Init	Description
ERR	<63:58>	RO	0	IRQ0 error interrupts <63> Chip detected MISC<NXM> <62> recommended hookup to Pchip0 error <61> recommended hookup to Pchip1 error Others per module designer's choice
RES	<57:56>	RO	0	Reserved
DEV	<55:0>	RO	0	IRQ1 PCI interrupts pending to the CPU

10.2.2.8 Device Raw Interrupt Request Register (DRIR – RO)

DRIR indicates which of the 64 possible device interrupts is asserted. Table 10–18 describes the device raw interrupt request register (DRIR).

Table 10–18 Device Raw Interrupt Request Register (DRIR)

Field	Bits	Type	Init	Description
DRIR	<63:0>	RO	0	Interrupts pending from devices

10.2.2.9 Probe Enable Register (PRBEN – RW)

This register is special in that reads do not return the value of the register, but rather cause the probe enable bit for the requesting CPU to be cleared. The return data is UNPREDICTABLE. Writing to this register causes the probe enable bit for the requesting CPU to be set, regardless of the value written. Table 10–19 describes the probe enable register (PRBEN).

Table 10–19 Probe Enable Register (PRBEN)

Field	Bits	Type	Init	Description
RES	<63:1>	MBZ	0	Reserved
PRBEN	<0>	RTC,WTS	0	Probe enable bit

10.2.2.10 Interval Ignore Count Register (IIC*n*, *n*=0,3 – RW)

Register *n* applies to CPU*n*. (Typhoon only: n=2,3.)

These registers are used for 21264 CPU sleep mode. They are written with a count of how many interval timer interrupts to suppress and count down to 0 as subsequent interval timer interrupts are asserted. They can be read at any time to find the remaining count. After the count has been decremented to 0, the next interval timer interrupt will

be sent through to the CPU. After the wake-up tick is received, the count goes negative, and the OF bit is set on the next timer interval tick. This allows the CPU to determine exactly how many interval timer ticks were skipped. Table 10–20 describes the interval ignore count register (IIC).

Table 10–20 Interval Ignore Count Register (IIC)

Field	Bits	Type	Init	Description
RES	<63:25>	MBZ,RAZ	0	Reserved
OF	<24>	RO	0	Overflow – Indicates negative count
ICNT	<23:0>	RW	0	Count of remaining interrupts to ignore

10.2.2.11 Wake-Up Delay Register (WDR – RW)

The WDR register determines how long (in system cycles) the chipset waits after a reset, or after sending a wake-up interrupt to a sleeping CPU, before deasserting **b_cfrst<1:0>**.

Table 10–21 Wake-Up Delay Register (WDR)

Field	Bits	Type	Init	Description
RES	<63:25>	MBZ	0	Reserved

10.2.2.12 Memory Programming Register (MPR0, MPR1, MPR2, MPR3 – WO)

A write to these registers causes a RAM program cycle (a mode register set command) to the associated memory array using the data written to the MPRDAT field. Table 10–22 describes the memory programming registers.

Table 10–22 Memory Programming Register (MPR*n*)

Field	Bits	Type	Init	Description
RES	<63:13>	MBZ	0	Reserved
MPRDAT	<12:0>	WO	—	Data to be written on address lines <12:0>

10.2.2.13 M-Port Control Register (MCTL – MBZ)

The M-port control register controls chipset debug features. It must be 0 for normal operations. In Typhoon, this register is replaced by the CMONCTL registers.

10.2.2.14 TIGbus Timing Register (TTR – RW)

The TIGbus timing register controls the nonaddress-specific timing of the TIGbus. Table 10–23 describes the TIGbus timing register (TTR). See Section 6.3 for timing diagrams and information.

Table 10–23 TIGbus Timing Register (TTR – RW)

Field	Bits	Type	Init	Description
RES	<63:15>	MBZ,RAZ	0	Reserved.
ID	<14:12>	RW	7	Interrupt starting device – If there are fewer than eight interrupt buffers present on the module, this field determines the lowest-order byte number that gets read in – Devices <7:ID> all present is a requirement.
RES	<11:10>	MBZ,RAZ	0	Reserved.
IRT	<9:8>	RW	3	Interrupt read time – The number of cycles that the interrupt driver is enabled on the TIGbus before the interrupt data is latched in the Cchip.

		Value	Cycles
		0	1 cycle
		1	2 cycles
		2	3 cycles
		3	4 cycles

Field	Bits	Type	Init	Description
RES	<7:6>	MBZ,RAZ	0	Reserved.
IS	<5:4>	RW	3	Interrupt setup time – The number of cycles that the Cchip drives the IRQ data on the TIGbus before asserting **b_tis** to strobe the data into a register on the module.

		Value	Cycles
		0	1 cycle
		1	2 cycles
		2	3 cycles
		3	4 cycles

Field	Bits	Type	Init	Description
RES	<3:2>	MBZ,RAZ	0	Reserved
AH	<1>	RW	0	Address hold after **as_l** before **cs_l**.

		Value	Cycles
		0	1 cycle
		1	2 cycles

Field	Bits	Type	Init	Description
AS	<0>	RW	0	Address setup to the address latch before **as_l**.

		Value	Cycles
		0	1 cycle
		1	2 cycles

10.2.2.15 TIGbus Device Timing Register (TDR – RW)

One 16-bit field of this register is selected by TIG address bits <23:22> to allow up to four different timing domains on the TIGbus. All values in the register are expressed in cycles. The state machine stays in each state one cycle longer than the number in the register (that is, a value of 0 means one cycle). See Section 6.3 for timing diagrams describing these fields.

Table 10–24 describes the TIGbus device timing register (TDR).

Table 10–24 TIGbus Device Timing Register (TDR)

Field	Bits	Type	Init	Description
WH3	<63>	RW	0	See bit WH0
WP3	<62:60>	RW	0	See bit WP0
RES	<59:58>	MBZ,RAZ	0	Reserved
WS3	<57:56>	RW	0	See bit WS0
RES	<55>	MBZ,RAZ	0	Reserved
RD3	<54:52>	RW	0	See bit RD0
RA3	<51:48>	RW	0	See bit RS0
WH2	<47>	RW	0	See bit WH0
WP2	<46:44>	RW	0	See bit WP0
RES	<43:42>	MBZ,RAZ	0	Reserved
WS2	<41:40>	RW	0	See bit WS0
RES	<39>	MBZ,RAZ	0	Reserved
RD2	<38:36>	RW	0	See bit RD0
RA2	<35:32>	RW	0	See bit RS0
WH1	<31>	RW	0	See bit WH0
WP1	<30:28>	RW	0	See bit WP0
RES	<27:26>	MBZ,RAZ	0	Reserved
WS1	<25:24>	RW	0	See bit WS0
RES	<23>	MBZ,RAZ	0	Reserved
RD1	<22:20>	RW	0	See bit RD0
RA1	<19:16>	RW	0	See bit RS0
WH0	<15>	RW	0	Write hold time – The number of cycles that **cs_l**, the address, and the data are held once **we_l** is deasserted
WP0	<14:12>	RW	0	Write pulse width – The number of cycles that **we_l** is held asserted
RES	<11:10>	MBZ,RAZ	0	Reserved
WS0	<9:8>	RW	0	Write setup time – The number of cycles that the address and data are held stable to the device before **we_l** is asserted

Table 10–24 TIGbus Device Timing Register (TDR) (Continued)

Field	Bits	Type	Init	Description
RES	<7>	MBZ,RAZ	0	Reserved
RD0	<6:4>	RW	0	Read output disable time – The number of cycles that the device takes to turn off its output drivers once **oe_l** is deasserted
RA0	<3:0>	RW	0	Read access time – The number of cycles that **cs_l** and **oe_l** are asserted to the device before the data is latched in from the TIG-bus

10.2.2.16 Power Management Control (PWR – RW)

This register controls chipset management features. To date, only SDRAM self-refresh mode is implemented.

Warning: Software must ensure that there are no DRAM accesses active in the 21272 when self-refresh mode is active.

Table 10–25 describes the power management control register (PWR).

Table 10–25 Power Management Control Register (PWR – RW)

Field	Bits	Type	Init	Value	Description
RES	<63:1>	MBZ,RAZ	0		Reserved.
SR	<0>	RW	0	0	Normal operation.
				1	Self-refresh mode; standard refreshing is suppressed, regardless of MTR<RI>.

10.2.3 Cchip Monitor Control (CMONCTLA, CMONCTLB – RW) – Typhoon only

All fields in the CMONCTLA and CMONCTLB registers are RW. They are cleared by reset. Some monitor signals are hardwired to select a specific CPU. The mask and match/entry fields provide wide flexibility in the selection of events to count.

Table 10–26 describes the Cchip monitor control register CMONCTLA and Table 10–27 describes register CMONCTLB.

Table 10–26 Cchip Monitor Control Register (CMONCTLA)

Field	Bits	Type	Init	Description
RES	<63:62>	MBZ,RAZ	0	Reserved.
MSK23	<61:52>	RW	0	Mask field – For ECNT2 and ECNT3, the match/entry fields can be used to qualify the value in the <SLCTn> field.
RES	<51:50>	MBZ,RAZ	0	Reserved.
MSK01	<49:40>	RW	0	Mask field – For ECNT0 and ECNT1.
STKDIS3	<39>	RW	0	ECNT3 stick disable.

Value	Description
0	ECNT3 sticks at all ones
1	ECNT3 wraps

Field	Bits	Type	Init	Description
STKDIS2	<38>	RW	0	
STKDIS1	<37>	RW	0	
STKDIS0	<36>	RW	0	
RES	<35:34>	MBZ,RAZ	0	Reserved.
SLCTMBL	<33:32>	RW	0	Select memory bus monitor low bits. Note: Memory bus monitor bits <20:16> are fixed.

Value	Group
0	mem bus monitor <15:0> = mgroup0
1	mem bus monitor <15:0> = mgroup1
2	mem bus monitor <15:0> = mgroup2
3	mem bus monitor <15:0> = mgroup3

Field	Bits	Type	Init	Description
SLCT3	<31:24>	RW	0	Select B MONITOR<3>; Select Event 3.

Table 10–26 Cchip Monitor Control Register (CMONCTLA) (Continued)

Field	Bits	Type	Init	Description
SLCT2	<23:16>	RW	0	Select B MONITOR<2>; Select Event 2.
SLCT1	<15:8>	RW	0	Select B MONITOR<1>; Select Event 1.
SLCT0	<7:0>	RW	0	Select B MONITOR<0>; Select Event 0.

Table 10–27 Cchip Monitor Control Register (CMONCTLB)

Field	Bits	Type	Init	Description
RES	<63:62>	MBZ,RAZ	0	Reserved
MTE3	<61:52>	RW	0	Match/entry field – for ECNT3 The match/entry and mask fields can be used to qualify the value in the <SLCTn> field.
RES	<51:50>	MBZ,RAZ	0	Reserved
MTE2	<49:40>	RW	0	Match/entry field – for ECNT2
RES	<39:38>	MBZ,RAZ	0	Reserved
MTE1	<37:28>	RW	0	Match/entry field – for ECNT1
RES	<27:26>	MBZ,RAZ	0	Reserved
MTE0	<25:16>	RW	0	Match/entry field – for ECNT0
RES	<15:1>	MBZ,RAZ	0	Reserved
DIS	<0>	RW	0	Disable monitor output signals:

Value	Description
0	B_MONITOR outputs in use for monitor
1	B_MONITOR outputs static at zero

10.2.3.1 Cchip Monitor Counters (CMONCNT01, CMONCNT23 – R0)

The 21272 has four 23-bit event counters. The event counted by counter *n* (ECNT*n*) is selected by the CMONCTL<SLCT*n*> field. One of the possible events selected is the carry-out of the previous counter, which allows both counters to be used as two 64-bit counters. In this case, the CMONCTL<STKDIS*n*> bit must be set to ensure that the low-order 32-bit counter does not stick at all ones.

Both counters hold their values for four cycles each time that a read to CMONCNT*x* is performed, so that a slight inaccuracy can result if the events being counted continue to occur at the time of reading.

The <SLCT*n*> field may specify the use of the CMONCTL fields <MTE*x*> and <MSK*y*> to further qualify the selection. In this case, a fixed correspondence occurs between the ECNT field to be updated and the combination MTE/MSK qualifier field used. Table 10–28 shows this correspondence.

Table 10–28 Correspondence Between ECNT and MTE/MSK

Field to Increment	MTE Field Used	MSK Field Used
ECNT3	MTE3	MSK23
ECNT2	MTE2	MSK23
ECNT1	MTE1	MSK01
ECNT0	MTE0	MSK01

CMONCNT01 Registers – Typhoon Only

All fields in the CMONCNT01 registers are Read/Write; however, the write feature is only for diagnostic purposes. Writing a value of all ones to any field of CMONCNT is not supported due to implementation considerations (the carry-out is precomputed).

All fields of CMONCNT are cleared by reset and when CMONCTLA or CMONCTLB is written. The expected usage is to write CMONCTL, wait for a while, read CMONCNT, and repeat. Table 10–29 shows the CMONCNT01 registers.

Table 10–29 CMONCNT01 Registers

Field	Bits	Type	Init	Values	Description
ECNT1	<63:32>	RW	0	—	Increments when Event 1 is true
ECNT0	<31:0>	RW	0	—	Increments when Event 0 is true

CMONCNT23 Registers – Typhoon Only

The operation of CMONCNT23 is the same as that for CMONCNT01. Table 10–30 shows the CMONCNT23 registers.

Table 10–30 CMONCNT23 Registers

Field	Bits	Type	Init	Values	Description
ECNT3	<63:32>	RW	0	—	Increments when Event 3 is true
ECNT2	<31:0>	RW	0	—	Increments when Event 2 is true

10.2.4 Dchip CSRs

Section 10.2.4.1 through Section 10.2.4.4 describe the Dchip register set.

10.2.4.1 Dchip System Configuration Register (DSC – RO)

Table 10–31 describes the Dchip system configuration register (DSC).

Table 10–31 Dchip System Configuration Register (DSC)

Field	Bits	Type	Init	Description
RES	<63:8>	Special[1]	0	—
RES	<7>	RAZ	0	Reserved
P1P	<6>	RO	—[2]	Pchip 1 present

Table 10–31 Dchip System Configuration Register (DSC) (Continued)

Field	Bits	Type	Init	Description
C3CFP	<5>	RO	—[2]	CPU3 clock forward preset (see Chapter 11)
C2CFP	<4>	RO	—[2]	CPU2 clock forward preset (see Chapter 11)
C1CFP	<3>	RO	—[2]	CPU1 clock forward preset (see Chapter 11)
C0CFP	<2>	RO	—[2]	CPU0 clock forward preset (see Chapter 11)
BC	<1:0>	RO	—[2]	Base configuration

Value	Configuration
0	2 Dchips, 1 memory bus
1	4 Dchips, 1 memory bus
2	4 Dchips, 2 memory buses
3	8 Dchips, 2 memory buses

[1] This is an 8-bit register that mirrors some information in CSC. It is special, however, in that it is byte-sliced across eight Dchips. Therefore, it is read as a quadword with the same value repeated in all eight bytes.

[2] This register powers up to the value present on bits <6:0> of the CPM command from the Cchip.

10.2.4.2 Dchip System Configuration Register 2 (DSC2 – R0)

These registers are for future use, for a Dchip that implements wide PADbus support. Table 10–32 describes the Dchip system configuration register 2.

Table 10–32 Dchip System Configuration Register 2 (DSC2)

Field	Bits	Type	Init	Description
RES	<63:5>	RO	0	Reserved
RES	<4:2>	RO	—[1]	Reserved
P1W	<1>	RO	—[1]	Reserved 0 = Wide PADbus1 (Typhoon only)
P0W	<0>	RO	—[1]	Reserved 0 = Wide PADbus0 (Typhoon only)

[1] This register powers up to the value present on bits <4:0> of the PADCMD bus from the Cchip.

10.2.4.3 System Timing Register (STR – RW)

When the system timing register is written, all Dchips, as well as the corresponding fields in the CSC register, are updated at the same time. The corresponding fields in the CSC register are read-only, so the only way to update them is to write this register.

Note: Follow the rules listed in Chapter 12 when writing to this CSR. After writing to this register, a delay is required to ensure that subsequent accesses to the 21272 will succeed.

IDDR, IDDW, and IRD (set in CSC) must be set as follows before accessing memory:

- RCD is the RAS-to-CAS delay in the DRAMs (set in CSC).
- CAT is the CAS access time in the DRAMs (set in CSC).

- SED is the SysDC extract delay (set in the CSC).

- b is the burst length (2 for 32-byte memories and 4 for 16-byte memories).

- p is the number of pipeline stages on the control signals between the Cchip and the SDRAMs (0, 1, or 2).

    $$IDDR = RCD + CAT + p + b - 1$$

    $$IDDW = MAX\,(RCD + p - 1,\ SED + 1,\ IDDR - 2b + 1)$$

    $$IRD = IDDW - RCD - p + 1$$

If software wishes to set IDDW to a value other than the power-up default, and knows that memory will not yet be accessed, then the restriction is relaxed to:

$$IDDW > SED$$

Table 10–33 describes the system timing register (STR).

Table 10–33 System Timing Register (STR)

Field	Bits	Type	Init	Description
RES	<63:8>	Special[1]	0	—[1]
RES	<7:6>	MBZ,RAZ	0	Reserved
IDDW	<5:4>	RW	2	Issue to data delay for all transactions except memory reads (see Table 7–5)
				Value	**Cycles**
				0	3 cycles
				1	4 cycles
				2	5 cycles
				3	6 cycles
IDDR	<3:1>	RW	4	Issue to data delay for memory reads (see Table 7–5)
				Value	**Cycles**
				0	5 cycles
				1	6 cycles
				2	7 cycles
				3	8 cycles
				4	9 cycles
				5	10 cycles
				6	11 cycles
				7	Reserved
AW	<0>	RW	0	Array width

Table 10–33 System Timing Register (STR) (Continued)

Field	Bits	Type	Init	Description	
				Value	**Cycles**
				0	16 bytes
				1	32 bytes

[1] This is an 8-bit register corresponding to bits CSC<13:8>. It is special, however, in that it must be written to up to eight Dchips simultaneously. Therefore, it is written as a quadword with the same value repeated in all eight bytes. That way, all Dchips are configured properly regardless of system configuration.

10.2.4.4 Dchip Revision Register (DREV – RO)

Table 10–34 describes the Dchip revision register (DREV).

Table 10–34 Dchip Revision Register (DREV)

Field	Bits	Type	Init	Description
RES	<63:60>	RAZ	0	Reserved
REV7	<59:56>	RO	1	Dchip 7 revision. This field indicates the latest revision of the Dchip.
RES	<55:52>	RAZ	0	Reserved
REV6	<51:48>	RO	1	Dchip 6 revision. This field indicates the latest revision of the Dchip.
RES	<47:44>	RAZ	0	Reserved
REV5	<43:40>	RO	1	Dchip 5 revision. This field indicates the latest revision of the Dchip.
RES	<39:36>	RAZ	0	Reserved
REV4	<35:32>	RO	1	Dchip 4 revision. This field indicates the latest revision of the Dchip.
RES	<31:28>	RAZ	0	Reserved
REV3	<27:24>	RO	1	Dchip 3 revision. This field indicates the latest revision of the Dchip.
RES	<23:20>	RAZ	0	Reserved
REV2	<19:16>	RO	1	Dchip 2 revision. This field indicates the latest revision of the Dchip.
RES	<15:12>	RAZ	0	Reserved
REV1	<11:8>	RO	1	Dchip 1 revision. This field indicates the latest revision of the Dchip.
RES	<7:4>	RAZ	0	Reserved
REV0	<3:0>	RO	1	Dchip 0 revision. This field indicates the latest revision of the Dchip.

10.2.5 Pchip CSRs

Section 10.2.5.1 through Section 10.2.5.12 describe the Pchip register set.

10.2.5.1 Window Space Base Address Register (WSBA*n* – RW)

Because the information in the WSBA*n* registers and WSM*n* registers (Section 10.2.5.2) is used to compare against the PCI address, a clock-domain crossing (from **i_sysclk** to **i_pclko<7:0>**) is made when these registers are written. Therefore, for a period of several clock cycles, a window is disabled when its contents are disabled. If PCI bus activity, which accesses the window in question, is not stopped before updating that window, the Pchip might fail to respond with **b_devsel_l** when it should. This would result in a master abort condition on the PCI bus. Therefore, before a window (base or mask) is updated, all PCI activity accessing that window must be stopped, even if only some activity is being added or deleted.

The contents of the window may be read back to confirm that the update has taken place. Then PCI activity through that window can be resumed.

Table 10–35 describes the window space base address registers WSBA0, 1, and 2. Table 10–36 describes WSBA3.

Table 10–35 Window Space Base Address Register (WSBA0, 1, 2)

Field	Bits	Type	Init	Description
RES	<63:32>	MBZ,RAZ	0	Reserved
ADDR	<31:20>	RW	0	Base address
RES	<19:2>	MBZ,RAZ	0	Reserved
SG	<1>	RW	0	Scatter-gather
ENA	<0>	RW	0	Enable

Table 10–36 Window Space Base Address Register (WSBA3)

Field	Bits	Type	Init	Description
RES	<63:40>	MBZ,RAZ	0	Reserved
DAC	<39>	RW	0	DAC enable
RES	<38:32>	MBZ,RAZ	0	Reserved
ADDR	<31:20>	RW	0	Base address if DAC enable = 0 Not used if DAC enable = 1
RES	<19:2>	MBZ,RAZ	0	Reserved
SG	<1>	RO	1	Scatter-gather always enabled
ENA	<0>	RW	0	Enable

10.2.5.2 Window Space Mask Register (WSM0, WSM1, WSM2, WSM3 – RW)

Table 10–37 describes the window space mask registers. Refer to the WSBA*n* register description (Section 10.2.5.1) for a brief description of the window space mask register.

Table 10–37 Window Space Mask Register (WSM*n*)

Field	Bits	Type	Init	Description
RES	<63:32>	MBZ,RAZ	0	Reserved
AM	<31:20>	RW	0	Address mask
RES	<19:0>	MBZ,RAZ	0	Reserved

10.2.5.3 Translated Base Address Register (TBA*n* – RW)

Table 10–38 describes the translated base address registers TBA0, 1, and 2. Table 10–39 describes TBA3.

Table 10–38 Translated Base Address Registers (TBA0, 1, and 2)

Field	Bits	Type	Init	Description
RES	<63:35>	MBZ,RAZ	0	Reserved
ADDR	<34:10>	RW	0	Translated address base
RES	<9:0>	MBZ,RAZ	0	Reserved

Table 10–39 Translated Base Address Registers (TBA3)

Field	Bits	Type	Init	Description
RES	<63:35>	MBZ,RAZ	0	Reserved
ADDR	<34:10>	RW	0	If DAC enable = 1, bits <34:22> are the Page Table Origin address <34:22> and bits <21:10> are ignored. If DAC enable = 0, this is the translated address base.
RES	<9:0>	MBZ,RAZ	0	Reserved

10.2.5.4 Pchip Control Register (PCTL – RW)

Table 10–40 describes the Pchip control register (PCTL).

Table 10–40 Pchip Control Register (PCTL)

Field	Bits	Type	Init	Description
RES	<63:48>	MBZ,RAZ	0	Reserved.
PID	<47:46>	RO	—[1]	Pchip ID.
RPP	<45>	RO	—[2]	Remote Pchip present.
PTEVRFY	<44>	RW	—	PTE verify for DMA read.

	Value	Description

Table 10–40 Pchip Control Register (PCTL) (Continued)

Field	Bits	Type	Init	Description
			0	If TLB miss, then make DMA read request as soon as possible and discard data if PTE was not valid – could cause Cchip non-existent memory error.
			1	If TLB miss, then delay read request until PTE is verified as valid – no request if not valid.
FDWDIS	<43>	RW	—	Fast DMA read cache block wrap request disable.

Value	Description
0	Normal operation
1	Reserved for testing purposes only

Table 10–40 Pchip Control Register (PCTL) (Continued)

Field	Bits	Type	Init	Description
FDSDIS	<42>	RW	—	Fast DMA start and SGTE request disable.

	Value	Description
	0	Normal operation
	1	Reserved for testing purposes only

Field	Bits	Type	Init	Description
PCLKX	<41:40>	RO	—[3]	PCI clock frequency multiplier

	Value	Multiplier
	0	x6
	1	x4
	2	x5
	3	Reserved

Field	Bits	Type	Init	Description
PTPMAX	<39:36>	RW	2	Maximum PTP requests to Cchip from both Pchips until returned on CAPbus, modulo 16 (minimum = 2) (use 4 for pass 1 Cchip and Dchip).
CRQMAX	<35:32>	RW	1	Maximum requests to Cchip from both Pchips until Ack, modulo 16 (use 4 for Cchip). (Use 3 or less for Typhoon because there is one less skid buffer in the C4 chip.)
REV	<31:24>	RO	0	In conjunction with the state of PMONCTL<0>, this field indicates the revision of the Pchip (see Section 8.10).
CDQMAX	<23:20>	RW	1	Maximum data transfers to Dchips from both Pchips until Ack, modulo 16 (use 4 for Dchip). Must be same as Cchip CSR CSC<FPQPMAX>.
PADM	<19>	RW	—[4]	PADbus mode.

	Value	Mode
	0	8-nibble, 8-check bit mode
	1	4-byte, 4-check bit mode

Field	Bits	Type	Init	Description
ECCEN	<18>	RW	0	ECC enable for DMA and SGTE accesses.
RES	<17:16>	MBZ,RAZ	0	Reserved.
PPRI	<15>	—	0	Arbiter priority group for the Pchip.
PRIGRP	<14:8>	RW	0	Arbiter priority group; one bit per PCI slot with bits <14:8> corresponding to input **b_req_l<6:0>**.

	Value	Group
	0	Low-priority group
	1	High-priority group

Field	Bits	Type	Init	Description
ARBENA	<7>	RW	0	Internal arbiter enable.
MWIN	<6>	RW	0	Monster window enable.
HOLE	<5>	RW	0	512KB-to-1MB window hole enable.

Table 10–40 Pchip Control Register (PCTL) (Continued)

Field	Bits	Type	Init	Description	
TGTLAT	<4>	RW	0	Target latency timers enable.	
				Value	**Mode**
				0	Retry/disconnect after 128 PCI clocks without data.
				1	Retry initial request after 32 PCI clocks without data; disconnect subsequent transfers after 8 PCI clocks without data.
CHAINDIS	<3>	RW	0	Disable chaining.	
THDIS	<2>	RW	0	Disable antithrash mechanism for TLB.	
				Value	**Mode**
				0	Normal operation
				1	Testing purposes only
FBTB	<1>	RW	0	Fast back-to-back enable.	
FDSC	<0>	RW	0	Fast discard enable.	
				Value	**Mode**
				0	Discard data if no retry after 2^{15} PCI clocks.
				1	Discard data if no retry after 2^{10} PCI clocks.

[1] This field is initialized from the PID pins.
[2] This field is initialized from the assertion of CREQRMT_L pin at system reset.
[3] This field is initialized from the PCI **i_pclkdiv<1:0>** pins.
[4] This field is initialized from a decode of the **b_cap<1:0>** pins.

10.2.5.5 Pchip Master Latency Register (PLAT – RW)

Table 10–41 describes the Pchip master latency register (PLAT).

Table 10–41 Pchip Master Latency Register (PLAT)

Field	Bits	Type	Init	Description
RES	<63:16>	MBZ,RAZ	0	Reserved
LAT	<15:8>	RW	0	Master latency timer
RES	<7:0>	MBZ,RAZ	0	Reserved

10.2.5.6 Pchip Error Register (PERROR – RW)

If any of bits <11:0> are set, then this entire register is frozen and the Pchip output signal **b_error** is asserted. Only bit <0> can be set after that. All other values will be held until all of bits <11:0> are clear. When an error is detected and one of bits <11:0> becomes set, the associated information is captured in bits <63:16> of this register. After the information is captured, the INV bit is cleared, but the information is not valid and should not be used if INV is set.

In rare circumstances involving more than one error, INV may remain set because the Pchip cannot correctly capture the SYN, CMD, or ADDR field.

Furthermore, if software reads PERROR in a polling loop, or reads PERROR before the Pchip's error signal is reflected in the Cchip's DRIR CSR, the INV bit may also be set. To avoid the latter condition, read PERROR only after receiving an IRQ0 interrupt, then read the Cchip DIR CSR to determine that this Pchip has detected an error.

Table 10–42 describes the Pchip error register (PERROR).

Table 10–42 Pchip Error Register (PERROR)

Field	Bits	Type	Init	Description
SYN	<63:56>	RO	0	ECC syndrome of error if CRE or UECC.
CMD	<55:52>	RO	0	PCI command of transaction when error detected if not CRE and not UECC.
				If CRE or UECC, then:
				<table><tr><td>**Value**</td><td>**Command**</td></tr><tr><td>0000</td><td>DMA read</td></tr><tr><td>0001</td><td>DMA RMW</td></tr><tr><td>0011</td><td>SGTE read</td></tr><tr><td>Others</td><td>Reserved</td></tr></table>
INV	<51>	RO Rev1 RAZ Rev0	0	Info Not Valid – only meaningful when one of bits <11:0> is set. Indicates validity of <SYN>, <CMD>, and <ADDR> fields.
				<table><tr><td>**Value**</td><td>**Mode**</td></tr><tr><td>0</td><td>Info fields are valid.</td></tr><tr><td>1</td><td>Info fields are not valid.</td></tr></table>
ADDR	<50:16>	RO	0	If CRE or UECC, then ADDR<50:19> = system address <34:3> of erroneous quadword and ADDR<18:16> = 0.
				If not CRE and not UECC, then ADDR<50:48> = 0; ADDR<47:18> = starting PCI address <31:2> of transaction when error was detected; ADDR<17:16> = 00 → not a DAC operation; ADDR<17:16> = 01 → via DAC SG Window 3; ADDR<17> = 1 → via Monster Window
RES	<15:12>	MBZ,RAZ	0	Reserved.
CRE	<11>	R,W1C	0	Correctable ECC error.
UECC	<10>	R,W1C	0	Uncorrectable ECC error.
RES	<9>	MBZ,RAZ	0	Reserved.
NDS	<8>	R,W1C	0	No **b_devsel_l** as PCI master.
RDPE	<7>	R,W1C	0	PCI read data parity error as PCI master.
TA	<6>	R,W1C	0	Target abort as PCI master.

Table 10–42 Pchip Error Register (PERROR)   (Continued)

Field	Bits	Type	Init	Description
APE	<5>	R,W1C	0	Address parity error detected as potential PCI target.
SGE	<4>	R,W1C	0	Scatter-gather had invalid page table entry.
DCRTO	<3>	R,W1C	0	Delayed completion retry timeout as PCI target.
PERR	<2>	R,W1C	0	**b_perr_l** sampled asserted.
SERR	<1>	R,W1C	0	**b_serr_l** sampled asserted.
LOST	<0>	R,W1C	0	Lost an error because it was detected after this register was frozen, or while in the process of clearing this register.

10.2.5.7 Pchip Error Mask Register (PERRMASK – RW)

If any of the MASK bits have the value 0, they prevent the setting of the corresponding bit in the PERROR register, regardless of the detection of errors or writing to PERR-SET. The default is for all errors to be disabled.

Beside masking the reporting of errors in PERROR, certain bits of PERRMASK have the following additional effects:

- If PERROR<RDPE> = 0, the Pchip ignores read data parity as the PCI master.

- If PERROR<PERR> = 0, the Pchip ignores write data parity as the PCI target.

- If PERROR<APE> = 0, the Pchip ignores address parity.

Table 10–43 describes the Pchip error mask register (PERRMASK).

Table 10–43 Pchip Error Mask Register (PERRMASK)

Field	Bits	Type	Init	Description
RES	<63:12>	MBZ,RAZ	0	Reserved
MASK	<11:0>	RW	0	PERROR register bit enables (see the text in this section and in Section 10.2.5.6)

10.2.5.8 Pchip Error Set Register (PERRSET – WO)

If any of the SET bits = 1, and the corresponding MASK bits in PERRMASK also = 1, they cause the setting of the corresponding bits in the PERROR register, the capture of the INFO into the corresponding bits in the PERROR register, and the freezing of the PERROR register. Zero (0) values in the PERRMASK register override one (1) values in the PERRSET register. If the PERROR register is already frozen when PERR-SET is written, only the LOST bit will be additionally set in PERROR.

Table 10–44 describes the Pchip error set register (PERRSET).

Table 10–44 Pchip Error Set Register (PERRSET)

Field	Bits	Type	Init	Description
INFO	<63:16>	WO	0	PERROR register information (see the text in this section)
RES	<15:12>	MBZ	0	Reserved
SET	<11:0>	WO	0	PERROR register bit set (see the text in this section and in Section 10.2.5.6)

10.2.5.9 Translation Buffer Invalidate Virtual Register (TLBIV – WO)

A write to this register invalidates all scatter-gather TLB entries that correspond to PCI addresses whose bits <31:16> and bit 39 match the value written in bits <19:4> and 27 respectively. This invalidates up to eight PTEs at a time, which are the number that can be defined in one 21264 cache block (64 bytes). Because a single TLB PCI tag covers four entries, at most two tags are actually invalidated. PTE bits <22:4> correspond to system address bits <34:16> – where PCI<34:32> must be zeros for scatter-gather window hits – in generating the resulting system address, providing 8-page (8KB) granularity.

Table 10–45 describes the translation buffer invalidate virtual register (TLBIV).

Table 10–45 Translation Buffer Invalidate Virtual Register (TLBIV)

Field	Bits	Type	Init	Description
RES	<63:28>	WO,MBZ	0	Reserved
DAC	<27>	WO	0	Only invalidate if match PCI address <39>
RES	<26:20>	WO,MBZ	0	Reserved
ADDR	<19:4>	WO	0	Only invalidate if match against PCI address <31:16>
RES	<3:0>	WO,MBZ	0	Reserved

10.2.5.10 Translation Buffer Invalidate All Register (TLBIA – WO)

A write to this register invalidates the scatter-gather TLB. The value written is ignored.

Table 10–46 describes the translation buffer invalidate all register (TLBIA).

Table 10–46 Translation Buffer Invalidate All Register (TLBIA)

Field	Bits	Type	Init	Description
RES	<63:0>	WO,MBZ	0	Reserved

10.2.5.11 Pchip Monitor Control Register (PMONCTL – RW)

This register has two fields — one each for selecting among a set of internal signals. The set of selectable signals is identical for each field. SLCT0 selects the signal that is brought to the chip output **b_monitor<0>**. SLCT1 selects the signal that is brought to the chip output **b_monitor<1>**. The chip monitor outputs are two **i_sysclk** cycles later than the defined signal. All of the defined signals are synchronized to the system clock (not the PCI clock). Also, some of the signals derived from PCI clocked signals are gated with UREN_D1_R (see Section 8.1.2.3) so that they can be used to count events that occur in the PCI clock domain, regardless of the PCI clock frequency. Others are not gated this way, so that durations can be measured in terms of system clocks.

In addition, **b_monitor<0>** is used as the input to the CNT0 field in PMONCNT, and **b_monitor<1>** is used as the input to the least significant bit in the CNT1 field in PMONCNT.

Writing any value to PMONCTL clears both fields of PMONCNT.

In normal operation, the two counters in PMONCNT stick at the value of all 1s (so that overflow can be detected). The two STKDIS control bits can disable this behavior for either counter, so that the associated counter wraps back to all 0s and continues counting. This is useful if one counter's carry-out is used as the input to the other counter.

Table 10–47 describes the Pchip monitor control register (PMONCTL).

Table 10–47 Pchip Monitor Control (PMONCTL)

Field	Bits	Type	Init	Description		
RES	<63:18>	MBZ,RAZ	0	Reserved		
STKDIS1	<17>	RW	0	Sticky count1 disable		
				Value	**Mode**	
				0	PMONCNT<CNT1> sticks at all 1s.	
				1	PMONCNT<CNT1> wraps at all 1s.	
STKDIS0	<16>	RW	0	Sticky count0 disable		
				Value	**Mode**	
				0	PMONCNT<CNT0> sticks at all 1s.	
				1	PMONCNT<CNT0> wraps at all 1s.	
SLCT1	<15:8>	RW	0	Selects chip output **b_monitor<1>**, which is also the input to the least significant bit of PMONCNT<CNT1>.		
SLCT0	<7:0>	RW	1	Selects chip output **b_monitor<0> on reset**; used to differentiate between current and previous revisions of the Pchip. Also input to the least significant bit of PMONCNT<CNT0>.		

10.2.5.12 Pchip Monitor Counters (PMONCNT – RO)

The two fields CNT0 and CNT1 count the system clock cycles during which the **b_monitor<0>** and **b_monitor<1>** signals respectively (selected by PMONCTL<SLCT0> and PMONCTL<SLCT1>) are asserted.

Both fields are cleared when any value is written to PMONCTL.

Each of the counters sticks at the value of all 1s, unless the associated STKDIS bit is set in PMONCTL.

The counters both hold their values for four cycles each time that a read to PMONCNT is performed. A slight inaccuracy can result if the events being counted continue to occur at the time of the reading.

Table 10–48 describes the Pchip monitor counters (PMONCNT).

Table 10–48 Pchip Monitor Counters (PMONCNT)

Field	Bits	Type	Init	Description
CNT1	<63:32>	RO	0	Counts **i_sysclk** cycles that **b_monitor<1>** is asserted
CNT0	<31:0>	RO	0	Counts **sysclk** cycles that **monitor<0>** is asserted

11

Chipset Clock Generation

This chapter describes the chipset input and output clocks, and their timing relationships.

11.1 Clock Generation

The 21272 chipset has seven unique clock types. They are:

- System reference clock pair (**i_sysclk**, **i_sysclk_l**)

- Forward reference clock pair (**i_fwdclk**, **i_fwdclk_l**)

- Input forwarded clocks – one per CPU (**b_c0clki_l**, **b_c1clki_l**)

- Output forwarded clocks – one per CPU (**b_c0clko_l**, **b_c1clko_l**)

- PCI output reference clocks (**b_pclko<7:0>**)

- PCI input clock (**i_pclki**)

- Memory reference clock (MEMCLK)

Signals **i_sysclk**, **i_sysclk_l**, **i_fwdclk**, **i_fwdclk_l**, **i_pclki**, **b_c0clki_l**, and **b_c1clki_l** are clock inputs to the 21272 chipset. Signals MEMCLK, **b_pclko<7:0>**, **b_c0clko_l**, and **b_c1clko_l** are output clocks generated by the 21272 chipset. The timing relationships of all 21272 clocks can be derived from input clock signal pair **i_sysclk{_l}** (see Table 11–1).

Table 11–1 Chipset Clocks

Clock	Relationship to sysclk	Signal Level	Comments
i_sysclk{_l}	–	Differential PECL	Main system clock
i_fwdclk{_l}	2***i_sysclk**	Differential PECL	Creates **b_c***n***clko_l**
pclko<*n***>/pclki**	(2***i_sysclk**)/(4 or 5 or 6)	LVTTL	Use divisor that yields a result of 33.3 MHz
MEMCLK	1***i_sysclk**	LVTTL	Address/data transfers on rising edge
b_c*n***clki_l/ b_c***n***clko_l**	2***i_sysclk** (FCLK)	Custom 2 V	Data clocked on both rising and falling edges

Figure 11–1 and Figure 11–2 show example block diagrams for a system clock implementation.

Figure 11–1 System Clock Implementation (Example 1)
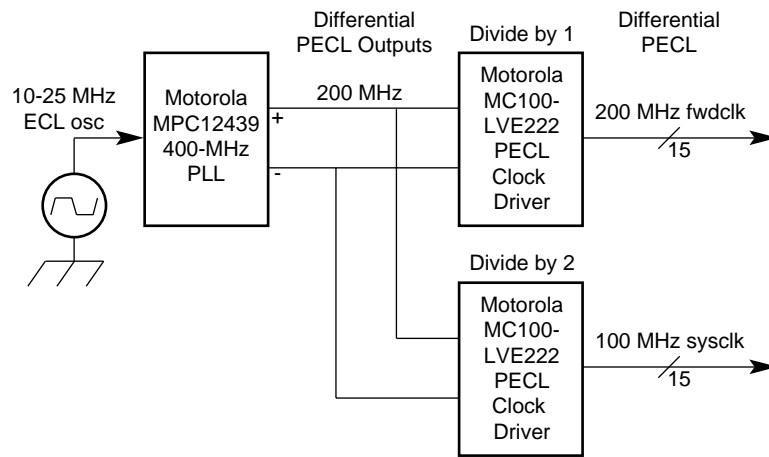
LJ-05524.AI7

Figure 11–2 System Clock Implementation (Example 2)


```
                                    ┌──────────────────┐
                                    │    12 Cache      │
                                    │    Memory        │
                                    └──────────────────┘
```

Clocks are all
Differential PECL

Cache Address,
Control and Clocking

200 MHz — osc

21264 Features:
1) 500-MHz internal clock
2) Divide by (1.5 to 3) I/O clock
3) Phase locks to system reference
    (sys ref = integer multiple of
    I/O clock)

100 MHz — ref

21264
CPU

CPU to Cchip and Dchip
interfaces are clock
forwarded

Control
(2V OD)

Forwarded
Clocks

Up to 200-MHz clock
is used when Cchip and
Dchips are sourcing data

200 MHz — pwdclk

Cchip

100 MHz — sysclk   mclk

mclk
(100 MHz)

200 MHz — pwdclk

Dchips

100 MHz — sysclk

2, 4, or 8 Dchips

3.3-V
Control

3.3-V
Data

200 MHz — pwdclk

Pchip

100 MHz — sysclk   pciclk

pclk<7:0>
/8

3.3-V Rail-
to-Rail
CMOS
Interface

mclk(100 MHz)

Motorola
MPC974
PLL

100 MHz

Up to 16

Main Memory-
SDRAM DIMMs

LJ-05559A.FH8

Figure 11–3 and Figure 11–4 show how the various clock domains for the 21272 chipset are connected within each ASIC.

Figure 11–3 Cchip/Dchip Clock System
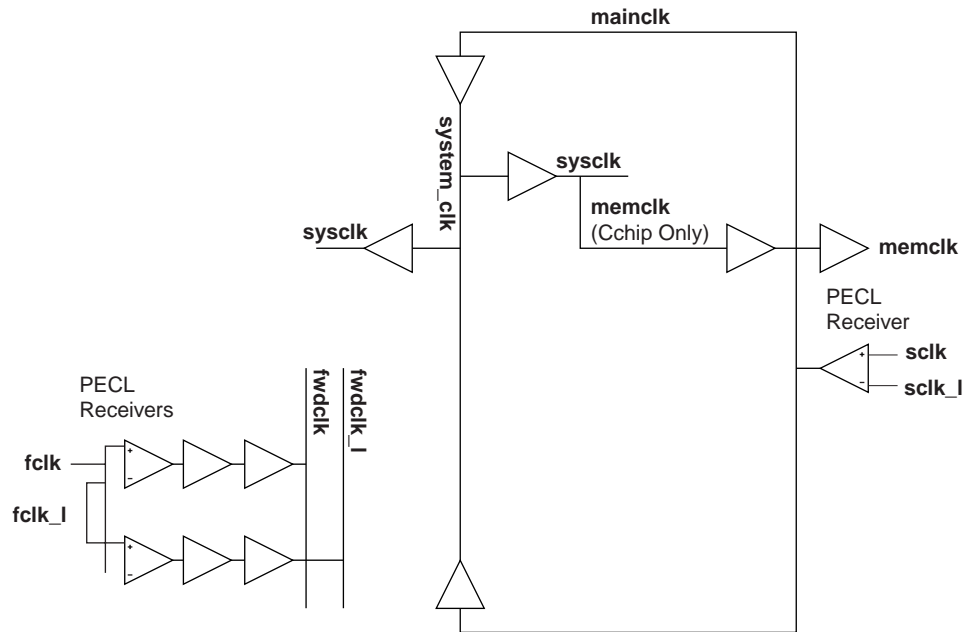
LKG-11034A-98WI

Figure 11–4 Pchip Clock System
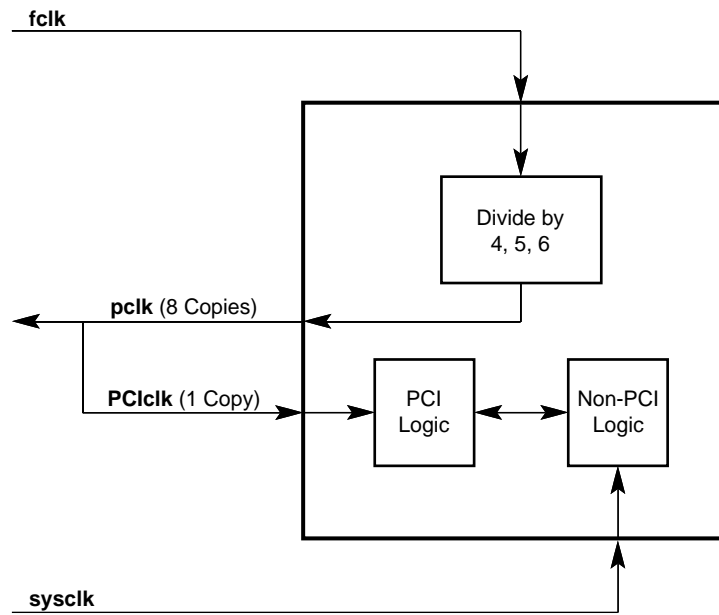
LJ-05526.AI4

11.2 PCI Bus Clocking

The 21272 chipset supports up to 33.3-MHz PCI operation and provides seven copies of the PCI clock (**b_pclko<_n_>**) for module use. An eighth copy is used as an input to the Pchip. Clocks **b_pclko<_n_>** are generated by internally dividing the **i_fwdclk** input by a user defined value of 4, 5, or 6.

11.3 SDRAM Clocking

The 21272 chipset supports synchronous DRAM interface timing. The Cchip provides a copy of the system clock for use as the memory reference clock (MEMCLK). Address and data are driven/received on the rising edge of **i_sysclk**.

11.4 Clock Skew

The 21272 chipset clock skew design parameters are listed in Table 11–2.

Table 11–2 Clock Skew Parameters

Clock	Intrachip Skew	Interchip Skew	Clock-to-Clock Skew	Duty Cycle Requirement
i_sysclk	200 ps	2.0 ns	**i_fwdclk** 100 ps **pclki** 3 ns MEMCLK 3 ns	Pulse width > 4 ns
i_fwdclk	100 ps	Unspecified	**i_sysclk** 100 ps	50/50 ±100 ps
i_pclki	300 ps	Unspecified	**i_sysclk** 3 ns	Pulse width > 11 ns
MEMCLK	Unspecified	Unspecified	**i_sysclk** 2 ns	Pulse width > 4 ns

11.5 CPU Interface Clock Forwarding

The following sections provide information about clock forwarding principles and the 21272-specific implementation.

11.5.1 Clock Forwarding Background

Clock forwarding is a technique to provide synchronous transfer of data between two chips whose I/O path delay and skew is greater than 1 clock period. For any implementation that takes multiple clock cycles to complete a data transfer, there must be a method to identify the individual clocks. In order for forwarded data transfers to work, careful matching of the delays of the data lines and the positioning of the forwarded clock with respect to the data must be performed. Because the data and the clock are sourced from the same device, and the wire delays have been carefully matched, these signals are said to be correlated. This means that the effects of process, voltage, and temperature affect the entire group in a similar fashion. The skew between the signals in the group is caused by errors in matching the signals to one another. Examples of mismatch errors can include etch variations, simultaneous switching effects, onchip process variations, and so forth.

The operation of the clock forwarding logic can be defined by a set of simple equations (not shown here), which define the following terms:

- Sample time – The elapsed time before data can be removed from the target buffer with respect to the source clock that sent the data.

- Recovered data valid time – The minimum data buffering time required on the target chip.

- Minimum bit time – The minimum time between data samples.

In practice, at least one extra target flip-flop is required to ensure proper operation. This extra flip-flop is used to ensure that the received data remains valid during the sample time. That is, it provides additional data hold time to ensure that the data valid window is greater than the uncertainty between the source and destination clocks.

Figure 11–5 shows the basic logic used for clock forwarding. The target receiver uses two input flip-flops and a multiplexer. This allows one flip-flop to be loaded while the other is being read. Figure 11–6 shows the timing of data flowing from the source chip into the receiver's clock forwarding logic, and the data as seen by the receiving chip. In Figure 11–5 the source and target clocks are operating at the same frequency. The clock forwarding clock is running at 1/2 the source clock frequency, but data is clocked on both the rising and falling edges. Flip-flop RXOUT0 clocks on the rising edge of the forwarded clock while flip-flop RXOUT1 uses the falling edge. The use of two flip-flops in this fashion doubles the data_out valid window as compared to a single flip-flop. This increased window allows sufficient time for the positioning of the target clock to successfully capture the source data into register RT.

Figure 11–5 Clock Forwarding Logic
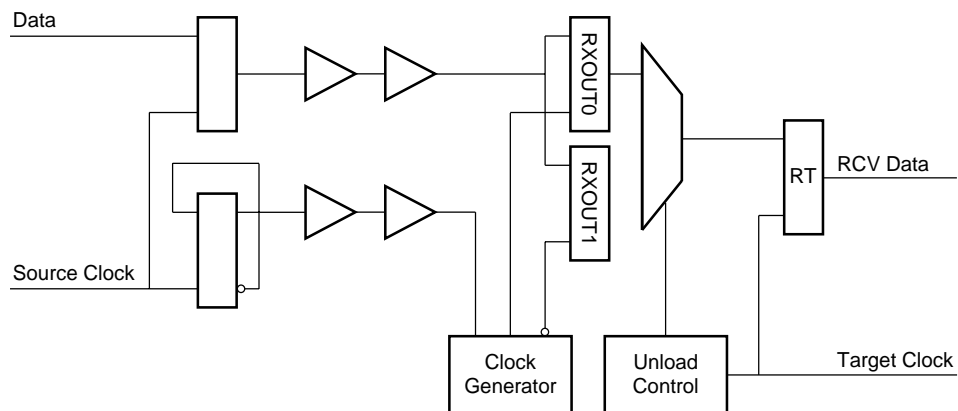
LJ-05527.AI4

Figure 11–6 Clock Forwarding Timing
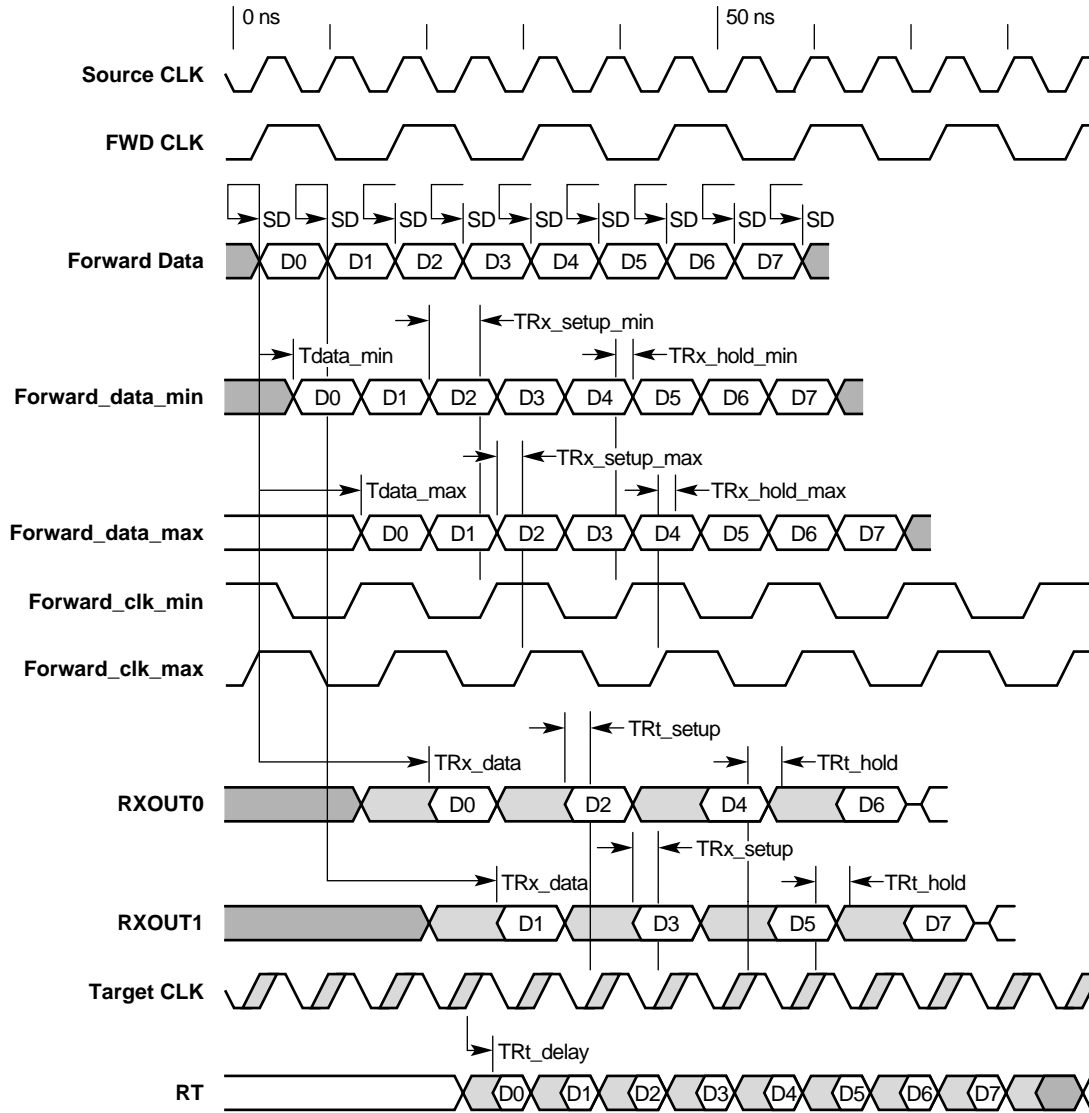
LJ-05528.AI4

11.5.2 21272 Chipset Clock Forwarding

With traditional clock forwarding, the source and target clock cycle times are identical. In the 21272 chipset design, the system clock is a multiple (1.5, 2.0, 2.5, or 3.0) of the CPU I/O clock cycle.

With the source and destination clocks running at two different frequencies, some changes to the 21272 clock forwarding logic are required to ensure that valid data can be captured from, or sent to, the CPU. Figure 11–7 shows the clock forwarding logic required for the 21264 CPU and the 21272 chipset. The figure shows the logic for data transfers in both directions. This logic assumes that the system clock cycle is 4X the CPU I/O clock cycle and that the forwarding clock is 2X. Because the CPU delivers four pieces of data for every system clock tick, the data paths in the 21272 chipset are 4X the size on the input data path. Figure 11–8 shows the logical timing operation of 21272 clock forwarding.

Figure 11–7 21272 Clock Forwarding Logic
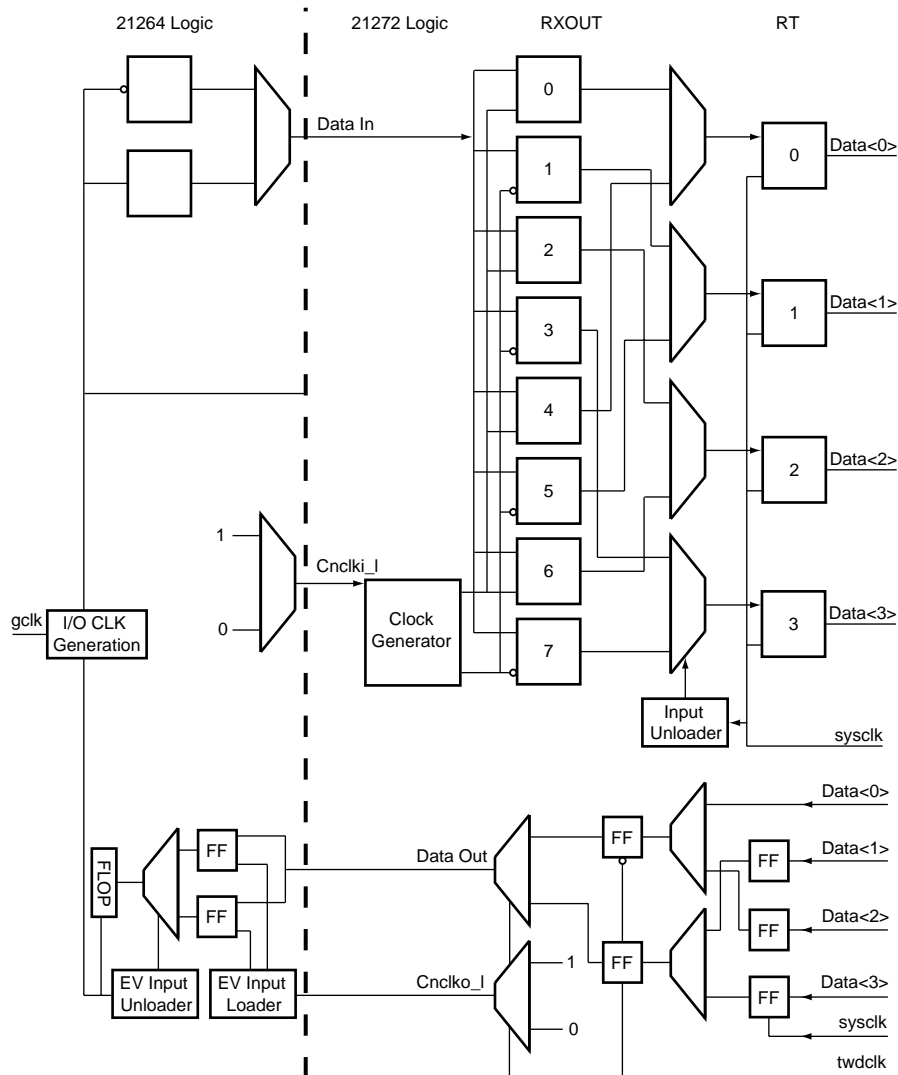
LJ-05529A.AI7

Figure 11–8 21272 Clock Forwarding Timing

LJ-05530.AI4

12
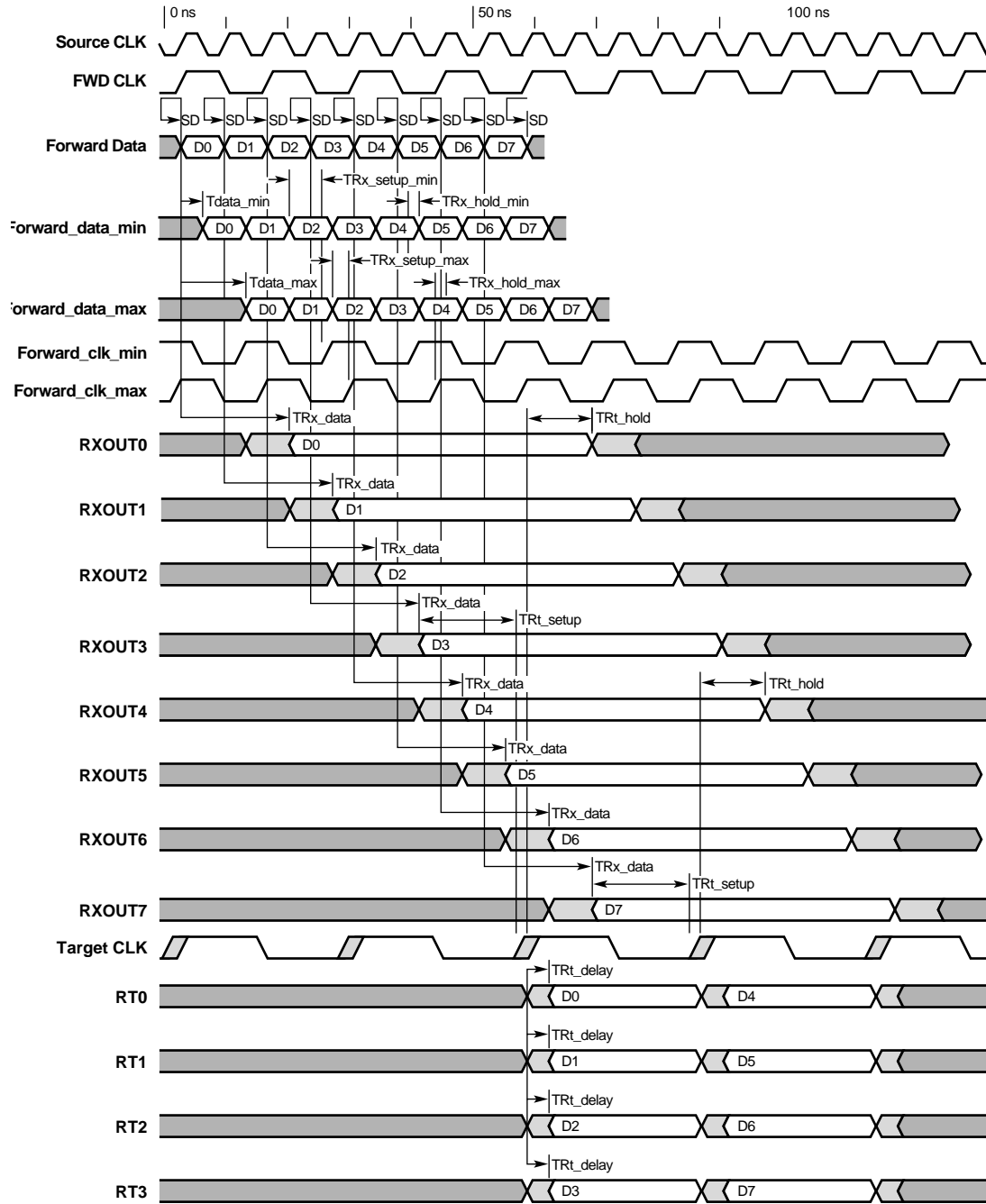
Reset, Initialization, and Power Management

This chapter describes the 21272/CPU/PCI hardware and firmware reset and initialization sequences. It also describes power management.

12.1 Hardware Initialization

Hardware initialization covers overall chipset and CPU reset timing, clock forward interface reset, and synchronous DRAM initialization.

12.1.1 Chipset Reset

The module reset signal (**b_modrst_l**) must be asserted at power-up, and held asserted for a minimum of 200 µs after power and clocks are stable. The Cchip takes as input the unsynchronized reset signal, **b_modrst_l**, and provides three copies of a reset signal with a synchronized deasserting edge (**b_sysrsta_l, b_sysrstb_l,** and **b_sysrstc_l**). Reset asserting edges are not synchronized, but are relayed by the Cchip immediately. While system reset (**i_sysrst_l**) is asserted, the Cchip holds the clock forward resets (**b_cfrst<1:0>**) asserted and tristates the TIGbus data lines (**b_td<7:0>**). Pull-ups and pull-downs on the TIGbus provide configuration information that is needed before firmware can continue the initialization process. Some of the TIGbus information is echoed by the Cchip onto the CPM command lines to the Dchips, and onto the CAPbus lines to the Pchips.

Signal **b_sysrstx_l** must be deasserted synchronously to **i_sysclk** such that all chipset chips see the deassertion within the same **i_sysclk** cycle. The Cchip provides three copies of a reset signal with a synchronous deasserting edge (**b_sysrsta_l**, **sysrstb_l**, and **sysrstc_l**). The Cchip also recirculates an internal copy of **i_sysrst_l** to maintain consistent timing with other 21272 chips. Distributing the synchronous deasserting edge through each 21272 chip takes two cycles. The synchronizer on the Cchip may take up to another three cycles. Therefore, the 21272 logic can see **i_sysrst_l** deassert up to five **i_sysclk** cycles after **b_modrst_l** deasserts.

At the deasserting edge of **sysrst_l**, all the chips store this configuration information into CSRs. The configuration information is listed in Table 12–1.

Table 12–1 Configuration Information

TIGbus Bit	Description	
<7>	Module specific spare	
<6>	SysDC fill delay	
	Value	**Cycles**
	0	2 cycles
	1	3 cycles
<5:4>	SysDC extract delay	
	Value	**Cycles**
	00	2 cycles
	01	3 cycles
	10	4 cycles
	11	5 cycles
<3>	CPU 1 clock forward preset	
<2>	CPU 0 clock forward preset	
<1:0>	Base configuration	
	Value	**Cycles**
	00	2 Dchips, 1 memory bus
	01	4 Dchips, 1 memory bus
	10	4 Dchips, 2 memory buses
	11	8 Dchips, 2 memory buses

The other CSR bits are initialized as indicated in the tables in Chapter 10. Based on these initial settings, the CSRs will be accessible from the CPU. After the **b_modrst_l** signal is deasserted, the CPU ramps up its internal clocks. When the CPU clocks are ramped up and the CPU is ready to perform its Built-In Self Test, Repair, and Initialize process (hereafter written as BiSt), it will signal the system by asserting **b_sromoe_l**. When the Cchip sees **b_sromoe_l** asserted, it sends a two SYSCLK-cycle pulse on the appropriate **b_cfrst<1:0>** line to notify each CPU to start its BiSt and SROM load sequence. When a CPU has completed its SROM load operation, it deasserts **b_sromoe_l**, indicating that the Cchip may now deassert **b_cfrst<1:0>** for that CPU. See Section 12.1.2 for more information about clock forward reset. The SROM code then initializes the TIGbus timing registers and loads the remaining firmware from a flash ROM attached to the TIGbus.

12.1.2 Clock Forward Interface Reset

The 21272 clock forward interfaces are reset during power-up reset. When **b_sysrst*x*_l** is asserted, the Cchip and Dchip asynchronously reset their clock forward interfaces, and the Chip also asserts **b_cfrst<1:0>** to reset the CPU's clock forwarding interfaces. The clock forward interface may also be reset in support of CPU sleep mode (Section 6.7). While the clock forward interfaces are in reset mode, the 21272 chips do not count incoming clock forward clocks, nor do the chips transmit clock forward clocks. When **i_sysrst_l** is deasserted, the Cchip holds **b_cfrst<1:0>** asserted until the CPU is ready to begin code execution.

The CPU deasserts **b_sromoe_l** to indicate that it is ready for the clock forward interfaces to begin operation. The Cchip deasserts the clock forward interface reset mode of operation by way of the following steps:

1. When the clock forward reset logic detects the deasserting edge of **b_sromoe_l** it notifies the Cchip issue unit, which issues a toggle clock forward reset command.

2. The Cchip issue unit waits for an idle cycle to insert the toggle clock forward reset command onto the CPM bus to the Dchip. When the issue unit sends the command on the CPM bus, it also notifies the CSR section that the command has been sent.

3. One cycle later, the CSR section deasserts the **b_cfrst<1:0>** signals to the CPU.

4. The Dchip receives and decodes the toggle clock forward reset command, and deasserts the clock forward reset signal onto the clock forward interface.

5. During this time, the Cchip's CSR section waits a number of cycles equivalent to the Dchip decode path, then deasserts the **b_cfrst<1:0>** signal onto the Cchip's clock forward interface receiving logic.

6. On both the Cchip and the Dchips, the clock forward interface aligns the deasserting edge of **b_cfrst<1:0>** to the framing clock, and then enables the clock forward interface receiving logic.

7. Four cycles later, the clock forward interfaces begin sending output clocks.

If the issue unit receives requests to toggle the clock forward interface reset for more than one CPU at a time, it prioritizes the requests to service the request for CPU0 first, and then CPU1.

For proper function, the clock forward receive circuitry must be out of reset before it receives as input, a clock from another device's clock forward transmit circuitry. The 21264 specifies a 3-cycle delay from the **i_sysclk** edge when **b_cfrst<1:0>** is seen deasserted at the CPU until the CPU's input and output circuits are operational.

12.1.3 SDRAM Initialization

The SDRAMs require the following sequence to initialize properly:

1. Pull **b_mcke** and **b_m*n*dqm<1:0>** inputs high, hold all other inputs at a no-op command, ramp up **Vdd**, and start the clock. The Cchip holds its **b_m*n*dqm<1:0>** outputs high and sends a no-op command to the SDRAMs during reset. Module components must ensure that the other requirements of this step are met.

2. Wait 100-200 μs with the inputs held to a no-op (timing depends on the SDRAM manufacturer's specifications). Signal **i_sysrst_l** must be held asserted for a minimum of 200 μs to accomplish this step.

3. Precharge all banks. To accomplish this step, the Cchip sends a precharge all banks command to the SDRAMs as soon as **i_sysrst_l** is deasserted. When the Cchip is reset, refreshing is turned off.

4. Perform a minimum of eight autorefresh cycles (fewer for some SDRAM manufacturers). To perform this step, firmware must set a refresh interval in the MTR register and wait for the proper amount of time (determined by the system cycle time and the value programmed into MTR<RI>) for the eight refresh cycles to occur. To accomplish this as fast as possible, set MTR<RI> = 4. Once the appropriate amount of time has passed, firmware should turn off refreshing again by writing a 0 into MTR<RI>. This prevents a refresh cycle from interfering with the mode register set command.

5. Perform a mode register set operation. Writing to the four MPR registers causes mode register set commands to be sent to the corresponding four memory arrays. The value written to the register is the value that will be put on the address lines to place into the mode register.

6. Set MTR<RI> to the final value for the refresh interval that the system will use.

The SDRAMs are now ready for operation.

12.2 Cchip Firmware Initialization Sequence

When the system comes out of reset, the only operations guaranteed to work are CSR reads and writes. Probes to the two CPUs are disabled. In a dual CPU system, the two CPUs can arbitrate to determine which CPU initializes the system by means of the MISC<ARB*n*> bits. The arbitration sequence is as follows:

1. Read MISC<CPUID> to obtain the CPU number.

2. Set MISC<ABT*n*> and MISC<ABW*m*> to request ownership of system initialization.

3. Issue a memory barrier instruction.

4. Read MISC<ABW>.

   If the bit corresponding to your CPU number is not set, you lost arbitration and should wait for an interprocessor interrupt from the winning CPU, notifying you that the initialization sequence is complete. Proceed with the initialization sequence.

If the bit corresponding to your CPU number is set, you have won arbitration. Read MISC<ABT> until the other CPUs have set their bits or until some timeout limit is reached. When other CPUs have set their bits, the winning CPU proceeds with the initialization process.

5. Wait for the other CPU to finish its read of MISC<ABW> so that a read does not occur during initialization of the timing registers (can be synchronized by way of an interprocessor interrupt).

The code held in the SROM is responsible for loading the rest of the firmware from a flash ROM on the TIGbus. In order to do so, it must initialize the TIGbus timing register (TTR) and the TIGbus device timing register (TDR) with the correct values for the components in the system. Also, the system timing register (STR) and Cchip system configuration register (CSC) should be initialized, although this can be delayed until after the flash ROM code is loaded into the CPU.

The correct initialization sequence for STR and CSC is as follows:

1. Read CSRs to obtain the necessary configuration information.

2. Issue a memory barrier instruction or register dependency on the last CSR read.

3. Write STR.

4. Issue a memory barrier instruction.

5. Wait 20 **i_sysclk** cycles.

6. Write CSC.

7. Issue a memory barrier instruction.

8. Wait 20 **i_sysclk** cycles.

No other sequence can guarantee correct behavior. Once this sequence is completed, no further changes are allowed to STR<IDDW> and STR<IDDR>, nor to any fields in CSC.

Next, the memory arrays must be initialized by programming the memory array sizes and timings into AAR*n* and MTR. The SDRAMs were initialized previously (as described in Section 12.1.3). This is accomplished as follows:

1. If serial presence detect pins are in use, read the SPD information from the serial ROM by using the MPD register and implementing the I^{2}C protocol in software. This provides the array sizes and SDRAM speeds. If serial presence detect pins are not in use, the timings can be held in the flash ROM, or firmware may want to use CSC<FW> to bring in timing parameters from pull-ups/pull-downs on the module.

2. Write MTR with the desired memory timing.

3. If a system has 32-byte memory buses, they may be only half-populated. If this is the case, set STR<AW> to 32 bytes and test the width by writing data and reading it back. Set STR<AW> to its proper value.

4. If serial presence detect is not in use, size the memory arrays by setting each array in turn to its largest possible size, and then, writing and reading back addresses to find out the highest order address bit in use. If serial presence detect is used, the serial data contains the SDRAM configuration.

5. Write AAR*n* with the determined memory configuration. Disable any arrays that are not present.

This concludes the memory array initialization. No further writes are allowed to AAR*n* and MTR.

Note: If array 0 is not detected, the firmware must not allow the system to boot.

Once the memory arrays are initialized and mapped, probes to the CPUs may be turned on and the I/O system initialized. The other CPU may be woken up at this time by means of an interprocessor interrupt.

12.3 PCI (Pchip) Reset

Upon assertion of **i_sysrst_l**, the Pchips asynchronously assert **b_prst_l** on the PCI bus. While **b_prst_l** is asserted, the Pchips also assert **b_req64_l**, which indicates to other PCI devices (if they see **b_req64_l** asserted) that the PCI bus is 64 bits wide.

Upon deassertion of **i_sysrst_l**, the Pchips deassert **b_prst_l** synchronously with **b_pclko<7:0>**. This synchronized deassertion occurs several cycles after the deassertion of **i_sysrst_l**. One PCI cycle after the deassertion of **b_prst_l**, the Pchips drive **b_req64_l** deasserted for one cycle, then tristate the **b_req64_l** driver.

Upon reset, the Pchip internal arbiter is disabled and remains disabled until reenabled by firmware. The Pchip contains weak pull-ups that pull these lines to their deasserted levels when the internal arbiter is disabled. The Pchip also contains weak pull-ups on its **b_req_l<6:1>** and **b_reqgnt_l<0>** inputs, and on the **b_ad<63:0>**, **b_par**, **b_par64**, and **b_cbe_l<7:0>** lines to prevent these lines from floating during reset.

12.4 SDRAM Self-Refresh/CPU and 21272 Power Down (ACPI S3)

The 21272 supports SDRAM self-refresh operations during which the CPUs and the 21272 chipset may be powered off. Additional support on the module is required for correct operation.

12.4.1 Entering SDRAM Self-Refresh

The following algorithm must be followed to enter a state where the SDRAMs maintain their contents by using self-refresh while the CPUs and 21272 chipset are powered off.

1. The CPU decides to power down and performs the housekeeping activities that use the system interface. All dirty cache data is written to system memory.

2. All DMA activity is halted.

3. Wait 100 us to ensure that the 21272 has completed any remaining SDRAM writes from the cache.

4. The CPU writes the Cchip PWR<SR> register bit, causing the 21272 to issue the self-refresh command to each of the memory arrays. Then as long as the 21272 has power, it holds the **b_mcke_l<3:0>** lines to the SDRAMs deasserted.

5. The CPU writes to the system power controller that maintains the power state. This controller can be in a South Bridge on the PCI bus, or on the 21272 TIGbus.

6. The power controller asserts a pin that will hold the **b_mcke_l<3:0>** lines deasserted while 21272 power is off.

7. The power controller turns off power to the 21272 and the CPUs.

12.4.2 Exiting SDRAM Self-Refresh

The power controller must be programmed to monitor conditions to detect a set of wake-up events. If one of these events occur, it uses the following algorithm to restore normal operation:

1. The power controller detects the enabled wake-up event.

2. The power controller turns on power to the CPUs and the 21272 chipset, and also causes the **b_modrst_l** signal to be asserted on the module, for a normal power-up.

3. As with a normal power-up sequence, the 21272 asserts the **b_mcke_l<3:0>** lines during reset, and waits to issue a precharge to all arrays when reset is deasserted.

4. Before deasserting reset, the module stops deasserting **b_mcke_l<3:0>** lines. This switches the **b_mcke_l<3:0>** lines to be asserted asynchronously to the SDRAM clocks, but that is permitted by the specification for exiting self-refresh mode. Because the 21272 is held in reset, it will not send any commands to the SDRAMs.

5. As the CPUs power up, they read the SROM and go through the power-up sequence, using the 21272 **srom_oe** handshake to enable the interface.

6. The SROM code performs a read from the power controller to determine that this is a return from a self-refresh state rather than a cold power-up. In the case of a cold power-up, the power controller has a bit that powers up in the off state.

7. Upon determining that this is not a cold power-up, the CPU firmware must program the 21272 chipset as described in Section 12.2. However, be careful that no actions are used that destroy SDRAM contents (for example, sizing of the memory array must be accomplished without reading or writing to them–presence detection methods must be used).

8. Once the 21272 is initialized, the power controller and any other devices can be allowed to cause interrupts to the 21272, which may be forwarded on to the appropriate CPUs.

Following is an alternative algorithm in which control of the **b_mcke_l<3:0>** lines is switched from the controller to the 21272 with the **b_mcke_l<3:0>** lines deasserted. This is another method for avoiding violation of the SDRAM specification for exiting self-refresh.

1. The controller notices the enabled wake-up event, powers up the 21272 and the CPU, and asserts **b_modrst_l** with the TIGbus containing the initial configuration information. Signal **b_modrst_l** is deasserted. The 21272 deasserts **b_mcke_l<3:0>** during and after reset, but the controller maintains **b_mcke_l<3:0>** deasserted on the SDRAMs.

2. The CPUs power up and the SROM initializes the 21272, recognizing that this is not a cold power-up. The 21272 is attempting to issue refresh commands to the SDRAMs, but the module switch is holding the **b_mcke_l<3:0>** lines deasserted so that these commands are not seen by the SDRAMs.

3. The CPU writes the Cchip PWR<SR> CSR bit to cause the 21272 to issue the self-refresh command (deasserting lines **b_mcke_l<3:0>**) to each of the memory arrays. This has no effect on the SDRAMs because **b_mcke_l<3:0>** is already deasserted; but it enables the next step to be completed safely.

4. The controller is directed to switch **b_mcke_l<3:0>** control from the module to the 21272. Both deassert **b_mcke_l<3:0>** at this time.

5. The CPU clears the Cchip PWR<CSR> CSR bit. The 21272 asserts **b_mcke_l<3:0>** to the SDRAMs and commences normal refresh operations one refresh later, avoiding violation of the SDRAM specification for exiting self-refresh.

6. Normal accesses to the SDRAMs through the 21272 are available to complete the power-up algorithm.

12.4.3 SDRAM Self-Refresh in Multiprocessing Systems

When exiting SDRAM self-refresh state in a multiprocessing system, the CPU must perform the sequence described in Section 12.2 to arbitrate so CPU attempts to initialize the 21272 chipset.

The individual CPUs may require the savings of unique PAL base locations while they are powered off. If necessary, this information can be saved in the power controller, or some other convention can be used to save this information in main memory (SDRAM). Description of the exact location is beyond the scope of this specification.

A
Technical Abbreviations

This appendix contains acronymns and abbreviations associated with the 21272.

Table A–1 Technical Abbreviations

Abbreviation	Description
AAR	array address register
ABT	arbitration try
ABW	arbitration won
ACL	arbitration clear
ASIC	application-specific integrated circuit
BiSt	built-in self test
BMB	byte-mask bypass
CMONCNT	Cchip monitor counters registers
CMONCTL	Cchip monitor control registers
CSALT	Technology used for 21272
CSC	Cchip system configuration register
DAC	dual-address cycle
DIM	device interrupt mask register
DIR	device interrupt request register
DQM	D/Q mask pins
DRC	delayed read completion
DREV	Dchip revision register
DRIR	Device raw interrupt request register
DRR	delayed read request
DSC	Dchip system configuration register
DWC	delayed write completion

Table A–1 Technical Abbreviations   (Continued)

Abbreviation	Description
DWR	delayed write request
ESBGA	enhanced super ball grid array
FPD	FromPchipData
FPQ	FromPchipQueue
FPR	From Pchip Requests
IIC	Cchip interval ignore count register
LDP	LoadP
MAF	miss address file
MB	memory barrier
MCTL	M-port control register
MEMCLK	memory reference clock
MPD	memory presence detect register
MPR	memory programming register
MSK	Masking
MTE	Matching
MTR	memory timing register
NXM	nonexistent memory error
NXS	nonexistent memory error (NXM) source
PCTL	Pchip control register
PECL	Pseudo ECL
PIO	Programmed I/O
PLAT	Pchip master latency register
PMONCNT	Pchip monitor counters register
PMONCTL	Pchip monitor control register
PMW	posted memory write
PRBEN	probe enable register
PTE	page table entry
PTP	peer-to-peer
PWR	power management control register
QDA	queue of downstream addresses
QDDR	queue of downstream data for reads
QDDW	queue of downstream data for writes
QUDR	queue of upstream data for reads
QUDW	queue of upstream data for writes

Table A–1 Technical Abbreviations   (Continued)

Abbreviation	Description
RMW	read-modify-write
RPB	release probe buffer
RVB	release victim buffer
SGTE	scatter-gather table entry
SPRST	soft PCI reset
STR	system timing register
TBA	translation base address
TCA	ToCpuAccumulator
TDR	TIGbus device timing register
TLB	translation lookaside buffer
TLBIA	translation buffer invalidate all register
TLBIV	translation buffer invalidate virtual register
TMA	ToMemoryAccumulator
TPD	To Pchip Data
TPQ	ToPchipQueue
TPQM	ToPchipQueueMemory
TPQP	ToPchipQueuePIO
TPR	ToPchip Requests
TTR	TIGbus timing register
VAF	victim address file
VDB	victim data buffer
WDR	wake-up delay register
WMB	WriteMergeBuffer
WQF	WaitQueueFrom Pchip
WQI	WaitQueueIssue
WQT	WaitQueueToPchip
WSBA	window space base address register
WSM	window space mask register

B
Support

B.1 Customer Support

The Alpha OEM website provides the following information for customer support.

Website	URL and Description
Alpha OEM	**http://www.digital.com/semiconductor/alpha/alpha.htm**
	Contains the following links:
	• **Developers' Area:** Development tools, code examples, driver developers' information, and technical white papers
	• **Motherboard Products:** Motherboard details and performance information
	• **Microprocessor Products:** Microprocessor details and performance information
	• **News:** Press releases
	• **Technical Information:** Motherboard firmware and drivers, hardware compatibility lists, and product documentation library
	• **Customer Support:** Feedback form

B.2 Part Numbers for Ordering Chips

To order the 21272 or 21274 chips, contact your local distributor and refer to the saleable part number as follows. The Compaq part number is used internally for tracking.

Chip	Saleable Part Number	Compaq Part Number
Tsunami Cchip	21272–C1	21–47312–03
Tsunami Dchip	21272–D1	21–47311–02
Tsunami Pchip	21272-P1	21–47310–03
Typhoon C4	21274–C1	21–49689–01
Typhoon D4	21274–D1	21–49733–01

B.3 Associated Documentation

The Alpha OEM Documentation Library is available at the following URL:

http://ftp.digital.com/pub/Digital/info/semiconductor/literature/dsc-library.html

The following table shows associated documentation that you can order from a vendor or download on the world wide web from the Alpha OEM Documentation Library.

Title and Order Number	Vendor or URL
Alpha Architecture Reference Manual EY–W938E–DP	Call your local distributor or call Butterworth-Heinemann (DIGITAL Press) at 1-800-366-2665
Alpha Architecture Handbook EC–QD2KB–TE	Alpha OEM Documentation Library: **http://ftp.digital.com/pub/Digital/info/semiconductor/ literature/dsc-library.html**
DIGITAL Alpha 21164 Microprocessor Hardware Reference Manual EC–QP99B–TE	See previous entry
DIGITAL Alpha 21164 Microprocessor Data Sheet EC–QP98C–TE	See previous entry
PCI Local Bus Specification, Revision 2.1 PCI Multimedia Design Guide, Revision 1.0 PCI System Design Guide PCI-to-PCI Bridge Architecture Specification, Revision 1.0 PCI BIOS Specification, Revision 2.1	PCI Special Interest Group U.S. 1–800–433–5177 International 1–503–797–4207 Fax 1–503–234–6762
82420/82430 PCIset ISA and EISA Bridges (includes 82378IB/ZB SIO) (PN 290483)	Intel Corporation Literature Sales P.O. Box 7641 Mt. Prospect, IL 60056 Phone: 1-800-628-8686 FaxBACK Service: 1-800-628-2283 BBS: 1-916-356-3600
Super I/O Combination Controller (FDC37C935) Data Sheet	Standard Microsystems Corporation 80 Arkay Drive Hauppauge, NY 11788 Phone: 1-516-435-6000 Fax: 1-516-231-6004

Index

Numerics

A

B

S

Scatter-gather
  associative TLB, 8-6
  mapped address translation, 10-11
  PTE structure, 10-12

SDRAM self-refresh
  entering, 12-6
  exiting, 12-7
  in multiprocessing systems, 12-8

Serial presence detect, 9-13

SERR error, 8-15

SGE error, 8-15

Shifting
  amount, 7-12
  CPU originated PIO operations, 7-11
  Pchip memory operations, 7-9
  PTP operations, 7-12

Signal name convention, xv

Skid buffers, 6-2

Sleep mode, 6-30
  entering, 6-30
  exiting, 6-31
  in multiprocessing systems, 6-32

Specifications
  ac, 4-6 to 4-16
  ac test, 4-16 to 4-19
  Cchip ac, 4-6, 4-8
  Cchip ac test, 4-16
  dc, 4-3
  Dchip ac, 4-12
  Dchip ac test, 4-18
  mechanical, 5-1
  Pchip ac, 4-14
  Pchip ac test, 4-19

Split array, 9-3

Splitting transactions, 8-10

srom_oe, 12-7

STR register, 9-13, 10-19, 10-42, 12-5

Subarray bit position, 9-10

sysclk, 6-31

SysDC
  command, 7-12
  extract delay, 10-43

System address space, 10-2

System clock implementation, 11-1

System memory, 9-1
  addressing, 9-7
  DRAM organizations, 9-4
  errors, 8-13
    CPU read/write, 6-30
    DMA read/write, 8-13
    SGTE read, 8-13
  programming, 9-7
  supported array sizes, 9-4

T

TA error, 8-14

TBAn registers, 10-3, 10-46

TDR register, 10-37, 12-5

Test specifications
  ac, 4-16 to 4-19
  Cchip ac, 4-16
  Dchip ac, 4-18
  Pchip ac, 4-19

tigadr[23:0], 10-9

TIGbus, 6-24
  address space translation, 10-9
  address timing, 6-27
  flash ROM, 6-25
  interrupt logic, 6-26
  interrupt timing, 6-26
  read timing, 6-27
  write timing, 6-27

Timing
  accumulate, 7-13
  data transfer, 7-7

TLB miss, 6-23

TLBIA register, 10-52

TLBIV register, 10-52

TPD, 6-16

TPQ, 6-10, 6-19, 7-5, 7-6, 7-7

TPQM, 7-5, 7-8

TPQP, 6-10, 6-18, 7-8

TPR, 6-16, 6-17

Transactions
  chaining, 8-10
  merging, 8-10
  splitting, 8-10

Translation
    CSR space, 10-9
    direct-mapped, 10-11
    DMA address, 10-9
    DMA monster window, 10-13
    linear configuration space, 10-6
    linear I/O space, 10-5
    linear IACK/special cycle space, 10-8
    linear memory space, 10-4
    PIO address, 10-4
    scatter-gather mapped, 10-11
    TIGbus address space, 10-9
TTR register, 10-36, 12-5

Twice split array, 9-3

U

Unaligned convention, xiv

W

WDR register, 6-30, 10-35

we_l, 10-37

Wrapping, 7-13

WSBAn registers, 10-3, 10-45

WSMn registers, 10-3, 10-11, 10-45, 10-46