DECchip 21071 and DECchip 21072 Core Logic Chipsets

Data Sheet

Order Number: EC–QAEMB–TE

Revision/Update Information: This document supersedes the *DECchip 21071 and DECchip 21072 Core Logic Chipsets Data Sheet,* (EC–QAEMA–TE).

Digital Equipment Corporation
Maynard, Massachusetts

January 1996

This document was prepared using VAX DOCUMENT Version 2.1.

DECchip 21071 and DECchip 21072 Core Logic Chipsets

Data Sheet

21071 and 21072 Features:

- Supports the entire family of DECchip 21064 Alpha AXP microprocessors
- DECchip 21071: 128-bit cache/64-bit memory
- DECchip 21072: 128-bit cache/128-bit memory
- System clock frequency up to 33 MHz
- Bcache/memory controller
  - Write-back cache
  - Bcache size from 128 KB to 16 MB
  - Bcache SRAMs, 17 ns and faster
  - 32-bit parity/32-bit ECC on Bcache (DECchip 21072 only)
  - 8 MB to 4 GB of memory supported
  - 267 MB/s CPU write bandwidth, 107 MB/s CPU read bandwidth
  - 32-bit parity/32-bit ECC on memory data (DECchip 21072 only)
  - RAS/CAS memory bus to industry-standard SIMMs
  - DRAM controller with fully programmable timing with 15 ns granularity
- High-performance PCI bridge
  120 MB/s DMA write bandwidth, 70 MB/s DMA read bandwidth, 82 MB/s programmed I/O write bandwidth, 22 MB/s programmed I/O read bandwidth
- Graphics support

The DECchip 21071 and DECchip 21072 core logic chipsets provide a cost-effective solution for designing uniprocessor systems using the DECchip 21064 family of Alpha AXP microprocessors. The chipsets include a secondary cache and memory controller, PCI interface, and corresponding data path functions. The chipsets provide ample flexibility to the system designer in building the memory and I/O subsystem, and they require minimal discrete logic on the module. The 21071 and 21072 chipsets contain three unique gate arrays:

- 21071-CA (Cache/memory controller) - 208 PQFP
- 21071-DA (PCI interface) - 208 PQFP
- 21071-BA (Data path) - 208 PQFP

Contents

1 DECchip 21071 and DECchip 21072 Core Logic Chipset Overview

Part I

2 DECchip 21071-CA Pin Descriptions

3 DECchip 21071-CA Architecture Overview

4 DECchip 21071-CA Programmer’s Reference

5 DECchip 21071-CA Transactions and Timing Diagrams

6 DECchip 21071-CA Electrical Data

7 DECchip 21071-CA Power-Up and Initialization

Part II

8 DECchip 21071-DA Pin Descriptions

9 DECchip 21071-DA Architecture Overview

10 DECchip 21071-DA Programmer's Reference

11 DECchip 21071-DA Transactions

12 DECchip 21071-DA Electrical Data

13 DECchip 21071-DA Power-Up and Initialization

Part III

14 DECchip 21071-BA Pin Descriptions

15 DECchip 21071-BA Architecture Overview

16 DECchip 21071-BA Transactions and Timing Diagrams

17 DECchip 21071-BA Electrical Data

18 DECchip 21071-BA Power-Up and Initialization

A Bcache PAL Equations

B Technical Support and Ordering Information

Figures

Tables

Preface

Purpose and Audience

This document is a support and reference document for engineers who design uniprocessor systems using an Alpha 21064 microprocessor.

Organization

This document is divided into the following parts:

- An overview of the DECchip 21071 and DECchip 21072 core logic chipsets precedes Part I.
- Part I contains information about the DECchip 21071-CA chip.
- Part II contains information about the DECchip 21071-DA chip.
- Part III contains information about the DECchip 21071-BA chip.
- Appendix A contains PAL programming equations.

This document contains the following chapters:

- Chapter 1 provides a brief overview of the DECchip 21071 and DECchip 21072 core logic chipset features.
- Chapter 2 describes the DECchip 21071-CA pin signals.
- Chapter 3 describes the DECchip 21071-CA architecture.
- Chapter 4 describes the DECchip 21071-CA control and status registers.
- Chapter 5 describes the transactions supported by the DECchip 21071-CA chip on the sysBus and memory interface.
- Chapter 6 describes the DECchip 21071-CA electrical requirements.
- Chapter 7 describes the behavior of the DECchip 21071-CA chip during power-up.
- Chapter 8 describes the DECchip 21071-DA pin signals.

- Chapter 9 describes the DECchip 21071-DA architecture.
- Chapter 10 describes the the DECchip 21071-DA control and status registers.
- Chapter 11 describes the transaction flows supported by the DECchip 21071-DA chip.
- Chapter 12 describes the DECchip 21071-DA electrical requirements.
- Chapter 13 describes the behavior of the DECchip 21071-DA chip during power-up.
- Chapter 14 describes the DECchip 21071-BA pin signals.
- Chapter 15 describes the DECchip 21071-BA architecture.
- Chapter 16 describes the flow of data within the DECchip 21071-BA chip for various transactions on the sysBus, memory data bus, and PCI bus.
- Chapter 17 describes the DECchip 21071-BA electrical requirements.
- Chapter 18 describes the behavior of the DECchip 21071-BA chip during power-up.

Conventions Used in this Document

The following conventions are used in this document:

Convention	Meaning
Note	Provides general information that could be useful.
Caution	Provides information to prevent damage to equipment.
Warning	Provides information to prevent personal injury.
Numbering	All numbers are decimal unless otherwise indicated. Numbers other than decimal are indicated with the name of the base following the number in parentheses. For example: FF (hex)
Ranges	Ranges are specified by a pair of numbers separated by two periods (..), and are inclusive. For example, a range of integers 0..4 includes the integers 0, 1, 2, 3 and 4.
Extents	Extents are specified by a pair of numbers in angle brackets separated by a colon (:), and are inclusive. For example, bits <7:3> specify an extent of bits including bits 7, 6, 5, 4, and 3.

Convention	Meaning
Clock edges	References to the rising and falling edges of clocks as defined by specifying the clock name followed by an R or F. For example, the rising edge of clk1 is referred to as clk1R and the falling edge of memClk is referred to as memClkF.
Signal edges	References to the assertion and deassertion of signals are defined by using the (∧) and (∨) characters to indicate signal rising and falling edges. For example, the deassertion of memRAS_l is referred to as memRAS_l∧.
sysBus	Refers to the DECchip 21064 pin bus (data, address, and controls) and the control signals between and the DECchip 21071-BA, DECchip 21071-CA, and the DECchip 21071-DA (or any other I/O bridge).
memClk cycle	Defined as the time from a memClk rising edge up to the next memClk rising edge. If a signal transitions in either the rising or falling edge of cycle N, then the signal is defined as occurring in cycle N.
GCR	Refers to general control register
Bcache	Refers to backup cache (secondary cache).
TLB	Refers to Translation lookaside buffer.
Byte	Contains 8 bits.
Word	Contains 16 bits.
Longword	Contains 32 bits.
Quadword	Contains 64 bits.
Octaword	Contains 128 bits.
Hexaword	Contains 256 bits (the size of one cache line).

1

DECchip 21071 and DECchip 21072 Core Logic Chipset Overview

1.1 DECchip 21071 and DECchip 21072 Core Logic Chipset Features

The DECchip 21071 and DECchip 21072 core logic chipsets provide a cost-competitive solution for designing uniprocessor systems that use the Alpha 21064 microprocessor. The DECchip 21071 chipset provides a 64-bit memory interface; the DECchip 21072 provides a 128-bit memory interface.

The chipsets include a Bcache (secondary cache) and memory controller, PCI interface, and corresponding data path functions. They provide ample flexibility to the system designer in building the memory and I/O subsystem and require minimal discrete logic on the module. The DECchip 21071 and DECchip 21072 chipsets contain three unique gate arrays:

- DECchip 21071-CA (cache/memory controller) - 208 PQFP
- DECchip 21071-DA (PCI interface) - 208 PQFP
- DECchip 21071-BA (data path) - 208 PQFP

The following list summarizes the major features of the DECchip 21071 and the DECchip 21072 chipsets:

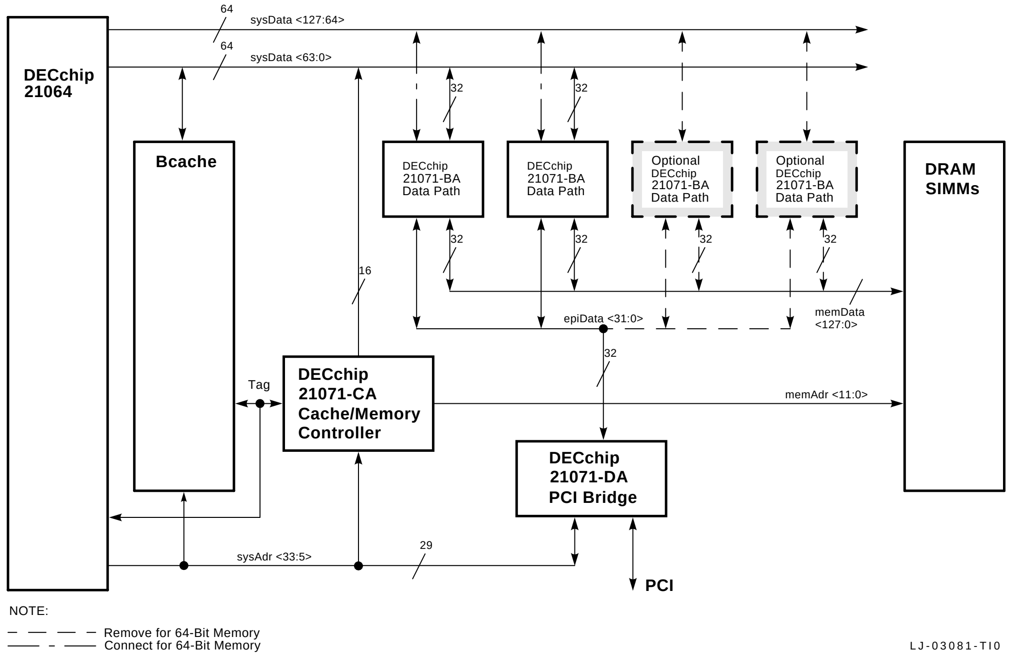
- Supports the Alpha 21064 microprocessor
- DECchip 21071 chipset:
  - Supports 128-bit cache/64-bit memory
  - Contains two DECchip 21071-BA chips
  - Contains one DECchip 21071-CA chip
  - Contains one DECchip 21071-DA chip

- DECchip 21072 chipset:
  - Supports 128-bit cache/128-bit memory
  - Contains four DECchip 21071-BA chips
  - Contains one DECchip 21071-CA chip
  - Contains one DECchip 21071-DA chip
- System clock frequency up to 33 MHz
- Bcache (secondary cache)/memory controller:
  - Write-back cache
  - Bcache size from 128 KB to 16 MB
  - Bcache SRAMs, 17 ns and faster
  - 32-bit parity/32-bit ECC on Bcache
  - 8 MB to 4 GB of memory supported
  - 267 MB/s CPU write bandwidth, 107 MB/s CPU read bandwidth
  - 32-bit parity/32-bit ECC on memory data (DECchip 21072 chipset only)
  - RAS/CAS memory bus to industry-standard SIMMs
  - DRAM controller with fully programmable timing with 15 ns granularity
  - Optional cache allocates for CPU writes
- High-performance PCI bridge:
  - 32-bit multiplexed address/data
  - Industry standard
  - No glue logic needed to connect PCI-compliant chips
  - 120 MB/s DMA write bandwidth, 70 MB/s DMA read bandwidth, 82 MB/s programmed I/O write bandwidth, 22 MB/s programmed I/O read bandwidth
  - Scatter/gather map support
- Graphics support:
  - High bandwidth memory data path to video RAM (VRAM)
  - Provides support for direct connection to VRAM frame buffer

1.2 System Overview

Figure 1–1 shows a block diagram of a system that is built using the DECchip 21071 and DECchip 21072 chipsets.

Figure 1–1 DECchip 21071 and DECchip 21072 System Block Diagram



The system is built using the following components:

- DECchip 21064 microprocessor
- DECchip 21071-BA chip
- DECchip 21071-CA chip
- DECchip 21071-DA chip
- Bcache data and tag RAMs
- Bcache control PALs
- Cache address buffers
- System clock generator

- Serial ROM interface
- Interrupt control/CPU configuration PALs
- Memory SIMMs
- PCI interrupt controller
- PCI peripherals
- PCI arbiter
- System ROM

1.2.1 Alpha 21064 Microprocessor

The DECchip 21071 and DECchip 21072 chipsets support the Alpha 21064 microprocessor. The microprocessor can run at cycle times which range between 3.3 ns and 10 ns.

The Alpha 21064 microprocessor contains two on-chip 8 KB direct-mapped caches, one for use as an instruction cache (Icache), the other for use as a data cache (Dcache). For details about the Alpha 21064 microprocessor, refer to the *Alpha 21064 and Alpha 21064A Microprocessors Hardware Reference Manual*.

1.2.2 Bcache Data and Tag RAMs

The DECchip 21071 chipset supports an optional write-back Bcache (secondary cache). System performance improves if the optional write-back Bcache is included. The size of the Bcache can range from 128 KB to 16 MB and the cache-line size is fixed at 32 bytes. The Bcache RAM data width is 128 bits. The speed of the Bcache RAMs generally ranges from 10 ns to 17 ns depending on cost and performance requirements, module routing delays of the targeted system, and the system clock cycle time.

The only restriction that the DECchip 21071 and DECchip 21072 chipsets place on the speed of the Bcache is that a read from the cache RAMs must be completed in one system clock cycle.

1.2.3 Bcache Control PALs

Systems that use the 21071-CA (cache/memory controller) chip and Bcache (secondary cache) need to implement two Bcache control PALs; these control PALs provide the tag and data RAMs with output enables, write enables, and lower address bits.

The Bcache control PALs are used to:

- Implement the NOR function between the processor-generated cache control signals and the system cache control signals.
- Generate timing of system cache control signals, so that the cache access loop generated by the 21071-CA chip can be better controlled.
- Generate some of the control signals for the processor data bus.

1.2.4 Cache Address Buffer

The cache address buffer is required to distribute the cache address to all the data and tag cache RAMs.

1.2.5 DECchip 21071-BA Features

The DECchip 21071-BA chip provides a 32-bit data path from the Alpha 21064 microprocessor to main memory and I/O. Depending on the selected width of the memory interface, two DECchip 21071-BA chips are required for a 64-bit interface, and four DECchip 21071-BA chips are required for a 128-bit interface.

The DECchip 21071-BA chip contains the cache and memory interface data path, which includes buffers for victim, noncacheable write, and DMA write operations. It also contains the I/O subsystem data path which provides buffering for DMA read and write data, and I/O read and write data.

The DECchip 21071-BA chip interfaces with the cache and CPU using the CPU sysBus (pin bus). It interfaces with the 21071-DA through the 32-bit epiBus.

The DECchip 21071-BA chip functions as the data path for the cache, memory, and I/O subsystem, and it contains the following data path functions:

Error Correction/Detection Logic: The DECchip 21071-BA chip supports longword (32 bits) parity in 64-bit and 128-bit memory mode. ECC mode may be used with 128-bit wide memory by using some of the unused higher order CPU data bits as check bits. Error checking/generation is done only on DMA-initiated transactions; error checking/generation on CPU-initiated transactions is performed by the CPU.

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

Memory Read Buffer: The memory read buffer is a one-cache-line (32 bytes), temporary holding buffer used to store data read from memory by the CPU or DMA requests.

Merge and I/O Read Buffer: The merge and I/O read buffer is a one-cache-line (32 bytes), temporary holding buffer used to store data written by the CPU on memory writes or to store data read from the PCI bus on CPU reads from I/O space.

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

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

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

1.2.6 DECchip 21071-CA Features

The DECchip 21071-CA chip provides the interface from the Alpha 21064 microprocessor to cache and main memory and includes the cache and memory controller. The DECchip 21017-CA chip controls and moves data to and from banks of main memory.

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

The DECchip 21071-CA chip (cache/memory controller) can directly control up to 16 banks of DRAM memory. Each bank may be composed of either DRAM parts or SIMMs.

Each DRAM may have 1M, 4M, or 16M addressable locations (1M x 1, 1M x 4, 4M x 1, 4M x 4, and 16M x 1 DRAM sizes are supported). Each location consists of either a quadword or octaword of data, for 64-bit and 128-bit data width, respectively. Maximum DRAM memory is 4 GB and minimum DRAM memory is 8 MB.

The DECchip 21071-CA chip provides support for a single video bankset of dual-port RAM (VRAM). This bankset can have 128K, 256K, or 512K locations. Each location consists of quadword data for a 64-bit interface, or octaword data for a 128-bit interface. VRAM capacity can vary from 1 MB to 8 MB.

The components of the cache and memory subsystem are distributed between the 21071-CA and 21071-BA. Together, the chips serve as an interface between the sysBus and memory subsystem (Figure 1–1).

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

The following list summarizes the major features of the 21071-CA chip:

- Provides control for filling the Bcache and extracting victims on CPU-initiated transactions.
- Provides control for probing the Bcache on DMA transactions and invalidating the Bcache on DMA write hits.
- Provides arbitration between the CPU and the DECchip 21071-DA chip for control of the sysBus.
- Stores addresses for the four-cache-line memory write buffer.
- Controls the loading of the I/O write buffer and the DMA read buffer.
- Uses fast-page mode on the DRAMs to improve performance on DMA burst reads and memory writes.
- Supports a frame buffer on the memory data bus.

1.2.7 DECchip 21071-DA Features

The DECchip 21071-DA chip functions as the bridge between the PCI and the CPU, its Bcache, and memory (Figure 1–1). The DECchip 21071-DA interface protocol is compliant with the PCI local bus. With the exception of a few pipeline registers and the parity tree, all the data path functions required to support the PCI reside in the DECchip 21071-BA chip.

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

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

The following list describes the major features of the DECchip 21071-DA chip:

- Scatter/gather mapping from the 32-bit PCI address to the 34-bit physical address, with on-chip, 8-entry translation lookaside buffer (TLB) for fast address translations. To reduce cost, the scatter/gather tables are stored in memory and are automatically read by the DECchip 21071-DA chip (PCI bridge) when a translation misses in the TLB.
- Supports a maximum PCI burst length of 16 longwords on PCI memory reads and writes.
- Supports two types of addressing regions on CPU-initiated transactions to PCI space.
  - Sparse space for accesses with byte and word granularities, and a maximum burst length of 2.
  - Dense space for burst lengths from 1 to 8 longwords on writes and a burst length of 2 longwords on reads. This region can be used for memory-like structures such as frame buffers, which require high bandwidth accesses.
- Stores address information for the DMA write buffer, and controls the loading of the DMA write buffer and I/O read buffer.
- Stores address information for the I/O write buffer and controls the unloading of the I/O write buffer and DMA read buffer.

Peripheral chips can be connected to the DECchip 21071-DA chip without any glue logic; however, logic that is external to the DECchip 21071-DA chip is required for interrupt arbitration, interrupt vector generation, DMA request generation, and interval timer implementation.

Note

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

1.2.8 System Clock Generator

Systems that use the DECchip 21071 or DECchip 21072 chipsets are targeted to run at system cycle times that range from 30 ns to 40 ns. The system clock generator must provide clk1x2 and clk2ref to each chip in the chipset.

Other system-specific clocks, for example, the PCI clock, must also be generated by the system clock generator. The system clock generator generates these clocks from the sysClkOut1_h and sysClkOut2_h signals, which are supplied by the Alpha 21064 microprocessor.

1.2.9 Serial ROM

The Alpha 21064 microprocessor provides an interface to a serial ROM, which can be used to initialize the instruction cache (Icache). The details for the implementation of this function can be found in the *DECchip 21064 and DECchip 21064A Alpha AXP Microprocessors Hardware Reference Manual*.

1.2.10 Interrupt Control/CPU Configuration PAL

The interrupt control/CPU configuration PAL provides system configuration information to the processor and six hardware interrupts. The PAL outputs connect to signals irq_h<5:0> from the Alpha 21064 microprocessor.

When reset_l is asserted, the PAL provides system clock configuration information and data bus width information to the processor on irq_h<5:0>.

When reset_l is deasserted, the PAL reflects the value of the system hardware interrupts to the processor on irq_h<5:0>.

1.2.11 Memory SIMMs

The DECchip 21071-CA chip (cache/memory controller) can directly control up to 16 banks of DRAM memory. Each bank may be composed of either DRAM parts or SIMMs.

Each DRAM may have 1M, 4M, or 16M addressable locations (1M x 1, 1M x 4, 4M x 1, 4M x 4, and 16M x 1 DRAM sizes are supported). Each location consists of either a quadword or octaword of data, for 64-bit and 128-bit data width, respectively. Maximum DRAM memory is 4 GB and minimum DRAM memory is 8 MB.

The DECchip 21071-CA chip provides support for a single video bankset of dual-port RAM (VRAM). This bankset can have 128K, 256K, or 512K locations. Each location consists of either a quadword or octaword of data, for 64-bit or 128-bit data width, respectively. Maximum VRAM memory is 8 MB and minimum VRAM memory is 1 MB.

1.2.12 PCI Interrupt Controller

An external interrupt controller is required to handle the interrupts posted by the PCI (and expansion bus) peripherals.

1.2.13 PCI Peripherals

The DECchip 21071 and DECchip 21072 chipsets, specifically the 21071-DA chip, can operate with any PCI-compliant, 32-bit peripheral.

1.2.14 PCI Arbiter

An external arbiter is required to determine ownership of the PCI bus during system operations.

1.2.15 System ROM

The system ROM contains all the console code and firmware that the system requires. The system ROM should be accessible to the DECchip 21071 and DECchip 21072 chipsets through the PCI bus.

Part I

Part I contains six chapters that provide information about the DECchip 21071-CA chip. The following table provides a brief description of each chapter:

Chapter	Description
2	Describes the DECchip 21071-CA pin signals.
3	Describes the DECchip 21071-CA architecture.
4	Describes the DECchip 21071-CA control and status registers.
5	Describes the transactions supported by the DECchip 21071-CA chip on the sysBus and memory interface.
6	Describes the DECchip 21071-CA electrical requirements.
7	Describes the behavior of the DECchip 21071-CA chip during power-up.

2

DECchip 21071-CA Pin Descriptions

This chapter provides a description of the DECchip 21071-CA pin signals.

2.1 DECchip 21071-CA Pin List

Table 2–1 lists the pin signals grouped by function. The information in the Type column identifies a signal as input (I), output (O), or bidirectional (B). The Buffer Strength column indicates the buffer drive strength.

All output and bidirectional pins, except pTestout, can be tristated.

Table 2–1 DECchip 21071-CA Pin List

Signal Name	Quantity	Type	Buffer Strength	Function
CPU/Bcache Signals (85 Total)				
sysData<15:0>	16	B	4 ma	Data pins for CSR data
sysAdr<33:5>	29	I	–	Address bus
tagAdr<31:17>	15	B	4 ma	Bcache tag
tagAdrP	1	B	4 ma	Bcache tag parity
tagCtlV	1	B	4 ma	Bcache valid bit
tagCtlD	1	B	4 ma	Bcache dirty bit
tagCtlP	1	B	4 ma	Bcache control parity bit
cpuCWMask<7:0>	8	I	–	Cycle write mask
cpuCReq<2:0>	3	I	–	Cycle request
cpuCAck<2:0>	3	O	4 ma	Command acknowledge
cpuDRAck<2:0>	3	O	4 ma	Data read acknowledge
cpuDWSel<1>	1	O	4 ma	Data word select
cpuDInvReq	1	O	4 ma	Dcache invalidate request
cpuHoldReq	1	O	4 ma	Hold request

(continued on next page)

Table 2–1 (Cont.) DECchip 21071-CA Pin List

Signal Name	Quantity	Type	Buffer Strength	Function
CPU/Bcache Signals (85 Total)				
cpuHoldAck	1	I	–	Hold acknowledge
Bcache/PAL Control Signals (9 Total)				
sysEarlyOEEn	1	O	8 ma	Early output enable
sysTagOEEn	1	O	8 ma	Bcache tag output enable
sysDataOEEn	1	O	8 ma	Bcache data output enable
sysDataALEn	1	O	8 ma	Bcache address bit enable low phase
sysDataAHEn	1	O	8 ma	Bcache address bit enable high phase
sysTagWE	1	O	8 ma	Bcache tag write enable
sysDataWEEn	1	O	8 ma	Bcache data short-write WE enable
sysDataLongWE	1	O	8 ma	Bcache data long-write write enable
sysDOE	1	O	8 ma	PAL CPU data output enable
PCI Bridge Interface Signals (9 Total)				
ioRequest<1:0>	2	I	–	21071-DA sysBus cycle request
ioGrant	1	O	8 ma	21071-DA sysBus cycle grant
ioCmd<2:0>	3	I	–	21071-DA command request
ioCAck<1:0>	2	O	8 ma	21071-DA command acknowledge
ioDataRdy	1	O	8 ma	21071-DA DMA read data ready

(continued on next page)

Table 2–1 (Cont.) DECchip 21071-CA Pin List

Signal Name	Quantity	Type	Buffer Strength	Function
Data Path Control Signals (16 Total)				
drvSysData	1	O	8 ma	Turns on the 21071-BA sysData<127:16> drivers
drvSysCSR	1	O	4 ma	Turns off the 21071-BA sysData<15:0> drivers
drvMemData	1	O	8 ma	Turns on the 21071-BA memData drivers
sysIORead	1	O	8 ma	Selects I/O read buffer to sysBus
sysReadOW	1	O	8 ma	Selects octaword to be returned on sysBus
subCmdA<1:0>	2	O	4 ma	Sub-commands for sysBus
subCmdB<1:0>	2	O	4 ma	Sub-commands for sysBus
subCmdCommon	1	O	8 ma	Sub-command for sysBus
sysCmd<2:0>	3	O	8 ma	Commands for sysBus side of 21071-BA chip
memCmd<3:1>	3	O	8 ma	Commands for memory side of the 21071-BA chip

(continued on next page)

Table 2–1 (Cont.) DECchip 21071-CA Pin List

Signal Name	Quantity	Type	Buffer Strength	Function
Memory Signals (39 Total)				
memAdr<11:0>	12	O	8 ma	Memory address
memRAS_l<8:0>	9	O	8 ma	Memory row address strobe
memRASB_l<8:0>	9	O	8 ma	Memory second subset RAS
memCAS_l<3:0>	4	O	8 ma	Memory column address strobe
memWE_l<1:0>	2	O	8 ma	Memory write enable
memPDClk	1	O	4 ma	Memory presence detect clock
memPDLoad_l	1	O	4 ma	Memory presence detect load enable
memPDDIn	1	I	4 ma	Memory presence detect data in
Video Support Signals (4 Total)				
vFrame_l	1	I	–	Video request for full serial register load
vRefresh_l	1	I	–	Video request for split serial register load
memDTOE_l	1	O	8 ma	Dual function data and output enable for VRAM bank
memDSF	1	O	8 ma	Special function output for VRAM bank

(continued on next page)

Table 2–1 (Cont.) DECchip 21071-CA Pin List

Signal Name	Quantity	Type	Buffer Strength	Function
Miscellaneous/Clock Signals (8 Total)				
wideMem	1	I	–	If true, indicates 128-bit wide memory
clk1x2	1	I	–	Clock input
clk2ref	1	I	–	Phase reference for clk1x2
reset_l	1	I	–	Reset
testMode	1	I	–	Test mode select
scanEnable	1	I	–	Scan enables
tristate_l	1	I	–	Tristates all output and bidirectional pins
pTestout	1	O	4 ma	Parametric NAND tree output

Pin Totals

Total input pins:	56
Total output pins:	114
Total signal pins:	170
Total power and ground pins:	36
Total pins:	206

2.2 DECchip 21071-CA Signal Descriptions

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

For simplicity, the signal sysclkOut1_h will be treated as clk1R. See Section 6.2.1 for more information about the clocks on the 21071-CA chip.

Signal descriptions are grouped by function and correspond to the pin list (Table 2–1).

Note

The Alpha 21064 microprocessor does not use clk1R; rather, it uses sysClkOut_h to generate and sample signals.

2.2.1 CPU/Bcache Signals

This section describes the CPU/Bcache signals.

2.2.1.1 sysData<15:0>

Signal Type: Bidirectional - (21071-BA, CPU, Bcache, 21071-CA)
Input Sampling Clock Edge: clk2F
Output Clock Edge: clk1R

sysData<15:0> is a bidirectional bus that provides data to and from the DECchip 21071-CA chip and the CPU. The default driver of sysData<15:0> is the CPU.

sysData<15:0> is used to read and write the CSR data for the 21071-CA chip. The 21071-CA chip does not support error checking on its CSR transactions, so corresponding sysCheck signals do not go to the 21071-CA. On a CSR read transaction, the 21071-CA chip drives sysData<15:0>. The rest of the bits are driven by the 21071-BA data chips.

2.2.1.2 sysAdr<33:5>

Signal Type: 21071-CA Input, CPU output, 21071-DA bidirectional
Input Sampling Clock Edge: Latch closes on clk1R or clk1F
Output Clock Edge: clk1R

sysAdr<33:5> contains the cache line address of sysBus transactions. sysAdr<33:32> indicates the address quadrant.

sysAdr<33:5> is driven by the CPU on CPU-initiated transactions and by the 21071-DA chip on DMA transactions.

- On CPU-initiated transactions, the 21071-CA chip latch opens when cpuCReq<2:0> becomes non-idle and closes on the next clk1R.
- On DMA transactions, the 21071-CA chip latch opens when DMA owns the sysBus and closes on the clk1F which is 1.5 cycles after the 21071-DA chip has driven the address.

2.2.1.3 tagAdr<31:17>

Signal Type: Bidirectional (21071-CA, Bcache), CPU input
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1F

tagAdr<31:17> carries Bcache tag information. The only addresses that are cached are those with sysAdr<33:32> = 00. Bits <33:32> of the tag are assumed to be 00.

The tagAdr<33:32> pins of the DECchip 21064 microprocessor should be tied to 00, and only bits <31:17> are variable. The number of significant bits of the tag depends on the depth of the Bcache RAMs and the maximum memory capacity of the system.

On a Bcache miss transaction, the tag address is driven onto tagAdr<31:17> by the 21071-CA chip and written into the tag data store.

tagAdr<31:17> is read by the processor during a cache probe. The processor does not drive these signals at any time.

The Bcache tag store drives tagAdr<31:17> with the assertion of sysEarlyOEEn, supplied by the 21071-CA chip on CPU read block, CPU write block, CPU, LD*x*_L, and CPU ST*x*_C transactions. On DMA transactions, the Bcache tag store drives tagAdr<31:17> when the 21071-CA chip asserts sysTagOEEn.

Unused tagAdr bits should be pulled down on the module.

2.2.1.4 tagAdrP

Signal Type: Bidirectional (21071-CA, CPU, Bcache)
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1F

tagAdrP is an even parity bit over the significant bits of tagAdr<33:17>. The number of bits that participate in the parity computation depend on the size of the Bcache.

2.2.1.5 tagCtlV

Signal Type: Bidirectional (21071-CA, CPU, Bcache)
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1F

tagCtlV indicates that the cache entry is valid. The 21071-CA chip sets this bit during cache fills and clears this bit during DMA writes that hit in the cache.

2.2.1.6 tagCtlD

Signal Type: Bidirectional (21071-CA, CPU, Bcache)
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1F

tagCtlD indicates that the cache entry is dirty. The 21071-CA chip sets this bit during write allocate cache fills. The processor sets this bit during CPU writes that hit in the Bcache.

2.2.1.7 tagCtlP

Signal Type: Bidirectional (21071-CA, CPU, Bcache)
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1F

tagCtlP is an even parity bit over tagCtlV and tagCtlD.

2.2.1.8 cpuCWMask<7:0>

Signal Type: 21071-CA Input
Signal Source: CPU
Input Sampling Clock Edge: clk1R and clk1F

cpuCWMask<7:0> is used on CPU-initiated read block and write block transactions. These signals carry different information on these transactions.

- On CPU write block and ST*x*_C transactions, these signals carry the longword mask for the whole cache line. An asserted cpuCWMask signal indicates that the corresponding longword from the cache line is valid and should be written.

  Any combination of mask bits is allowed on cpuCWMask<7:0> during a CPU write block transaction. CPU ST*x*_C transactions can only have combinations that correspond to a single quadword or longword.

- On CPU read block and LD*x*_L transactions, the cpuCWMask<7:0> signals carry additional information about the read transaction. cpuCWMask<1:0> carries address bits <4:3>, which indicate the address of the actual quadword that missed.

  This information can be used to implement quadword granularity to I/O space, as well as to provide wrapping in memory space. cpuCWMask<2> indicates the type of read reference. cpuCWMask<2> is true if the miss is a Dstream reference, and it is false if the miss is an Istream reference. cpuCWMask<6> is ignored, but it contains longword or quadword information on LD*x*L transactions in the Alpha 21064 microprocessor.

2.2.1.9 cpuCReq<2:0>

Signal Type: 21071-CA Input
Signal Source: CPU
Input Sampling Clock Edge: clk1F

Whenever the processor wants to initiate an external transaction, it puts a transaction type code onto cpuCReq<2:0>. Table 2–2 lists the encodings for the different transaction types.

Table 2–2 CPU-Initiated Transaction Encodings

cpuCReq<2:0>	Transaction
000	Idle
001	Barrier
010	Fetch
011	FetchM
100	Read block
101	Write block
110	LD*x*_L
111	ST*x*_C

The transaction types are held on cpuCReq<2:0> until the end of the transaction; therefore, there is no need to latch these signals.

Transactions on cpuCReq<2:0> are ignored by the 21071-CA and 21071-DA chips when the bus is granted to the 21071-DA chip for DMA transactions. cpuCReq<2:0> are ignored from the cycle that cpuHoldReq was asserted by the 21071-CA through the cycle after cpuHoldAck is deasserted at the end of the DMA transaction.

2.2.1.10 cpuCAck<2:0>

Signal Type: 21071-CA Output
Signal Destination: CPU
Output Clock Edge: clk1R

The 21071-CA chip provides transaction acknowledge information to the CPU on cpuCAck<2:0>. The 21071-CA chip is the only driver of these signals. On CPU-initiated transactions addressed to the 21071-DA or the PCI, the 21071-CA chip receives transaction acknowledge information from the 21071-DA chip on ioCmd<2:0> and forwards it to the CPU on cpuCAck<2:0> in the following cycle.

Table 2–3 lists the encodings for cpuCAck<2:0>.

Table 2–3 cpuCAck Encodings

cpuCAck<2:0>	Acknowledge	Description
000	Idle	—
001	Hard_Error	Transaction failed in a catastrophic manner.
010	Soft_Error	A failure occurred in the transaction, but was corrected. (Not used.)
011	ST*x*_C_Fail	CPU ST*x*_C transaction failed.
100	OK	Transaction completed successfully.

2.2.1.11 cpuDRAck<2:0>

Signal Type: 21071-CA Output
Signal Destination: CPU
Output Clock Edge: clk1R

The 21071-CA chip indicates to the CPU that valid read data is on the sysBus, indicates whether the data should be cached and indicates whether ECC checking and correction or parity checking should be performed. Table 2–4 lists the encodings of cpuDRAck<2:0>.

The 21071-CA chip is the only driver of these signals. On CPU-initiated transactions addressed to the 21071-DA chip or the PCI, the 21071-CA chip receives transaction acknowledge information from the 21071-DA chip on ioCmd<2:0> and forwards it to the CPU on cpuDRAck<2:0> in the following cycle.

Table 2–4 cpuDRAck Encodings

cpuDRAck<2:0>	Acknowledge	Description
000	Idle	—
100	ok_NCache_NChk	Data valid, don't cache, don't check.
101	ok_NCache	Data valid, don't cache, check ECC or parity. (Not used.)
110	ok_NChk	Data valid, cache, don't check. (Not used.)
111	ok	Data valid, cache, check ECC or parity.

2.2.1.12 cpuDWSel<1>

Signal Type: 21071-CA Output
Signal Destination: CPU
Output Clock Edge: clk1R

During a CPU write, the 21071-CA chip uses cpuDWSel<1> to indicate to the processor which data word should be driven on the sysBus.

When the CPU owns the sysBus, cpuDWSel<1> is asserted to the CPU as soon as the 21071-CA chip has decoded a write block or ST*x*_C command on cpuCReq<2:0>. Once the high octaword of CPU data has been loaded into the 21071-BA chips, cpuDWSel<1> is deasserted.

Note

The 21071-CA chip controls the rate at which CPU write data is available on the sysBus with cpuDWSel<1>. The 21071-DA chip (I/O bridge) is always capable of accepting all the data on a CPU-initiated I/O write transaction on the sysBus. The I/O write can be stalled on the sysBus by delaying cpuCAck<2:0> after all the data has been latched.

2.2.1.13 cpuDInvReq

Signal Type: 21071-CA Output
Signal Destination: CPU
Output Clock Edge: clk1R

The 21071-CA chip asserts cpuDInvReq when it needs to invalidate an entry in the CPU internal Dcache. The signal is asserted while the index to the Dcache is stable on the IAdr<12:5> pins (a buffered or unbuffered version of sysAdr<12:5>) of the CPU. This signal should be tied to the dInvReq pins of the CPU.

2.2.1.14 cpuHoldReq

Signal Type: 21071-CA Output
Signal Destination: CPU
Output Clock Edge: clk1R

The 21071-CA chip asserts cpuHoldReq to get ownership of the Bcache when the 21071-DA chip has won arbitration for the sysBus. If an external transaction is present on the sysBus, cpuHoldReq is asserted at the end of that transaction. If the bus is idle or if the 21071-DA chip is requesting preemption, cpuHoldReq is asserted right away.

2.2.1.15 cpuHoldAck

Signal Type: 21071-CA Input
Signal Source: CPU
Input Sampling Clock Edge: clk1F

The processor asserts cpuHoldAck to indicate that it has given up control of the cache to the 21071-CA chip. The minimum delay from the assertion of cpuHoldReq to the assertion of cpuHoldAck is two sysBus cycles.

The deassertion of cpuHoldReq causes cpuHoldAck to deassert in one sysBus cycle. When the processor asserts cpuHoldAck, it will have turned off its external drivers on or before clk1R. When the processor deasserts cpuHoldAck, it does not turn on its drivers for two CPU cycles after clk1R.

2.2.2 Bcache/PAL Control Signals

This section describes the Bcache/PAL control signals.

2.2.2.1 sysEarlyOEEn

Signal Type: 21071-CA Output
Signal Destination: Bcache PAL
Output Clock Edge: clk1R

sysEarlyOEEn is asserted during sysBus idle cycles to allow CPU data bus drivers, as well as data and tag RAM output enables to be asserted from the PALs as quickly as possible when the CPU asserts cpuCReq<2:0>.

sysEarlyOEEn is asserted on clk1R in the idle cycle where cpuCReq<2:0> may be asserted. When sysEarlyOEEn is asserted, cpuCReq<2:0> will cause various outputs to assert, as shown in Table 2–5.

Table 2–5 sysEarlyOEEn Effect on bcTagOE_l and bcDataOE_l

cpuCReq<2:0>	Command	bcTagOE_l	bcDataOE_l	cpuDOE_l
000	Idle	F	F	F
001	Barrier	T	T	F
010	Fetch	T	T	F
011	FetchM	T	T	F
100	Read block	T	T	F
101	Write block	T	F	T
110	LD*x*_L	T	T	F
111	ST*x*_C	T	F	T

2.2.2.2 sysTagOEEn

Signal Type: 21071-CA Output
Signal Destination: Bcache PAL
Output Clock Edge: clk1F or clk1R

sysTagOEEn is asserted by the 21071-CA chip during DMA transactions after the processor has given ownership of the cache by asserting cpuHoldAck. sysTagOEEn is also asserted during CPU-initiated, non-cacheable transactions to avoid long tristate times on tagAdr<31:17> and sysTagCtl.

sysTagOEEn is asserted on clk1R in the first cycle of a DMA transaction (the cycle when ioCmd<2:0> is driven). During all other cycles it is asserted and deasserted on clk1F.

2.2.2.3 sysDataOEEn

Signal Type: 21071-CA Output
Signal Destination: Bcache PAL
Output Clock Edge: clk1F, clk2F or clk1R

sysDataOEEn is asserted by the 21071-CA chip whenever it needs to read data from the Bcache. This occurs during a victim read, during an LD*x*_L or ST*x*_C transaction that hits in the cache, and during all DMA transactions, because the data cache is never written during DMA.

sysDataOEEn is asserted on clk1R in the first cycle of a DMA transaction (the cycle when ioCmd<2:0> is driven).

sysDataOEEn is asserted on clk1F in the cycle before CPU Write Allocate Victim data is read from the cache. (This makes the access time path from the SRAM output enable 1¼ cycles.)

In all other cases sysDataOEEn is asserted and deasserted on clk2F.

2.2.2.4 sysDataALEn

Signal Type: 21071-CA Output
Signal Destination: Bcache PAL
Output Clock Edge: clk2R
Input Sampling Clock Edge: clk2F

sysDataALEn and sysDataAHEn are sent to the PAL to generate the lower address bit for the Bcache data RAMs. The lower address bit must be toggled to the Bcache during cache fills, victim reads, reads that hit the cache, and during LD*x*_L and ST*x*_C hits.

The PAL receives the sysDataALEn signal to enable bcDataA<4> for the period when clk2 is low.

2.2.2.5 sysDataAHEn

Signal Type: 21071-CA Output
Signal Destination: Bcache PAL
Output Clock Edge: clk2F
Input Sampling Clock Edge: clk2R

sysDataALEn and sysDataAHEn are sent to the PAL to generate the lower address bit for the Bcache data RAMs. The lower address bit must be toggled to the Bcache during cache fills, victim reads, reads that hit the cache, and during LD*x*_L and ST*x*_C hits.

The PAL receives the sysDataAHEn signal to enable bcDataA<4> for the period when clk2 is high.

2.2.2.6 sysTagWE

Signal Type: 21071-CA Output
Signal Destination: Bcache PAL
Output Clock Edge: clk1R
Input Sampling Clock Edge: RAM WE

This signal is asserted when a write to the tag address and control cache RAMs is needed. sysTagWE is NORed with the CPU write enable pulse to generate the tag control write enable. sysTagWE is inverted to generate the tag address write enable.

2.2.2.7 sysDataWEEn

Signal Type: 21071-CA Output
Signal Destination: Bcache PAL
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk1F

This signal is asserted to the PALs when a write to the data cache RAMs is needed. sysDataWEEn is used if the system is performing *short writes*. The actual write enable pulse is generated by the PAL by ANDing sysDataWEEn with an inverted clk1 signal. It is then NORed with the CPU write enable signal to generate the data RAM write enable.

2.2.2.8 sysDataLongWE

Signal Type: 21071-CA Output
Signal Destination: Bcache PAL
Output Clock Edge: clk1F
Input Sampling Clock Edge: RAM WE

This signal is asserted to the PALs when a write to the data cache RAMs is needed. sysDataLongWE is used if the system is doing *long writes*. The write enable pulse is NORed with the CPU write enable pulse to generate the data RAM write enable.

2.2.2.9 sysDOE

Signal Type: 21071-CA Output
Signal Destination: PAL
Output Clock Edge: clk1R
Input Sampling Clock Edge: Flow through

sysDOE enables the processor data output enable during CPU external write cycles. sysDOE flows through the PAL and causes cpuDOE_l to assert.

2.2.3 PCI Bridge Interface Signals

This section describes the PCI bridge interface signals.

2.2.3.1 ioRequest<1:0>

Signal Type: 21071-CA Input
Signal Source: 21071-DA
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1R

The 21071-DA chip asserts ioRequest<1:0> to request ownership of the sysAdr lines to perform a DMA transaction. ioRequest<1:0> is acknowledged using ioGrant.

A request may be asserted for three cycles before the bus is actually required, because three cycles are required to acquire ownership of the Bcache from the CPU. When a DMA transaction is started, ioRequest<1:0> should be returned to idle in the same cycle as ioCmd<2:0> if no further DMA transactions are required. Table 2–6 lists the encodings for ioRequest<1:0>.

Table 2–6 ioRequest<1:0> Encodings

ioRequest<1:0>	Function
00	Idle
01	DMA preempt request
10	DMA request
11	DMA atomic request

When the 21071-DA chip uses the DMA request encoding, the bus arbiter determines who will get the bus based on which node currently has the bus and programmed priority.

The 21071-DA chip uses the DMA atomic request encoding when it needs to do multiple DMA transactions on the sysBus without the intervention of transactions from the CPU. For the first transaction, the 21071-DA chip uses the DMA request encoding. After that request has been granted, the 21071-DA chip changes ioRequest<1:0> to the DMA atomic request encoding.

Assertion of a DMA preempt request should be done only during memory barriers or to avoid deadlocks when the CPU owns the sysBus and is addressing the 21071-DA chip address space. A preempt request forces the 21071-DA chip to win arbitration and causes the 21071-CA chip to assert cpuHoldReq in the middle of the CPU transaction. The 21071-DA chip can keep DMA preempt request asserted for consecutive DMA transactions. For example, when a CPU request needs to be preempted by a DMA write transaction to flush the DMA write buffer, the 21071-DA chip should keep a DMA preempt request asserted through the entire flush of the buffer until all DMA write transactions have been completed.

2.2.3.2 ioGrant

Signal Type: 21071-CA Output
Signal Destination: 21071-DA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk1F

The 21071-CA chip indicates to the 21071-DA chip that it has won ownership of the sysBus by asserting ioGrant in response to ioRequest<1:0>. On assertion of ioGrant, the 21071-DA chip must not begin any new CPU transactions. When ioGrant and cpuHoldAck are both asserted, the 21071-DA chip may begin a new DMA transaction. If the 21071-DA chip samples ioGrant as deasserted in any cycle, its sysAdr drivers must be tristated on the next clk1R. The 21071-DA chip uses the ioGrant in combination with cpuHoldAck to determine if cpuCReq<2:0> should be ignored.

2.2.3.3 ioCmd<2:0>

Signal Type: 21071-CA Input
Signal Source: 21071-DA
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1R

The 21071-DA chip asserts ioCmd<2:0> to request an action by the 21071-CA chip. When the 21071-DA chip has the sysBus, ioCmd<2:0> is used to request a bus transaction.

When the CPU has the bus, ioCmd<2:0> is used to request assertion of the cpuCAck<2:0> and cpuDRAck<2:0> signals.

Note

There is no encoding for cpuDRAck<2:0> ok_NChk. The 21071-DA chip never returns cacheable, non-checkable read data.

A cpuCAck<2:0> or cpuDRAck<2:0> request must *not* be sent during DMA, one cycle after the 21071-CA chip sends ioGrant, or one cycle after the 21071-DA chip requests a preempt. Table 2–7 lists the encodings for ioCmd<2:0>.

Table 2–7 ioCmd<2:0> Encodings

ioCmd<2:0>	CPU Owns sysBus	21071-DA Owns sysBus
000	Idle	Idle
001	ClrLock	Flush
010	cpuDRAck ok_NCache_NChk	Write
011	cpuDRAck ok_NCache	Write masked
100	cpuCAck ok	Read
101	cpuCAck Hard_Error	Read burst
110	cpuCAck Soft_Error	Read wrapped
111	cpuCAck ST*x*C_Fail	Read burst wrapped

2.2.3.4 ioCAck<1:0>

Signal Type: 21071-CA Output
Signal Destination: 21071-DA
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1R

The 21071-CA chip asserts ioCAck<1:0> to acknowledge a DMA transaction. ioCAck<1:0> indicates that the DMA transaction has been completed. If any error occurs during the transaction, an error response is sent. Table 2–8 lists the encodings for ioCAck<1:0>.

Table 2–8 ioCAck<1:0> Encodings

ioCAck<1:0>	Function
00	Idle
01	Reserved/unused
10	DMA cycle acknowledge
11	DMA cycle error

2.2.3.5 ioDataRdy

Signal Type: 21071-CA Output
Signal Destination: 21071-DA
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1R

During any DMA read, ioDataRdy is asserted when read data is ready on the sysBus. ioDataRdy is used by the 21071-DA chip to get an early start on getting read data from the DMA read buffer without having to wait for ioCAck<1:0>. When the 21071-DA chip receives ioDataRdy, data will be available on epiData<31:0> in the next cycle.

Note

The number of ioDataRdy assertions may not correspond to the number of octawords loaded into the DMA read buffer. The 21071-DA chip must ignore ioDataRdy if a DMA read is not in progress.

When the 21071-DA chip receives ioCAck<1:0>, the entire cache block is available in the DMA read buffer. The data may be read out on epiData<31:0> two cycles after acknowledge of ioCAck<1:0> is received. (See Figure 16–1.)

2.2.4 Data Path Control Signals

This section describes the data path control signals.

2.2.4.1 drvSysData

Signal Type: 21071-CA Output
Signal Destination: 21071-BA
Output Clock Edge: clk2R assertion, clk2F deassertion
Input Sampling Clock Edge: clk1R assertion, clk1F deassertion.

drvSysData is asserted by the 21071-CA chip to indicate that the 21071-BA chip should drive sysData and sysCheck on the next clk1R. When deasserted, drvSysData indicates to the 21071-BA chip that it should tristate the sysBus on the next clk1F.

2.2.4.2 drvSysCSR

Signal Type: 21071-CA Output
Signal Destination: 21071-BA
Output Clock Edge: clk2R
Input Sampling Clock Edge: clk1R

drvSysCSR is asserted by the 21071-CA chip to indicate that the 21071-CA chip is driving sysData<15:0> on the next clk1R, and that the lower order 21071-BA chip should not drive these lines.

The drvSysCSR signal is normally deasserted, except during CSR reads. When drvSysData is asserted and drvSysCSR is not asserted, the 21071-BA chips will drive all sysData<127:0> lines.

On a CSR read to the 21071-CA chip, both drvSysData and drvSysCSR are asserted. This will result in the 21071-BA chips driving sysData<127:16> and the 21071-CA chip driving sysData<15:0>.

2.2.4.3 drvMemData

Signal Type: 21071-CA Output
Signal Destination: 21071-BA
Input Sampling Clock Edge: Flow through
Output Clock Edge: memClkR

drvMemData is asserted by the 21071-CA chip to indicate that the 21071-BA chips should drive memData on the next memClkR.

2.2.4.4 sysIORead

Signal Type: 21071-CA Output
Signal Destination: 21071-BA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

sysIORead is asserted by the 21071-CA chip and drvSysData to indicate that the contents of the I/O read buffer should be driven onto the sysBus.

2.2.4.5 sysReadOW

Signal Type: 21071-CA Output
Signal Destination: 21071-BA
Input Sampling Clock Edge: clk2F
Output Clock Edge: clk1R

sysReadOW is asserted by the 21071-CA chip to indicate to the 21071-BA chips that the upper octaword of data should be taken from the memory read, merge, and I/O read buffers.

2.2.4.6 subCmdA<1:0>, subCmdB<1:0>, subCmdCommon

Signal Type: 21071-DA Output
Signal Destination: 21071-BA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

The subCmd signals are asserted to further qualify the sysCmd<2:0> signals (Table 2–10). Table 2–9 describes how to connect the various subCmd pins from the 21071-CA chip to the 21071-BA chips.

Table 2–9 SubCmd Connections

21071-CA Pin	21071-BA Pin, 64-Bit Memory DECchip 21071 Configuration	21071-BA Pin, 128-Bit Memory DECchip 21072 Configuration
subCmdA<0>	21071-BA 0 subCmd<0>	21071-BA 0 subCmd<0>
subCmdA<1>	21071-BA 0 subCmd<1>	21071-BA 2 subCmd<0>
subCmdB<0>	21071-BA 1 subCmd<0>	21071-BA 1 subCmd<0>
subCmdB<1>	21071-BA 1 subCmd<1>	21071-BA 3 subCmd<0>
subCmdCommon	Not applicable	21071-BA 0-3 subCmd<1>

2.2.4.7 sysCmd<2:0>

Signal Type: 21071-CA Output
Signal Destination: 21071-BA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

The sysCmd<2:0> signals, in combination with the subCmd<1:0> signals indicate to the 21071-BA chip the action to take on the sysData bus. In general, they echo the actions taking place on the sysBus during the previous cycle. The bits are decoded into various actions based on the information in the following table.

Table 2–10 sysCmd<2:0> and subCmd<1:0> Encodings

sysCmd	subCmd	Mnemonic	Function
000	0X	RESET	The merge bits in the merge buffer are cleared. All sysBus counters are reset. The data in the pad latches is held (to save power).
000	1X	NOP	The data in the pad latches is held in the latches, and new data will not be clocked into them. Used during reads or to hold the first transfer of write data when the write buffer is full.
001	XX	LOAD	No write action is performed. Sent when waiting for write data to be ready. Data from the sysData bus is loaded into the pad flops.
010	XX	RDDMAS WRIO	Data in the sysData pad latches is loaded into the DMA read buffer, which also serves as the I/O write buffer. A counter is incremented so that the next RDDMAS will load data into the next sub-cache line of the buffer.
011	XX	RDDMAM	Data in the memory read buffer is loaded into the DMA read buffer. A counter is incremented so that the next RDDMAM will load data into the next sub-cache line of the buffer.

(continued on next page)

Table 2–10 (Cont.) sysCmd<2:0> and subCmd<1:0> Encodings

sysCmd	subCmd	Mnemonic	Function
100	00	MERGE00	Nothing is loaded into the merge buffer. A counter is incremented so that the next MERGE*nn* will load data into the next sub-cache line of the buffer. During ST*x*_C transactions that hit in the cache, each sub-cache line of the merge buffer is loaded twice: once with the CPU write data using MERGE (that is, MERGE01) and once with the cache data using MERGE with inverted enables, called an overlay (that is, OVLY10).
100	01	MERGE01	Same as MERGE00, except longword in the sysData pad latches is loaded into the read/merge buffer, and the merge bit that corresponds to longword 0 is set.
100	10	MERGE10	Same as MERGE00, except longword in the sysData pad latches is loaded into the read/merge buffer, and the merge bit that corresponds to longword 1 is set.
100	11	MERGE11	Same as MERGE00, except longwords 0 and 1 in the sysData pad latches are loaded into the read/merge buffer, and the merge bits that correspond to longwords 0 and 1 are set.
101	00	WRSYS0	Data in the sysData pad latches is loaded into the memory write buffer, which represents cache line 0. A counter is incremented so that the next WRSYS0 will load data into the next sub-cache line of cache line 0.
101	01	WRSYS1	Data in the sysData pad latches is loaded into the memory write buffer, which represents cache line 1. A counter is incremented so that the next WRSYS1 will load data into the next sub-cache line of cache line 1.
101	10	WRSYS2	Data in the sysData pad latches is loaded into the memory write buffer, which represents cache line 2. A counter is incremented so that the next WRSYS2 will load data into the next sub-cache line of cache line 2.

(continued on next page)

Table 2–10 (Cont.) sysCmd<2:0> and subCmd<1:0> Encodings

sysCmd	subCmd	Mnemonic	Function
101	11	WRSYS3	Data in the sysData pad latches is loaded into the memory write buffer, which represents cache line 3. A counter is incremented so that the next WRSYS3 will load data into the next sub-cache line of cache line 3.
110	00	WRDMAS0	Data in the sysData pad latches is merged with the DMA write buffers and loaded into the memory write buffer, which represents cache line 0. A counter is incremented so that the next WRDMAS0 will load data into the next sub-cache line of cache line 0.
110	01	WRDMAS1	Data in the sysData pad latches is merged with the DMA write buffers and loaded into the memory write buffer, which represents cache line 1. A counter is incremented so that the next WRDMAS1 will load data into the next sub-cache line of cache line 1.
110	10	WRDMAS2	Data in the sysData pad latches is merged with the DMA write buffers and loaded into the memory write buffer, which represents cache line 2. A counter is incremented so that the next WRDMAS2 will load data into the next sub-cache line of cache line 2.
110	11	WRDMAS3	Data in the sysData pad latches is merged with the DMA write buffers and loaded into the memory write buffer, which represents cache line 3. A counter is incremented so that the next WRDMAS3 will load data into the next sub-cache line of cache line 3.
111	00	WRDMAM0	Data in the memory read buffer is merged with the DMA write buffers and loaded into the memory write buffer, which represents cache line 0. A counter is incremented so that the next WRDMAM0 will load data into the next sub-cache line of cache line 0.

(continued on next page)

Table 2–10 (Cont.) sysCmd<2:0> and subCmd<1:0> Encodings

sysCmd	subCmd	Mnemonic	Function
111	01	WRDMAM1	Data in the memory read buffer is merged with the DMA write buffers and loaded into the memory write buffer, which represents cache line 1. A counter is incremented so that the next WRDMAM1 will load data into the next sub-cache line of cache line 1.
111	10	WRDMAM2	Data in the memory read buffer is merged with the DMA write buffers and loaded into the memory write buffer, which represents cache line 2. A counter is incremented so that the next WRDMAM2 will load data into the next sub-cache line of cache line 2.
111	11	WRDMAM3	Data in the memory read buffer is merged with the DMA write buffers and loaded into the memory write buffer, which represents cache line 3. A counter is incremented so that the next WRDMAM3 will load data into the next sub-cache line of cache line 3.

2.2.4.8 memCmd<3:1>

Signal Type: 21071-CA Output
Signal Destination: 21071-BA
Output Clock Edge: clk2R
Input Sampling Clock Edge: clk1R

The memCmd<3:1> signals indicate to the 21071-BA chips the action to take on the memData bus. memCmd<3:1> is driven by the 21071-CA chip on clk2R and latched by the 21071-BA chip on clk1R.

The bits are decoded into various actions. Table 2–11 provides a complete description of the memCmd<3:1> encodings.

Table 2–11 memCmd<3:1> Encodings

memCmd	Mnemonic	Function
010	NOP	No operation.
011	RESET	All memory pointers in the 21071-BA chip are reset.
000	RDIMM	Read data is loaded into the read/merge buffer on the next memClkR. A counter is incremented so that the next RD*xxx* will load data into the next available sub-cache line of the read buffer.
001	RDDLY	Read data is loaded into the read/merge buffer on the memClkR after the next memClkR. A counter is incremented so that the next RD*xxx* will load data into the next available sub-cache line of the read buffer.
100	WRIMM	Data from the memory write buffer is driven to memory on the next memClkR. A counter is incremented so that the next WR*xxx* will drive the next sub-cache line to memory.
101	WRDLY	Data from the memory write buffer is driven to memory on the memClkR after the next memClkR. A counter is incremented so that the next WR*xxx* will drive the next sub-cache line to memory.
110	WRIMML	Data from the memory write buffer is driven to memory on the next memClkR. After the write, the quadword pointer is reset to 0, and the cache line pointer is incremented so that the next WR*xxx* will drive the first sub-cache line of the next cache line to memory.
111	WRDLYL	Data from the memory write buffer is driven to memory on the memClkR after the next memClkR. After the write, the quadword pointer is reset to 0, and the cache line pointer is incremented so that the next WR*xxx* will drive the first sub-cache line of the next line to memory.

2.2.5 Memory Signals

This section describes the memory signals.

2.2.5.1 memAdr<11:0>

Signal Type: 21071-CA Output
Signal Destination: Memory
Output Clock Edge: memClkR

memAdr<11:0> is the time multiplexed address bus that provides the row and column addresses to the memory.

2.2.5.2 memRAS_l<8:0>

Signal Type: 21071-CA Output
Signal Destination: Memory
Output Clock Edge: memClkR (Programmable)

memRAS_l<8:0> is asserted on memory read or write transactions and video serial register loads to indicate the presence of a valid row address on memAdr<11:0>. Each memRAS_l<8:0> signal corresponds to one of the nine banksets as determined by the memory address decode logic. memRAS_l<8:0> is asserted on memory reads and writes only if the subbank number is 0, or if subbanks for that bank are disabled (Bx_SUBENA=0). On memory refresh transactions, memRAS_l<8:0> is asserted.

2.2.5.3 memRASB_l<8:0>

Signal Type: 21071-CA Output
Signal Destination: Memory
Output Clock Edge: memClkR (Programmable)

memRASB_l<8:0> functions similarly to the memRAS_l<8:0> signals, except that memRASB_l<8:0> is asserted on memory reads and writes only if the subbank number is 1. If subbanks for that bank are disabled (B*x*_SUBENA=0), the memRASB_l line of that bank will assert only for refreshes.

2.2.5.4 memCAS_l<3:0>

Signal Type:21071-CA Output
Signal Destination: Memory
Output Clock Edge: memClkR (Programmable)

memCAS_l<3:0> signals are used during memory reads and writes to indicate that a valid column address is on memAdr<11:0>. During memory writes, memCAS_l<3:0> asserts if the respective memory longwords are being written. On memory reads, all memCAS_l bits are asserted. memCAS_l<3:0> is also asserted during refreshes and video serial register loads.

2.2.5.5 memWE_l<1:0>

Signal Type: 21071-CA Output
Signal Destination: Memory
Output Clock Edge: memClkR (Programmable)

memWE_l<1:0> signals are asserted on a memory write transaction to indicate that valid write data is present on the memData outputs. memWE_l<0> and memWE_l<1> are identical copies provided to reduce loading.

2.2.5.6 memPDClk

Signal Type: 21071-CA Output
Signal Destination: Presence Detect Shift Register
Output Clock Edge: clk2R

memPDClk provides a clock at one-fourth the clk1 frequency. This clock is connected to the presence detect shift registers. memPDLoad_l and the sampling of memPDDIn are referenced to memPDClk. memPDClk starts as soon as reset_l is deasserted, and discontinues after all data has been shifted into the presence detect Control Status Registers (CSRs).

2.2.5.7 memPDLoad_l

Signal Type: 21071-CA Output
Signal Destination: Presence Detect Shift Register
Output Clock Edge: clk2R

memPDLoad_l asserts to indicate that the presence detect pins should be loaded into the presence detect shift register. When memPDLoad_l is asserted, at least one memPDClk will occur. This enables the use of either asynchronous or synchronous loading shift registers.

2.2.5.8 memPDDIn

Signal Type: 21071-CA Input
Signal Source: Presence Detect Shift Register
Input Clock Edge: clk2R

The memPDDIn signal contains the data from the presence detect shift register. The value of memPDDIn is shifted into the 21071-CA chip presence detect registers one sysClock after memPDClk deasserts (which is three sysClocks after memPDClk asserts). The data is loaded Most Significant Bit (MSB) first into the registers (a shift right).

2.2.6 Video Support Signals

This section describes the video support signals.

2.2.6.1 vFrame_l

Signal Type: 21071-CA Input
Signal Source: External logic
Input Clock Edge: Asynchronous

Assertion of vFrame_l causes the video display pointer to be loaded with the contents of the video frame pointer register which is located in the 21071-CA chip. A full serial register load to the video bank is requested at the video display pointer address.

The vFrame_l signal is edge sensitive and asynchronous with the 21071-CA chip clocks. Assertion of vFrame_l is detected and synchronized with memClk before being used.

vFrame_l has a weak internal pull-up to support systems that do not use the video support functionality provided by the 21071-CA chip.

2.2.6.2 vRefresh_l

Signal Type: 21071-CA Input
Signal Source: External logic
Input Clock Edge: Asynchronous

Assertion of vRefresh_l causes the incremented value of the video display pointer to be latched into the video display pointer. A split serial register load cycle to the video bank is requested at the video display pointer address.

The vRefresh_l signal is edge sensitive and asynchronous with the 21071-CA chip clocks. Assertion of vRefresh_l is detected and synchronized with memClk before being used.

VRefresh_l has a weak internal pull-up to support systems that do not use the video support functionality provided by the 21071-CA chip.

2.2.6.3 memDTOE_l

Signal Type: 21071-CA Output
Signal Destination: Memory
Output Clock Edge: memClkR

The memDTOE_l signal has two functions and is intended to be used only by the single video bank. During random access reads and writes, memDTOE_l is held deasserted before asserting memRAS_l. For random reads, memDTOE_l is asserted with the first column address. During a serial register load, memDTOE_l is asserted with the row address. This signal, along with memDSF, is used at memRAS_l<8> or memRASB_l<8> assertion by the VRAMs to perform full or split register loads.

2.2.6.4 memDSF

Signal Type: 21071-CA Output
Signal Destination: Memory
Output Clock Edge: memClkR

The memDSF signal is used at memRAS_l<8> assertion by the single video bank to choose between full and split serial register loads. memDSF is driven with the row address in order to set up memRAS_l<8> or memRASB_l<8>.

2.2.7 Miscellaneous/Clock Signals

This section describes the miscellaneous and clock signals.

2.2.7.1 wideMem

Signal Type: 21071-CA Input
Signal Source: Static
Input Clock Edge: Static

The wideMem signal, an input to the 21071-CA and 21071-BA chips, indicates the size of the memory data bus. wideMem is tied high to indicate a 128-bit wide memory data bus (four 21071-BA chips). wideMem is tied low to indicate a 64-bit wide memory data bus (two 21071-BA chips).

wideMem has a weak internal pull down and a Schmitt trigger input.

2.2.7.2 clk1x2

Signal Type: 21071-CA Input
Signal Source: Clock Generator

clk1x2 is a clock input which supplies a clock at twice the frequency of the DECchip 21064 sysClkOut1 signal, with a minimum period of 15 ns and a 50 percent duty cycle.

2.2.7.3 clk2ref

Signal Type: 21071-CA Input
Signal Source: Clock Generator

clk2ref is a signal input which is low when the assertion of clk1x2 corresponds to the assertion of sysClkOut1. The received signal must be set up to the assertion of clk1x2.

2.2.7.4 reset_l

Signal Type: 21071-CA Input
Signal Source: External Logic
Input Clock Edge: Asynchronous on assertion, clk1R on deassertion

Assertion of reset_l sets all internal logic and state machines to their initialized states. During reset, the memory data bus is driven, and the sysBus data and tag buses are tristated. All signals that are sent to the Alpha 21064 microprocessor are guaranteed to be tristated or held low, to prevent more than 3.0 volts from entering the Alpha 21064 microprocessor during reset.

2.2.7.5 testMode

Signal Type: 21071-CA Input
Signal Source: Test logic
Input Clock Edge: Asynchronous

Assertion of testMode places the chip into a mode for chip testing. testMode is intended to be used only during chip testing and must be tied low during normal system operation.

testMode has a weak internal pull down and a Schmitt trigger input.

2.2.7.6 scanEnable

Signal Type: 21071-CA Input
Signal Source: Test logic

Assertion of scanEnable places all internal flops in their scan state. scanEnable is intended to be used only during chip testing and must be tied low during normal system operation.

scanEnable has a weak internal pull down and a Schmitt trigger input.

2.2.7.7 tristate_l

Signal Type: 21071-CA Input
Signal Source: External logic
Input Clock Edge: Asynchronous

Assertion of this signal tristates all output and bidirectional drivers. tristate_l is intended for use only during chip testing and power-up.

tristate_l has a weak internal pull-up and a Schmitt trigger input.

2.2.7.8 pTestout

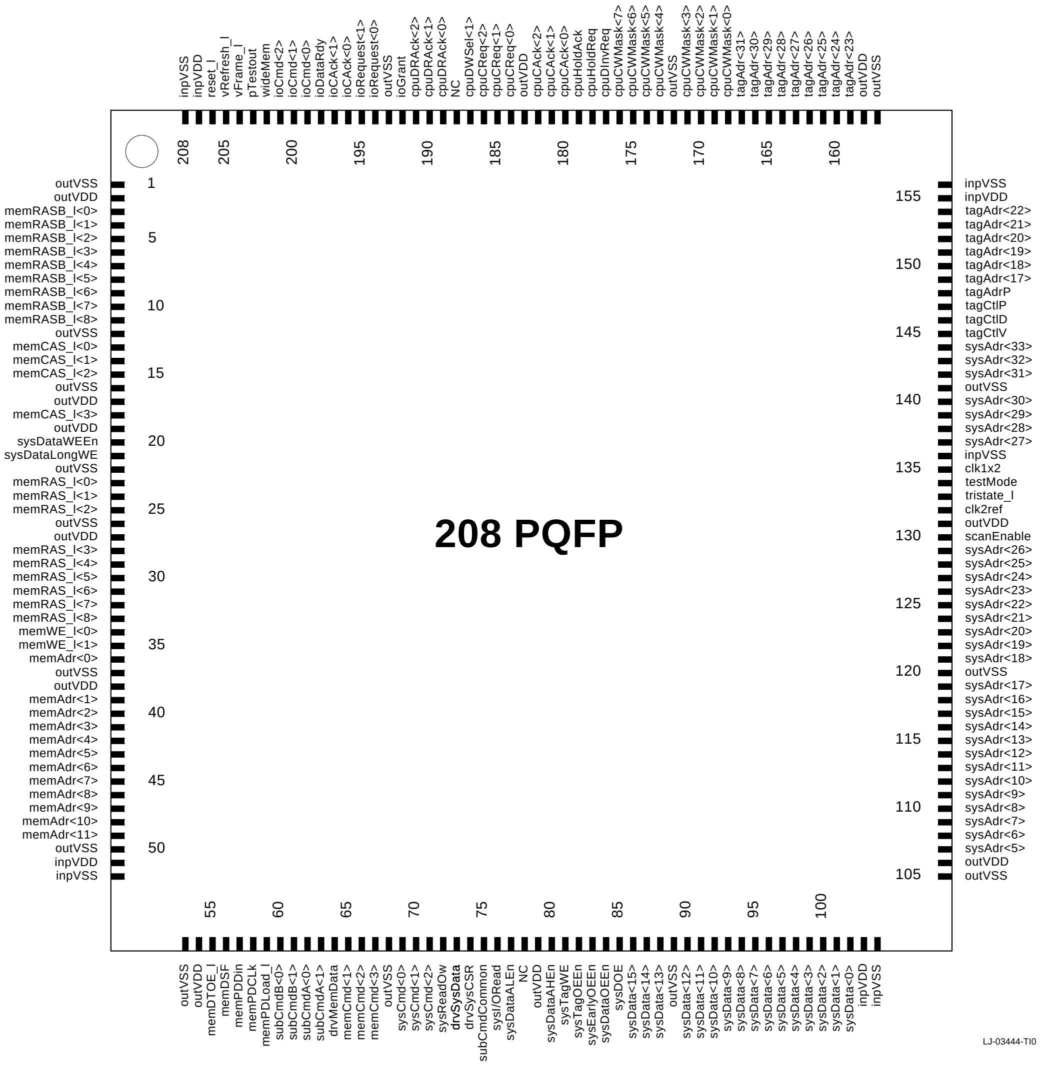
Signal Type: 21071-CA Output
Signal Destination: Test logic
Output Clock Edge: Flow through

The pTestout signal contains the output from the parametric NAND tree, as required for testing. The tristated signal must be asserted for pTestout to be valid. pTestout is intended for use only during chip or module testing.

2.3 DECchip 21071-CA Pin Assignment

The DECchip 21071-CA is a 208-pin plastic quad flat pack (PQFP). Figure 2–1 shows the signal assignments. Sections 2.3.1 and 2.3.2 provide alphabetical and numerical pin listings.

Figure 2–1 DECchip 21071-CA Pinout Diagram



2.3.1 DECchip 21071-CA Alphabetical Pin Assignment List

Table 2–12 lists the DECchip 21071-CA pins in alphabetical order. The following abbreviations are used in the Type column of the table:

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

Table 2–12 DECchip 21071-CA Alphabetical Pin Assignment List

Pin Name	Pin	Type	Pin Name	Pin	Type
clk1x2	135	I	**InpVdd**	155	P
clk2ref	132	I	**InpVdd**	207	P
cpuCAck<0>	180	O	**InpVdd**	103	P
cpuCAck<1>	181	O	**InpVdd**	51	P
cpuCAck<2>	182	O	**InpVss**	104	P
cpuCReq<0>	184	I	**InpVss**	156	P
cpuCReq<1>	185	I	**InpVss**	136	P
cpuCReq<2>	186	I	**InpVss**	52	P
cpuCWMask<0>	168	I	**InpVss**	208	P
cpuCWMask<1>	169	I	**ioCack<0>**	196	O
cpuCWMask<2>	170	I	**ioCack<1>**	197	O
cpuCWMask<3>	171	I	**ioCmd<0>**	199	I
cpuCWMask<4>	173	I	**ioCmd<1>**	200	I
cpuCWMask<5>	174	I	**ioCmd<2>**	201	I
cpuCWMask<6>	175	I	**ioDataRdy**	198	O
cpuCWMask<7>	176	I	**ioGrant**	192	O
cpuDinvReq	177	O	**ioRequest<0>**	194	I
cpuDRack<0>	189	O	**ioRequest<1>**	195	I
cpuDRack<1>	190	O	**memAdr<0>**	36	O
cpuDRack<2>	191	O	**memAdr<1>**	39	O
cpuDWSel<1>	187	O	**memAdr<2>**	40	O
cpuHoldAck	179	I	**memAdr<3>**	41	O
cpuHoldReq	178	O	**memAdr<4>**	42	O
drvMemData	64	O	**memAdr<5>**	43	O
drvSysCSR	74	O	**memAdr<6>**	44	O
drvSysData	73	O	**memAdr<7>**	45	O

Pin Name	Pin	Type	Pin Name	Pin	Type
memAdr<8>	46	O	**memRAS_l<1>**	24	O
memAdr<9>	47	O	**memRAS_l<2>**	25	O
memAdr<10>	48	O	**memRAS_l<3>**	28	O
memAdr<11>	49	O	**memRAS_l<4>**	29	O
memCAS_l<0>	13	O	**memRAS_l<5>**	30	O
memCAS_l<1>	14	O	**memRAS_l<6>**	31	O
memCAS_l<2>	15	O	**memRAS_l<7>**	32	O
memCAS_l<3>	18	O	**memRAS_l<8>**	33	O
memCmd<1>	65	O	**memWE_l<0>**	34	O
memCmd<2>	66	O	**memWE_l<1>**	35	O
memCmd<3>	67	O	**nc***	188	—
memDSF	56	O	**nc***	78	—
memDTOE_l	55	O	**outVdd**	19	P
memPDClk	58	O	**outVdd**	27	P
memPDDin	57	I	**outVdd**	79	P
memPDLoad_l	59	O	**outVdd**	183	P
memRASB_l<0>	3	O	**outVdd**	17	P
memRASB_l<1>	4	O	**outVdd**	38	P
memRASB_l<2>	5	O	**outVdd**	54	P
memRASB_l<3>	6	O	**outVdd**	158	P
memRASB_l<4>	7	O	**outVdd**	106	P
memRASB_l<5>	8	O	**outVdd**	131	P
memRASB_l<6>	9	O	**outVdd**	2	P
memRASB_l<7>	10	O	**outVss**	1	P
memRASB_l<8>	11	O	**outVss**	37	P
memRAS_l<0>	23	O	**outVss**	120	P

*nc—Do not connect these pins on board.

Pin Name	Pin	Type	Pin Name	Pin	Type
outVss	16	P	**sysAdr<19>**	122	I
outVss	68	P	**sysAdr<20>**	123	I
outVss	22	P	**sysAdr<21>**	124	I
outVss	50	P	**sysAdr<22>**	125	I
outVss	12	P	**sysAdr<23>**	126	I
outVss	172	P	**sysAdr<24>**	127	I
outVss	105	P	**sysAdr<25>**	128	I
outVss	89	P	**sysAdr<26>**	129	I
outVss	157	P	**sysAdr<27>**	137	I
outVss	141	P	**sysAdr<28>**	138	I
outVss	26	P	**sysAdr<29>**	139	I
outVss	193	P	**sysAdr<30>**	140	I
outVss	53	P	**sysAdr<31>**	142	I
pTestout	203	O	**sysAdr<32>**	143	I
reset_l	206	I	**sysAdr<33>**	144	I
scanEnable	130	I	**sysAdr<5>**	107	I
subCmdA<0>	62	O	**sysAdr<6>**	108	I
subCmdA<1>	63	O	**sysAdr<7>**	109	I
subCmdB<0>	60	O	**sysAdr<8>**	110	I
subCmdB<1>	61	O	**sysAdr<9>**	111	I
subCmdCommon	75	O	**sysCmd<0>**	69	O
sysAdr<10>	112	I	**sysCmd<1>**	70	O
sysAdr<11>	113	I	**sysCmd<2>**	71	O
sysAdr<12>	114	I	**sysDataAHEn**	80	O
sysAdr<13>	115	I	**sysDataALEn**	77	O
sysAdr<14>	116	I	**sysDataLongWE**	21	O
sysAdr<15>	117	I	**sysDataOEEn**	84	O
sysAdr<16>	118	I	**sysDataWEEn**	20	O
sysAdr<17>	119	I	**sysData<0>**	102	B
sysAdr<18>	121	I	**sysData<1>**	101	B

Pin Name	Pin	Type	Pin Name	Pin	Type
sysData<10>	92	B	**tagAdr<18>**	150	B
sysData<11>	91	B	**tagAdr<19>**	151	B
sysData<12>	90	B	**tagAdr<20>**	152	B
sysData<13>	88	B	**tagAdr<21>**	153	B
sysData<14>	87	B	**tagAdr<22>**	154	B
sysData<15>	86	B	**tagAdr<23>**	159	B
sysData<2>	100	B	**tagAdr<24>**	160	B
sysData<3>	99	B	**tagAdr<25>**	161	B
sysData<4>	98	B	**tagAdr<26>**	162	B
sysData<5>	97	B	**tagAdr<27>**	163	B
sysData<6>	96	B	**tagAdr<28>**	164	B
sysData<7>	95	B	**tagAdr<29>**	165	B
sysData<8>	94	B	**tagAdr<30>**	166	B
sysData<9>	93	B	**tagAdr<31>**	167	B
sysDOE	85	O	**tagCtlD**	146	B
sysEarlyOEEn	83	O	**tagCtlP**	147	B
sysIORead	76	O	**tagCtlV**	145	B
sysReadOW	72	O	**testMode**	134	I
sysTagOEEn	82	O	**triState_l**	133	I
sysTagWE	81	O	**vFrame_l**	204	I
tagAdrP	148	B	**vRefresh_l**	205	I
tagAdr<17>	149	B	**wideMem**	202	I

2.3.2 DECchip 21071-CA Numerical Pin Assignment List

Table 2–13 lists the DECchip 21071-CA pins in numerical order. The following abbreviations are used in the Type column of the table:

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

Table 2–13 DECchip 21071-CA Numerical Pin Assignment List

Pin Name	Pin	Type	Pin Name	Pin	Type
outVss	1	P	**outVss**	26	P
outVdd	2	P	**outVdd**	27	P
memRASB_l<0>	3	O	**memRAS_l<3>**	28	O
memRASB_l<1>	4	O	**memRAS_l<4>**	29	O
memRASB_l<2>	5	O	**memRAS_l<5>**	30	O
memRASB_l<3>	6	O	**memRAS_l<6>**	31	O
memRASB_l<4>	7	O	**memRAS_l<7>**	32	O
memRASB_l<5>	8	O	**memRAS_l<8>**	33	O
memRASB_l<6>	9	O	**memWE_l<0>**	34	O
memRASB_l<7>	10	O	**memWE_l<1>**	35	O
memRASB_l<8>	11	O	**memAdr<0>**	36	O
outVss	12	P	**outVss**	37	O
memCAS_l<0>	13	O	**outVdd**	38	P
memCAS_l<1>	14	O	**memAdr<1>**	39	O
memCAS_l<2>	15	O	**memAdr<2>**	40	O
outVss	16	P	**memAdr<3>**	41	O
outVdd	17	P	**memAdr<4>**	42	O
memCAS_l<3>	18	O	**memAdr<5>**	43	O
outVdd	19	O	**memAdr<6>**	44	O
sysDataWEEn	20	O	**memAdr<7>**	45	O
sysDataLongWE	21	O	**memAdr<8>**	46	O
outVss	22	P	**memAdr<9>**	47	O
memRAS_l<0>	23	O	**memAdr<10>**	48	O
memRAS_l<1>	24	O	**memAdr<11>**	49	O
memRAS_l<2>	25	O	**outVss**	50	P

Pin Name	Pin	Type	Pin Name	Pin	Type
inpVdd	51	P	**outVdd**	79	P
inpVss	52	P	**sysDataAHEn**	80	O
outVss	53	P	**sysTagWE**	81	O
outVdd	54	P	**sysTagOEEn**	82	O
memDTOE_l	55	O	**sysEarlyOEEn**	83	O
memDSF	56	O	**sysDataOEEn**	84	O
memPDDIn	57	O	**sysDOE**	85	O
memPDClk	58	O	**sysData<15>**	86	B
memPDLoad_l	59	O	**sysData<14>**	87	B
subCmdB<0>	60	O	**sysData<13>**	88	B
subCmdB<1>	61	O	**outVss**	89	P
subCmdA<0>	62	O	**sysData<12>**	90	B
subCmdA<1>	63	O	**sysData<11>**	91	B
drvMemData	64	O	**sysData<10>**	92	B
memCmd<1>	65	O	**sysData<9>**	93	B
memCmd<2>	66	O	**sysData<8>**	94	B
memCmd<3>	67	O	**sysData<7>**	95	B
outVss	68	P	**sysData<6>**	96	B
sysCmd<0>	69	O	**sysData<5>**	97	B
sysCmd<1>	70	O	**sysData<4>**	98	B
sysCmd<2>	71	O	**sysData<3>**	99	B
sysReadOW	72	O	**sysData<2>**	100	B
drvSysData	73	O	**sysData<1>**	101	B
drvSysCSR	74	O	**sysData<0>**	102	B
subCmdCommon	75	O	**inpVdd**	103	P
sysIORead	76	O	**inpVss**	104	P
sysDataALEn	77	O	**outVss**	105	P
nc[*]	78	–	**outVdd**	106	P

[*] **nc**—Do not connect these pins on board.

Pin Name	Pin	Type	Pin Name	Pin	Type
sysAdr<5>	107	I	**sysAdr<28>**	138	I
sysAdr<6>	108	I	**sysAdr<29>**	139	I
sysAdr<7>	109	I	**sysAdr<30>**	140	I
sysAdr<8>	110	I	**outVss**	141	P
sysAdr<9>	111	I	**sysAdr<31>**	142	I
sysAdr<10>	112	I	**sysAdr<32>**	143	I
sysAdr<11>	113	I	**sysAdr<33>**	144	I
sysAdr<12>	114	I	**tagCtlV**	145	B
sysAdr<13>	115	I	**tagCtlD**	146	B
sysAdr<14>	116	I	**tagCtlP**	147	B
sysAdr<15>	117	I	**tagAdrP**	148	B
sysAdr<16>	118	I	**tagAdr<17>**	149	B
sysAdr<17>	119	I	**tagAdr<18>**	150	B
outVss	120	P	**tagAdr<19>**	151	B
sysAdr<18>	121	I	**tagAdr<20>**	152	B
sysAdr<19>	122	I	**tagAdr<21>**	153	B
sysAdr<20>	123	I	**tagAdr<22>**	154	B
sysAdr<21>	124	I	**inpVdd**	155	P
sysAdr<22>	125	I	**inpVss**	156	P
sysAdr<23>	126	I	**outVss**	157	P
sysAdr<24>	127	I	**outVdd**	158	P
sysAdr<25>	128	I	**tagAdr<23>**	159	B
sysAdr<26>	129	I	**tagAdr<24>**	160	B
scanEnable	130	I	**tagAdr<25>**	161	B
outVdd	131	P	**tagAdr<26>**	162	B
clk2Ref	132	I	**tagAdr<27>**	163	B
tristate_l	133	I	**tagAdr<28>**	164	B
testMode	134	I	**tagAdr<29>**	165	B
clk1x2	135	I	**tagAdr<30>**	166	B
inpVss	136	P	**tagAdr<31>**	167	B
sysAdr<27>	137	I	**cpuCWMask<0>**	168	I

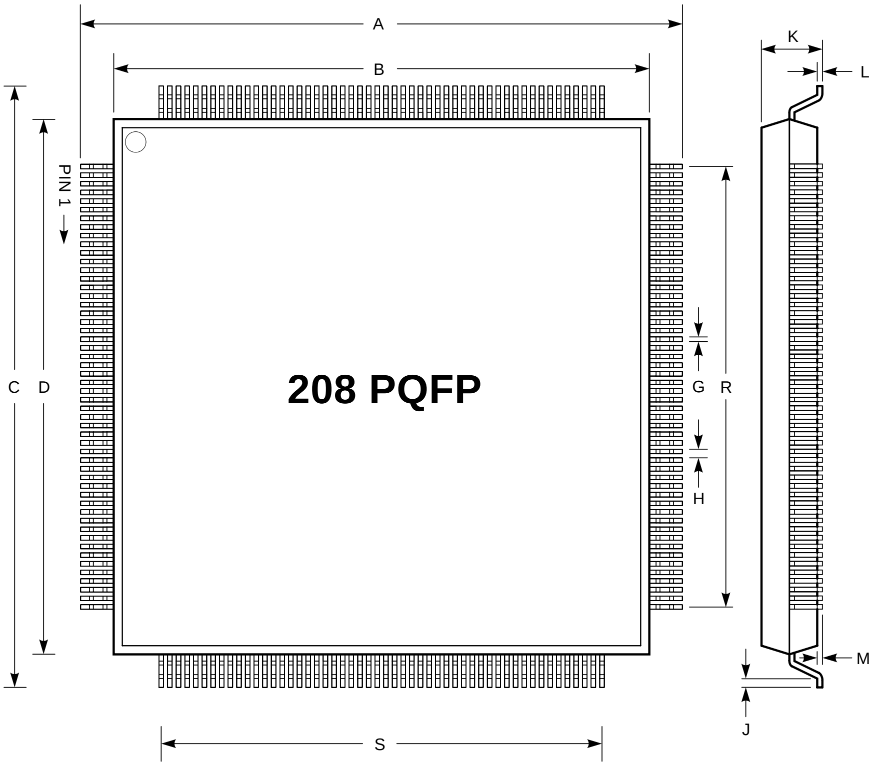
Pin Name	Pin	Type	Pin Name	Pin	Type
cpuCWMask<1>	169	I	**ioCAck<0>**	196	O
cpuCWMask<2>	170	I	**ioCAck<1>**	197	O
cpuCWMask<3>	171	I	**ioDataRdy**	198	O
outVss	172	P	**ioCmd<0>**	199	I
cpuCWMask<4>	173	I	**ioCmd<1>**	200	I
cpuCWMask<5>	174	I	**ioCmd<2>**	201	I
cpuCWMask<6>	175	I	**wideMem**	202	I
cpuCWMask<7>	176	I	**pTestout**	203	I
cpuDInvReq	177	O	**vFrame_l**	204	I
cpuHoldReq	178	O	**vRefresh_l**	205	I
cpuHoldAck	179	I	**reset_l**	206	I
cpuCAck<0>	180	O	**inpVdd**	207	P
cpuCAck<1>	181	O	**inpVss**	208	P
cpuCAck<2>	182	O			
outVdd	183	P			
cpuCReq<0>	184	I			
cpuCReq<1>	185	I			
cpuCReq<2>	186	I			
cpuDWSel<1>	187	O			
nc[*]	188	—			
cpuDRAck<0>	189	O			
cpuDRAck<1>	190	O			
cpuDRAck<2>	191	O			
ioGrant	192	O			
outVss	193	P			
ioRequest<0>	194	I			
ioRequest<1>	195	I			

[*] **nc**—Do not connect these pins on board.

2.4 DECchip 21071-CA Mechanical Specifications

Figure 2–2 shows the DECchip 21071-CA package dimensions.

Figure 2–2 DECchip 21071-CA Package Dimensions



DIM	Millimeters		Inches	
	MIN	MAX	MIN	MAX
A	30.50	30.77	1.201	1.211
B	27.90	28.10	1.098	1.106
C	30.50	30.77	1.201	1.211
D	27.90	28.10	1.098	1.106
G	0.23	0.33	0.009	0.013
H	.500 BSC		0.0197 BSC	
J	0.45	0.62	0.018	0.024
K	3.45	3.85	0.136	0.152
L	0.13	0.23	0.005	0.009
M	0.25	0.35	0.010	0.012
R	25.5 REF		1.004 REF	
S	25.5 REF		1.004 REF	

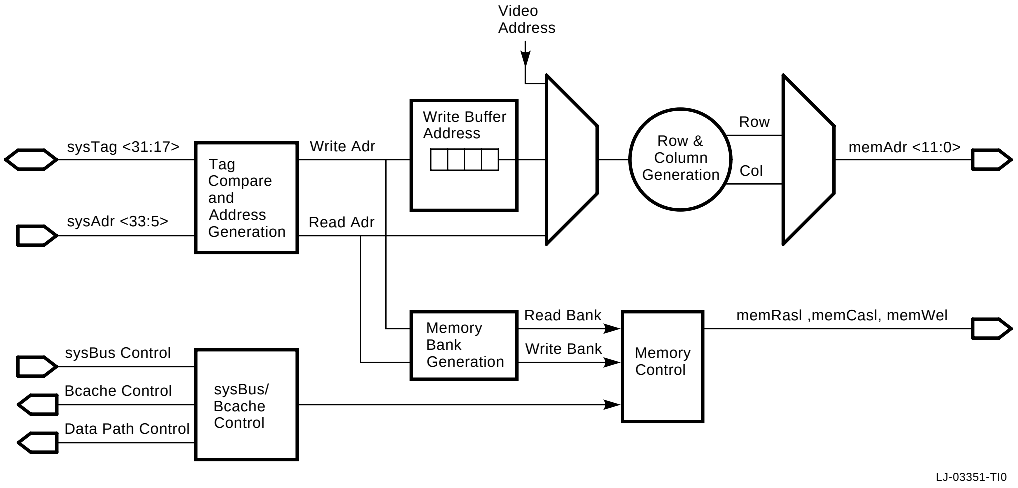
LJ-03666-TI0

3

DECchip 21071-CA Architecture Overview

This chapter describes the DECchip 21071-CA architecture. The 21071-CA chip provides both second-level cache and memory control functions. The 21071-CA chip also controls the cache/memory data path located on the 21071-BA chip. Figure 3–1 shows a block diagram of the 21071-CA chip.

Figure 3–1 DECchip 21071-CA Block Diagram



3.1 sysBus Interface Architecture

The CPU, 21071-DA chip, 21071-BA chips, cache, and 21071-CA chip communicate with each other over the sysBus. The sysBus is essentially the processor pinbus with additional signals for DMA transaction control, arbitration, and cache control.

The sysBus interface contains:

- sysBus arbiter
- Bcache controller
- Write buffer address and control
- Read/merge buffer control
- Lock register

3.1.1 sysBus Arbitration

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

3.1.1.1 Arbitration CSRs

The arbitration policy of the 21071-CA chip can be programmed by setting up the DMA_ARB CSR field to select whether the CPU or the 21071-DA chip has highest priority. There are three possible priority encodings:

- **CPU priority**

  When the CPU and DMA are simultaneously requesting the sysBus, the CPU is given the priority.

- **DMA priority**

  DMA is given priority over the CPU, and the bus is released to the cache on DMA cache misses or noncacheable DMA transactions.

- **DMA strong priority**

  DMA is given priority over the CPU. If another ioRequest<1:0> is pending, the bus is not released to the cache on DMA cache misses or noncacheable DMA transactions.

3.1.1.2 DECchip 21071-DA Requests

The 21071-CA arbiter monitors requests for the sysBus by decoding the cpuCReq<2:0> and ioRequest<1:0> fields. cpuCReq<2:0> is not a bus request; it is a cycle command that indicates that the CPU has started a transaction on the sysBus.

When the 21071-CA arbiter detects the assertion of ioRequest<1:0> and when DMA has won arbitration, it makes a request to the CPU for control of the Bcache by asserting cpuHoldReq to the CPU.

The 21071-DA chip can make three types of requests for the sysBus:

- **Atomic Request**

  This request is used if the 21071-DA chip wants to do multiple transactions without interruption from the CPU. When the 21071-DA chip already has a DMA transaction in progress, the assertion of atomic request will override programmed priority. If the 21071-DA chip does not already have a transaction in progress, the assertion of atomic request is equivalent to sending a plain DMA request.

  Note

  To guarantee atomicity, the 21071-DA chip must assert an atomic request in the cycle that it drives the command on the ioCmd<2:0> lines for the first transaction.

- **Preempt Request**

  This request should be used by the 21071-DA chip for deadlock prevention. A preempt request causes the arbiter to request the CPU to suspend a transaction in progress. If the 21071-DA chip must do multiple DMA transactions for deadlock prevention, it must keep preempt request or atomic asserted until all deadlocked transactions have completed.

  When the 21071-DA chip changes ioRequest<1:0> from preempt to idle or plain DMA request, the arbiter will allow the suspended CPU transaction to resume.

A preempt request must be used *only* on CPU transactions addressed to the 21071-DA chip; that is, I/O reads, I/O writes, fetch, fetchM to 21071-DA space, and barriers. Preempt must not be asserted when the sysBus is idle or on any transactions not addressed to the 21071-DA chip.

Note

Because a preempt request suspends the CPU transaction in progress, it should be used only if that transaction cannot complete without the completion of the requesting DMA transaction.

- **DMA Request**

  This is the ordinary DMA request. No special priority is given to DMA request unless the arbiter is so programmed.

3.1.1.3 Arbitration Cycles

The cycle in which arbitration occurs depends on whether the CPU or the 21071-DA chip has control of the bus. Arbitration will occur at the following times:

- When a CPU transaction is in progress, arbitration will occur up to two cycles before the assertion of cpuCAck<2:0> to the CPU. Table 3–1 shows the arbitration cycles of CPU transactions.

  If the arbiter receives ioRequest<1:0> at this time, the 21071-DA is granted (independent of programmed priority), and cpuHoldReq is asserted to get control of the Bcache.

Table 3–1 Arbitration Cycles of CPU Transactions

Two Cycles Before cpuCAck	One Cycle Before cpuCAck
CPU read block, CSR or memory	CPU read block, I/O space
CPU write block, CSR or memory	CPU write block, I/O space
CPU fetch, CSR or memory	CPU fetch, I/O space
CPU ST*x*_C hit	CPU ST*x*_C fail
CPU LD*x*_L hit dirty	CPU memory barrier
—	Any error

- When a DMA transaction is in progress, arbitration will occur one cycle before ioCAck<1:0> is sent to the 21071-DA chip. The result of arbitration depends on programmed priority if both the CPU and the 21071-DA chip are requesting the bus.
- When the sysBus is idle, arbitration occurs every cycle. When a sysBus idle cycle is followed by requests from both the CPU and the 21071-DA chip, the CPU will be granted (independent of programmed priority or the ioRequest field). This is because the CPU has already started the transaction on the sysBus, and the 21071-DA chip cannot stall write data transfers soon enough.

3.1.1.4 Grant Mechanism

After the 21071-DA chip has made a request, and the arbiter has determined that the 21071-DA chip should be granted the bus, the 21071-CA chip asserts ioGrant to the 21071-DA chip and cpuHoldReq to the CPU in the same cycle. After cpuHoldReq has been asserted, the 21071-CA and 21071-DA chips must ignore cpuCReq<2:0> until the transaction is complete (ioCAck<1:0> has been returned) and cpuHoldAck has been deasserted.

After the 21071-DA chip detects that both ioGrant and cpuHoldAck have been asserted, it will drive its command address and data lines as appropriate.

Note

The ioCmd<2:0> encodings change as soon as the 21071-DA chip has the bus.

After the 21071-DA chip has received cpuHoldAck, it is expected to take away ioRequest<1:0> in the cycle it drives ioCmd<2:0>, unless it has another transaction to do. The 21071-DA chip may choose to withdraw ioRequest<1:0> without doing a transaction; in this case it should drive IDLE on the ioCmd<2:0> pins, until it removes ioRequest<1:0>. If the 21071-DA chip withdraws ioRequest<1:0> for one or more cycles after receiving cpuHoldAck, performance may be affected, but no other adverse behavior will occur.

During DMA transactions, the 21071-DA chip will drive the DMA address on the sysAdr lines until the 21071-CA chip has completed the Bcache probe and latched the DMA address. After the address is latched by the 21071-CA chip, the arbiter may decide that it wants to release the cache back to the CPU by deasserting cpuHoldReq.

When this *release* decision has been made, ioGrant will be deasserted, indicating to the 21071-DA chip that it needs to tristate its address lines. The arbiter releases the cache on either DMA read or masked write transactions that don't use the cache or DMA full write transactions if the memory write buffer is full. The release will not occur if the programmed arbitration priority is DMA strong and the ioRequest<1:0> lines are non-idle or if the 21071-DA chips ioRequest<1:0> lines are driving DMA atomic or DMA preempt.

3.1.1.5 Releases

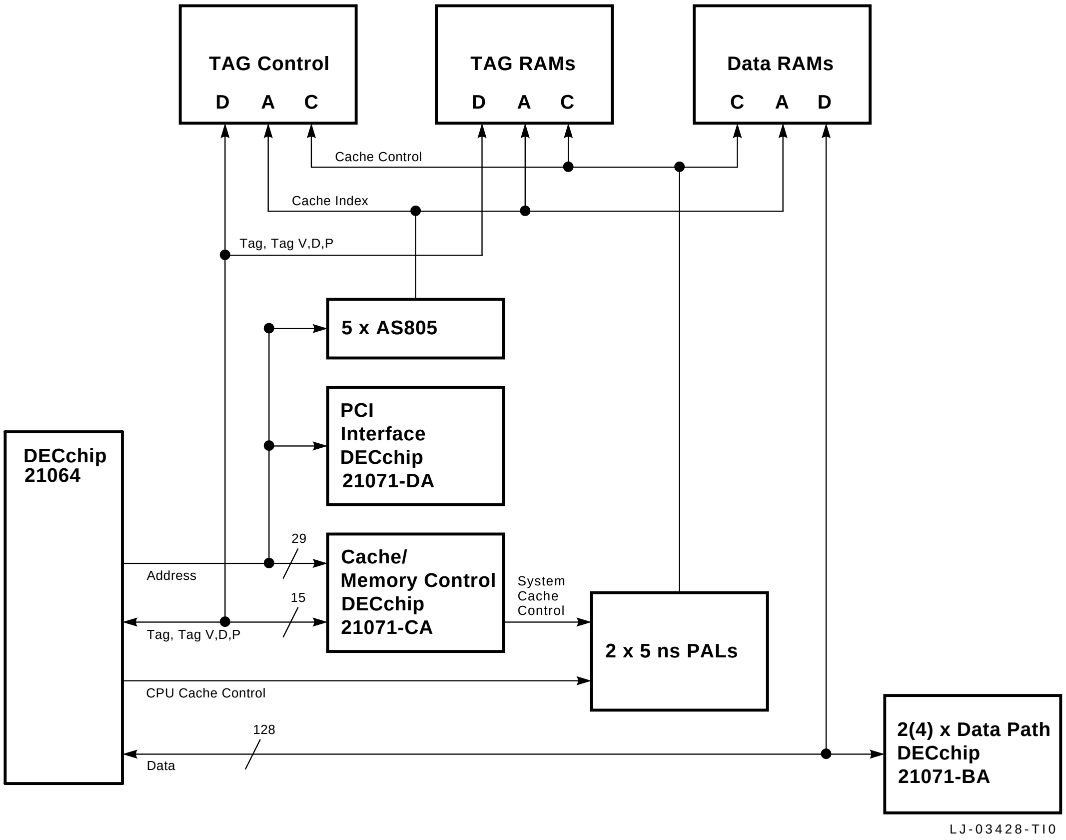
When the cache has been released (during a DMA transaction in progress), arbitration occurs one cycle before ioCAck<1:0> is sent to the 21071-DA chip. It could also occur up to one cycle after ioCack<1:0>, if ioCack<1:0> occurred while the sysBus was being released. The result of arbitration depends on programmed priority if both the CPU and 21071-DA chip are requesting the bus.

3.1.2 Bcache Control

The Bcache controller provides control for the secondary cache on CPU-initiated memory read/write transactions that miss and on all CPU-initiated memory LD*x*_L and ST*x*_C transactions (hits and misses). On DMA initiated transactions, the Bcache controller provides control for probing the cache and extracting or invalidating the cache line when required. The 21071-CA chip supports only a write-back cache.

Figure 3–2 shows the implementation of a cache subsystem with a 512 KB cache.

Figure 3–2 Cache Subsystem for a 512 KB Cache



The following sections describe the salient features of the Bcache controller.

3.1.2.1 Bcache Width, Size, and Speed

The 21071-CA chip supports only a secondary cache width of 128 bits. A 64-bit wide cache is not supported.

The Bcache controller can support Bcache sizes from 128 KB to 16 MB. The controller needs to know the Bcache size to perform a Tag compare on the appropriate bits. The 21071-CA chip uses a register to enable the appropriate bits of the tag address. Software is required to program this register based on the size of the cache. Refer to Section 4.2.3 for additional information.

The only restriction that the Bcache controller places on the speed of the Bcache is that a 21071-CA initiated read from the cache RAMs be completed in one sysClk cycle. Bcache writes can be programmed to take one or two sysClk cycles.

3.1.2.2 Bcache Allocation Policy

The 21071-CA chip supports a write-back Bcache (secondary cache). The Bcache is allocated on CPU memory read misses. The 21071-CA chip supports an optional allocation policy on writes. Allocation on CPU memory writes can be turned off by setting a bit in a register. Refer to Section 4.2.1 for additional information.

3.1.2.3 Bcache Write Granularity

The Bcache controller in the 21071-CA chip supports octaword write granularity to the Bcache. This has implications in the way ST*x*_C hit transactions are handled. ST*x*_C transactions are either quadword or longword in length. Since less than an octaword cannot be written into the cache, the 21071-CA chip has to perform a read-modify-write transaction on the Bcache when an ST*x*_C hits in the cache.

On partial writes or ST*x*_C transactions that miss in the cache, the 21071-CA chip has to merge the write data with data from memory and write it into the cache if allocation is enabled. If allocation is disabled, the write is sent directly to memory through the memory write buffer.

3.1.2.4 CPU-Initiated Bcache Operations

For CPU requests, the 21071-CA chip performs the following operations on the Bcache data and tag RAMs:

- Extracts victim blocks from the Bcache into the write buffer when the Bcache has to be allocated.
- Writes the Bcache with fill data in parallel with returning it to the CPU during a read block to cacheable memory space.
- Writes the Bcache with the updated block of data during a write block to cacheable memory space when write allocate mode is enabled.
- Performs a tag probe and compare for LD*x*_L and ST*x*_C requests.
- Provides data from the Bcache to the CPU on LD*x*_L transactions that hit in the cache.
- Writes the Bcache tag store with the appropriate address and control bits during the previously listed operations.

3.1.2.5 DMA-Initiated Bcache Operations

During DMA requests, after the 21071-CA chip—using the cpuHoldReq/cpuHoldAck mechanism—has received ownership of the Bcache, the chip performs the following operations:

- Performs a tag probe to determine if the DMA block is in the Bcache.
- Reads a block of data from the Bcache and loads it into the DMA read buffer if a DMA read hits in the Bcache.
- Reads a block of data from the Bcache, merges it with DMA write buffer data, and loads it into the memory write buffer if a DMA write mask transaction hits in the Bcache.
- Invalidates the cache block if a DMA write hits in the Bcache.

3.1.2.6 External Logic Requirement

The 21071-CA chip requires external logic (PALs) to generate the controls for the cache RAMs. It supplies cache control signals to external PALs which NOR them with the CPU cache control signals. The Bcache PALs clock the system cache control signals according to the specific timing requirements of that system before NORing with the CPU signals. The 21071-CA chip sends data and tag RAM output enables, write enables, and lower address bit signals to the Bcache PAL logic.

3.1.2.7 Tag Compare Logic

As part of its function to support a system with a backup cache, the 21071-CA chip is responsible for comparing the upper bits of sysAdr<33:5> with address bits stored in the tag RAMs. The 21071-CA chip does this tag comparison during LD*x*_L and ST*x*_C CPU requests and during DMA transactions to cacheable memory space. The number of bits that are used in the address comparison and the parity check is controlled by the tag enable register in the 21071-CA chip. In the case of a system that implements the smallest cache size of 128 KB Bcache, the 21071-CA chip compares sysAdr<31:17> to the tagAdr<31:17> bits read from the tag RAMs. In the other extreme of a 16 MB Bcache, the 21071-CA chip performs the comparison only on bits<31:24>.

3.1.2.8 CPU Primary Cache Invalidates

The 21071-CA chip Bcache controller is responsible for ensuring that the CPU Dcache is always a subset of the external Bcache. Maintaining system cache coherency is accomplished by asserting cpuDInvReq to the CPU at the following times:

- When a valid Bcache block is replaced during a fill of the Bcache with CPU Istream read data.
- When a valid Bcache block is replaced during a fill of the Bcache with write allocate data.
- During a Bcache invalidate that is due to a DMA write or write masked transaction that hits in the cache.
- During all DMA writes when the Bcache is disabled or when no Bcache is present in the system.

The 21071-CA chip assumes that sysAdr<12:5> are logically connected (either directly or indirectly) to the CPU cpuInvAdr<12:5> pins so that the correct Dcache block is invalidated.

3.1.3 sysBus Controller

The sysBus controller consists of a sequencer that receives CPU and DMA command fields for decode, results from the sysBus arbiter logic, and status from the memory controller logic. The sequencer supplies state that is used to generate Bcache control and read requests to the memory controller. The state controls the loading of data from the sysBus into the read/merge buffer and write buffer, and it acknowledges cycles to the CPU and 21071-DA chip.

3.1.3.1 Wrapping

The sysBus controller supports wrapping on the sysBus. On read transactions, the requested octaword is returned to the CPU or the 21071-DA chip.

Note

Wrapping is not optional in the sysBus controller. The processor must be configured with wrapping enabled.

3.1.4 Address Decoding

The 21071-CA sysBus interface logic decodes the sysBus address for both CPU and DMA requests in order to determine what action needs to be taken. It supports cacheable and noncacheable memory accesses, as well as accesses to its CSR space.

Table 3–2 provides an exact mapping of this address space.

Table 3–2 sysBus Address Map

sysAdr<33:32>	sysAdr<31:29>	Address Space	Notes
00	XXX	Cacheable memory space	Accessed by the CPU instruction stream (Istream) or data stream (Dstream). Accessed by DMA.
01	0XX	Noncacheable memory space	Accessed by the CPU (Istream/Dstream). Accessed by DMA; can be used for a frame buffer on the DRAM bus.
01	100	21071-CA CSRs	The 21071-CA chip will respond to all addresses in this space. Dstream access only.
01	101	Reserved for 21071-DA	The 21071-CA expects the 21071-DA to respond to addresses in this range. CPU Dstream access only.
01	11X	Reserved for 21071-DA	The 21071-CA expects the 21071-DA to respond to addresses in this range. CPU Dstream access only.

(continued on next page)

Table 3–2 (Cont.) sysBus Address Map

sysAdr<33:32>	sysAdr<31:29>	Address Space	Notes
10	XXX	Reserved for 21071-DA	The 21071-CA expects the 21071-DA to respond to addresses in this range. CPU Dstream access only.
11	XXX	Reserved for 21071-DA	The 21071-CA expects the 21071-DA to respond to addresses in this range. CPU Dstream access only.

3.1.4.1 Cacheable Memory Space

0 0000 0000 .. 0 FFFF FFFF

The 21071-CA chip recognizes the 4 GB of quadrant 0 (corresponding to sysBus address<33:32> = 00) to be cacheable memory space. The 21071-CA chip responds to all read/write accesses in this space. If the Bcache is enabled, cache probes, allocates, deallocates, and invalidates happen according to the protocols described in Chapter 5. Some or all of main memory can be programmed to be in this cacheable space.

3.1.4.2 Noncacheable Memory Space

1 0000 0000 .. 1 7FFF FFFF

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

3.1.4.3 21071-CA CSR Space

1 8000 0000 .. 1 9FFF FFFF

The 21071-CA must respond to all accesses in this space. Exact CSR addresses are defined in Chapter 4.

3.1.5 Lock Address Register and Lock Bit

The 21071-CA chip implements the lock address register and lock bit as required by the Alpha architecture. The lock register contains sysAdr<32:5> and gets loaded with the sysAdr during all LD*x*_L transactions. The 21071-CA chip locks 32 bytes of data at a time. All LD*x*_L transactions also set the lock bit associated with the address register.

The following conditions clear the lock bit:

- Chip reset
- A DMA write address matches the lock address
- Any ST*x*_C command
- A CPU write to any I/O address (to allow PALcode to reset the lock flag)
- The assertion of the ioClrLock command from the 21071-DA

  This command is used by the 21071-DA to keep the lock flag clear as long as memory is locked by a device on the PCI.

Note

The state of the lock bit is unpredictable after ST*x*_C and LD*x*_L transactions that have tag parity or non-existent memory errors.

3.1.6 Memory Write Buffer

The 21071-CA chip has a memory writer buffer that supports buffering of up to four memory write transactions. This write buffer is used to buffer data on its way to memory for the following types of transactions:

- DMA writes
- Victim data from the Bcache
- CPU noncacheable memory write data (which includes all CPU writes when allocate mode is disabled)

The 21071-CA chip stores the cache line address, longword masks, memory bankset bank numbers, and a cache line valid bit per entry of the memory buffer.

3.1.6.1 Write Buffer Address Comparison

The 21071-CA chip architecture allows memory read requests to bypass writes as long as the read address does not match an address in the memory write buffer. The 21071-CA chip compares the incoming memory read address against the addresses of the valid entries of the memory write buffer.

If there is a match, then the memory controller will continue to dump the contents of the write buffer to memory, one cache line at a time, until the write buffer hit condition no longer exists. The memory controller is then free to start the original memory read transaction, which resulted from the CPU or DMA request.

If there is no match, then the memory read is allowed to proceed ahead of the buffered writes. The memory read transaction may be initiated by a CPU or DMA read from memory, a DMA masked write transaction, or a partial cacheable write transaction from the CPU.

3.1.6.2 Write Buffer Flushing

The 21071-CA chip allows the 21071-DA chip to flush the memory write buffer with a special DMA command.

3.1.6.3 Write Buffer Full Condition

If the memory write buffer is full, then the 21071-CA chip accepts the first data from the sysBus and stores it in a temporary latch until one write transaction has been retired to memory. The second data is stalled on the bus until then. The write buffer full condition can happen on CPU memory writes (noncacheable or nonallocate), DMA writes, and victim reads from the cache.

3.1.7 Read/Merge Buffer Control

The 21071-CA chip controls the read/merge buffer from the 21071-BA chip. The read/merge buffer is a cache line buffer which is used for four main purposes:

- Buffering read data from memory until the sysBus is ready to receive it.
- Supporting Bcache write allocation by providing a mechanism to merge CPU partial writes to the cache with the rest of the cache line from memory.
- Supporting ST*x*_C transactions that hit in the cache.
- Supporting LD*x*_L transactions that hit in the cache.

The read/merge buffer consists of two cache line buffers— the read buffer and the merge buffer. The read buffer is used to store memory read data, and the merge buffer is used to store write data from the CPU or data read from the cache.

During a CPU read block or DMA read transaction, memory data is loaded into the read buffer before being sent out to the sysBus or DMA read buffer. The read buffer acts as a timing stage to phase align the memory timing to the sysBus timing. After the memory controller has loaded an entry of memory data into the read buffer, it sets that entry's valid bit to indicate to the sysBus controller logic that data is ready to be returned to the sysBus. During these memory read transactions, the buffer is also used for storage, because the sysBus could be busy transferring victim data from the cache.

During a cacheable write block transaction with allocate mode enabled, the valid longwords of CPU data are loaded into the merge buffer while the memory controller is fetching the rest of the cache line. Because data could return from memory before all of the CPU data has been loaded, the read and merge buffers can be loaded simultaneously.

During the special case of an ST*x*_C transaction that hits in the Bcache, the merge buffer is used to merge the valid longwords of CPU write data with the rest of the cache line read from the Bcache. After the data has been merged in the buffer, the entire block is then written back to the Bcache.

During an LD*x*_L transaction that hits in the Bcache, the 21071-CA reads the data from the cache into the merge buffer, and then drives the requested data on the sysBus.

3.1.8 sysBus Transactions

This section describes the sysBus transactions.

3.1.8.1 CPU Transactions

This section describes the CPU transactions.

- Read Block From Memory

  A read block from memory can be from cacheable or noncacheable memory space. Data is read from memory and returned to the CPU. On a cacheable read transaction, a victim, if any, is extracted from the cache, and then the cache is filled with the memory data. Only one octaword is transferred on noncacheable reads. The Dcache is invalidated on Istream reads.

- Read Block to I/O Space

  A read block from I/O space may be directed to the 21071-CA CSR or to the 21071-DA chip. On a read block from the 21071-CA CSR, the data is returned by the 21071-CA chip.

  A read block that does not fall within the 21071-CA CSR address range is assumed to belong to the 21071-DA chip.

  The 21071-DA chip is expected to receive the command request, take appropriate action, and notify the 21071-CA chip when data is ready to be returned to the CPU. The 21071-CA chip then provides cpuDRack<2:0> and cpuCAck<2:0> to the CPU, and the transaction is terminated.

  Note

  The 21071-DA chip cannot directly respond to the CPU with cpuDRack<2:0> and cpuCAck<2:0>. It must respond through the 21071-CA chip.

  An I/O read addressed to the 21071-DA chip can be preempted by the 21071-DA chip for deadlock resolution.

- Write Block to Memory

  If allocates are turned on and the transaction is to cacheable space, a cache fill is performed at the end of the write. The cache is filled with data received from the CPU if the whole cache line is being written. In the case of a partial write, CPU write data is merged with memory data before writing in the cache. In either case, a victim (if there is one) is extracted before the fill.

  If allocates are turned off, or if the write is noncacheable, the write data from the CPU is loaded into the write buffer from where it gets written to memory.

- Write Block to I/O Space

  A write block to I/O space may be directed to a 21071-CA CSR or the 21071-DA chip. On a write block to a 21071-CA CSR, data is written to the CSR, and the transaction is completed.

  An I/O write block that does not fall within the 21071-CA CSR address range is assumed to belong to the 21071-DA chip. The 21071-DA chip is expected to notify the 21071-CA chip when the transaction has to be terminated and the 21071-CA chip asserts cpuCAck<2:0> to the CPU. An I/O write addressed to the 21071-DA chip can be preempted by it for deadlock resolution.

- LD*x*_L

  The Bcache controller performs a cache probe. If the address is a miss, then the behavior is exactly the same as that of a memory read block, except that the cache line address is stored in the lock register and the lock flag is set. The same is true of a noncacheable address.

  If the address is a hit, data is read from the cache into the merge buffer and is then returned to the CPU. As in the miss case, the lock address is captured, and the lock flag is set.

  An LD*x*_L to I/O space is handled as a read block to I/O space.

- ST*x*_C

  The 21071-CA chip responds only to ST*x*_C transactions that are addressed to memory space or to its CSR space. On an ST*x*_C transaction in memory space, the state of the lock flag is checked. If the lock flag is clear, the ST*x*_C fails, and the transaction is terminated with an ST*x*_C fail CACK. If the lock flag is set, the transaction proceeds as outlined below.

  A cache probe is done to detect a hit or a miss. If it hits in the cache, the write data is loaded into the merge buffer, and a read of the cache is performed. The read data is merged with the write data and is then written to the cache. This is necessary because an ST*x*_C transaction is always less than an octaword, and the write granularity of the Bcache is an octaword.

  If the cache probe failed, the remainder of the flow looks like a write block. As in the write block flow, the write data enters the merge buffer if Bcache write allocate is enabled, otherwise it is stored in the memory write buffer.

  An ST*x*_C to the 21071-CA chip CSR space is handled as a write block. Error checking takes precedence over checking the lock flag.

- Barrier

  A barrier transaction has no effect on the 21071-CA chip. However, instead of terminating the transaction right away, the 21071-CA chip allows the 21071-DA chip to respond to a barrier. Therefore, the 21071-DA chip has to notify the 21071-CA chip when it wants the barrier terminated.

Note

The 21071-CA chip requires the 21071-DA chip to respond to a barrier instruction using ioCAck<1:0>. Failure to comply with this condition will cause the transaction to hang.

- Fetch, FetchM

  A fetch, fetchM transaction has no effect on the 21071-CA chip. If a fetch or a fetchM is within memory or the 21071-CA CSR space, the transaction is simply acknowledged as OK. The 21071-DA chip must decode and request acknowledgment of fetch and fetchM transactions if they are within the 21071-DA chip address space.

3.1.8.2 DMA Transactions

After DMA wins arbitration, it may request a transaction with the 21071-CA chip. Unlike the CPU transactions, the only unit of transfer for DMA transactions is a cache line.

- DMA Read

  A DMA read command is sent by the 21071-DA chip to indicate that it wants the lower octaword of the cache line first, followed by the upper octaword. The whole cache line is always returned. A DMA read transaction to cacheable space causes the Bcache controller to do a cache probe. If the address hits in the cache, data is read from the cache and returned to the 21071-DA chip. If the address is noncacheable or if the address misses in the cache, the data is read from memory.

- DMA Read Wrapped

  The only difference between a DMA read and DMA read wrapped transaction is that the requested data in the DMA read wrapped transaction is the upper octaword in the cache line, which should be returned first.

- DMA Read Burst

  The DMA read burst command is similar to the DMA read command. It is used by the 21071-DA chip to give a page mode hint to the 21071-CA chip, and it may cause the memory controller to remain in page mode at the end of this read transaction.

- DMA Read Wrapped Burst

  The DMA read wrapped burst command is similar to the DMA read wrapped command. It is used by the 21071-DA chip to give a page mode hint to the 21071-CA chip, and it will cause the memory controller to remain in page mode at the end of this read transaction.

- DMA Write Full

  This command indicates that the whole cache line has to be written to memory. If the address is in cacheable space, the cache is probed. If there is a cache hit, the corresponding location is invalidated in the Bcache and Dcache. The write data is loaded into the write buffer from where it is written to memory. Except for the cache invalidate, the operation is the same on noncacheable writes or cache miss writes.

  If the Bcache is disabled (bc_En clear) or not present on the system, every DMA write will cause a CPU data cache (Dcache) invalidate.

- DMA Write Masked

  The 21071-DA chip requests a DMA write masked transaction when only a subset of the bytes in a cache line are to be written. The 21071-CA chip begins the transaction by performing a DMA read. As the read data is received from the Bcache or memory, it is merged with DMA write data and loaded into the memory write buffer. If the cache was hit, the cache is invalidated.

  If the Bcache is disabled (bc_En clear) or not present on the system, every DMA write will cause a CPU Dcache invalidate.

- DMA Flush

  This command should be used by the 21071-DA chip when it wants to flush the memory write buffer. The 21071-CA chip will acknowledge the transaction after all buffered writes have been written to memory.

3.1.9 Error Handling

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

- Bcache tag address parity error
- Bcache tag control parity error
- Non-existent memory error

When one or more errors are detected on a transaction, the 21071-CA chip signals the errors to the CPU or the 21071-DA chip at the end of the transaction by acknowledging hard error on the cpuCAck<2:0> or ioCAck<1:0> field. The current sysAdr<33:5> is logged in the error address register and error status is logged in the error and diagnostics status register. These CSRs are locked until the CPU clears all the error status bits by writing the CSR. Refer to Chapter 4 for additional information.

If errors occur on a transaction while the error address and status are locked, the transaction is acknowledged with hard error on the cpuCAck<2:0> or ioCAck<1:0> command fields. The LostErr bit in the error and diagnostics status Register is set, and neither the error address nor the error status of the lost error are recorded.

The hard error indication overrides ST*x*_C fail. The lock bit is unpredictable after LD*x*_L transactions that have errors.

3.2 Memory Controller

This section describes memory organization and memory controller features.

3.2.1 DRAM and SIMM Requirements

The I/O pins for all the SIMMs or RAMs must be TTL compatible. DRAM output drivers are controlled by using only the memCAS_l and memWE_l pins. The VRAM drivers use memDTOE_l and memDSF pins in addition to the memCAS_l and memWE_l pins. The OE_l pins on the DRAMs should be grounded. A separate CAS per longword must be used at the RAMs. CAS-before-RAS refresh must be supported. The expected RAS-access time is 50 ns to 100 ns, with page mode CAS-access time between 10 ns and 50 ns.

3.2.2 Memory Organization

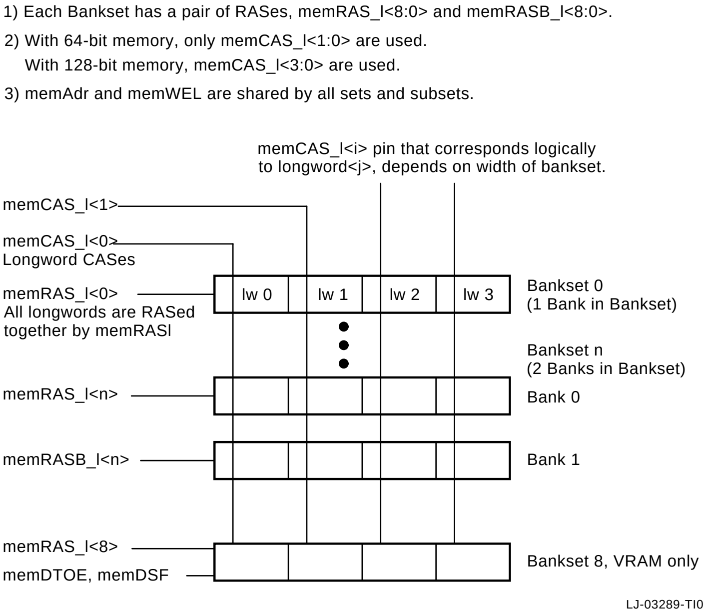
The 21071-CA chip supports between 8 MB and 4 GB of dynamic random-access memory (DRAM) and an additional 1 MB to 8 MB of dual port random-access memory (VRAM).

Memory can be accessed in two widths—64 bits and 128 bits. The actual number of bits required is higher depending on the mode of error detection. Longword parity requires 66 or 132 bits, and longword ECC requires 78 or 156 bits corresponding to 64-bit and 128-bit wide memory respectively.

The 21071-CA chip supports up to 8 banksets of DRAM and 1 bankset of VRAM. Each bankset can be made up of one or two banks. A bank of memory refers to one width of DRAMs. It may be implemented using SIMMs or by directly soldering DRAMs on the module. A SIMM implementation requires more than one SIMM to form one memory bank. For instance, four 33-, 36-, or 40-bit SIMMs would be required to form a bank width of 128. The two banks in a bankset should be identical in configuration, size, and speed. The 21071-CA chip has a pair of RAS signals that corresponding to a bankset—memRAS_l and memRASB_l. Each bank in a bankset should be connected to one of these RAS pins. If the bankset has only one bank of RAMs, memRAS_l should be used, and memRASB_l should be left unconnected.

Figure 3–3 shows the memory set organization.

Figure 3–3 Memory Set Organization



3.2.2.1 Memory Bankset Characteristics

Each memory bankset must conform to the following characteristics:

- Width: All the banksets in a system must have the same memory width.
- Banks: The banks in a bankset should be identical in DRAM size and speed.
- Longword writes: Each bankset must support longword write capability. The 21071-CA chip generates longword CASes for writes. For banksets implemented using 33-, 36- or 40-bit SIMMs, each SIMM should receive a unique memCAS_l pin. Table 3–3 shows the CAS connections.
- Address Range: Each bankset has a programmable base address and size. The base address of a bankset must be aligned to the natural size boundary. For example, an 8 MB bankset must start on an 8 MB boundary.

Table 3–3 Longword Number to memCAS_l[n] Correspondence

Memory	memCAS_l			
Width	<0>	<1>	<2>	<3>
64	LW0	LW1	Unused	Unused
	LW2	LW3	Unused	Unused
	LW4	LW5	Unused	Unused
	LW6	LW7	Unused	Unused
128	LW0	LW1	LW2	LW3
	LW4	LW5	LW6	LW7

A detailed description of the banksets is given in the following sections.

3.2.2.2 Bankset0..Bankset7

Bankset0 through bankset7 are intended for DRAMs; they have the same features.

- DRAM Type: 1M x 1, 1M x 4, 4M x 1, 4M x 4 and 16M x 1 DRAMs are supported. Both symmetrical (11,11) and asymmetrical (12,10) addressing for 16 MB DRAMs are supported. Typical expected RAS access time is 50 to 100 ns. CAS-before-RAS refresh is used to refresh all banksets simultaneously.
- Bankset Size (MB): A bankset may be made up of 1 or 2 banks, giving a total of 1M, 2M, 4M, 8M, 16M or 32M addressable locations depending on the depth of the DRAMs used. Each location consists of 8 bytes for 64-bit memory or 16 bytes for 128-bit memory. Table 3–4 lists supported bankset sizes and the possible DRAM configurations that can be used to get these sizes.

Table 3–4 Supported Bankset Sizes and DRAM Configurations for Different Memory Widths

Locations	Bankset Size		Number of	DRAM
in Bankset	64-Bit	128-Bit	Subbanks	Configurations
1M	8 MB	16 MB	1	1M x 1 / 1M x 4
2M	16 MB	32 MB	2	1M x 1 / 1M x 4
4M	32 MB	64 MB	1	4M x 1 / 4M x 4
8M	64 MB	128 MB	2	4M x 1 / 4M x 4
16M	128 MB	256 MB	1	16M x 1
32M	256 MB	512 MB	2	16M x 1

3.2.2.3 Bankset8

A single, fixed bankset location for VRAMs simplifies the support logic and reduces CSR bits. As bankset8 provides from 1 MB to 8 MB of VRAM, more than one VRAM bankset is not required.

- VRAM Type: 128K x 4, 128K x 8, 256K x 4, and 256K x 8 VRAMs are supported. The number of rows in the VRAM must be 512. This is required for the video display pointer logic to increment correctly. Typical expected RAS-access-time to the RAM port of the VRAM is 50 ns to 100 ns. CAS-before-RAS refresh is used.
- Bankset8 Size: Bankset8 can have 1 or 2 banks giving a total of 128K, 256K or 512K addressable locations. This provides 1 MB, 2 MB, or 4 MB of VRAM for 64-bit memory; and 2 MB, 4 MB, or 8 MB for 128-bit memory.

3.2.2.4 Supported Memory SIMMs

The 21071-CA chip supports industry-standard 33-, 36-, and 40-bit SIMMs. 33- and 36-bit SIMMs are used when longword parity is the error detection mode, and 40-bit SIMMs are used when longword ECC is used. Table 3–4 lists the DRAM sizes and widths that are supported. Split RAS SIMMs are supported by the 21071-CA chip. Split RAS SIMMs have two banks of RAMs, one on each side. A split RAS SIMM can therefore be considered as a bankset with two banks, and the corresponding memRAS_l and memRASB_l can be used to select between either side of the SIMM.

3.2.3 Memory Address Generation

Note

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

This section describes the generation of row and column addresses from the address originating on the sysBus, that is, the physical address PA<33:5>. The 21071-CA chip sysBus interface decodes accesses to memory space and the 21071-CA chip I/O space. The physical addresses received by the 21071-CA chip memory control logic are always in memory space.

For memory reads, the address comes directly from sysAdr<33:5>. For memory writes, the write buffer provides the initial value of PA<33:4>. For video serial register loads, the address is derived internally.

Each bankset has a programmable base address and size. The incoming physical address is compared in parallel with the memory ranges of all banksets present. Depending on the size of the bankset, a variable number of PA and base address bits from the CSR are compared. Table 3–5 describes the base address bits and the subbank bit for the allowed bankset sizes.

Table 3–5 Base Address Comparison

Bankset Size	Compared	Subbank
512 MB	PA<33:29>	PA<28>
256 MB	PA<33:28>	PA<27>
128 MB	PA<33:27>	PA<26>
64 MB	PA<33:26>	PA<25>
32 MB	PA<33:25>	PA<24>
16 MB	PA<33:24>	PA<23>
8 MB	PA<33:23>	PA<22>
4 MB	PA<33:22>	PA<21>
2 MB	PA<33:21>	PA<20>
1 MB	PA<33:20>	PA<19>

Note

Bankset0 through bankset7 have a minimum size of 8 MB. VRAM bankset8 has a maximum size of 8 MB.

The memory address depends on the width of memory and the number of row and column bits per bankset. Program Sn_ColSel according to Table 3–6 and Table 3–7.

Table 3–6 Row and Column Address Decode for Bankset0..7

Sn_ColSel	Memory Width	Row,Column Bits	Row Address	Column Address <11:0>
000	64	12,12	PA<24:13>	PA<26,25,12:3>
001	64	12,10 or 11,11	PA<24:13>	PA<x,24,12:3>
011	64	10,10	PA<xx,22:13>	PA<xx,12:3>
000	128	12,12	PA<25,24,22:13>	PA<27,26,23,12:4>
001	128	12,10 or 11,11	PA<25,24,22:13>	PA<x,25,23,12:4>
011	128	10,10	PA<xx,22:13>	PA<xx,23,12:4>

Table 3–7 Row and Column Address Decode for Bankset8

S8_ColSel	Memory Width	Row,Id Bits	Row Address<11:0>	Column Address<11:0>
100	64	9,9	xxx,<20:12>	xxx,<11:3>
101	64	9,8	xxx,<19:11>	xxxx,<10:3>
100	128	9,9	xxx,<21:13>	xxx,<12:4>
101	128	9,8	xxx,<20:12>	xxxx,<11:4>

Note

BankSet0 through bankset7 cannot have less than 10 column bits as the smallest DRAM size supported is 1M x 1. Bankset8 cannot have more than 9 column bits as the largest VRAM supported is 256K x 8.

3.2.4 Performance Optimizations

The following sections describe performance optimizations.

3.2.4.1 Memory Page Mode Support

The 21071-CA chip supports page mode within CPU read transactions. Page mode between transactions is supported on DMA read burst transactions and on memory write transactions.

The following page mode features are supported by the 21071-CA chip:

- A refresh transaction never starts in page mode. If any memRAS_l is asserted when the refresh transaction is selected, the controller waits for the duration of the RAS precharge before doing the refresh.
- A video transaction never starts in page mode. If memRAS_l<8> or memRASB_l<8> are asserted when the video transaction is selected, the controller waits for the duration of the RAS precharge before doing the transaction.
- A memory read transaction will start in page mode if the preceding transaction was a memory read initiated by a DMA read burst command on the sysBus, and the row address, bankset, and subbank of the current transaction are the same as that of the previous transaction. Furthermore, a memory read initiated by a CPU transaction (read or partial write) will never start in page mode. This is because the sysBus controller notifies the memory controller to deassert RAS if the sysBus has been given to the CPU after a DMA read burst.

- A memory write transaction starts in page mode, only if the previous transaction was a write, and the row address, bankset, and subbank of the current write are the same as that of the previous transaction.

In all of the previous cases, the transaction will not start in page mode if the maximum RAS width counter has overflowed. The RAS has to be precharged even if there is a page hit.

A transaction that does not start in page mode may or may not have the extra latency of RAS precharge. If the current transaction is to a different bankset than the previous one, the RAS for the previous transaction is deasserted, and at the same time, the RAS for the current one is asserted.

3.2.4.2 Read Latency Minimization

In order to minimize the read latency seen by devices on the sysBus, the memory controller performs certain optimizations in the way transactions are selected. In general, because writes can go into a deep write buffer, reads are given priority over writes, to the extent that in some cases the memory controller waits for a read to happen even if there are writes queued up in the write buffer. These situations are described here:

Following a memory read initiated by a CPU or DMA transaction on the sysBus (CPU read or a partial write), the 21071-CA chip does not service a write from the write buffer for 12 memClk cycles after the last read data has latched, unless the write buffer is full. The reason for doing this is that there is a delay between the completion of the read by the memory controller and the initiation of another read on the sysBus. Servicing a write from the write buffer would add latency to the following read. This will definitely happen on reads that have Bcache victims, because every read will be accompanied by a write. The write will add latency to the next read, and the effect of the victim buffer will be minimal. This condition is called Wait After Read (WAR). Waiting after a DMA read also helps in the case of a scatter/gather read performed in guaranteed-access-time (GAT) mode. See Section 9.4.3 for more details about GAT mode.

The writes are held off only if the write buffer is not full.

3.2.5 Transaction Scheduler

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

Table 3–8 describes the priority scheme.

Table 3–8 Memory Transaction Scheduling

Refresh Request	Read Request	WBuf Hit[2]	Write Request	WBuf Full[3]	Video Request	RB[4]	WAR[5]	Select
1	X[1]	X	X	X	X	X	X	Refresh
0	X	X	X	X	1	X	X	Video
0	1	0	X	X	0	X	X	Read
0	1	1	X	X	0	X	X	Write
0	0	X	X	1	0	0	X	Write
0	0	X	1	0	0	X	0	Write
0	0	X	X	X	0	1	X	None
0	0	X	X	0	0	X	1	None

[1] X : Don't care

[2] WBuf Hit: Read address matches buffered write.

[3] WBuf Full: Write buffer full.

[4] RB: Read burst. Hint to stay in read page mode.

[5] WAR: Wait after read. Internal stall signal.

3.2.6 Programmable Memory Timing

The memory control state machine sequences through all the memory transactions. On memory read and write transactions, it has to communicate with the 21071-BA chips so that data may be latched from the memData bus or driven onto the memData bus respectively. All memory signals are generated on memClkR. However commands from the 21071-CA chip to the 21071-BA chip are sent on sysClocks (clk2R). Because the sysClock cycle time is twice that of the memClk, the 21071-BA chips have to be informed which memClk the data has to be latched on. This is done by sending immediate and delayed commands. Immediate commands require that data is latched (or driven) on the next memClk rising edge, and the delayed commands require that data be latched (or driven) on the second memClkR.

The memory control state machine is actually made up of two separate state machines — one is the master, which does all the RAS and CAS assertion, and controls when the other state machines start; the second is the read/write state machine, which does all the sequencing for generating the memCmds to read or write memory data. The read/write state machine is started by the master, and then it sequences independently. Each state machine uses some of the programmed timing parameters to generate the corresponding memory control signals.

Note

While programming the memory timing, ensure that the parameters used for the address, RAS, and CAS are compatible with the ones used for data; otherwise, operation on the memory interface will be incorrect.

Because memCmds have to be sent to the 21071-BA chips on clk2R, the memory controller synchronizes the start of all transactions to clk2R. This way, the memory control signals track the memory data according to the programmed values. This synchronization may add an extra delay of one memClk on memory transactions. When the memory controller is idle, sysBus reads or writes do not have the extra delay, because the corresponding requests are generated synchronous to sysClock.

3.2.7 Presence Detect Logic

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

As soon as the internal synchronized version of reset deasserts, the loading process begins. First, the data is loaded into the shift register by asserting memPDLoad_l and pulsing memPDClk. Then a bit is loaded by toggling memPDLoad_l. Either edge of memPDClk may be used to shift memPDDIn, as memPDDIn is sampled when memPDClk is stable. Once all 32 bits have been loaded, memPDClk stops and the presence detect registers may be read. See Figure 3–4, which shows the operation of the presence detect logic.

Figure 3–4 Presence Detect Logic Operation

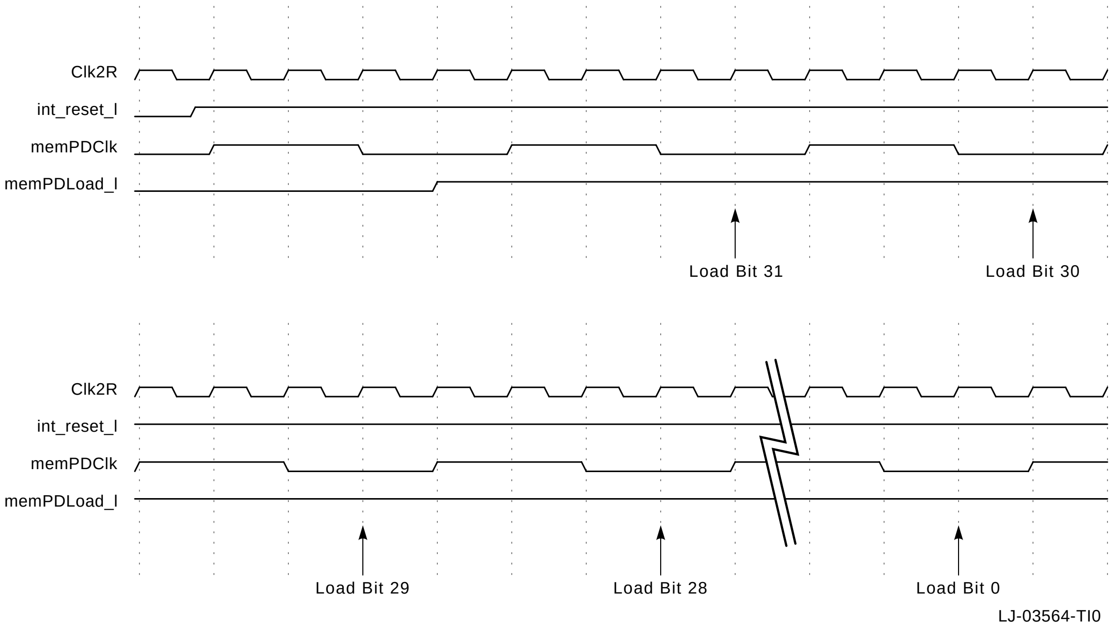


Table 3–9 shows the presence detect shift registers that are supported.

Table 3–9 Supported Presence Detect Shift Registers

Part	Bits[1]	/Load [2]	clk[3]	Din[4]	Dout[5]	Vcc[6]	Gnd[7]
74F166	8	/PE	CP	DS	Q7	—	/CE
74F194	4	*S1	CP	DSR	Q3	/MR,S0	—
74F195	4	/PE	CP	J,/K	Q3	/MR	—
74F199	8	/PE	CP	J,/K	Q7	/MR	/CE
74F299	8	*S1	CP	DSR	Q0	/MR,/OE	—
74F322	8	S/P	CP	D0	Q7	/MR,/SE	/RE,S
74F323	8	*S1	CP	DSR	Q0	/SR	/OE
74F395	4	*PE	/CP	DS	QS	/MR	/OE
74F674	16	R/W	/CP	NotSup	Q15	M	/CS
74F676	16	*M	/CP	SI	SO	—	/CS

[1] Number of presence detect pins supported.

[2] Pins to tie to memPDLoad_l. Asterisk (*) indicates that signals must be inverted on module.

[3] Pins to tie to memPDClk.

[4] Pins to daisy chain data into.

[5] Pins to daisy chain to next shift register or to memPDDIn.

[6] Pins to be tied high.

[7] Pins to be tied low.

3.2.8 Video Support Logic

The 21071-CA chip provides the logic and control to perform full and split serial register loads to the VRAM bankset8. The 21071-CA chip does regular CPU/DMA accesses to the random port of bankset8 if the address matches the bank's base address, just like for any other bankset. In addition, the 21071-CA chip does serial register loads in response to vframe_l or vRefresh_l pin assertions. When the 21071-CA chip does a serial register load, the VRAM latches the data in the accessed row into its serial register. Other external logic then shifts out the serial register through the VRAM's serial port. The 21071-CA chip does not provide any support for unloading the serial port of the VRAM. Figure 3–5 shows an implementation of a video subsystem using a dumb frame buffer in bankset8.

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

In terms of timing, a serial register load is identical to a memory read to bankset8, with the exception of memDTOE and memDSF. The data on memData<31:0> is ignored during serial register loads.

The 21071-CA chip provides the logic and control to perform full and split serial register loads to the VRAM bankset8. The Video Frame Pointer (VFP) CSR provides the start address of the video frame buffer in memory. An internal set of latches, called the Video Display Pointer (VDP), contains the subset, row, and column addresses for video shift register loads.

Following a vFrame_l assertion, the Video Frame Pointer is latched into the VDP. A full serial register load is performed at the subbank and row address indicated in the VDP, with an all-zero column address. At the end of the load, the row address in the VDP is incremented (mod 512) to point to the next row. In case of overflow, the subbank bit in the VDP is toggled if subbanks are enabled for bankset 8. The column MSB in the VDP is toggled.

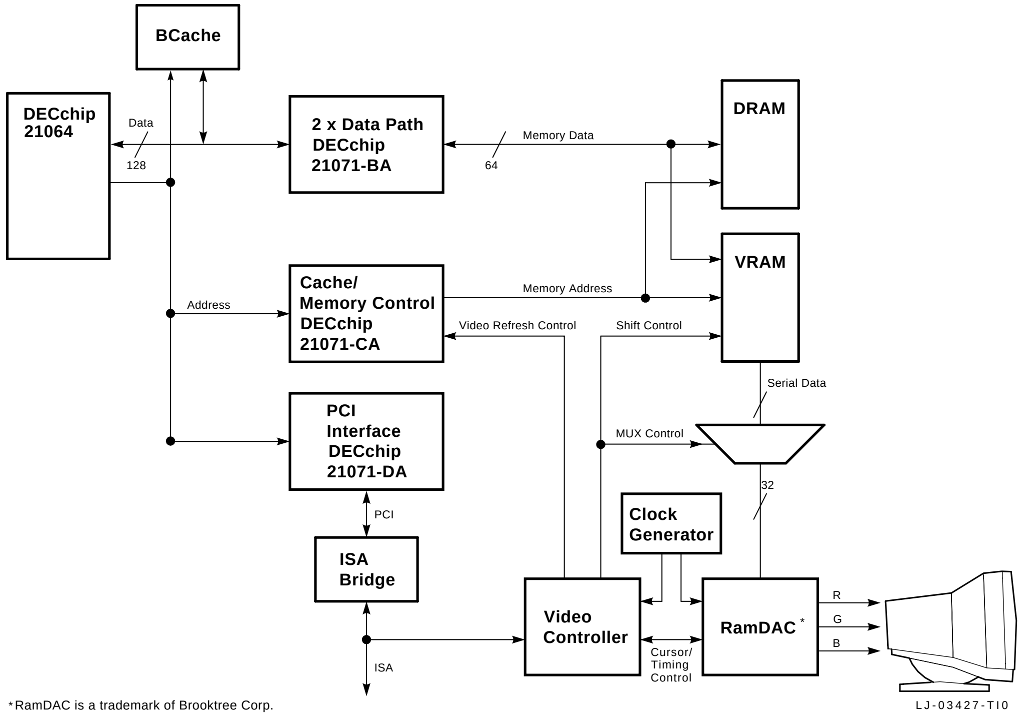
Following a vRefresh_l assertion, a split serial register load is performed at the subbank and row address indicated in the VDP. The column MSB in the VDP is toggled. If the new column MSB equals 0, the row address in the VDP is incremented. If the row address overflows (mod 512), the subbank bit in the VDP is toggled if the subbank is enabled.

The memory controller can take up to 135 sysClk cycles to complete a serial register load after the assertion of vFrame_l or vRefresh_l. If a request is reasserted before the previous request has been completed, the second request may either override the first request or it may be ignored.

Simultaneous assertion of vFrame_l and vRefresh_l can cause one of the requests to be serviced while the other is lost.

Figure 3–5 shows a video subsystem using a DECchip 21071 chipset and a dumb frame buffer.

Figure 3–5 Video Subsystem Using a DECchip 21071 Chipset and a Dumb Frame Buffer



*RamDAC is a trademark of Brooktree Corp.

4

DECchip 21071-CA Programmer's Reference

This chapter describes the 21071-CA control and status registers (CSRs). It also provides information about how to program memory timing, configure memory, and initialize the Bcache.

4.1 Register Descriptions

This section describes the 21071-CA control and status registers (CSRs). These CSRs are 16 bits wide and addressed on cache-line boundaries only. Writes to read-only registers could result in unpredictable behavior. Reads are nondestructive. Only zeros should be written to unspecified bits within a CSR. Only bits <15:0> of each CSR are defined. Other bits are undefined. CSRs are initialized as specified in the register descriptions.

Table 4–1 shows the base address and name of all the control and status registers.

Table 4–1 DECchip 21071-CA Register Summary

Address	Name
1 8000 0000	General control register
1 8000 0020	Reserved
1 8000 0040	Error and diagnostic status register
1 8000 0060	Tag enable register
1 8000 0080	Error low address register
1 8000 00A0	Error high address register
1 8000 00C0	LD*x*_L low address register
1 8000 00E0	LD*x*_L high address register

(continued on next page)

Table 4–1 (Cont.) DECchip 21071-CA Register Summary

Address	Name
1 8000 0200	Global timing register
1 8000 0220	Refresh timing register
1 8000 0240	Video frame pointer register
1 8000 0260	Presence detect low data register
1 8000 0280	Presence detect high data register
1 8000 0800	Bank 0 base address register
1 8000 0820	Bank 1 base address register
1 8000 0840	Bank 2 base address register
1 8000 0860	Bank 3 base address register
1 8000 0880	Bank 4 base address register
1 8000 08A0	Bank 5 base address register
1 8000 08C0	Bank 6 base address register
1 8000 08E0	Bank 7 base address register
1 8000 0900	Bank 8 base address register
1 8000 0A00	Bank 0 configuration register
1 8000 0A20	Bank 1 configuration register
1 8000 0A40	Bank 2 configuration register
1 8000 0A60	Bank 3 configuration register
1 8000 0A80	Bank 4 configuration register
1 8000 0AA0	Bank 5 configuration register
1 8000 0AC0	Bank 6 configuration register
1 8000 0AE0	Bank 7 configuration register
1 8000 0B00	Bank 8 configuration register
1 8000 0C00	Bank 0 timing register A
1 8000 0C20	Bank 1 timing register A
1 8000 0C40	Bank 2 timing register A
1 8000 0C60	Bank 3 timing register A
1 8000 0C80	Bank 4 timing register A
1 8000 0CA0	Bank 5 timing register A
1 8000 0CC0	Bank 6 timing register A
1 8000 0CE0	Bank 7 timing register A
1 8000 0D00	Bank 8 timing register A
1 8000 0E00	Bank 0 timing register B
1 8000 0E20	Bank 1 timing register B

(continued on next page)

Table 4–1 (Cont.) DECchip 21071-CA Register Summary

Address	Name
1 8000 0E40	Bank 2 timing register B
1 8000 0E60	Bank 3 timing register B
1 8000 0E80	Bank 4 timing register B
1 8000 0EA0	Bank 5 timing register B
1 8000 0EC0	Bank 6 timing register B
1 8000 0EE0	Bank 7 timing register B
1 8000 0F00	Bank 8 timing register B

4.2 General Registers

This section describes the 21071-CA general registers. These registers control the sysBus state machine and associated logic.

4.2.1 General Control Register

The general control register contains status information that affects the major operational modes of the entire 21071-CA chip. Figure 4–1 shows the register bit assignments, and Table 4–2 provides the bit descriptions for the general control register.

Figure 4–1 General Control Register

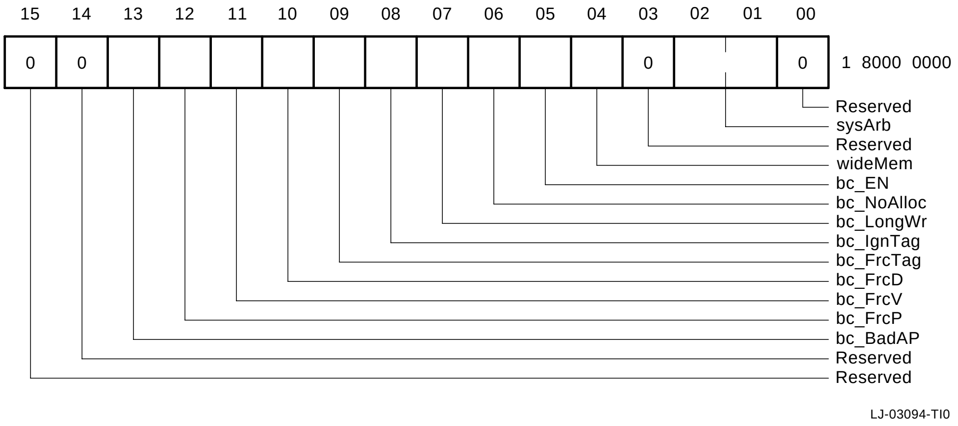


Table 4–2 General Control Register

<table>
<tr><th>Field</th><th>Bits</th><th>Type,
Reset</th><th>Description</th></tr>
<tr><td>Reserved</td><td><0></td><td>MBZ</td><td>—</td></tr>
<tr><td>sysArb</td><td><2:1></td><td>RW,0</td><td>DMA arbitration mode. Determines arbitration scheme for sysBus transactions.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0X</td><td>CPU priority</td></tr>
<tr><td>10</td><td>DMA priority</td></tr>
<tr><td>11</td><td>DMA strong priority</td></tr>
</table>
See Section 3.1.1 for a detailed description of these fields.</td></tr>
<tr><td>Reserved</td><td><3></td><td>MBZ</td><td>—</td></tr>
<tr><td>wideMem</td><td><4></td><td>RO,–</td><td>Memory size. Reads the status of the wideMem input pin. Returns 1 if the memory is 128 bits wide, or 0 if 64 bits wide.</td></tr>
<tr><td>bc_En</td><td><5></td><td>RW,0</td><td>Bcache enable. When clear, the Bcache is disabled and the cache state machine will not probe the cache.</td></tr>
<tr><td>bc_NoAlloc</td><td><6></td><td>RW,0</td><td>Bcache no allocate mode. When set, CPU writes to cacheable memory space will not be allocated into the cache.</td></tr>
<tr><td>bc_LongWr</td><td><7></td><td>RW,0</td><td>Bcache long writes. When set, two sysBus cycles are required to write to the cache data RAMs. See Section 5.1.4.</td></tr>
<tr><td>bc_IgnTag</td><td><8></td><td>RW,0</td><td>Bcache ignore tag. When set, Bcache probes will act as if the valid bit was invalid. All tag results will be ignored, and any victims will be lost. Tag and address parity will be ignored. May be used to fill the cache with valid data.</td></tr>
</table>

(continued on next page)

Table 4–2 (Cont.) General Control Register

Field	Bits	Type, Reset	Description
bc_FrcTag	<9>	RW,0	Bcache force tag. When set, the Bcache will be probed for victims, and the line will be invalidated using the values in the bc_FrcD, bc_FrcV, and bc_FrcP. CSRs will be used as the tag controls. Although the line is invalidated (assuming bc_FrcV is reset), the data is loaded into the cache and will be returned to the CPU as cacheable. Used for diagnostic testing of the cache RAM, and for flushing the cache by setting this bit, clearing bc_FrcV, and cycling through the address range present in the cache.
bc_FrcD	<10>	RW,0	Bcache force dirty. When set, the dirty bit will be set on the next cache fill.
bc_FrcV	<11>	RW,0	Bcache force valid. When set, the valid bit will be set on the next cache fill.
bc_FrcP	<12>	RW,0	Bcache force parity. When set, the parity bit will be set on the next cache fill.
bc_BadAP	<13>	RW,0	Bcache force bad address Parity. When set, the tag address parity will be loaded as bad. This bit is independent of the bc_FrcTag bit.
Reserved	<15:14>	MBZ	—

4.2.2 Error and Diagnostic Status Register

The error and diagnostic status register contains status information for diagnostics and for error analysis. The occurrence of an error sets one or more error bits (bc_TAPErr, bc_TCPErr, nxMErr) and locks the address of the error. After the address is locked, any additional error will set lostErr and will not affect the address or other error bits (bc_TAPErr, bc_TCPErr, nxMErr). Clearing all of the error bits (not the lostErr bit) unlocks the address.

Figure 4–2 shows the register bit assignments, and Table 4–3 provides the bit descriptions for the error and diagnostic status register.

Figure 4–2 Error and Diagnostic Status Register

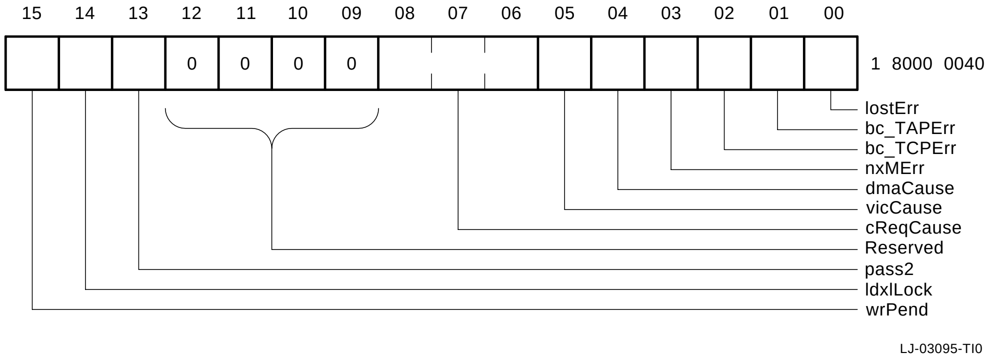


Table 4–3 Error and Diagnostic Status Register

Field	Bits	Type, Reset	Description
lostErr	<0>	RW1C,0	Multiple errors. When set, indicates that additional errors occurred when an error address was already locked. No address or cause information is latched for the error. Cleared by writing a 1 to lostErr.
bc_TAPErr	<1>	RW1C,0	Bcache tag address parity error. When set, indicates that a tag probe encountered bad parity in the tag address RAM. Set only when address is unlocked.
bc_TCPErr	<2>	RW1C,0	Bcache tag control parity error. When set, indicates that a tag probe encountered bad parity in the tag control RAM. Set only when address is unlocked.
nxMErr	<3>	RW1C,0	Nonexistent memory error. When set, indicates that a read or write occurred to an invalid address that does not map to any memory bank, CSR, or I/O quadrant. Set only when address is unlocked.

(continued on next page)

Table 4–3 (Cont.) Error and Diagnostic Status Register

Field	Bits	Type, Reset	Description
dmaCause	<4>	RO,–	DMA transaction caused error. When set, indicates that the bc_TAPErr, bc_TCPErr, or nxMErr was caused by a DMA transaction. Locked with the error address. Only valid when an error is indicated on bc_TAPErr, bc_TCPErr, or memErr.
vicCause	<5>	RO,–	Victim write caused error. When set, indicates that an NXM error was caused by a victim write transaction. Undefined for other types of errors. Locked with the error address. Only valid when an error is indicated on bc_TAPErr, bc_TCPErr, or memErr.
cReqCause	<8:6>	RO,–	Cycle request that caused error. Indicates the DMA or CPU cycle request type that caused the error. Copy of either the cpuCReq or ioCmd lines depending on the DmaCause CSR. Locked with the error address. Only valid when a error is indicated on bc_TAPErr, bc_TCPErr, or memErr.
Reserved	<12:9>	MBZ	—
pass2	<13>	RO,1	Chip version reads low on pass1 and high on pass2.
ldxlLock	<14>	RO,–	LD*x*_L locked. When set, indicates that the lock bit for LD*x*_L is set, and that the next ST*x*_C may succeed. Writing to any CSR or I/O space location clears this lock bit.
wrPend	<15>	RO,0	Write pending. When set, indicates that valid write data is stored in the write buffer.

4.2.3 Tag Enable Register

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

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

To construct the tagEn bits, see Tables 4–4 and 4–5. The value shown in Table 4–4 (based on the cache size) is ANDed with the value in Table 4–5 (based on the maximum cacheable system memory).

The following example shows how to program a system with a 16 MB cache and a maximum of 1 GB of cacheable memory:

```
1111 1111 0000 000X   (ANDed with (16 MB, Table 4-4))
0011 1111 1111 111X   (gives   (1 GB, Table 4-5))

0011 1111 0000 000X   (value is put into tag enable register)
```

Figure 4–3 shows the register bit assignments for the tag enable register, Table 4–4 provides the cache size tag enable values, and Table 4–5 provides the maximum memory tag enable values.

Figure 4–3 Tag Enable Register

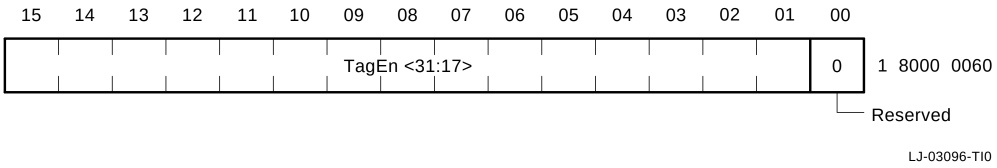


Table 4–4 Cache Size Tag Enable Values

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

Table 4–5 Maximum Memory Tag Enable Values

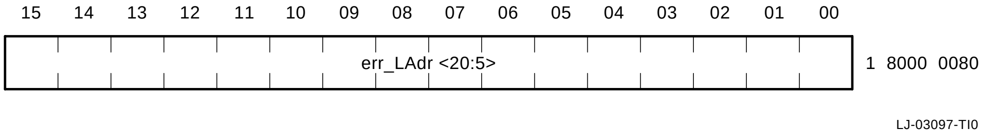
tagEn<15:0>	Compared	Memory Size
1111 1111 1111 111X	<31:17>	4 GB
0111 1111 1111 111X	<30:17>	2 GB
0011 1111 1111 111X	<29:17>	1 GB
0001 1111 1111 111X	<28:17>	512 MB
0000 1111 1111 111X	<27:17>	256 MB
0000 0111 1111 111X	<26:17>	128 MB
0000 0011 1111 111X	<25:17>	64 MB
0000 0001 1111 111X	<24:17>	32 MB
0000 0000 1111 111X	<23:17>	16 MB
0000 0000 0111 111X	<22:17>	8 MB
0000 0000 0011 111X	<21:17>	4 MB
0000 0000 0001 111X	<20:17>	2 MB
0000 0000 0000 111X	<19:17>	1 MB
0000 0000 0000 011X	<18:17>	512 KB
0000 0000 0000 001X	<17:17>	256 KB
0000 0000 0000 000X	None	128 KB

4.2.4 Error Low Address Register

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

Figure 4–4 shows the register bit assignments for the error low address register.

Figure 4–4 Error Low Address Register

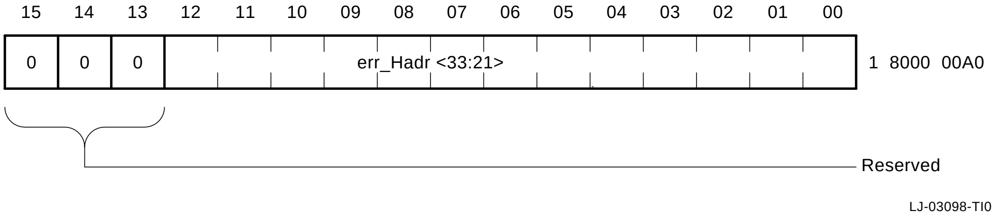


4.2.5 Error High Address Register

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

Figure 4–5 shows the register bit assignments for the error high address register.

Figure 4–5 Error High Address Register

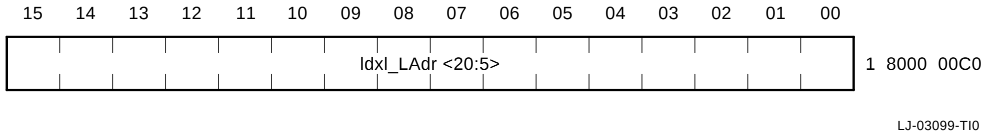


4.2.6 LD*x*_L Low Address Register

The LD*x*_L low address register stores the low order bits of the last locked address. Bits <15:0> in the register represent sysAdr<20:5>. This register is read-only, and it is not initialized.

Figure 4–6 shows the register bit assignments for the LD*x*_L low address register.

Figure 4–6 LD*x*_L Low Address Register

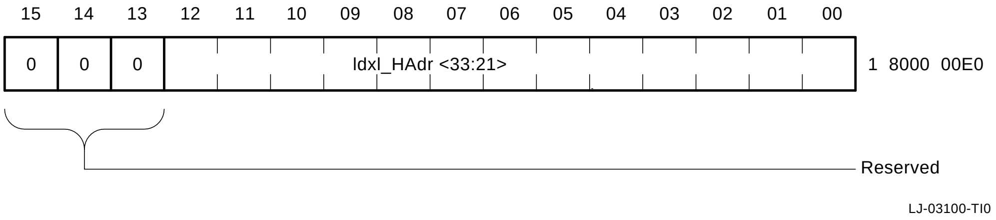


4.2.7 LD*x*_L High Address Register

The LD*x*_L high address register stores the high order bits of the locked address. Bits <12:0> in the register represent sysAdr<33:21>. This register is read-only, and it is not initialized.

Figure 4–7 shows the register bit assignments for the LD*x*_L high address register.

Figure 4–7 LD*x*_L High Address Register



4.3 Memory Registers

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

4.3.1 Video Frame Pointer Register

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

Figure 4–8 shows the register bit assignments, and Table 4–6 provides the bit descriptions for the video frame pointer register.

Figure 4–8 Video Frame Pointer Register

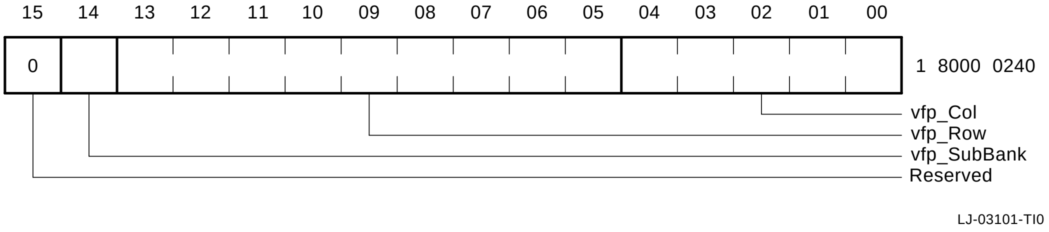


Table 4–6 Video Frame Pointer Register

Field	Bits	Type, Reset	Description
vfp_Col<4:0>	<4:0>	RW,–	Video frame column address pointer. vfp_Col<4:0> are used as column address <6:2> for all serial register loads.
vfp_Row<8:0>	<13:5>	RW,–	Video frame row address pointer. Row address of the start of the frame buffer.

(continued on next page)

Table 4–6 (Cont.) Video Frame Pointer Register

Field	Bits	Type, Reset	Description
vfp_SubBank	<14>	RW,–	Video frame subbank pointer. Subbank for the start of the frame buffer. If the subbank is enabled by setting s8_SubEna in the bankset8 configuration register, setting the vfp_SubBank bit causes the 21071-CA chip to assert memRASB_l<8> instead of memRAS_l<8> on full serial register loads. vfp_SubBank is ignored if s8_SubEna is cleared.
Reserved	<15>	MBZ	—

4.3.2 Presence Detect Low Data Register

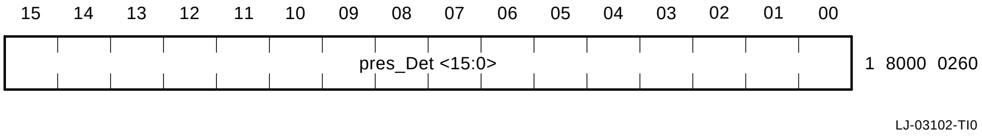
The presence detect low data register stores the low order bits of the presence detect information that was shifted in after reset. Bits <15:0> in the register represent data bits <15:0> that were shifted in.

Note

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

Figure 4–9 shows the register bit assignments for the presence detect low data register.

Figure 4–9 Presence Detect Low Data Register



4.3.3 Presence Detect High Data Register

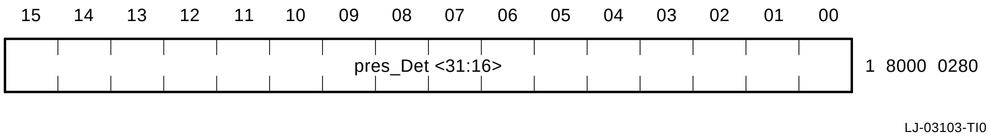
The presence detect high data register stores the high order bits of the presence detect information that was shifted in after reset. Bits <15:0> in the register represent data bits <31:16> that were shifted in.

Note

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

Figure 4–10 shows the register bit assignments for the presence detect high data register.

Figure 4–10 Presence Detect High Data Register



4.3.4 Base Address Registers

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

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

Note

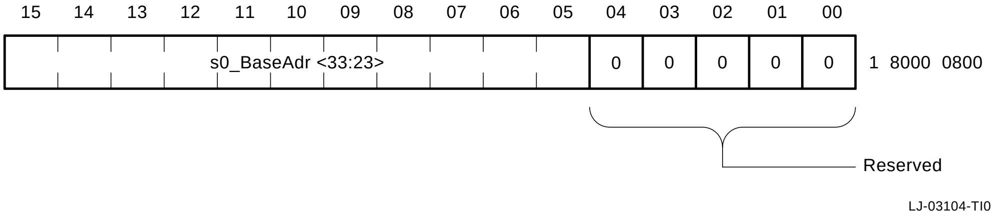
Software could require contiguous memory. Because banksets must be naturally aligned, the programmer should ensure that the largest bankset is placed at the lowest base address, the next largest bankset is placed at a base address following the end of the largest bankset, and so on, to create contiguous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

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

If bankset8 has parity or ECC checking disabled (s8_Check bit clear), then bankset8 must be mapped into noncacheable space (S8_BaseAdr<32> set).

Figure 4–11 shows the register bit assignments for the bankset0 base address register.

Figure 4–11 Bankset0 Base Address Register



4.3.5 Configuration Registers

Each memory bankset has a corresponding configuration register. This register contains mode bits and bits for memory address generation, as well as bankset decoding. Banksets 0 through 7 have the same limits on bankset size and type of DRAMs used. The format of the configuration register is the same for banksets 0 through 7. Bankset8 is the VRAM bank. It supports different minimum DRAM sizes and configurations; therefore, its configuration register is different.

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

Figure 4–12 shows the register bit assignments, and Table 4–7 provides the bit descriptions for the bankset 0 through 7 configuration registers.

Figure 4–12 Bankset 0 Configuration Register

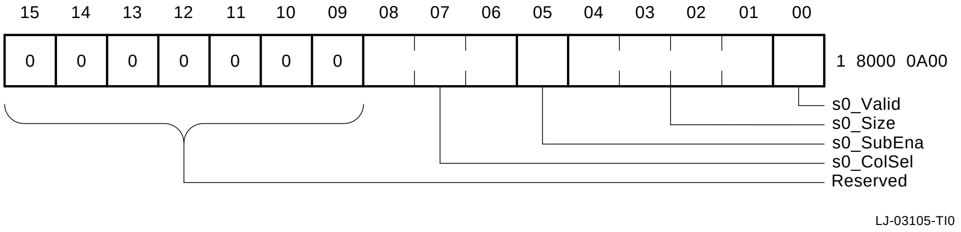


Table 4–7 Bankset0 Configuration Register

<table>
<tr><th>Field</th><th>Bits</th><th>Type, Reset</th><th>Description</th></tr>
<tr><td>s0_Valid</td><td><0></td><td>RW,0</td><td>Bankset0 valid. If set, all timing and configuration parameters for bankset0 are valid, and access to bankset0 is allowed. If cleared, access to bankset0 is not allowed.</td></tr>
<tr><td>s0_Size<3:0></td><td><4:1></td><td>RW,–</td><td>Bankset0 size in MB. Indicates the size of the bankset in order to determine which bits are used to compare the bankset base address with the physical address (PA) and to generate the subset. Corresponds to the total size of the bankset, including subbanks, if present. s0_Size<3> must be set to 0.
<table>
<tr><th>S0_ Size<3:0></th><th>Compared</th><th>Subset</th><th>Set Size</th></tr>
<tr><td>0000</td><td>—</td><td>—</td><td>Reserved</td></tr>
<tr><td>0001</td><td>PA<33:29></td><td>PA<28></td><td>512 MB</td></tr>
<tr><td>0010</td><td>PA<33:28></td><td>PA<27></td><td>256 MB</td></tr>
<tr><td>0011</td><td>PA<33:27></td><td>PA<26></td><td>128 MB</td></tr>
<tr><td>0100</td><td>PA<33:26></td><td>PA<25></td><td>64 MB</td></tr>
<tr><td>0101</td><td>PA<33:25></td><td>PA<24></td><td>32 MB</td></tr>
<tr><td>0110</td><td>PA<33:24></td><td>PA<23></td><td>16 MB</td></tr>
<tr><td>0111</td><td>PA<33:23></td><td>PA<22></td><td>8 MB</td></tr>
<tr><td>1XXX</td><td>—</td><td>—</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>s0_SubEna</td><td><5></td><td>RW,0</td><td>Enable subbanks. When set, subbanks are enabled and determined according to the previous table. When clear, subbanks are disabled, and the memRASB_l pins will be asserted only during refreshes.</td></tr>
</table>

(continued on next page)

Table 4–7 (Cont.) Bankset0 Configuration Register

Field	Bits	Type, Reset	Description
s0_ColSel<2:0>	<8:6>	RW,–	Column address selection. Indicates the number of valid column bits expected at the DRAMs. Used along with memory width information to generate row or column addresses. Memory width is determined by the wideMem pin. See Table 3–6 for more information.
Reserved	<15:9>	MBZ	—

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

Figure 4–13 shows the register bit assignments, and Table 4–8 provides the bit descriptions for the bankset8 configuration register.

Figure 4–13 Bankset8 Configuration Register

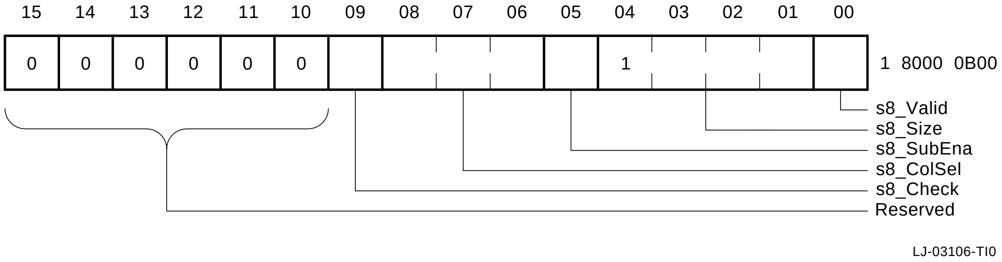


Table 4–8 Bankset 8 Configuration Register

Field	Bits	Type, Reset	Description
s8_Valid	<0>	RW,0	Valid. If set, all parameters are valid, and access to bankset8 is allowed. If cleared, no accesses to bankset8 are allowed.
s8_Size<3:0>	<4:1>	RW,0	Size. Indicates the size of the bankset in order to determine which bits are used to compare the base address with the physical address and to select the subset (if s8_SubEna is set). Corresponds to the total size of bankset8, including subbanks, if present.

s8_ Size<3:0>	Compared	Subbank	Bankset Size
0XXX	—	—	Reserved
1000	—	—	Reserved
1001	PA<33:23>	PA<22>	8 MB
1010	PA<33:22>	PA<21>	4 MB
1011	PA<33:21>	PA<20>	2 MB
1100	PA<33:20>	PA<19>	1 MB
1101	—	—	Reserved
1110	—	—	Reserved
1111	—	—	Reserved

(continued on next page)

Table 4–8 (Cont.) Bankset 8 Configuration Register

Field	Bits	Type, Reset	Description
s8_SubEna	<5>	RW,0	Enable subbanks. When set, subbanks are enabled and determined according to the previous table. When clear, subbanks are disabled, and the memRASB_l pins will only be asserted during refresh.
s8_ColSel<2:0>	<8:6>	RW,–	Column address selection. Indicates the number of valid column bits expected at the DRAMs. Used along with memory width information to generate column row or column addresses. Memory width is determined by the wideMem pin. See Table 3–7 for more information. **S8_ColSel** — **Row, Column Bits** 0XX — Reserved 100 — 9, 9 101 — 9, 8 11X — Reserved
s8_Check	<9>	RW,0	Enable ECC/parity checking. When set, accesses to bankset8, like other banksets, will have their parity or ECC checked. When clear, parity or ECC will not be checked. When clear, bankset8 must be mapped into noncacheable space. Only bankset8 has this feature. DMA accesses to this bank should not be performed when error checking is disabled.
Reserved	<15:10>	MBZ	—

4.3.6 Bankset Timing Registers A and B

Each bankset has two timing registers associated with it. These registers contain the timing parameters required to perform memory read and write transactions. The format of the timing registers is identical for all 9 banksets.

On reset, all the parameters are set to the maximum value. This may cause improper operation of the memory interface. The timing registers should be programmed by software before setting the corresponding bankset valid bit in the configuration register.

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

See Section 4.4 to determine how the timing register should be programmed for particular memory transactions.

Figure 4–14 shows the register bit assignments, and Table 4–9 provides the bit descriptions for the bankset timing register A.

Figure 4–14 Bankset Timing Register A

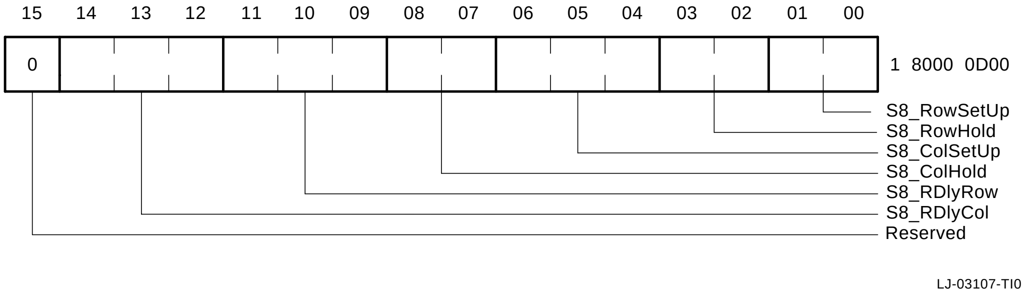


Table 4–9 BankSet Timing Register A

Field	Bits	Type, Reset	Description
s8_RowSetup<1:0>	<1:0>	RW,1s	Row address setup (t_{ASR}). Used to generate memRAS_l assertion from row address. Programmed_Value = Desired_Value –1
s8_RowHold<1:0>	<3:2>	RW,1s	Row address hold (t_{RAH}). Used to switch memAdr from row to column after memRAS_l assertion. Programmed_Value = Desired_Value –1

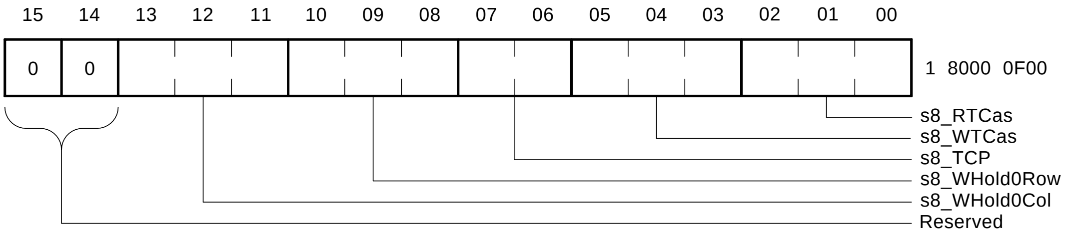
(continued on next page)

Table 4–9 (Cont.) BankSet Timing Register A

Field	Bits	Type, Reset	Description
s8_ColSetup<2:0>	<6:4>	RW,1s	Column address setup (t_{ASC}) to first CAS assertion and write enable setup (t_{CWL}) to CAS assertion. Used to determine first memCAS_l assertion after column address and memCAS_l assertion after memWE_l. The maximum of the two setup values should be programmed. A programmed value of 7 is illegal. Programmed_Value = Desired_Value –1
s8_ColHold<1:0>	<8:7>	RW,1s	Column hold (t_{CAH}) from memCAS_l assertion. Used to determine when the current column address can be changed to the next column or row address. Programmed_Value = Desired_Value –1
s8_RDlyRow<2:0>	<11:9>	RW,1s	Read delay from row address. Delay from row address to latching first valid read data. Programmed_Value = Desired_Value –4
s8_RDlyCol<2:0>	<14:12>	RW,1s	Read delay from column address. Used only when starting in page mode. Delay from column address to latching first valid read data. Programmed_Value = Desired_Value –2
Reserved	<15>	MBZ	—

Figure 4–15 shows the register bit assignments, and Table 4–10 provides the bit descriptions for the bankset timing register B.

Figure 4–15 Bankset Timing Register B



LJ-03108-TI0

Table 4–10 Bankset Timing Register B

Field	Bits	Type, Reset	Description
s8_RTCas<2:0>	<2:0>	RW,1s	Read CAS width (t_{CAS}). Used on reads to generate the memCAS_l deassertion from the assertion of memCAS_l. Note: RTCas and TCP should be programmed so that their sum is $\leq$ 5. Programmed_Value = Desired_Value –2
s8_WTCas<2:0>	<5:3>	RW,1s	Write CAS width (t_{CAS}). Used on writes to generate the memCAS_l deassertion from the assertion of memCAS_l. Note: WTCas and TCP should be programmed so that their sum is $\leq$ 5. Programmed_Value = Desired_Value –2
s8_TCP<1:0>	<7:6>	RW,1s	CAS precharge (t_{CP}). Delay from memCAS_l deassertion to the next assertion of memCAS_l in page mode. Programmed_Value = Desired_Value –1
s8_WHold0Row<2:0>	<10:8>	RW,1s	Write hold time from row address. Hold time of first write data from first row address. The first write data is valid with the row address and is held valid s8_WHold0Row + 2 cycles after the row address. Used when not starting in page mode. A programmed value of zero is illegal. Programmed_Value = Desired_Value –2
s8_WHold0Col<2:0>	<13:11>	RW,1	Write hold time from column address is used only for the first data when starting in page mode. Write data is valid with the column address and is held valid S8_WHold0Col + 2 cycles after the column address. Programmed_Value = Desired_Value –2
Reserved	<15:14>	MBZ	—

4.3.7 Global Timing Register

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

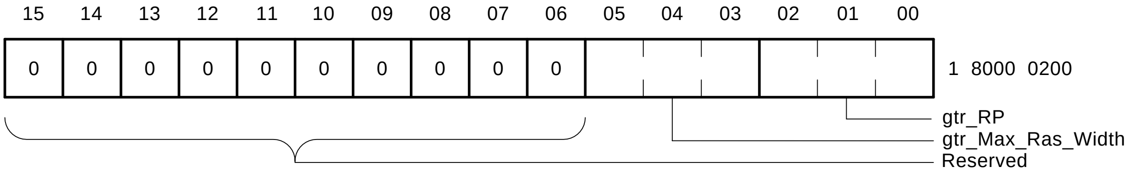
Note

The 21071-CA chip requires the RAS precharge interval to be smaller than the length of a complete memory transaction. For values of gtr_RP less than or equal to 4 (which gives a 6 memClk precharge), there are no restrictions. For RAS precharge greater than 6 memClk cycles, each valid bankset must satisfy the following conditions:

$$\text{gtr\_RP} \leq \text{RowHold} + \text{ColSetup} + \text{WTCas} + 4$$
$$\text{gtr\_RP} \leq \text{RowHold} + \text{ColSetup} + \text{RTCas} + 4$$

Figure 4–16 shows the register bit assignments, and Table 4–11 provides the bit descriptions for the global timing register.

Figure 4–16 Global Timing Register



LJ-03109-TI0

Table 4–11 Global Timing Register

Field	Bits	Type, Reset	Description
gtr_RP<2:0>	<2:0>	RW,1s	Minimum number of RAS precharge cycles. memRAS_l deassertion to next assertion of the same memRAS_l pin. Corresponds to DRAM parameter t_{RP}. Programmed_Value = Desired_Value –2
gtr_Max_Ras_Width<2:0>	<5:3>	RW,1s	Maximum RAS assertion width as a multiple of 128 memClk cycles. When this count is reached, the asserted memRAS_l is deasserted at the end of the ongoing transaction. This value should be programmed with sufficient margin to allow for the timer overflowing during a transaction. Corresponds to DRAM parameter t_{RAS}. When programmed to a 0, page mode between transactions will be disabled.
Reserved	<15:6>	MBZ	—

4.3.8 Refresh Timing Register

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

The observed refresh interval may be greater than the value programmed in ref_interval by the number of memClk cycles required to perform a read or write plus a RAS precharge interval. The programmer must account for this behavior when choosing the value of ref_interval.

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

Figure 4–17 shows the register bit assignments, and Table 4–12 provides the bit descriptions for the refresh timing register.

Figure 4–17 Refresh Timing Register

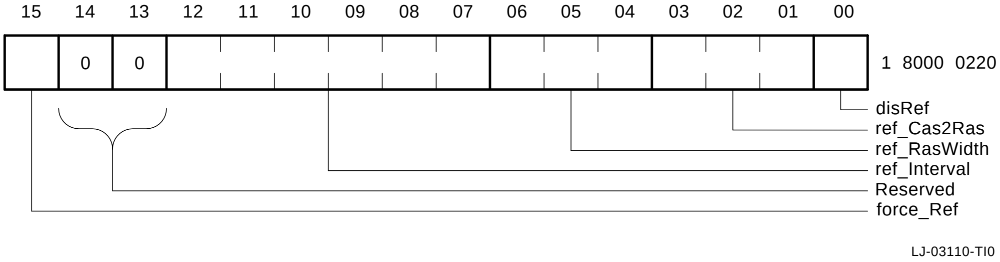


Table 4–12 Refresh Timing Register

Field	Bits	Type, Reset	Description
disRef	<0>	RW,0	Disable refresh. Refresh operations will not be performed when disRef is set.
ref_Cas2Ras<2:0>	<3:1>	RW,1s	Refresh CAS assertion to RAS assertion cycles. Corresponds to DRAM parameter t_{CSR}. Programmed_Value = Desired_ Value –2
ref_RasWidth<2:0>	<6:4>	RW,1s	Refresh RAS assertion width, from memRAS_l assertion to memRAS_l deassertion. memCAS_l is deasserted with memRAS_l for refresh. Corresponds to DRAM parameter t_{RAS}. Programmed_Value = Desired_ Value –3

(continued on next page)

Table 4–12 (Cont.) Refresh Timing Register

Field	Bits	Type, Reset	Description
ref_Interval<5:0>	<12:7>	RW,000001	Refresh interval. Multiplied by 64 to generate number of memClk cycles between refresh requests. A programmed value of zero is illegal.
Reserved	<14:13>	MBZ	—
force_Ref	<15>	WO,–	Force refresh. Writing a 1 to this bit causes a single memory refresh. Reads as 0. Resets the internal refresh interval counter. The other timings in this register should not be changed while setting this bit. Force refresh overrides disable refresh.

4.4 Programming Memory Timing

This section describes how a system designer should program the memory timings for a particular memory configuration, DRAM speed, and sysClk cycle time. The system designer should:

1. Develop a timing diagram for memory reads, writes, refreshes, page mode reads, and page mode writes for the chosen memory configuration and sysClk cycle time.
2. Count the number of cycles required for a particular parameter. This is the desired value that is referred to in the description of the various parameters. For each parameter there is an equation to generate the programmed value from the desired value (generally by subtracting a constant from the desired value).

Warning

The memData driving and latching state machines run independently from the state machine that controls memRas_l, memCas_l, memAdr, and the other controls. The two machines start at the same time, and then use the programmed timing to cycle through the transaction. Arbitrarily programming RDlyRow, RDlyCol, WHold0Row, and WHold0Col could result in illegal memory transactions.

Tables 4–13 and 4–14 provide equations that must be applied while programming the memory timings.

Table 4–13 Read Timings: Equations for Programmed Values

$\text{RDlyROW} = \text{RowSetUp} + \text{RowHold} + \text{ColSetUp} + \text{Taccess}^{1} - 1$

$\text{RDlyCol} = \text{ColSetUp} + \text{Taccess} - 1$

$\text{RTCas} \geq \text{Taccess} - 2$

$\text{RTCas} + \text{TCP} \leq 5$

[1] Taccess is the access time in memClks for data from CAS assertions, determined by module signal integrity and DRAM timing.

Table 4–14 Write Timings: Equations for Programmed Values

$\text{WHold0Row} = \text{RowSetUp} + \text{RowHold} + \text{ColSetUp} + \text{TDataHold}^{1} + 1$

$\text{WHold0Col} = \text{ColSetUp} + \text{TDataHold} - 1$

$\text{WTCas} \geq \text{TDataHold} - 2$

$\text{WTCas} + \text{TCP} \leq 5$

[1] TDataHold is the data hold time, in memClk cycles from CAS assertions, determined by module signal integrity and DRAM timing.

Figures 4–18 and 4–19 show the timing for a memory write and memory read, respectively. Assume that the two timing diagrams shown are for the same bankset. The programming for these transactions is shown in Table 4–15.

Table 4–15 Programming Memory Timings

Parameter	Desired Value	Programmed Value	Timing Diagram
RowSetUp	2	1	Read, Write
RowHold	2	1	Read, Write
ColSetUp	2	1	Read, Write
ColHold	2	1	Read, Write
RTCas	3	1	Read
TCP	1	0	Read, Write
RDlyRow	9	5	Read
WTCas	2	0	Write
WHold0Row	8	6	Write
gtr_RP	4	2	Read, Write

Figure 4–18 Memory Write Timing

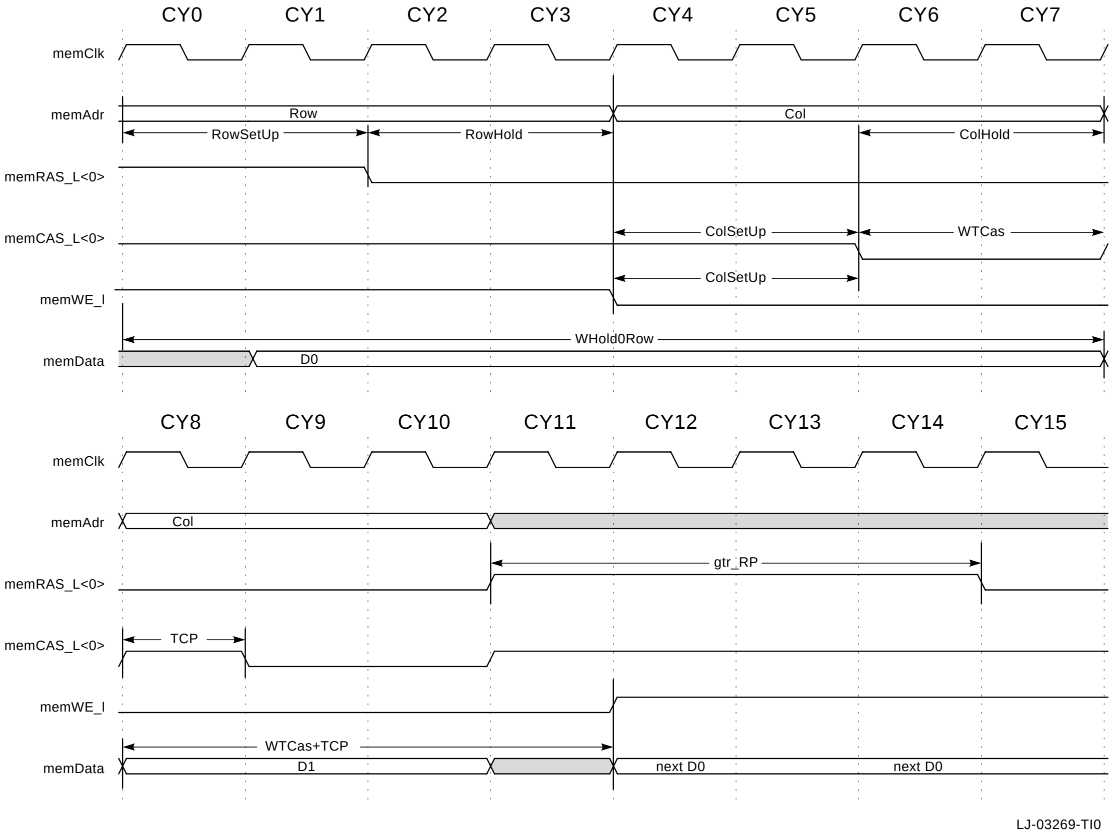
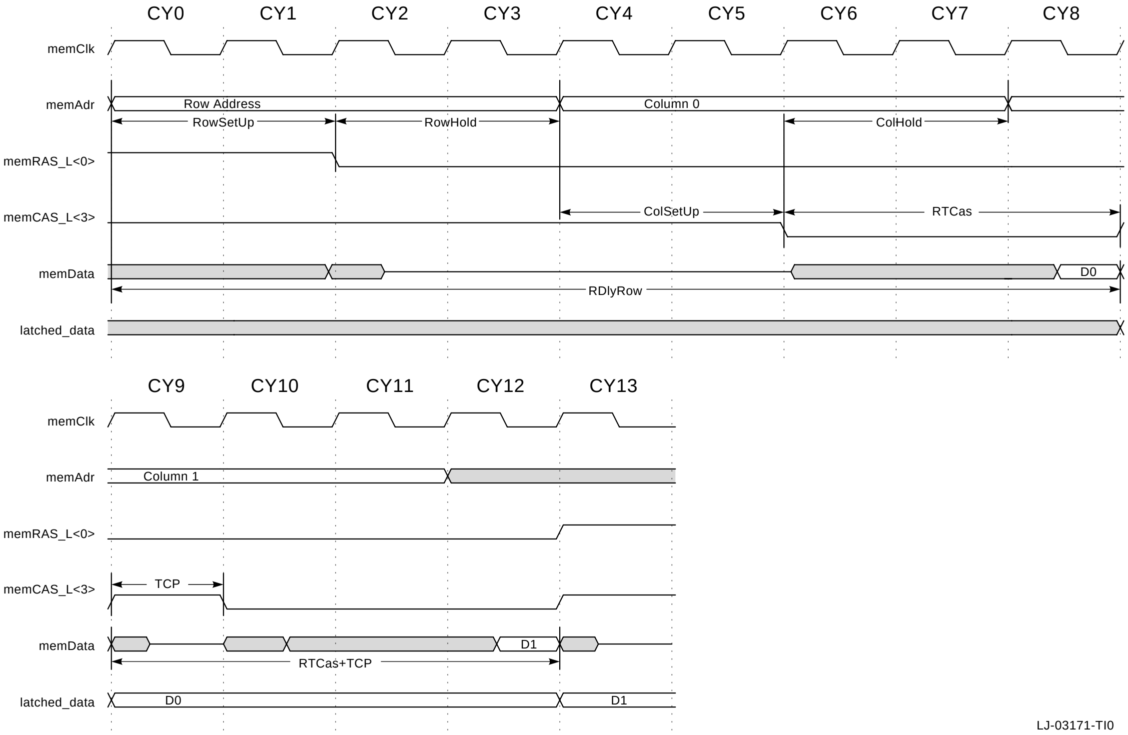


Figure 4–19 Memory Read Timing



4.5 Configuring Memory

The 21071-CA memory configuration and timing registers must be set up before memory can be read and written by the CPU. Firmware must determine the number of memory banksets in the system and the speed and size of the memory SIMMs used.

The 21071-CA provides two methods for determining memory configuration.

4.5.1 Using the 21071-CA Presence Detect Registers to Configure Memory

The system designer could use the presence detect registers in the 21071-CA to load in the value of the presence detect pins of the memory SIMMs following the deassertion of reset. See Section 3.2.7 for the details of this operation.

4.5.2 Polling Memory to Configure Memory

This method can be used if the presence detect pins are not accessible via the 21071-CA presence detect registers. The following algorithm can be used by the firmware to determine memory configuration.

1. Configure all banksets invalid by writing 0 to the 21071-CA bankset configuration registers.
2. Read the general control register to determine whether memory is 128 bits wide or 64 bits wide.

   The procedure for determining the configuration is the same in both cases, except that the sizes in MB mentioned in the following steps should be halved for 64-bit wide memory. Start with bankset0.
3. Configure bankset as valid with a base address = 0, bankset size = 512 MB, ColSel = 000 (12,12 DRAMs), and subEna = 1 (subbanks enabled).
4. Configure the bankset timing registers for slow memory. For example, Timing Register A = 4F99#16 and Timing Register B = 17D2#16.
5. Write 11111111#16 to address 0.
6. Write 22222222#16 to address 10#16.
7. Read address 0; if the data is not 11111111#16, bankset has no memory. Store this information; configure the bankset as invalid. Go back to step 3 and start with the next bankset. If the data read is 11111111#16, bankset has memory; go to the next step.
8. Write 33333333#16 to address 128 MB.
9. Write 44444444#16 to address 0.
10. Read address 128 MB. If the data returned is 44444444#16, the bankset has wrapped back to address 0; the bankset under investigation is not a 12,12 bankset. Go to step 11. If the data is not 44444444#16, the bankset is a 12,12 bankset. Determine whether it has subbanks:
    - Write address 256 MB with 55555555#16; this attempts to write the upper subbank.
    - Write address 0 with 66666666#16.
    - Read address 256 MB. If the data is 55555555#16, the subbank exists; if not, this bankset does not have subbanks.
    - At this point, all the information for this bankset is known. Store this information and configure the bankset as invalid. Go to step 3 and start with the next bankset.

11. Write 77777777#16 to address 16 MB.
12. Write 88888888#16 to address 0.
13. Read address 16 MB. If data returned is 88888888#16, the bankset is not a 12,10 or 11,11 bankset; the bankset under investigation is not a 12,10 or 11,11 bankset. Go to step 15. If the data is not 88888888#16, the bankset is a 12,10 or 11,11 bankset. Determine whether it has subbanks:
    - Configure the bankset size to 128 MB with subEna = 1 (subbanks enabled). The ColSel and base address should remain unchanged.
    - Write address 64 MB with 99999999#16; this attempts to write the upper subbank.
    - Write address 0 with AAAAAAAA#16.
    - Read address 64 MB. If the data is 99999999#16, the subbank exists; if not, this bankset does not have subbanks.
    - At this point, all the information for this bankset is known. Store this information and configure the bankset as invalid. Go to step 3 and start with the next bankset.
14. Write BBBBBBBB#16 to address 8 MB.
15. Write CCCCCCCC#16 to address 0.
16. Read address 8 MB. If the data returned is CCCCCCCC#16, the bankset is not a 10,10 bankset. An illegal bankset has been inserted. If the data returned is not CCCCCCCC#16, the bankset is a 10,10 bankset. Determine whether it has subbanks:
    - Configure bankset size to 32 MB with subEna = 1 (subbanks enabled). The ColSel and base address should remain unchanged.
    - Write address 16 MB with DDDDDDDD#16; this attempts to write the upper subbank.
    - Write address 0 with EEEEEEEE#16.

    - Read address 16 MB. If the data is DDDDDDDD#16, the subbank exists; if not, this bankset does not have subbanks.
    - At this point, all the information for this bankset is known. Store this information and configure the bankset as invalid. Go to step 3 and start with the next bankset.

17. When the configurations of all the banksets are known, set up the base addresses of each bankset. The largest bankset should be mapped to the lowest base address.

4.6 Bcache Initialization

Firmware has to initialize the Bcache and memory before booting the operating system. The following sections describe the two methods used to initialize the Bcache and memory.

4.6.1 Primary Method to Initialize the Bcache

1. Disable the Bcache—BIU_CTL<bc_En>=0 and 21071-CA GCR<bc_En>=0, GCR<bc_IgnTag>=0.
2. Disable machine checks ABOX_CTL<MCHK_EN>=0.
3. Write something to *all* locations throughout the available memory. This will put good data parity/ECC in the memory SIMMs.
4. Enable Bcache in 21071-CA only—21071-CA GCR<bc_En>=1, GCR<bc_IgnTag>=1.
5. Clear all bits of the 21071-CA Tag Enable Register.
6. Read all cache locations between location zero and (cache_size — 1 byte). Because bc_IgnTag is set, the DECchip 21071-CA will fetch data from memory and put it in the cache as a clean block with correct tag parity.

Warning

Reading an area of memory other than between location zero and (cache_size —1 byte) will result in leaving the Bcache and main memory in an incoherent state.

7. Set the 21071-CA Tag Enable Register to the appropriate value based on system cache and memory size.

8. Clear the DECchip 21071-CA bc_IgnTag—GCR<bc_IgnTag>= 0.
9. Enable Bcache in the Alpha 21064 microprocessor— BIU_CTL<bc_En>=1.
10. Enable machine checks (if desired) ABOX_CTL<MCHK_EN>=1.

4.6.2 Alternative Method to Initialize the Bcache

1. Enable the Bcache—BIU_CTL<BC_EN>=1 and 21071-CA GCR<bc_En>=1, GCR<bc_IgnTag>=1, GCR<bc_NoAlloc>=1.
2. Disable machine checks ABOX_CTL<MCHK_EN>=0.
3. Clear all bits of the 21071-CA Tag Enable Register.
4. Read all cache locations between location zero and (cache_size — 1 byte). (Due to random initialization of the Bcache Ram bits some of these reads will rarely hit in the Bcache.)
5. Set the 21071-CA Tag Enable Register to the appropriate value based on system cache and memory size.
6. Read all cache locations between location cache_size and ((2 x cache_size) — 1 byte). All of these reads will result in a Bcache miss, and the 21071-CA chip will read uninitialized data from memory and put it in the cache as a clean block with correct tag parity.
7. Clear 21071-CA bcIgn_Tag—GCR<bcIgn_Tag>= 0.
8. Write something to *all* locations throughout the available memory. This will result in all of memory having correct data parity/ECC.
9. Enable machine checks (if desired) ABOX_CTL<MCHK_EN>=1.

Note

BIU_CTL and ABOX_CTL are registers in the Alpha 21064 microprocessor.

5

DECchip 21071-CA Transactions and Timing Diagrams

This chapter describes the transactions that are supported by the 21071-CA chip on the sysBus interface and the memory interface. When a topic is discussed, refer to the associated timing diagram.

5.1 sysBus Transactions

The following sections describe the CPU, DMA, arbitration, and write speed transactions.

5.1.1 CPU Transactions

This section describes the CPU transactions.

5.1.1.1 Idle

When the CPU is idle, the 21071-CA chip prepares for the next CPU transaction. The cache controls, with the exception of sysEarlyOEEn, are disabled. This will enable the cache tags on a CPU read or write, and enable the cache data on a read.

5.1.1.2 Read Block

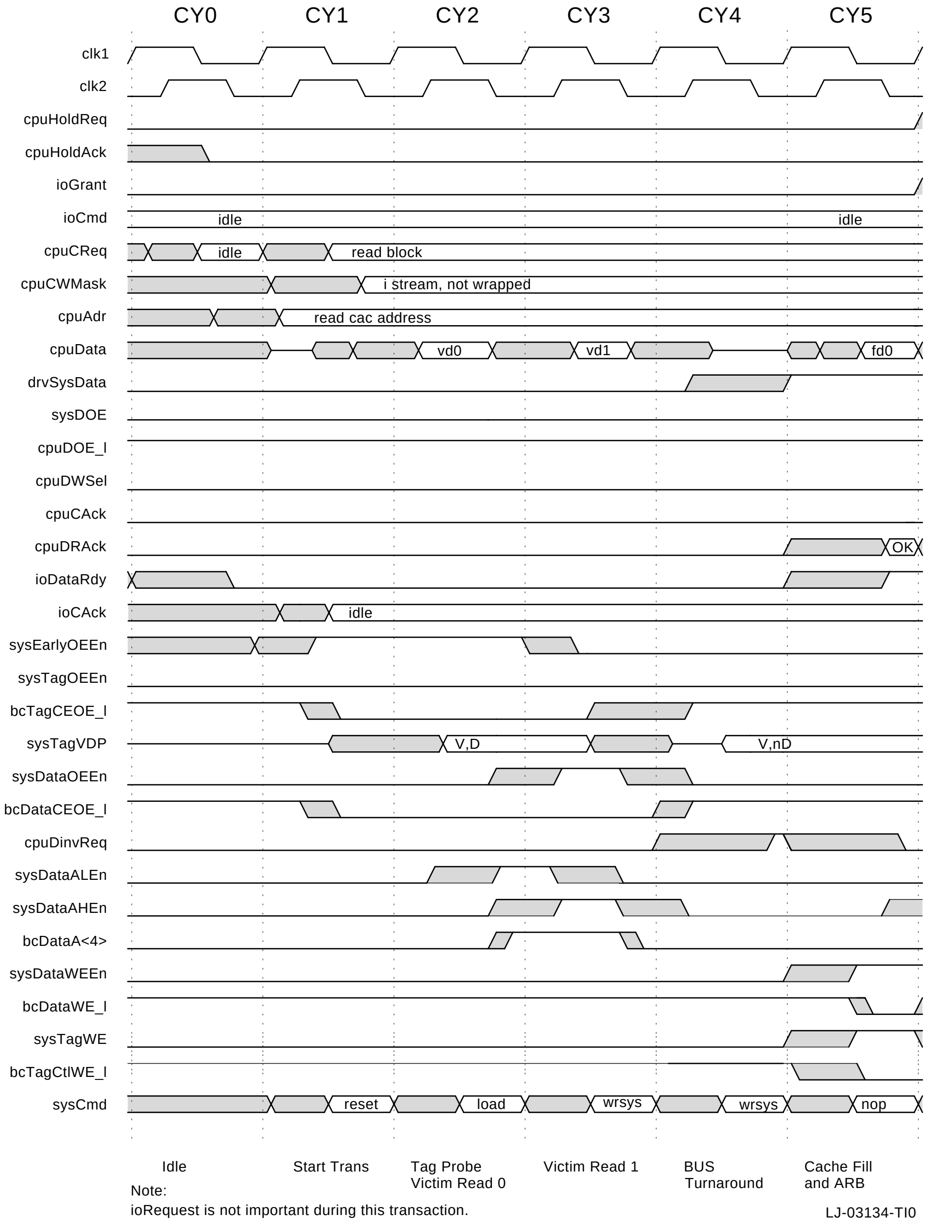
This section describes the read block transactions.

5.1.1.2.1 Cacheable With Victim The following table describes the cycles for a CPU read block transaction in cacheable space with a victim, as shown in Figure 5–1.

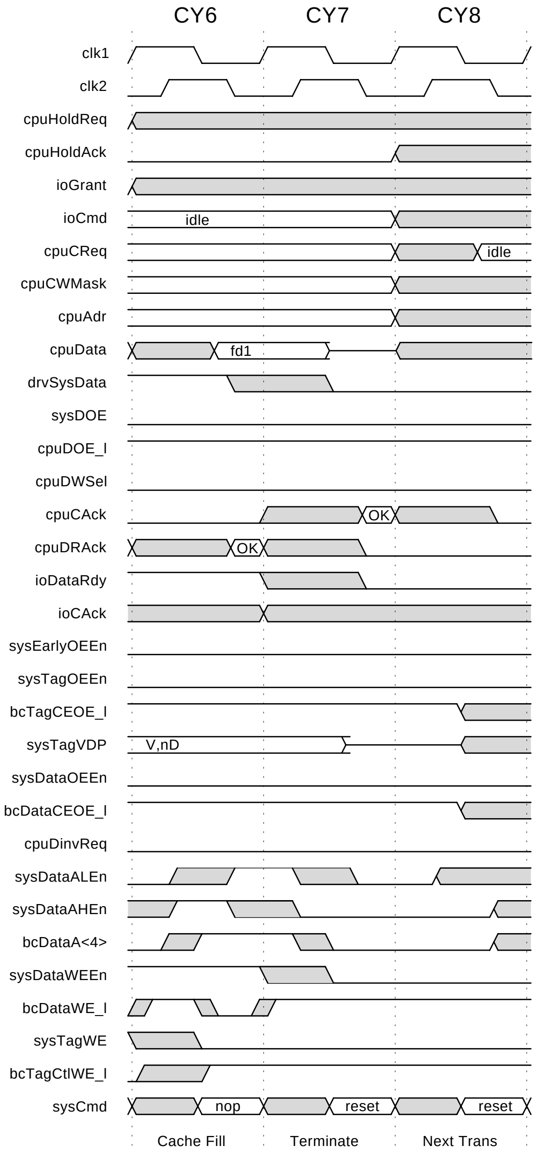
Cycle	Description
0	A read block begins during the idle cycle. The address is becoming valid because the CPU is doing a probe of the Bcache.
1	The CPU requests a read block with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcDataOE and bcTagOE.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in cacheable memory space. Also, the cache tag is available and indicates a victim must be processed. The first octaword of victim data is already on sysData<127:0>. To prepare for the rest of the victim, sysDataALEn is asserted, followed one-half cycle later by sysDataAHEn. These will produce a one cycle pulse on bcDataA4 beginning on clk2F. To maintain the data output from the cache, sysEarlyOEEn is left asserted and sysDataOEEn is asserted.
3	The second octaword of the victim is received. The 21071-CA chip prepares to drive the bus by deasserting the cache controls sysEarlyOEEn and sysDataOEEn.
4	The read of the victim is complete. The cache tags are driven by the 21071-CA chip with the tag information for the fill data (valid and clean). If the CPU requested a wrapped read, bcDataA<4> would be asserted for the first time. Figure 5–8 shows a wrapped LD*x*_L read. If the read is in the instruction stream, as indicated by cpuCWMask<2> being false, and the cache line was previously valid, the CPU internal Dcache is invalidated using cpuDInvReq.
5	The system may stall for any number of cycles waiting for the read data to be available, although in this example the read data is ready now. The read data is driven onto the data bus. Using cpuDRAck<2:0>, the data is acknowledged as OK. sysTagWE is asserted, which generates bcTagCtlWE and bcTagAdrWE to write the tags into the cache. SysDataWEEn is asserted, in turn generating bcDataWE, which writes the data into the cache. To prepare to write, the second octaword bcDataA<4> is asserted. Figure 5–1 shows a single write pulse of half the system clock width. The 21071-CA chip also supports a write pulse of twice that duration. (See Section 5.1.4.)

Cycle	Description
6	The second octaword is written with sysDataWEEn, and again acknowledged as OK using cpuDRAck<2:0>. bcDataA<4> is deasserted once we are done with the write. The arbiter could decide that DMA will be granted the bus, as indicated by the unknown (X's) on cpuHoldReq and ioGrant. For more information about arbitration, see Section 5.1.3.
7	The cycle is acknowledged with cpuCAck<2:0>, and the data drivers and cache controls are returned to their default state. It is not possible to assert cpuCAck<2:0> sooner, because the CPU data bus drivers could have created a bus contention with the memory output buffers.
8	The transaction is complete and the next transaction is ready to begin. If the CPU won arbitration, sysEarlyOEEn will be asserted in the next cycle in preparation for the next transaction. If the 21071-DA chip won arbitration, this cycle is used for bus turnaround.

Figure 5–1 Timing of CPU Read Block, Cacheable, Victim



Note:
ioRequest is not important during this transaction.



Note:
ioRequest is not important during this transaction.

LJ-03135-TI0

5.1.1.2.2 Cacheable Without Victim A read without a victim is similar to Figure 5–1. When the tag results are clean or invalid in cycle 2, the information read out of the cache is discarded. The transaction has the same length and control signals as the victim case described in Section 5.1.1.2.1.

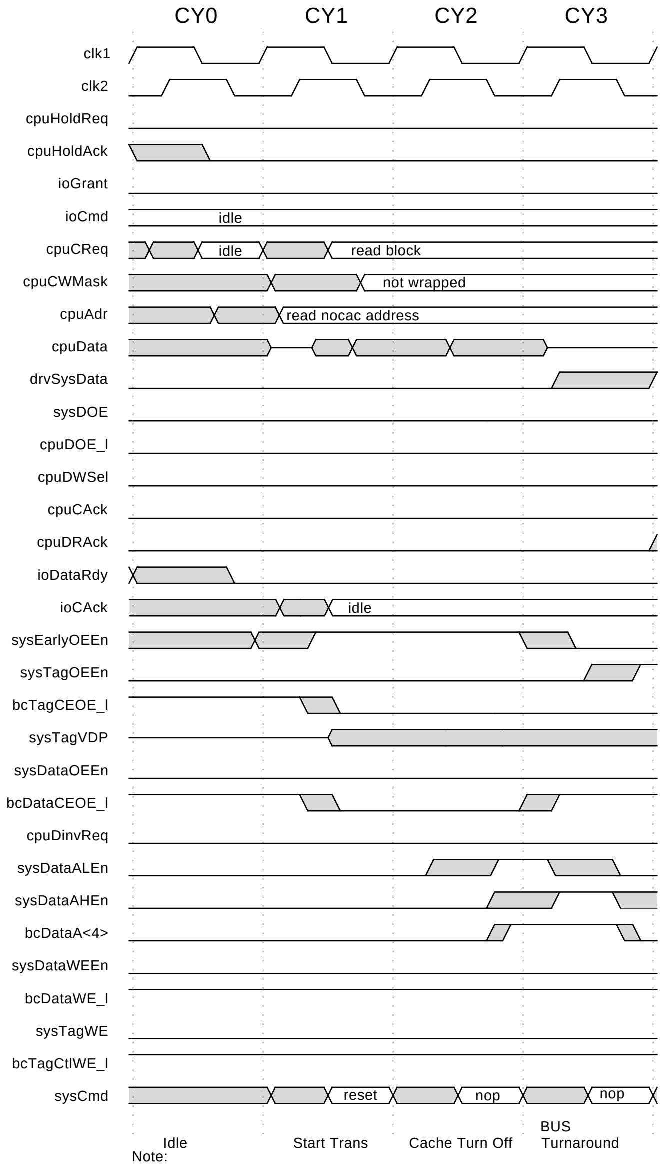
5.1.1.2.3 Noncacheable The following table describes the cycles for a CPU read block transaction in noncacheable space, as shown in Figure 5–2.

Cycle	Description
0	In read block to noncacheable space, the address is placed on the bus one CPU cycle (as little as 3 ns) before clk1R.
1	The CPU requests a read block with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcDataOE and bcTagOE.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it is in noncacheable memory space. The 21071-CA chip prepares to drive the bus so sysEarlyOEEn is deasserted, which deasserts bcDataOE and bcTagOE.
3	SysTagOE is asserted to prevent the cache tags from floating. The 21071-CA chip waits for the cache data to tristate.
4	The read data is ready and is driven onto the data bus. Using cpuDRAck<2:0>, the data is acknowledged as noncacheable. CpuDInvReq does not assert in noncacheable space. The CPU does not require more than an octaword of data; therefore, only one data transfer is required.
5	The cycle is acknowledged with cpuCAck<2:0> and the data drivers are returned to their default state.
6	The transaction is complete, and the next transaction is ready to begin.

Note

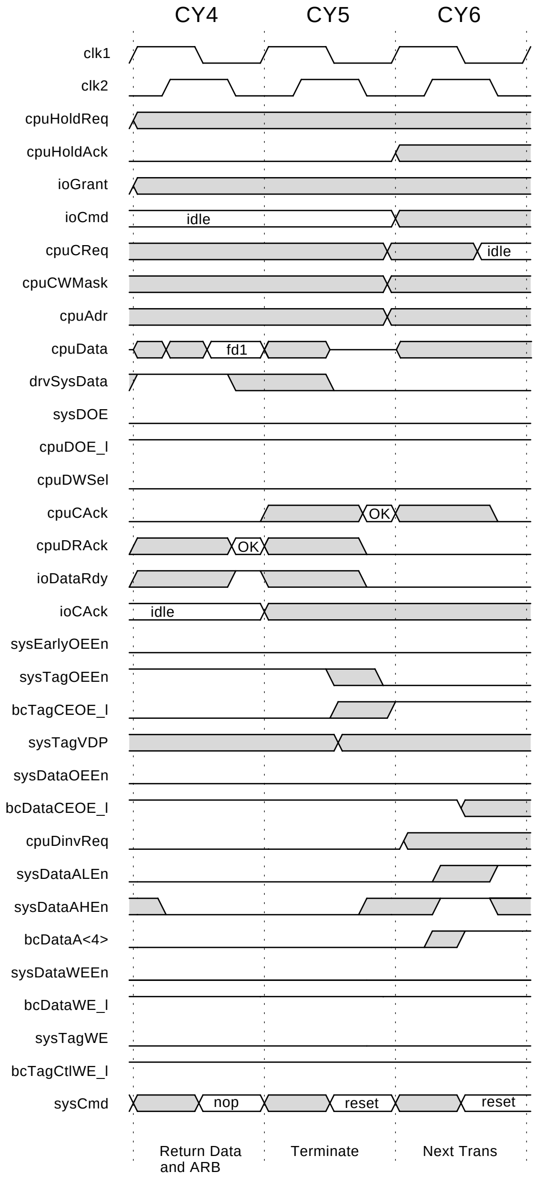
A read block with the cache disabled is similar to a noncacheable read. However, a full hexaword is returned, and OK will be sent on cpuDRack<2:0> so that the CPU will place the data in its Dcache or Icache.

Figure 5–2 Timing of CPU Read Block, Noncacheable



Note:
ioRequest is not important during this transaction.

LJ-03160-TI0



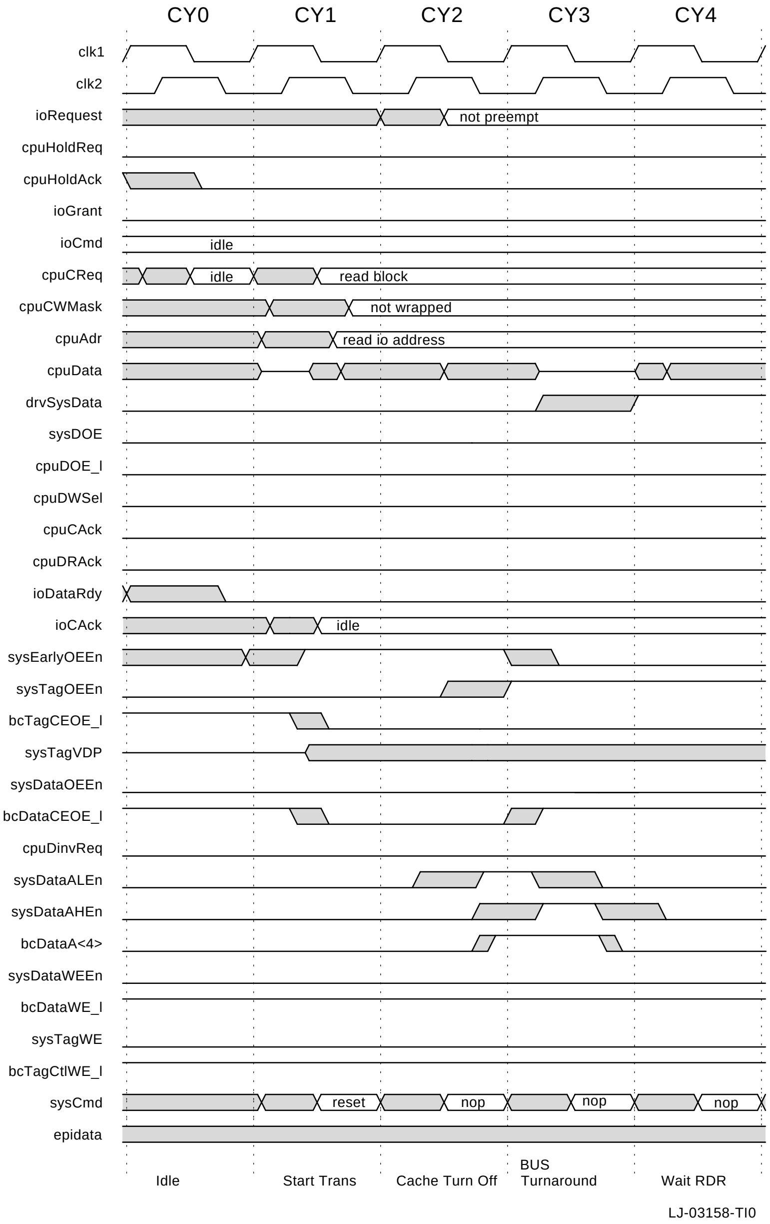
Note:
ioRequest is not important during this transaction.

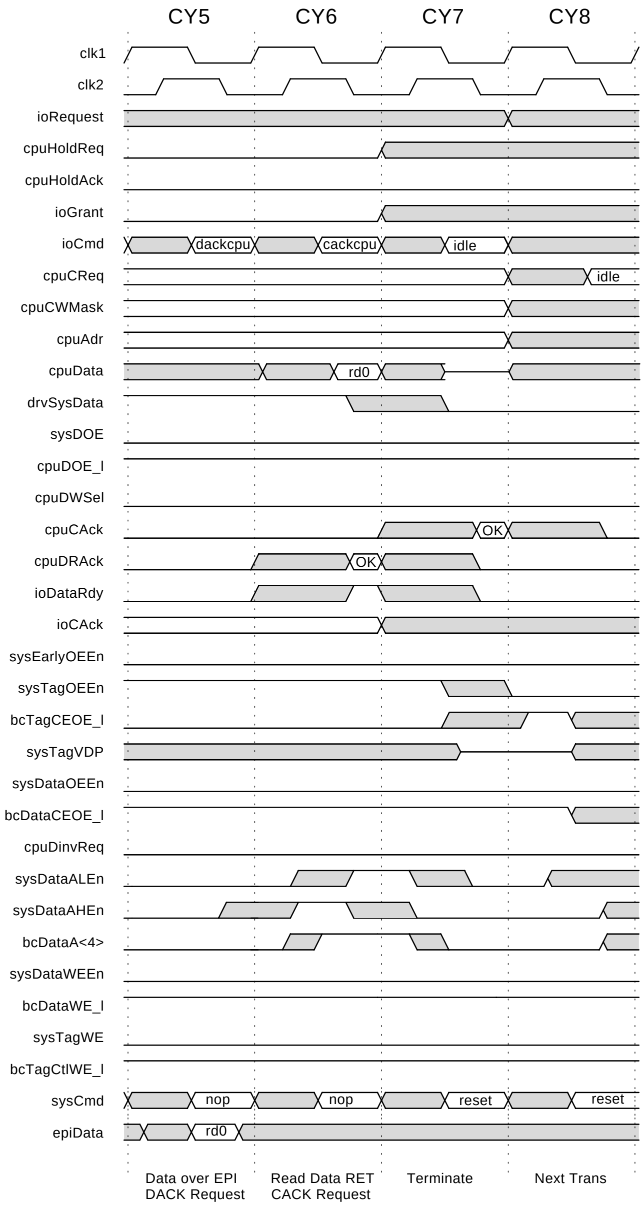
LJ-03161-TI0

5.1.1.2.4 I/O Space The following table describes the cycles for a CPU read block transaction in remote I/O space, as shown in Figure 5–3.

Cycle	Description
0	As I/O space is noncacheable, the address is placed on the bus one CPU cycle (as little as 3 ns) before clk1R.
1	The CPU requests a read block with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcDataOE and bcTagOE.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in I/O space. To get the cache off the bus, while preventing the tag from floating, sysEarlyOEEn is deasserted and sysTagOEEn is asserted.
3	The 21071-CA chip waits for the cache data to tristate. The 21071-DA chip processes the I/O read.
4	The 21071-CA chip could return data in this cycle, but the data is not ready for two more cycles.
5	The 21071-DA chip loads the merge and I/O read buffer on the 21071-BA chip using the epiBus. It indicates that the read data is loaded and can be sent to the CPU in the next cycle, so it requests a cpuDRAck<2:0> using ioCmd<2:0>. If more than one longword is being read, multiple epiData transfers are required; the last epiData transfer has the cpuDRack request.
6	The read data is ready and is driven onto the data bus. The 21071-CA chip receives the cpuDRAck<2:0> request on ioCmd<2:0> and asserts cpuDRAck<2:0> as noncacheable. A CPU cycle acknowledge is requested using ioCmd<2:0>.
7	The 21071-CA chip receives ioCmd<2:0>, tristates its data drivers, and acknowledges the cycle with cpuCAck<2:0>. The cache is turned off by deasserting sysTagOEEn.
8	The transaction is complete, and the next transaction is ready to begin.

Figure 5–3 Timing of CPU Read Block, Remote I/O Space




CY5
CY6
CY7
CY8
clk1
clk2
ioRequest
cpuHoldReq
cpuHoldAck
ioGrant
ioCmd
dackcpu
cackcpu
idle
cpuCReq
idle
cpuCWMask
cpuAdr
cpuData
rd0
drvSysData
sysDOE
cpuDOE_l
cpuDWSel
cpuCAck
OK
cpuDRAck
OK
ioDataRdy
ioCAck
sysEarlyOEEn
sysTagOEEn
bcTagCEOE_l
sysTagVDP
sysDataOEEn
bcDataCEOE_l
cpuDinvReq
sysDataALEn
sysDataAHEn
bcDataA<4>
sysDataWEEn
bcDataWE_l
sysTagWE
bcTagCtlWE_l
sysCmd
nop
nop
reset
reset
epiData
rd0
Data over EPI
DACK Request
Read Data RET
CACK Request
Terminate
Next Trans

LJ-03159-TI0

5.1.1.3 Write Block

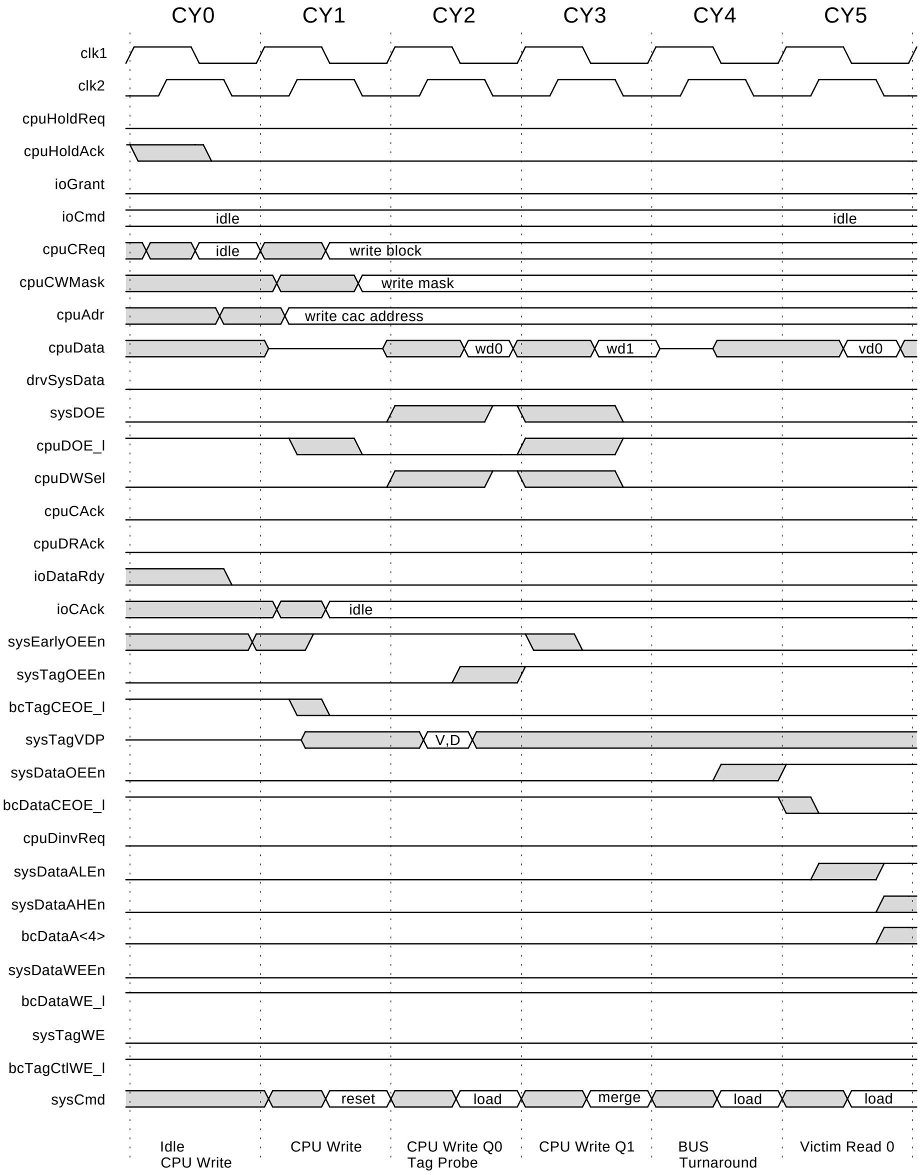
This section describes the write block transactions.

5.1.1.3.1 Cacheable Allocate With Victim The following table describes the cycles for a CPU write block transaction in cacheable space with a victim and with write allocation enabled, as shown in Figure 5–4.

Cycle	Description
0	A write block begins during the idle cycle. The address is becoming valid due to the CPU doing a probe of the Bcache. Systems may rely on cacheable address being set up for the time it takes the CPU to do a probe (a minimum of 10 ns).
1	The CPU requests a write block with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcTagOE and cpuDOE_l.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in cacheable memory space. The CPU sees the assertion of cpuDOE_l; the first octaword of write data is placed on the cpuData bus and is latched by the 21071-BA chip. The 21071-CA chip asserts sysDOE to ensure that cpuDOE_l will not deassert too soon. The 21071-CA chip asserts sysTagOEEn to prevent the tag bus from floating. The 21071-CA chip also asserts cpuDWSel to get the second octaword of write data. The cache tag indicates that a victim must be processed.
3	The CPU sees the assertion of cpuDWSel and places the second octaword of write data on the cpuData bus. The 21071-CA chip deasserts sysEarlyOEEn. The data is latched by the 21071-BA chip. The 21071-CA chip deasserts sysDOE and cpuDWSel.
4	The sysData bus is tristated by the CPU. The 21071-CA chip asserts sysDataOEEn causing the cache to begin driving the data bus. sysDataOEEn is asserted on clk1F, rather than on the normal clk2F, to allow additional cache output enable access time.
5	The first octaword of victim data is on sysData<127:0> and is latched by the 21071-BA chip. To prepare for the rest of the victim, bcDataA<4> is asserted.
6	The second octaword of the victim is received. The 21071-CA chip prepares to drive the bus so that sysTagOEEn and sysDataOEEn are deasserted.
7	The read of the victim is complete. The cache tags are driven by the 21071-CA chip with the tag information for the fill data (valid and dirty). sysDataWEEn and sysTagWE are asserted to write the cycle data tags.

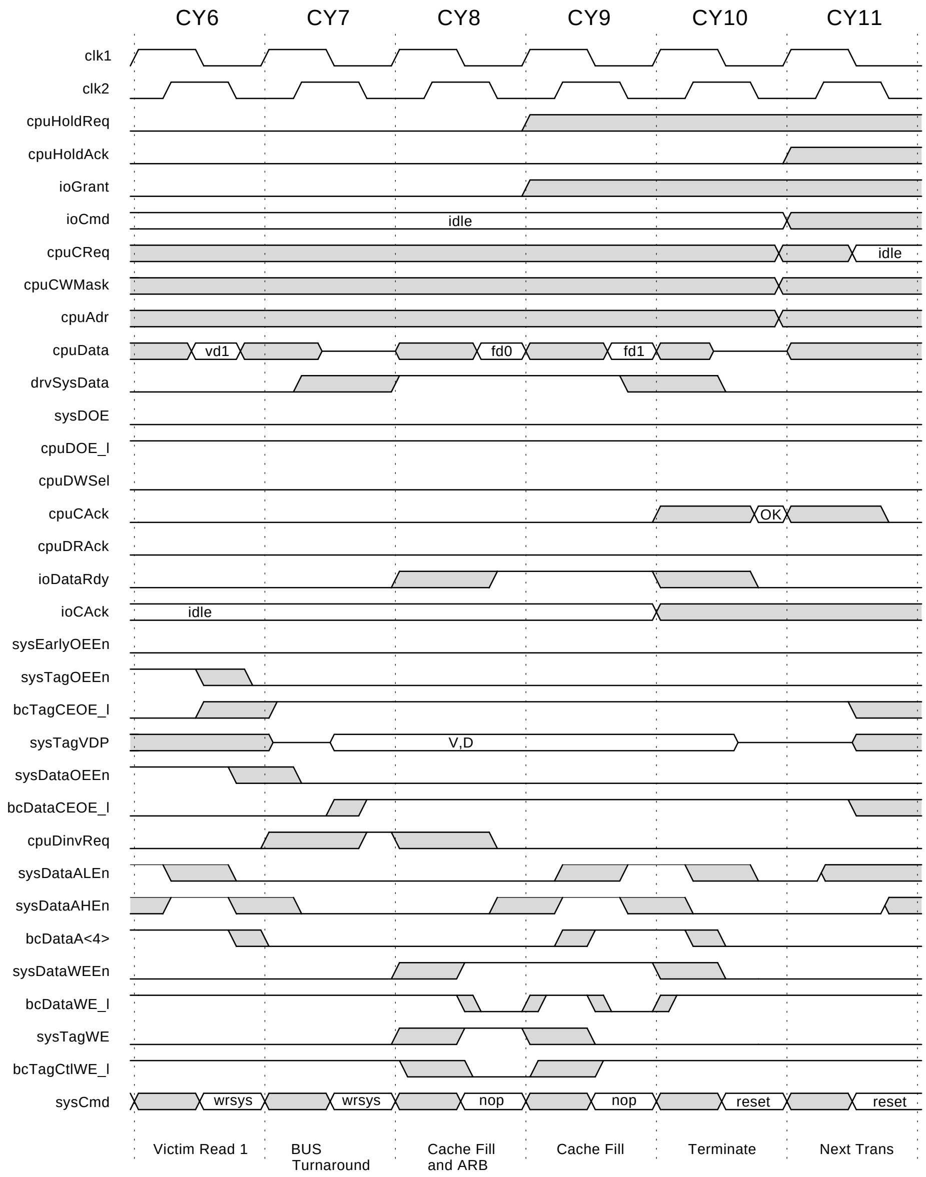
Cycle	Description
8	The fill data is ready and is driven on sysData<127:0>. If the CPU wrote a full cache line, the fill data is simply the same as the data written in cycle 2. Otherwise, the 21071-CA chip reads a line from memory and merges it with the write data to create an updated line of data. The CPU internal Dcache is invalidated using cpuDInvReq. To prepare to write the second octaword, bcDataA<4> will change on clk2R because write timing is being used.
9	The second octaword is written with sysDataWEEn. bcDataA<4> is deasserted after the write is done.
10	The cycle is acknowledged with cpuCAck<2:0>, and the cache controls are returned to their default state.
11	The transaction is complete, and the next transaction is ready to begin.

Figure 5–4 Timing of CPU Write Block, Cacheable, Allocate, Victim



Note:
ioRequest is not important during this transaction.

LJ-03140-TI0



Note:
ioRequest is not important during this transaction.

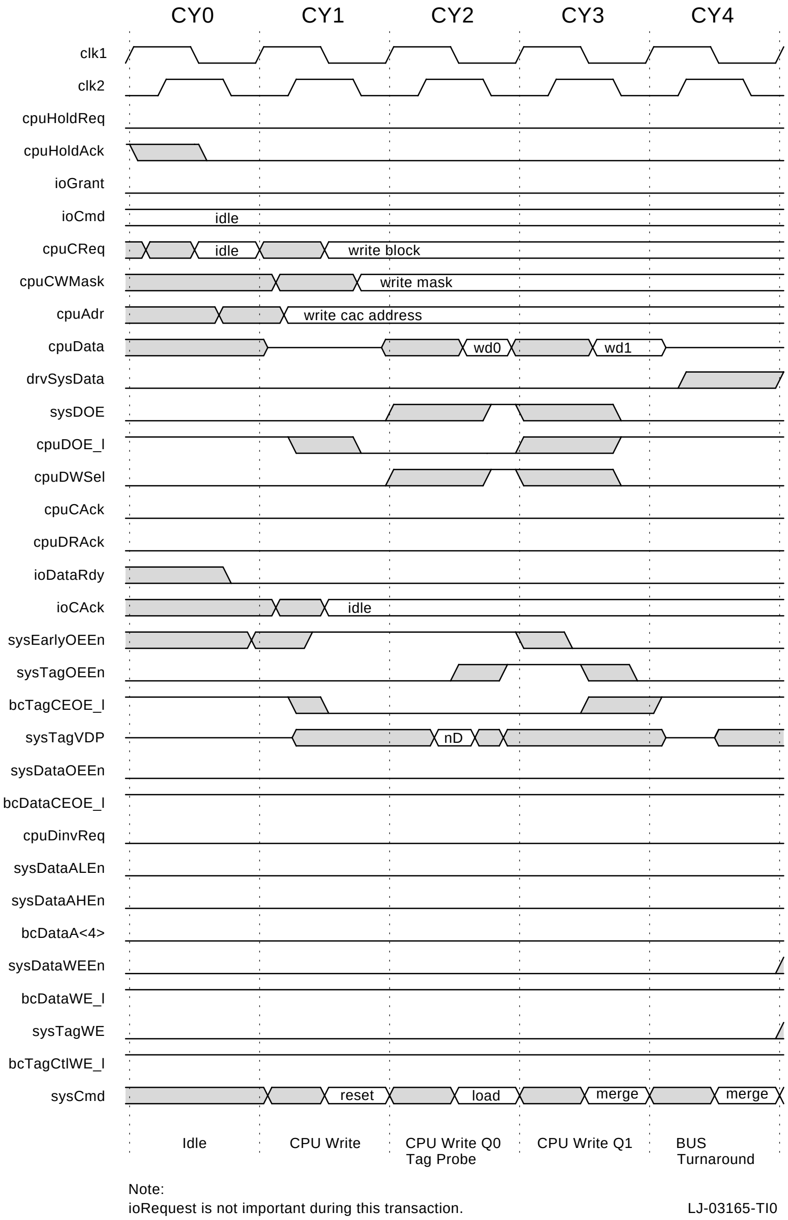
LJ-03141-TI0

5.1.1.3.2 Cacheable Allocate Without Victim

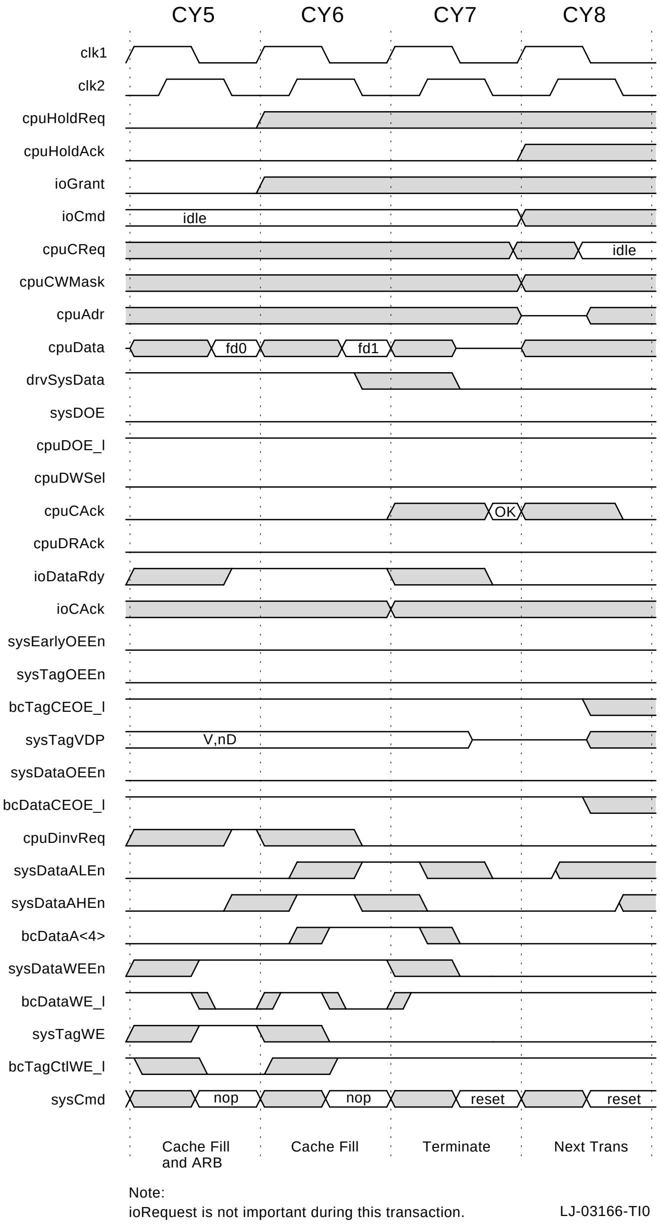
The following table describes the cycles for a CPU write block transaction in cacheable space without a victim and with write allocation enabled, as shown in Figure 5–5.

Cycle	Description
0	A write block begins during the idle cycle. The address is becoming valid due to the CPU doing a probe of the Bcache. Systems may rely on cacheable address being set up for the time it takes the CPU to do a probe (a minimum of 10 ns).
1	The CPU requests a write block with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcTagOE and cpuDOE_l.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in cacheable memory space. The CPU sees the assertion of cpuDOE_l; the first octaword of write data is placed on the cpuData bus and latched by the 21071-BA chip. The 21071-CA chip asserts sysDOE. The 21071-CA chip asserts sysTagOEEn to prevent the tag bus from floating. The 21071-CA chip also asserts cpuDWSel to get the second octaword of write data. The cache tag indicates no victim.
3	The CPU sees the assertion of cpuDWSel and places the second octaword of write data on the cpuData bus. The 21071-CA chip deasserts sysEarlyOEEn. The data is latched by the 21071-BA chip. The 21071-CA chip deasserts sysDOE and cpuDWSel. The 21071-CA chip prepares to drive the bus so that sysTagOEEn is deasserted.
4	The sysData bus is tristated by the CPU. The cache tags are driven by the 21071-CA chip with the tag information for the fill data (valid and dirty).
5	The fill data is ready and is driven on sysData<127:0>. If the CPU wrote a full cache line, the fill data is simply the same as the data written in cycle 2. Otherwise, the 21071-CA chip reads a line from memory and merges it with the write data to create an updated line of data. If the old cache line was valid, the CPU internal Dcache is invalidated using cpuDInvReq. sysDataWEEn and sysTagWE are asserted, in turn generating bcDataWE and bcTagWE, which write the data and tags into the cache. To prepare to write the second octaword bcDataA<4> is asserted.
6	The second octaword is written with sysDataWEEn.
7	The cycle is acknowledged with cpuCAck<2:0> and the data drivers are returned to their default state. cpuDOE_l is reasserted because the 21071-CA chip is finished with the data bus.
8	The transaction is complete, and the next transaction is ready to begin.

Figure 5–5 Timing of CPU Write Block, Cacheable, Allocate, No Victim



Note:
ioRequest is not important during this transaction.

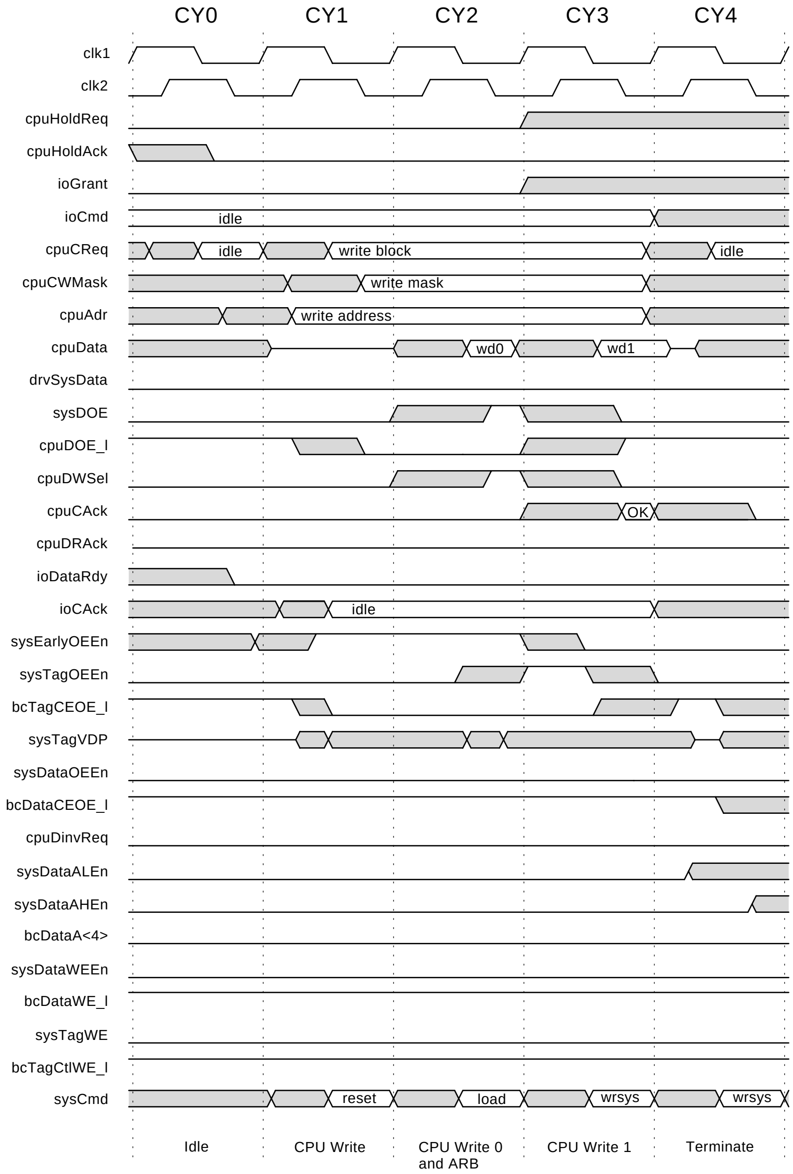


Note:
ioRequest is not important during this transaction.

5.1.1.3.3 Cacheable No Allocate The following table describes the cycles for a CPU write block transaction with write allocation disabled, as shown in Figure 5–6.

Cycle	Description
0	A write block begins during the idle cycle. The address is becoming valid. This transaction does not discriminate between cacheable and noncacheable, so the address is set up for only 4 ns.
1	The CPU requests a write block with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcTagOE and cpuDOE_l.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in cacheable memory space. The CPU sees the assertion of cpuDOE_l; the first octaword of write data is placed on the cpuData bus and is latched by the 21071-BA chip. The 21071-CA asserts sysDOE. The 21071-CA chip asserts sysTagOEEn to prevent the tag bus from floating. The 21071-CA chip also asserts cpuDWSel to get the second octaword of write data.
3	The CPU sees the assertion of cpuDWSel and places the second octaword of write data on the cpuData bus. The 21071-CA chip deasserts sysEarlyOEEn. The data is latched by the 21071-BA chip. The 21071-CA chip deasserts sysDOE and cpuDWSel. The cache is disabled by deasserting sysTagOEEn. The cycle is acknowledged with cpuCAck<2:0>.
4	The sysData bus is tristated by the CPU.

Figure 5–6 Timing of CPU Write Block, Noncacheable or No Allocate



Note:
ioRequest is not important during this transaction.

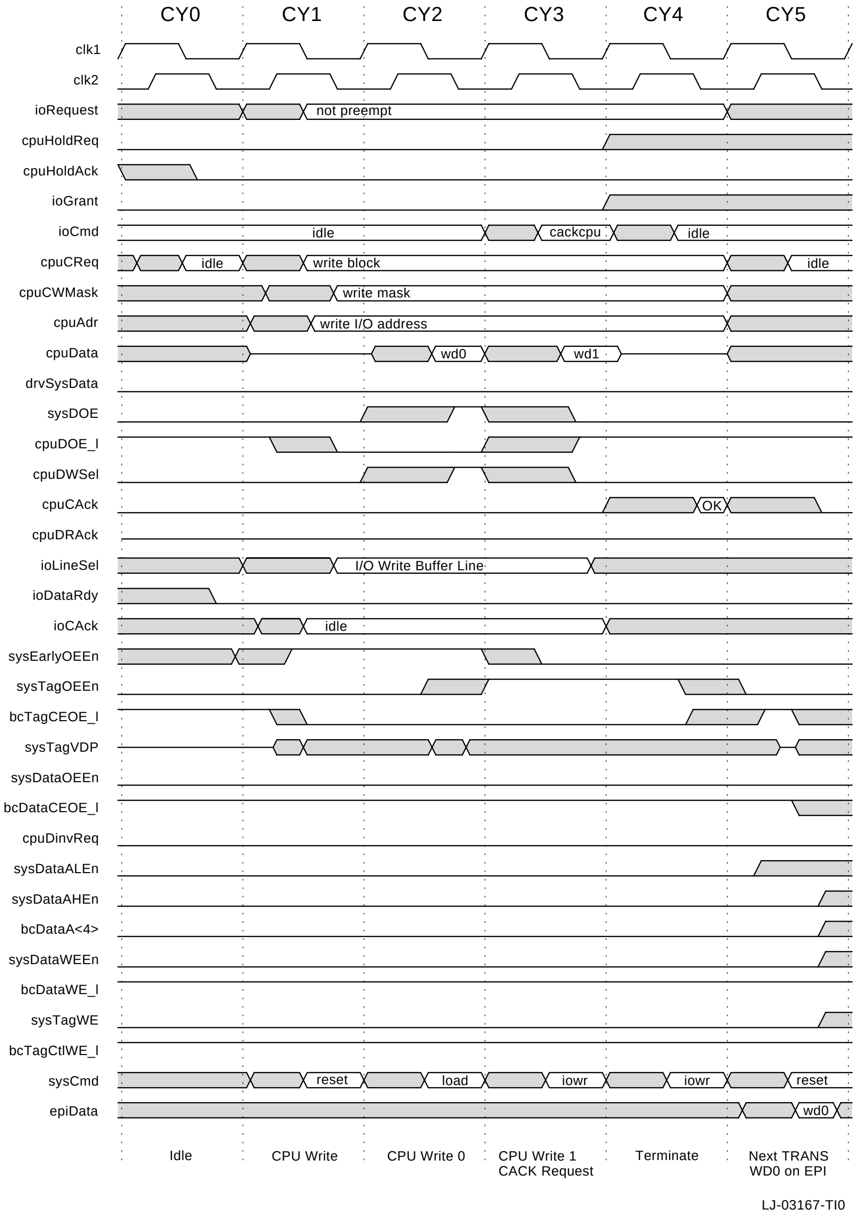
LJ-03170-TI0

5.1.1.3.4 Noncacheable A write block transaction to noncacheable space is identical to a write block with write allocation disabled. See Section 5.1.1.3.3 for a description of the transaction.

5.1.1.3.5 I/O Space The following table describes the cycles for a CPU write block transaction in remote I/O space, as shown in Figure 5–7.

Cycle	Description
0	During the entire time that the CPU has ownership of the bus, the 21071-DA chip, using ioLineSel<1:0>, provides a pointer to a free cache line buffer in the DMA read and I/O write buffer.
1	The CPU requests a write block with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcTagOE and cpuDOE_l.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in cacheable memory space. The CPU sees the assertion of cpuDOE_l, and the first octaword of write data is placed on the cpuData bus. The 21071-BA chip loads the data into the DMA read and I/O write buffer at the line selected by ioLineSel<1:0>. The 21071-CA chip asserts sysDOE. The 21071-CA chip asserts sysTagOEEn to prevent the tag bus from floating. The 21071-CA chip also asserts cpuDWSel to get the second octaword of write data.
3	The CPU sees the assertion of cpuDWSel and places the second octaword of write data on the cpuData bus. The data is latched by the 21071-BA chip. The 21071-CA chip deasserts sysDOE and cpuDWSel. The 21071-DA chip, using ioCmd<2:0>, is ready to end the transaction in the next cycle, so cpuCAck is requested.
4	The 21071-CA chip receives ioCmd<2:0> and acknowledges the cycle with cpuCAck<2:0>. The cache is turned off by deasserting sysTagOEEn. The 21071-DA chip is free to unload the data using the epiBus. If the 21071-DA chip did not request a cpuCAck by this cycle, then sysDataOEEn will be asserted to prevent sysData<127:0> from floating.
5	The transaction is complete, and the next transaction is ready to begin.

Figure 5–7 Timing of CPU Write Block, Remote I/O Space



5.1.1.4 LD*x*_L

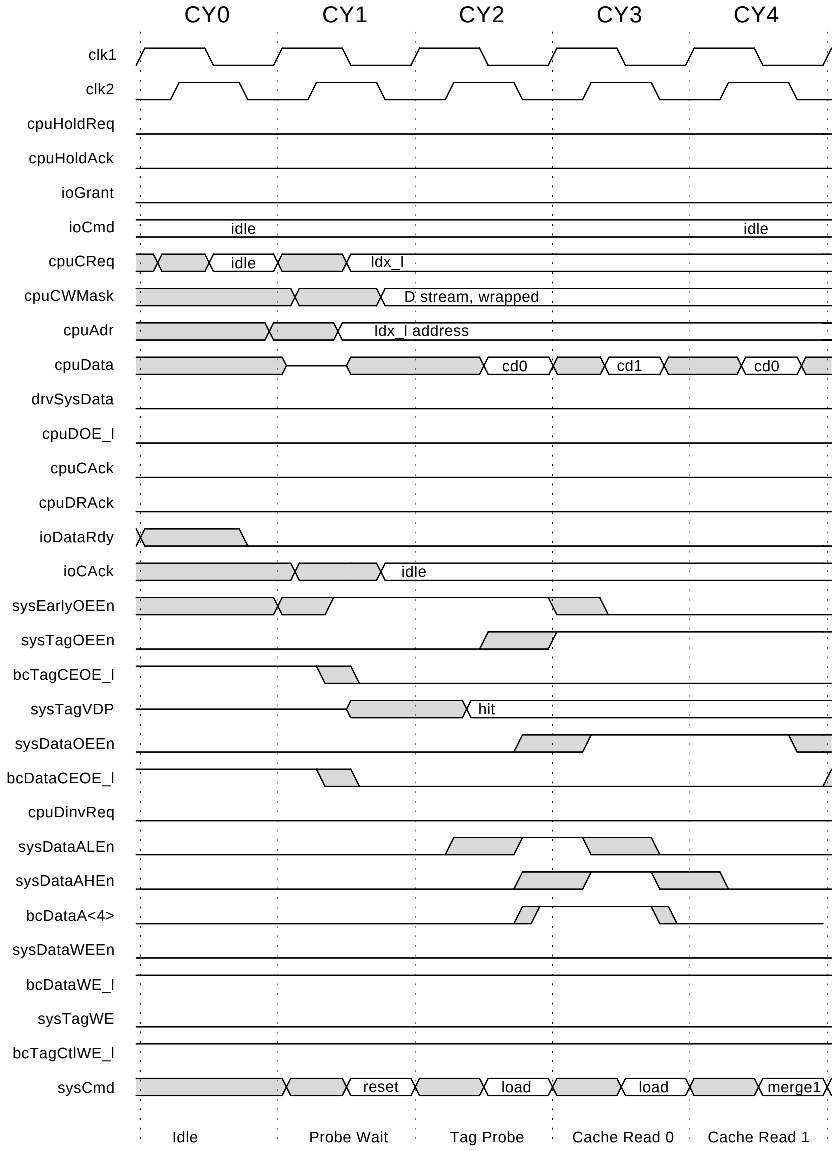
In general an LD*x*_L transaction looks like a read block. There are two major differences. The first is that the architecturally defined lock bit and lock address are set. The second is that in contrast to the read block transaction, the cache must be probed. (The Alpha 21064 microprocessor does not probe on LD*x*_L or ST*x*_C.)

5.1.1.4.1 Cacheable Hit Figure 5–8 shows an LD*x*_L transaction in cacheable space that hits. Data is not returned directly from the cache, to avoid an address-to-data race through the cache RAMs. Although the CPU should not issue one, a read block that hits in the cache will be treated as an LD*x*_L hit without the lock bit being set.

The following table describes the cycles for an LD*x*_L transaction in cacheable space that hits, as shown in Figure 5–8.

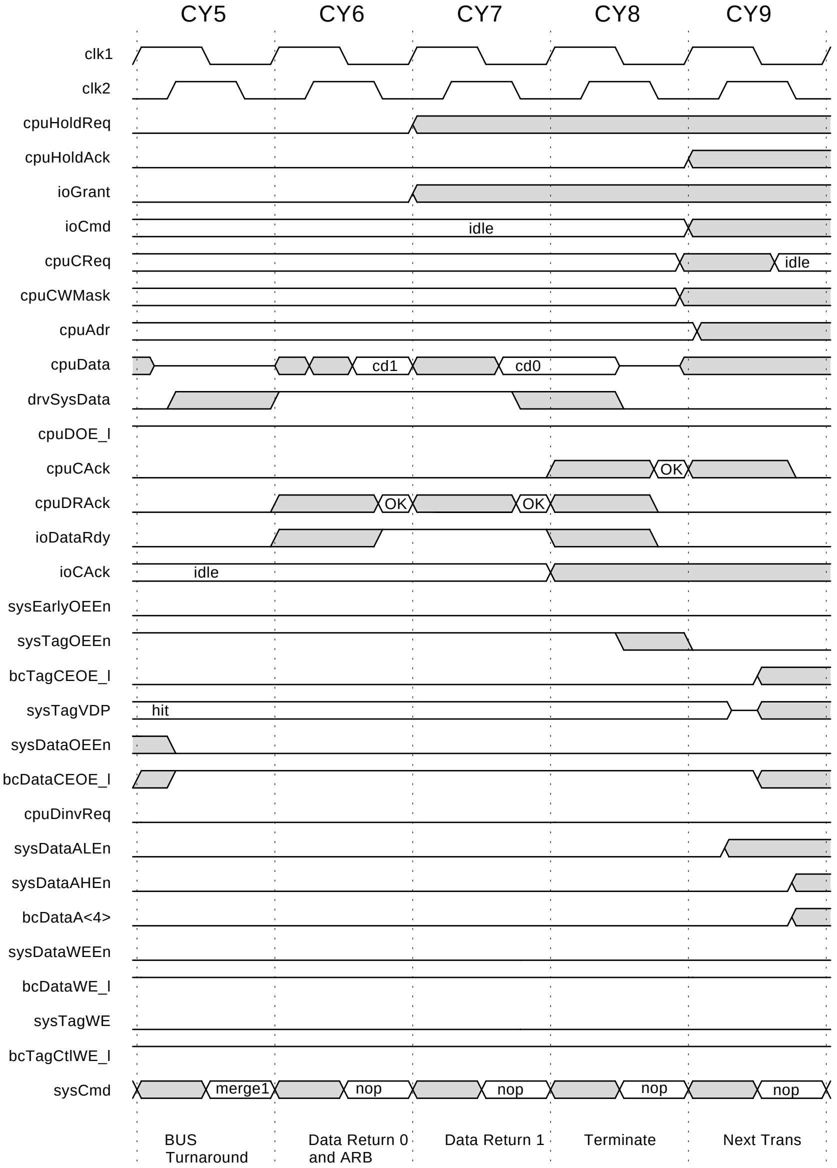
Cycle	Description
0	An LD*x*_L begins during the idle cycle. The address is becoming valid one CPU cycle before clk1F, because the CPU did not probe the cache.
1	The CPU requests an LD*x*_L with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcDataOE and bcTagOE. Wrapped return data is requested by asserting cpuCWMask<1>.
2	The LD*x*_L locked bit is set, and the LD*x*_L locked address is loaded from sysAdr<33:5>. If the 21071-DA chip is sending ClrLock on ioCmd<2:0>, then the lock bit is not set, and it is forced to remain clear for as long as the ClrLock is being sent. The cache tag indicates a hit. SysDataAEn is asserted as the data must be returned in wrapped order. If the cache line is clean, data will be wrapped from the memory, as in a regular wrapped read operation.
3	Data from the cache is loaded into the 21071-BA chip merge buffer. To prepare to read the first octaword (since it is wrapped), bcDataA<4> is deasserted. The 21071-CA chip prepares to drive the bus so that sysEarlyOEEn is deasserted.
4	The second octaword is loaded into the 21071-BA chip merge buffer.
5	The 21071-CA chip waits for the cache data to tristate.
6	The merge buffer data is driven on sysData<127:0> and acknowledged with cpuDRack<2:0>.
7	The second octaword is driven and acknowledged.
8	The cycle is acknowledged with cpuCAck<2:0> and the data drivers are returned to their state.
9	The transaction is complete, and the next transaction is ready to begin.

Figure 5–8 Timing of CPU LD*x*_L, Wrapped, Cacheable Hit



Note:
ioRequest, sysDOE, and cpuDWSel are not important during this transaction.

LJ-03138-TI0



Note:
ioRequest, sysDOE, and cpuDWSel are not important during this transaction.

LJ-03139-TI0

5.1.1.4.2 Cacheable Miss An LD*x*_L transaction that misses in cacheable space is similar to Figure 5–1. Also see the description in Section 5.1.1.2.2.

5.1.1.4.3 Noncacheable An LD*x*_L transaction to noncacheable space is identical to a read block to noncacheable space (Section 5.1.1.2.3), except that the lock bit and lock address must be set.

5.1.1.4.4 I/O Space An LD*x*_L transaction to I/O space is treated by the 21071-CA chip as a regular read to I/O space (although the lock bit is set). An implementation may treat the LD*x*_L as a regular read block in I/O space, flag an error, or implement the LD*x*_L.

5.1.1.5 ST*x*_C

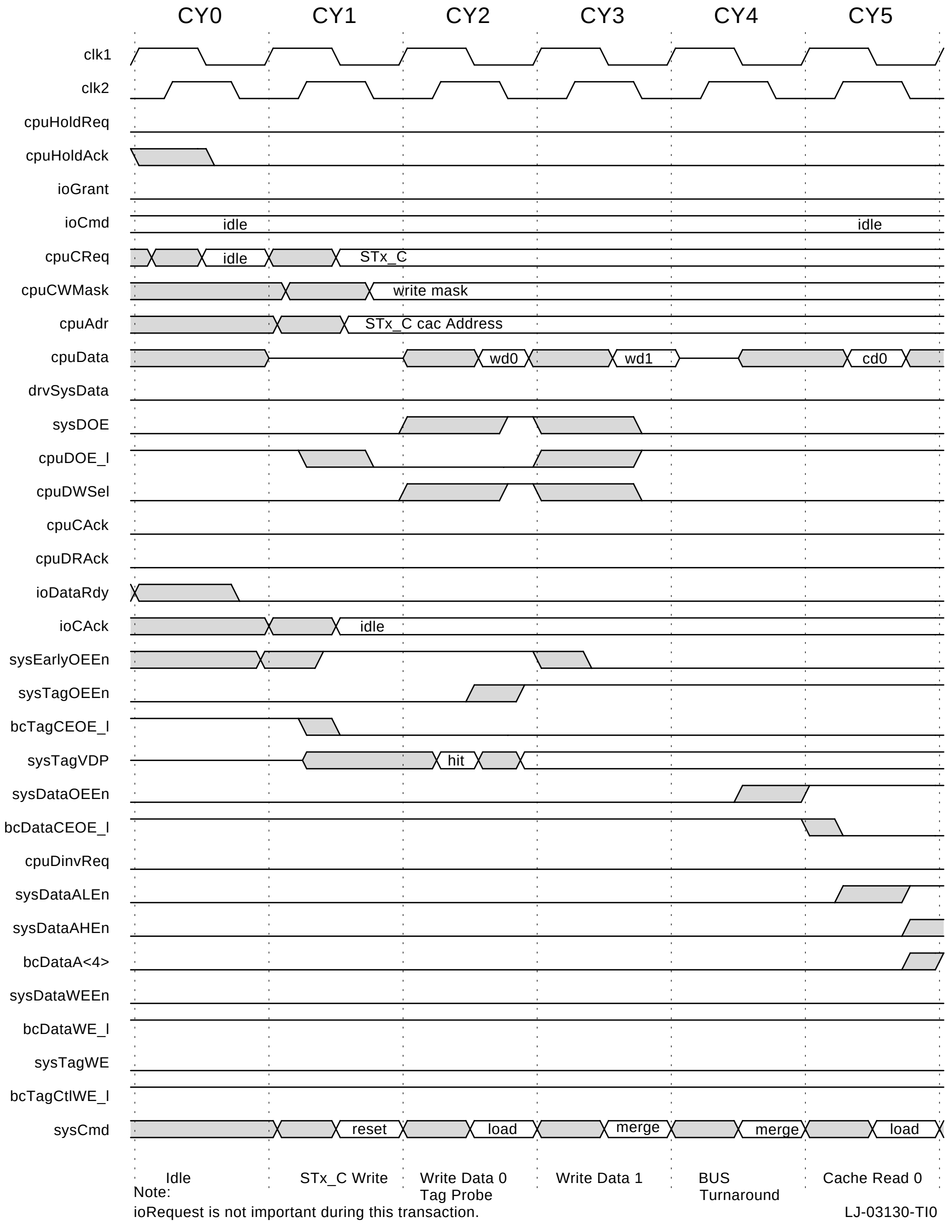
In general, an ST*x*_C transaction looks like a write block. Also, the transaction may be aborted by the lock bit being cleared. The 21071-DA chip may ensure that ST*x*_C to memory always fails by using the ClrLock command on ioCmd<2:0>. For ClrLock to affect a CPU ST*x*_C transaction, the ClrLock command must be asserted in or before the first cycle of the ST*x*C transaction flow. For example, a DMA read miss transaction that needs to clear the lock flag must do so before one cycle after the ioCAck<1:0> for the DMA read. This is because an ST*x*_C transaction may potentially start in the cycle after ioCAck<1:0>.

5.1.1.5.1 Cacheable Hit The following table describes the cycles for an ST*x*_C transaction to cacheable space that hits in the cache, as shown in Figure 5–9.

Cycle	Description
0	An ST*x*_C begins during the idle cycle. An address is placed on the bus one CPU cycle before clk1F.
1	The CPU requests an ST*x*_C with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcTagOE and cpuDOE_l.
2	The CPU sees the assertion of cpuDOE_l; the first octaword of write data is placed on the cpuData bus and is latched by the 21071-BA chip. The 21071-CA chip recognizes the transaction and tests the LD*x*_L lock bit, which is set (success). The cache tag indicates a cache hit. The 21071-CA chip asserts sysDOE. The 21071-CA chip asserts sysTagOEEn to prevent the tag bus from floating. The 21071-CA chip also asserts cpuDWSel to get the second octaword of write data.
3	The CPU sees the assertion of cpuDWSel and places the second octaword of write data on the cpuData bus. The data is latched. The 21071-CA chip deasserts sysDOE, cpuDWSel, and sysEarlyOEEn.

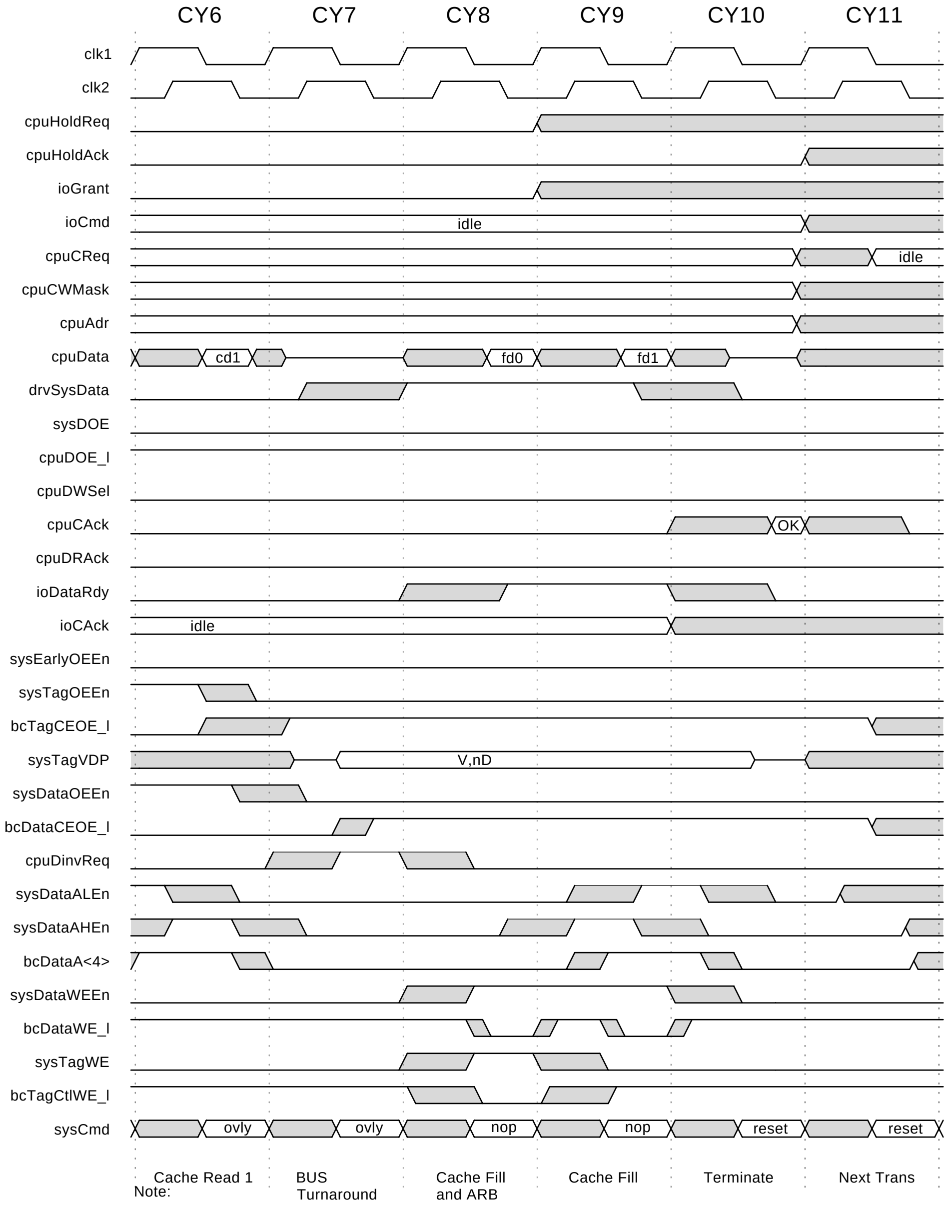
Cycle	Description
4	The sysData bus is tristated by the CPU. The 21071-CA chip asserts sysDataOEEn, causing the cache to begin driving the data bus.
5	The first octaword of cache data is on sysData<127:0> and is latched by the 21071-BA chip. To prepare for the rest of the data, bcDataA<4> is asserted.
6	The second octaword of cache data is received. The 21071-CA chip prepares to drive the bus by deasserting sysEarlyOEEn and sysDataOEEn.
7	The cache read is complete. The cache tags are driven by the 21071-CA chip with the tag information for the fill data (valid and dirty).
8	The fill data is ready and is driven on sysData<127:0>. The fill data is a merge of the data read from the cache, overlaid by the quadword or longword written by the ST*x*_C. The CPU internal Dcache is not invalidated, as the CPU handles this case itself. sysDataWEEn and sysTagWE are asserted, in turn generating bcDataWE and bcTagWE, which writes the data and tags into the cache. To prepare to write the second octaword, bcDataA<4> is asserted.
9	The second octaword is written with sysDataWEEn.
10	The cycle is acknowledged with cpuCAck<2:0>, and the data drivers are returned to their default state.
11	The transaction is complete, and the next transaction is ready to begin.

Figure 5–9 Timing of CPU ST*x*_C Succeeds, Hit, Cacheable, Allocate



Note:
ioRequest is not important during this transaction.

LJ-03130-TI0



Note:
ioRequest is not important during this transaction.

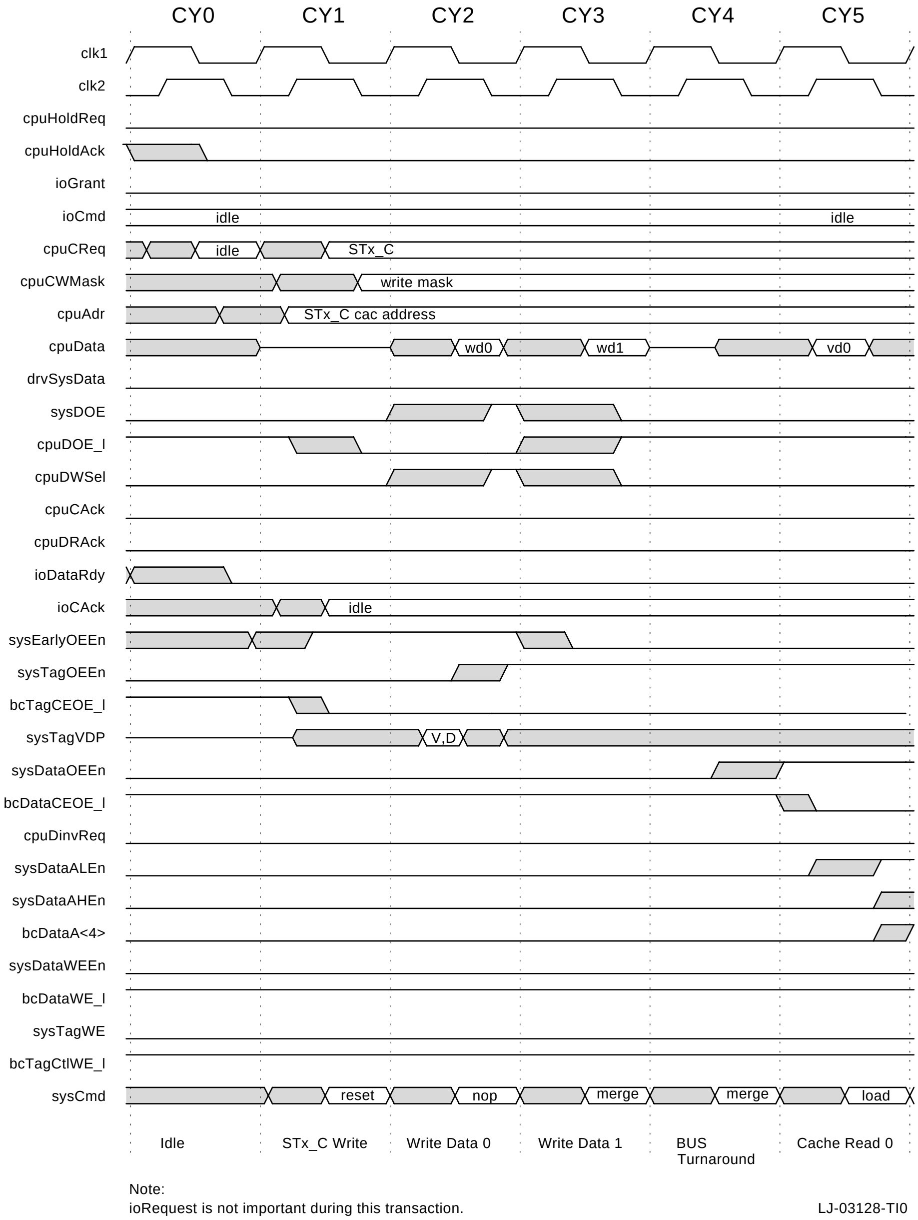
LJ-03131-TI0

5.1.1.5.2 Cacheable Miss Figure 5–10 shows an ST*x*_C transaction to cacheable space that misses the cache. The figure shows only a write allocation and victim. See Figure 5–5 and Figure 5–6 for examples of no victim and no write allocation.

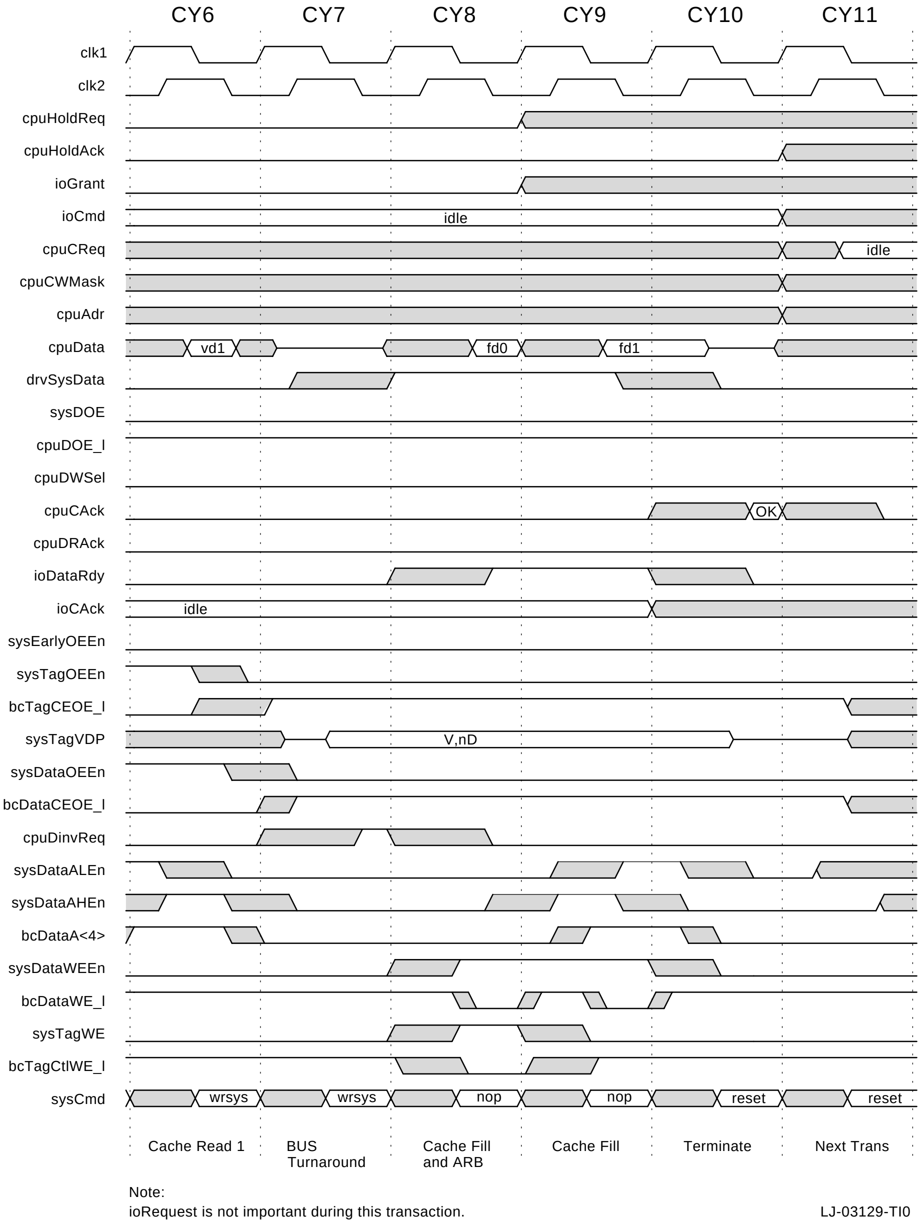
The following table describes the cycles for a CPU ST*x*_C transaction to cacheable space that hits in the cache, as shown in Figure 5–10.

Cycle	Description
0	An ST*x*_C begins during the idle cycle. An address is placed on the bus one CPU cycle before clk1F.
1	The CPU requests an ST*x*_C with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcTagOE and cpuDOE_l.
2	The CPU sees the assertion of cpuDOE_l; the first octaword of write data is placed on the cpuData bus and is latched by the 21071-BA. The 21071-CA chip recognizes the transaction and tests the LD*x*_L lock bit, which is set (success). The cache tag indicates a cache miss. This cycle and the remaining cycles of the ST*x*_C miss transaction are the same as write block cycles 2 and onward of Sections 5.1.1.3.1 (victim), 5.1.1.3.2 (no victim), and 5.1.1.3.3 (write allocation disabled).

Figure 5–10 Timing of CPU ST*x*_C Succeeds, Miss, Cacheable, Allocate, Victim



Note:
ioRequest is not important during this transaction.



Note:
ioRequest is not important during this transaction.

5.1.1.5.3 Noncacheable The following table describes the cycles for a CPU ST*x*_C transaction to noncacheable space. This transaction looks the same as the noncacheable write block transaction shown in Figure 5–6.

Cycle	Description
0	An ST*x*_C begins during the idle cycle. An address is placed on the bus one CPU cycle before clk1F.
1	The CPU requests an ST*x*_C with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcTagOE and cpuDOE_l.
2	The 21071-CA chip recognizes the transaction and tests the LD*x*_L lock bit, which is set (success). It decodes sysAdr<33:5> and finds it in cacheable memory space. This cycle and the remaining cycles of the ST*x*_C noncacheable transaction are the same as noncacheable write block cycles 2 and onward, which are described in Section 5.1.1.3.3.

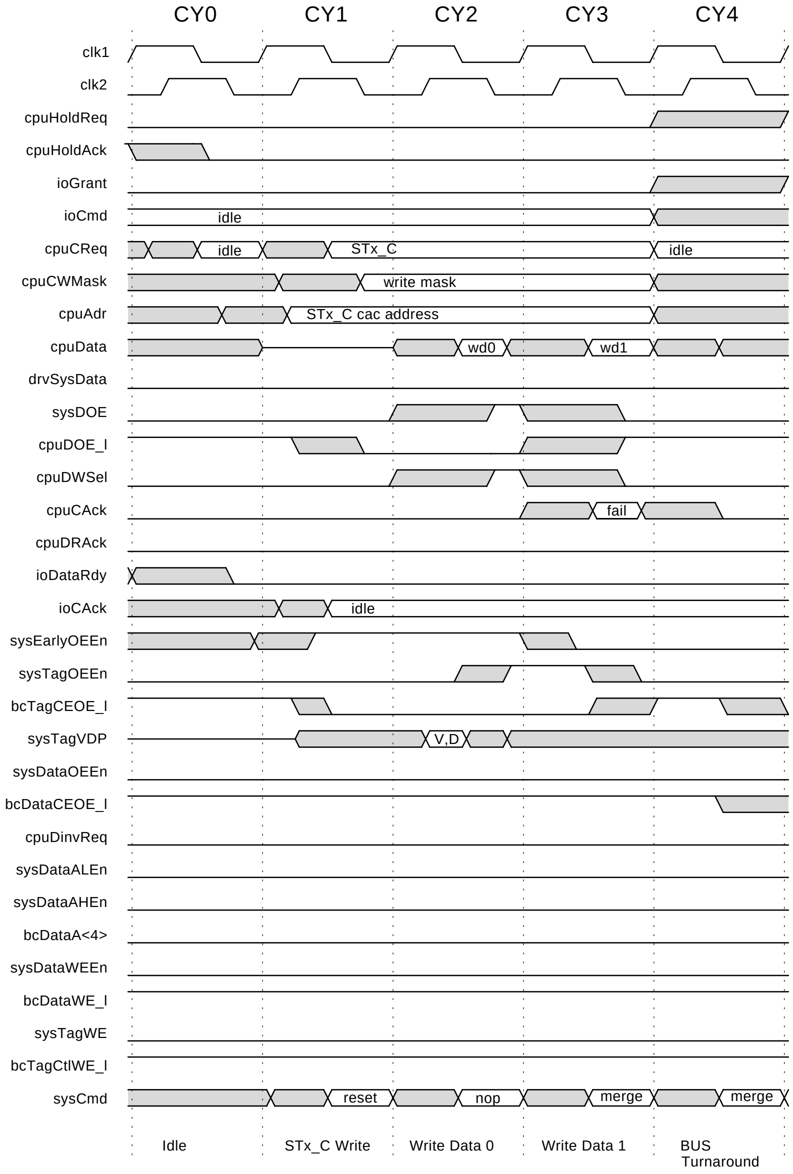
5.1.1.5.4 I/O Space Similar to LD*x*_L, an ST*x*_C transaction to I/O space is treated by the 21071-CA chip as a write block to I/O space. An implementation may perform the ST*x*_C or flag an error.

5.1.1.5.5 Fail If the LD*x*_L lock bit is not set, or if the 21071-DA chip is sending ClrLock on ioCmd<2:0> (forcing the lock bit to remain clear), an ST*x*_C instruction will fail.

The following table describes the cycles for a CPU ST*x*_C fail transaction, as shown in Figure 5–11.

Cycle	Description
0	An ST*x*_C begins during the idle cycle. An address is placed on the bus one CPU cycle before clk1F, as the CPU did not probe the cache.
1	The CPU requests the transaction with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers the assertion of bcTagOE and cpuDOE_l.
2	The first octaword of data is received. The 21071-CA chip asserts sysDOE, cpuDWSel, and sysTagOEEn.
3	The 21071-CA chip recognizes the transaction and tests the LD*x*_L lock bit, which is clear (fail). The latched write data is discarded. The 21071-CA chip deasserts sysEarlyOEEn and acknowledges the cycle with cpuCAck<2:0>.
4	The transaction is complete, and the next transaction is ready to begin.

Figure 5–11 Timing of CPU ST*x*_C Fails



Note:
ioRequest is not important during this transaction.

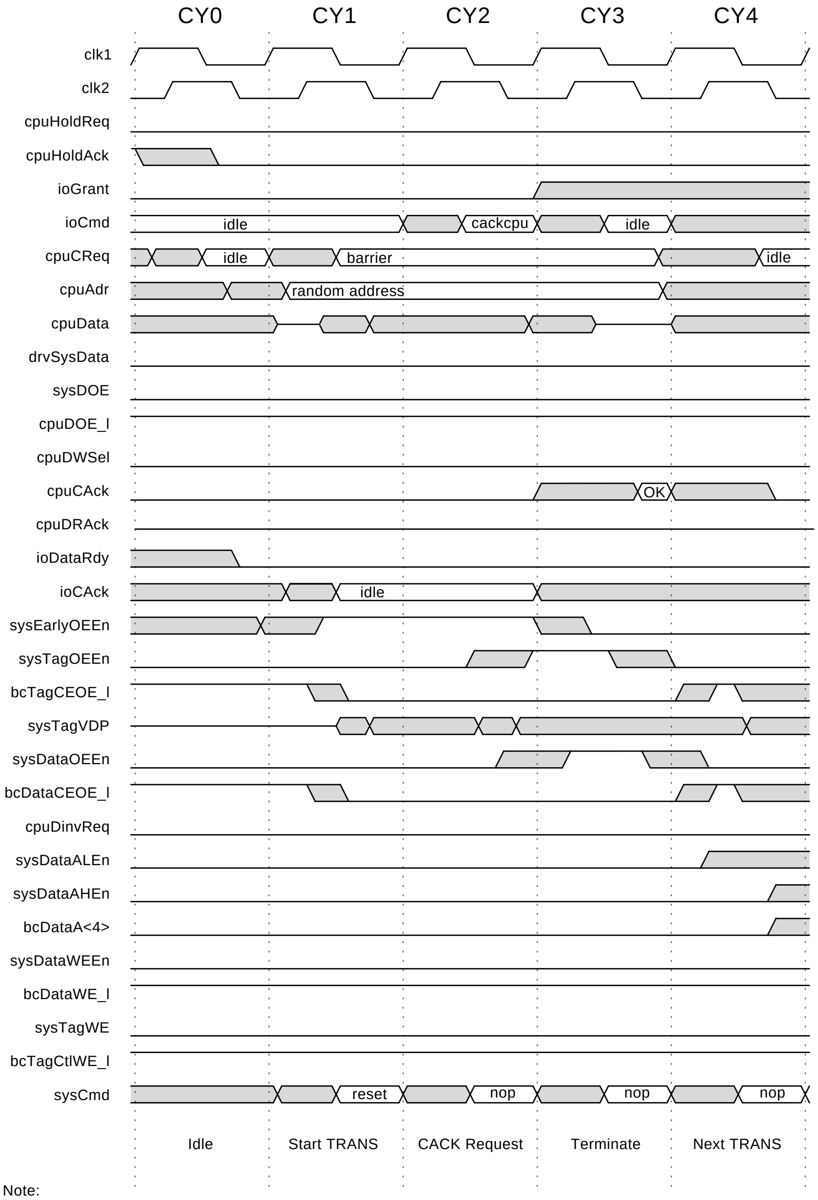
LJ-03720-TI0

5.1.1.6 Barrier

The following table describes the cycles for a memory barrier transaction, as shown in Figure 5–12.

Cycle	Description
0	A barrier begins during the idle cycle. An address is placed on the bus one CPU cycle before clk1F, but is ignored.
1	The CPU requests the transaction with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers bcDataOE and bcTagOE to turn on. (This is done to avoid having the data and tag buses float, because the CPU does not drive the data or tags during these transactions.)
2	The 21071-DA chip recognizes the transaction and requests that an OK be sent using cpuCAck<2:0> in the next cycle. The 21071-DA chip could also preempt the barrier at this point. The 21071-CA chip asserts sysDataOEEn and sysTagOEEn.
3	The 21071-CA chip receives the request on ioCmd<2:0>, deasserts sysTagOEEn, sysDataOEEn, and sysEarlyOEEn, and acknowledges the cycle with cpuCAck<2:0>.
4	The transaction is complete, and the next transaction is ready to begin.

Figure 5–12 Timing of CPU Barrier or Fetch or FetchM



Note:
ioRequest and cpuCWMask are not important during this transaction.

LJ-03144-TI0

5.1.1.7 Fetch, FetchM

These CPU transactions, similar to those shown in Figure 5–12, may be supported as desired by a particular implementation. The simplest implementation looks like an ST*x*_C fail.

The following table describes the cycles for a fetch or fetchM transaction.

Cycle	Description
0	A fetch or fetchM begins during the idle cycle. An address is placed on the bus one CPU cycle before clk1F, but is ignored.
1	The CPU requests the transaction with cpuCReq<2:0>. Because sysEarlyOEEn was asserted, this triggers bcDataOE and bcTagOE to turn on. (This is done to avoid having the data and tag buses float, because the CPU does not drive the data or tags during these transactions.)
2	A wait state is performed.
3	The 21071-CA chip recognizes the transaction, deasserts sysEarlyOEEn, and acknowledges the cycle with cpuCAck<2:0>.

5.1.2 DMA Transactions

After DMA wins arbitration, it may initiate a transaction with the 21071-CA chip. Unlike the CPU transactions, the only unit of transfer for DMA transactions is a cache line.

5.1.2.1 DMA Idle

When DMA has the bus, the CPU is isolated by holding cpuDWSel, and sysEarlyOEEn is deasserted. The cache is prepared for a probe by the 21071-CA chip asserting sysDataOEEn and sysTagOEEn in the first cycle that a DMA transaction may begin. The cache also drives the data bus in case a DMA read or write hits the cache.

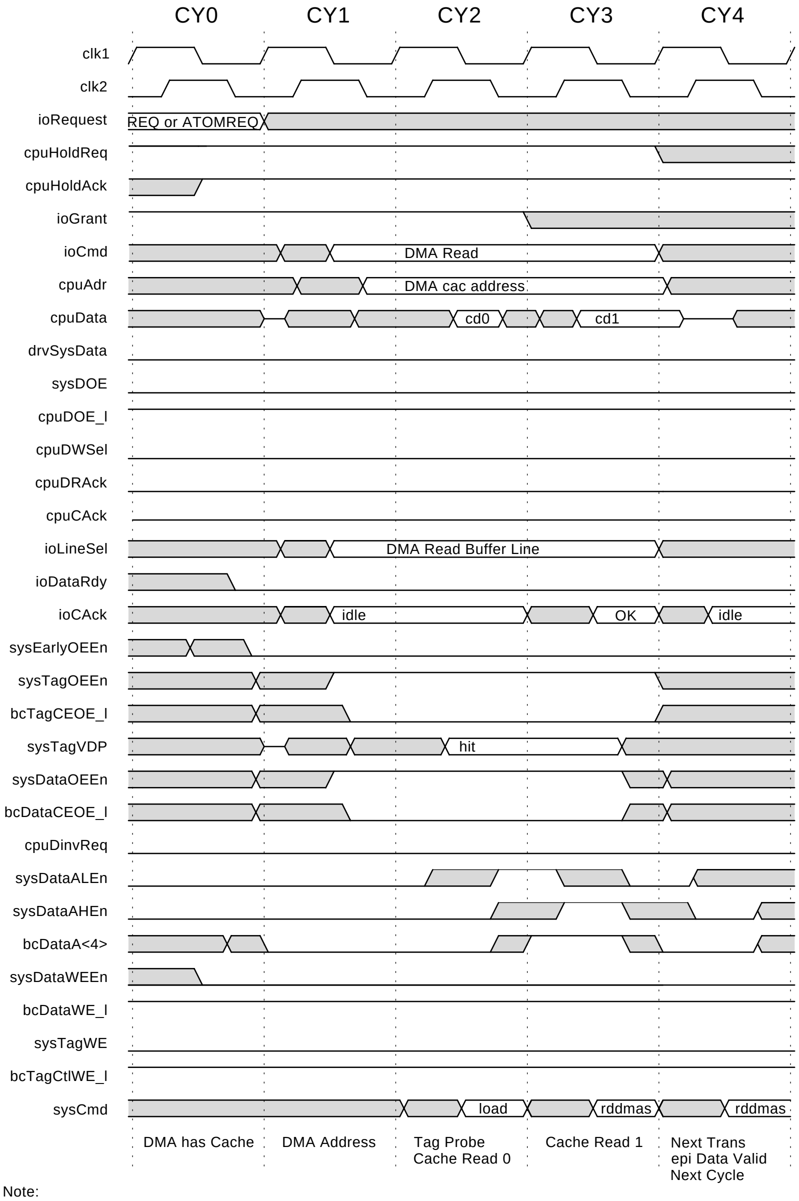
5.1.2.2 DMA Read

This section describes the DMA read transactions.

5.1.2.2.1 Cacheable Hit The following table describes the cycles for a DMA read transaction in cacheable space that hits, as shown in Figure 5–13.

Cycle	Description
0	The transaction begins with DMA owning the bus, as indicated by the assertion of ioGrant.
1	The 21071-DA chip requests a DMA read with ioCmd<2:0>, places the address on sysAdr<33:5>, and points to a line to be loaded in the DMA read and I/O write buffer with ioLineSel<1:0>.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in cacheable memory space. The 21071-CA chip waits for the cache probe, which indicates a cache hit. The first octaword of data is already on the data bus, so it is loaded into the DMA read buffer. (If the read was wrapped, the data would be invalid and would have to be loaded in the next cycle.) To prepare for reading the second octaword, bcDataA<4> is asserted.
3	The 21071-CA chip loads the second octaword of read data into the DMA read buffer and indicates data ready with ioDataRdy. The transaction is acknowledged with ioCAck<1:0>. If the 21071-DA chip won arbitration, it may start a new read transaction in the next cycle. If the CPU won arbitration, this cycle is used for bus turnaround.
4	The transaction is complete, and the next transaction is ready to begin.

Figure 5–13 Timing of DMA Read, Cacheable, Hit



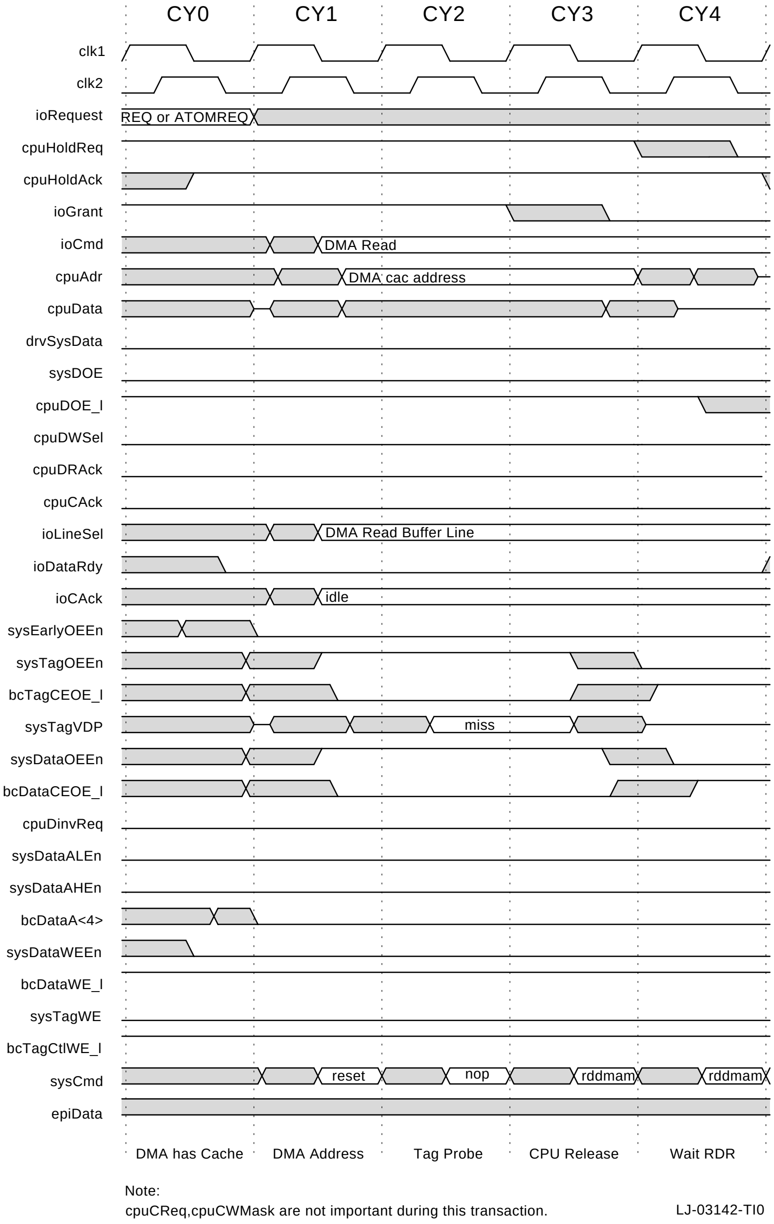
Note:
cpuCReq and cpuCWMask are not important during this transaction.

LJ-03147-TI0

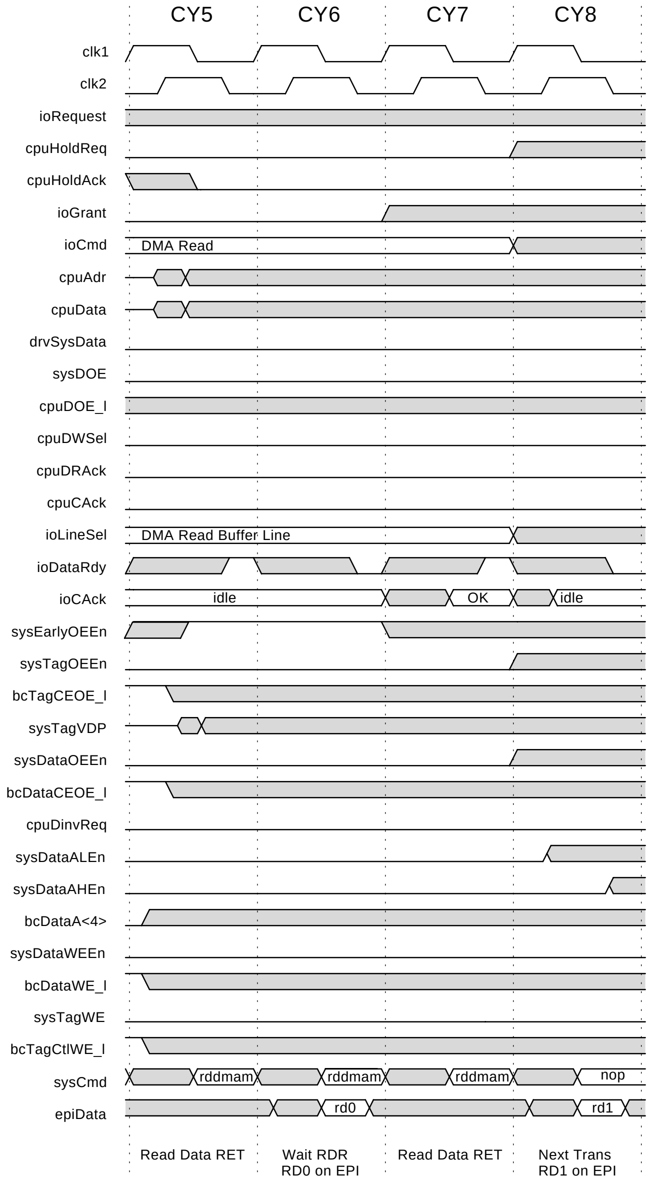
5.1.2.2.2 Cacheable Miss The following table describes the cycles for a DMA read transaction in cacheable space that misses, as shown in Figure 5–14.

Cycle	Description
0	The transaction begins with DMA owning the bus, as indicated by the assertion of ioGrant.
1	The 21071-DA chip requests a DMA read with ioCmd<2:0>, places the address on sysAdr<33:5>, and points to a line to be loaded in the DMA read and I/O write buffer with ioLineSel<1:0>.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in cacheable memory space. Also, the cache tag, available this cycle, indicates a cache miss.
3	The read data could be returned to the 21071-DA chip in this cycle, although it is shown to take until cycle 5. If the arbitration allows a release, and the 21071-CA chip is not in the middle of a preemption, the CPU may be released to use the cache. If so, the 21071-CA chip deasserts cpuHoldReq, sysTagOEEn, and sysDataOEEn. Section 5.1.3.2.4 describes returning from a released CPU to a DMA transaction.
4	The 21071-CA chip waits for read data to return. SysEarlyOEEn is asserted so that if the CPU starts an external transaction, the tag and sysData buses will not float.
5	The first octaword of read data is loaded into the DMA read and I/O write buffer. The 21071-CA chip indicates the transfer by asserting ioDataRdy.
6	The 21071-CA chip waits for the second quadword of read data to return.
7	The second octaword is loaded into the DMA read and I/O write buffer and is acknowledged with ioDataRdy. The transaction is acknowledged with ioCAck<1:0>.
8	The transaction is complete, and the next transaction is ready to begin.

Figure 5–14 Timing of DMA Read, Cacheable, Miss



Note:
cpuCReq,cpuCWMask are not important during this transaction.



Note:
cpuCReq and cpuCWMask are not important during this transaction.

LJ-03143-TI0

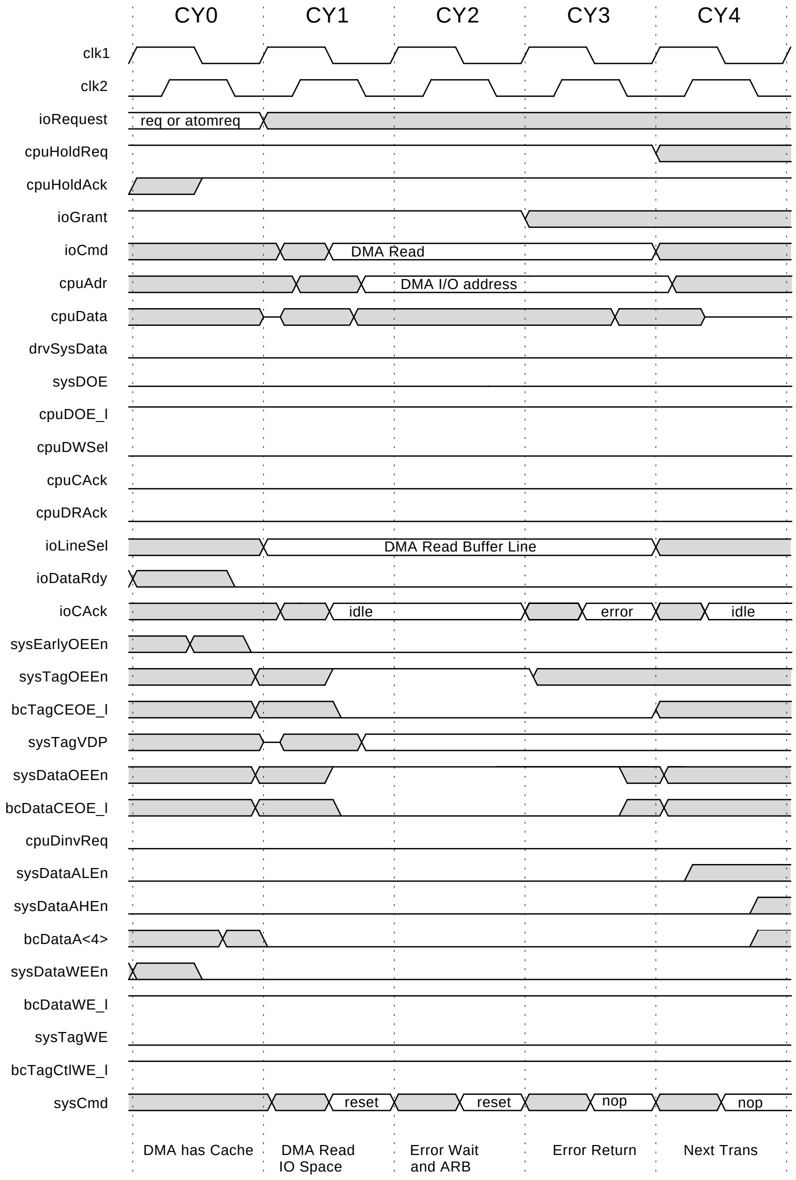
5.1.2.2.3 Noncacheable A DMA read transaction to noncacheable space is similar to the cacheable miss shown in Figure 5–14. Due to internal timing issues, the probe cycle still exists, but the probe results are ignored.

5.1.2.2.4 I/O Space DMA transactions are not supported to I/O space; they should be responded to as an error using ioCAck<1:0> (Figure 5–15).

The following table describes the cycles for a DMA read, I/O space transaction, as shown in Figure 5–15.

Cycle	Description
0	The transaction begins with DMA owning the bus, as indicated by the assertion of ioGrant.
1	The 21071-DA chip requests a DMA read with ioCmd<2:0>, places the address on sysAdr<33:5>, and points to a line to be loaded in the DMA read and I/O write buffer with ioLineSel<1:0>.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it is in I/O space. The 21071-CA chip turns on its sysData drivers for this one cycle to prevent a floating bus.
3	The cycle is acknowledged as an error with ioCAck<1:0>.
4	The transaction is complete, and the next transaction is ready to begin.

Figure 5–15 Timing of DMA Read, I/O Space (Error)



Note:
cpuCReq and cpuCWMask are not important during this transaction.

LJ-03148-TI0

5.1.2.3 DMA Read Wrapped

The transaction for DMA read wrapped is the same as that of DMA read. The return read data is returned with octaword 1, followed by octaword 0. This is done by asserting sysDataAEn for the first octaword and deasserting it for the second.

5.1.2.4 DMA Read Burst

The transaction for DMA read burst is the same as that of DMA read, except that the transaction includes a hint that the next transaction will be to the next cache line address.

5.1.2.5 DMA Read Wrapped Burst

The transaction for DMA read wrapped burst is the same as that of DMA read, except that it contains the next line hint in DMA read burst and includes the wrapping in DMA read wrapped.

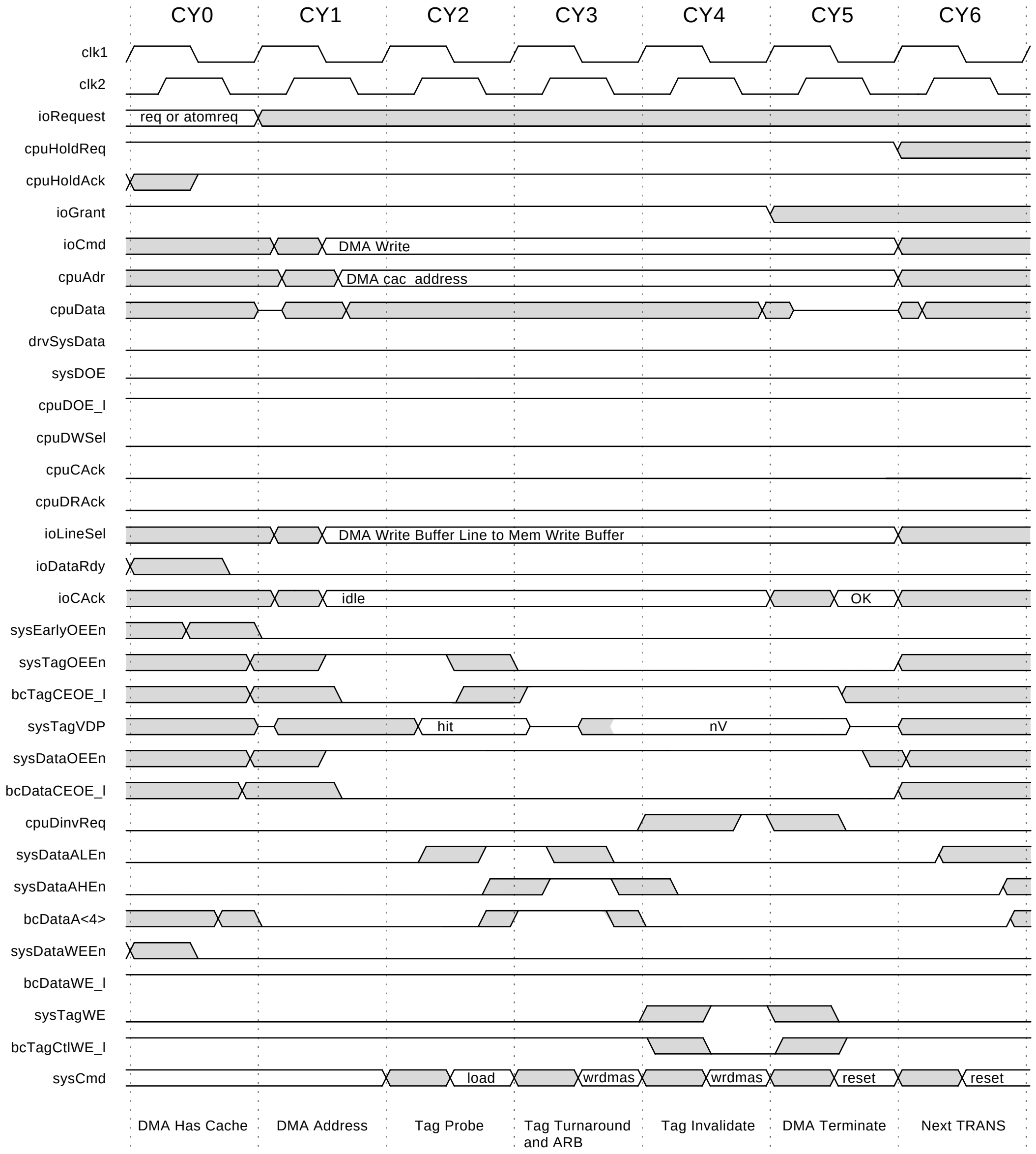
5.1.2.6 DMA Write

A DMA write releases the cache when the memory write buffer is full and the write does not hit in the cache.

5.1.2.6.1 Cacheable Hit The following table describes the cycles for a DMA write transaction in cacheable space that hits, as shown in Figure 5–16. The cache is invalidated rather than updated.

Cycle	Description
0	The transaction begins with DMA owning the bus, as indicated by the assertion of ioGrant.
1	The 21071-DA chip requests a DMA write with ioCmd<2:0>, places the address on sysAdr<33:5>, and points to the DMA write buffer cache line with write data using ioLineSel<1:0>.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in cacheable memory space. Also, the cache tag indicates a cache hit. The 21071-BA chips internally transfer the first octaword of DMA write data to the memory write buffer. To prepare to invalidate the cache, sysTagOEEn is deasserted.
3	The cache tags are driven by the 21071-CA chip as invalid. The second octaword is transferred.
4	The tags are written by asserting sysTagWE for one cycle. The cache data is not written. bc_LongWR does not affect this transaction.
5	The 21071-CA chip tristates the tags. The transaction is acknowledged with ioCAck<1:0>. (The acknowledgment could not be done in cycle 4 because the address was still required to do the invalidate.)
6	The transaction is complete, and the next transaction is ready to begin.

Figure 5–16 Timing of DMA Write, Cacheable, Hit, Followed by DMA Read



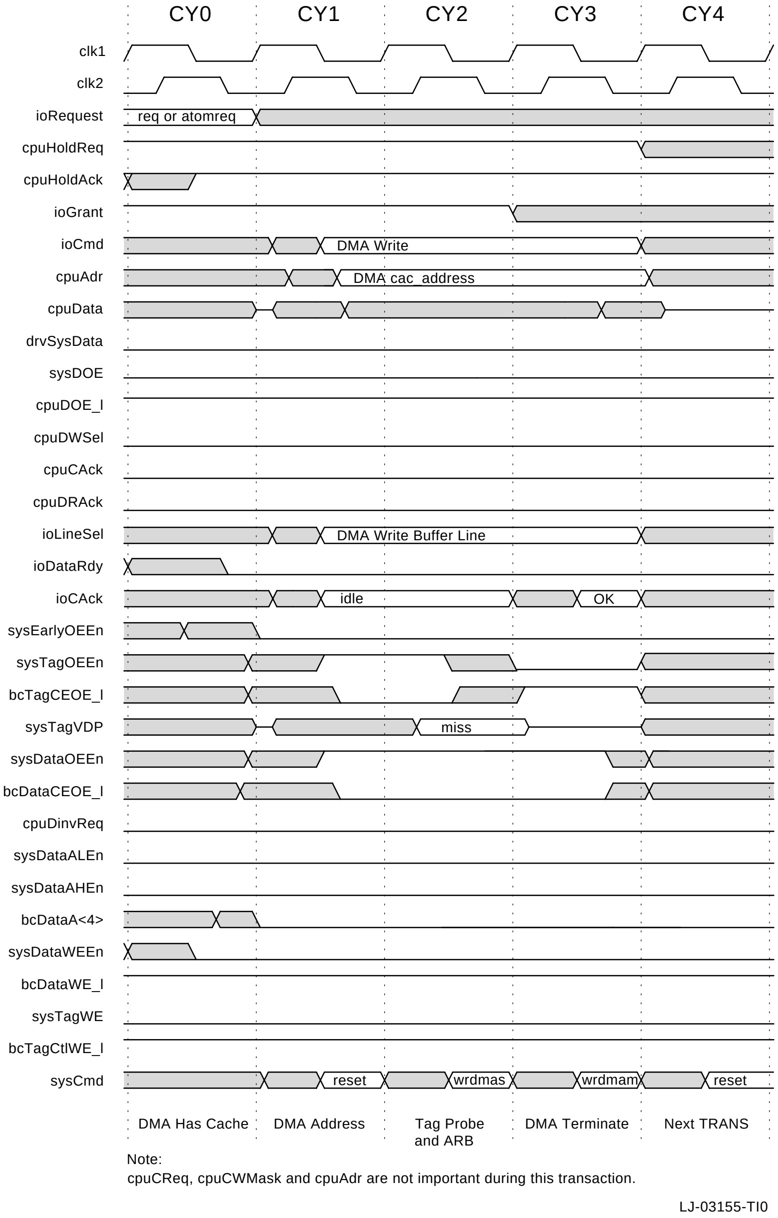
Note:
cpuCReq and cpuCWMask are not important during this transaction.

LJ-03153-TI0

5.1.2.6.2 Cacheable Miss The following table describes the cycles for a DMA write transaction in cacheable space, as shown in Figure 5–17.

Cycle	Description
0	The transaction begins with DMA owning the bus, as indicated by the assertion of ioGrant.
1	The 21071-DA chip requests a DMA write with ioCmd<2:0>, places the address on sysAdr<33:5>, and points to the DMA write buffer cache line with write data using ioLineSel<1:0>.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in cacheable memory space. Also, the cache tag indicates a cache miss. The 21071-BA chips internally transfer the first octaword of DMA write data to the memory write buffer. If the cache is disabled (bc_EN =0), the tag probe results are ignored (assumes a miss), and the CPU internal Dcache is invalidated with cpuDinvReq. If the memory write buffer was full, the probe is completed and the write data is not transferred. If the probe missed, the arbitration may release the cache, and the transaction will continue when the memory write buffer is no longer full.
3	The transaction is acknowledged with ioCAck<1:0>. (The acknowledgment could not be done in cycle 2 because the tag results were not available yet.)
4	The transaction is complete, and the next transaction is ready to begin.

Figure 5–17 Timing of DMA Write, Cacheable, Miss, Followed by CPU Write



Note:
cpuCReq, cpuCWMask and cpuAdr are not important during this transaction.

5.1.2.6.3 Noncacheable A DMA write transaction in noncacheable space is similar to a DMA write miss, as shown in Figure 5–17. Although the tag probe results do not matter, the timing of internal transfers and the acknowledgment are the same. The acknowledgment cannot be done in cycle 2 because of the time required to determine whether the transaction is to a valid memory location.

5.1.2.6.4 I/O Space DMA transactions are not supported to I/O space; they should be responded to as an error using ioCAck<1:0>. This is shown in Figure 5–15 and is described in Section 5.1.2.2.4.

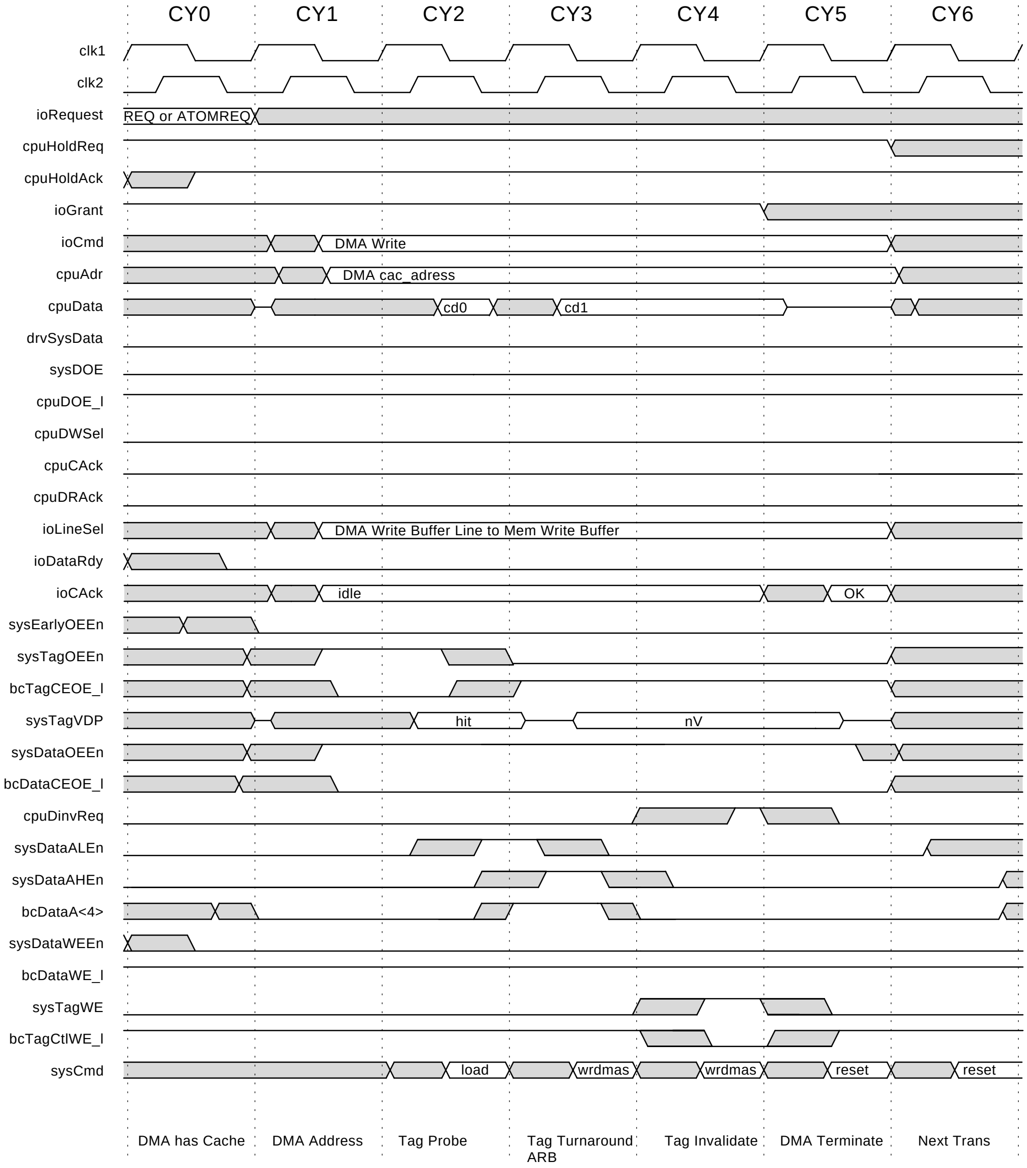
5.1.2.7 DMA Write Masked

A DMA write masked transaction is a combination of the DMA read and DMA write transactions. In a DMA write masked transaction, the cache or memory is read as it is in a DMA read transaction. The results of the read are combined with the DMA write buffer and are loaded into the memory write buffer.

5.1.2.7.1 Cacheable Hit The following table describes the cycles for a DMA write masked transaction in cacheable space that hits, as shown in Figure 5–18.

Cycle	Description
0	The transaction begins with DMA owning the bus, as indicated by the assertion of ioGrant.
1	The 21071-DA chip requests a DMA write masked with ioCmd<2:0>, places the address on sysAdr<33:5>, and points to the DMA write buffer cache line with write data using ioLineSel<1:0>.
2	The 21071-CA chip decodes sysAdr<33:5> and finds it in cacheable memory space. The 21071-CA chip waits for the cache probe, which indicates a cache hit. The first octaword of data is already on the data bus. The data is merged (based on the byte enables) with the DMA write buffer and is loaded into the memory write buffer. To prepare for reading the second octaword, bcDataA<4> is asserted.
3	The cache tags are driven by the 21071-CA chip as invalid. The 21071-CA chip reads the second octaword of cache data, merges it, and places it into the memory write buffer.
4	The tags are written by asserting sysTagWE for one cycle.
5	The 21071-CA chip tristates the tags. The transaction is acknowledged with ioCAck<1:0>. (The acknowledgment could not be done in cycle 4 because the address was still required in order to do the invalidate.)
6	The transaction is complete, and the next transaction is ready to begin.

Figure 5–18 Timing of DMA Write Masked, Cacheable, Hit



Note:
cpuCReq, cpuCWMask, and cpuAdr are not important during this transaction.

LJ-03154-TI0

5.1.2.7.2 Cacheable Miss A DMA write masked transaction that misses the cache looks externally identical to a DMA read that misses the cache. This is described in Section 5.1.2.2.2. (The internal merge and transfer to the memory write buffer is invisible.) The cache may be released.

5.1.2.7.3 Noncacheable A DMA write masked transaction to noncacheable space looks externally identical to a regular noncacheable DMA read. This is described in Section 5.1.2.2.3.

5.1.2.7.4 I/O Space Any DMA transaction to I/O space is an error and is described in Section 5.1.2.2.4.

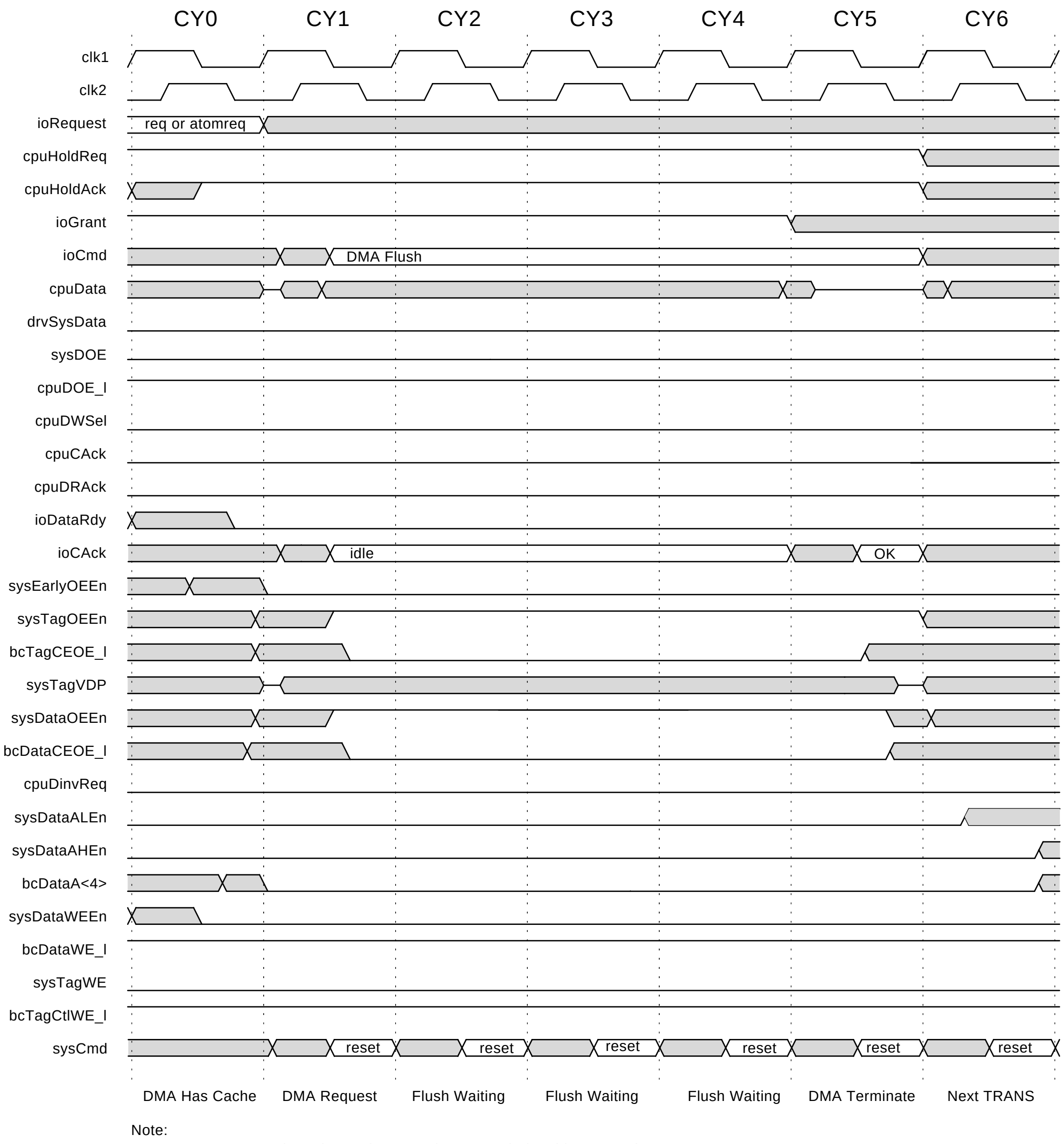
5.1.2.8 DMA Flush

A DMA flush transaction is used to ensure that the 21071-CA chip write buffer is empty. This may be required to guarantee the limited memory access time required by ISA and EISA devices.

The following table describes the cycles involved for a DMA flush transaction, as shown in Figure 5–19.

Cycle	Description
0	The transaction begins DMA owning the bus, as indicated by the assertion of ioGrant.
1	The 21071-DA chip requests a DMA flush with ioCmd<2:0> and places an arbitrary address on sysAdr<33:5>.
2	The 21071-CA chip checks to see if its write buffer is empty. In this figure, it is not emptied for two cycles, so the 21071-CA chip waits.
3	If the write buffer was empty, ioCAck<1:0> would be in this cycle. It is not, so the 21071-CA chip continues to wait.
4	The 21071-CA chip continues to wait.
5	The 21071-CA chip determines that its write buffer is empty, and the transaction is acknowledged with ioCAck<1:0>.
6	The transaction is complete, and the next transaction is ready to begin.

Figure 5–19 Timing of DMA Flush



Note:
cpuCReq, cpuCWMask, and cpuAdr are not important during this transaction.

LJ-03132-TI0

5.1.3 Arbitration Transactions

This section describes the arbitration transactions.

5.1.3.1 Back-to-Back Transactions

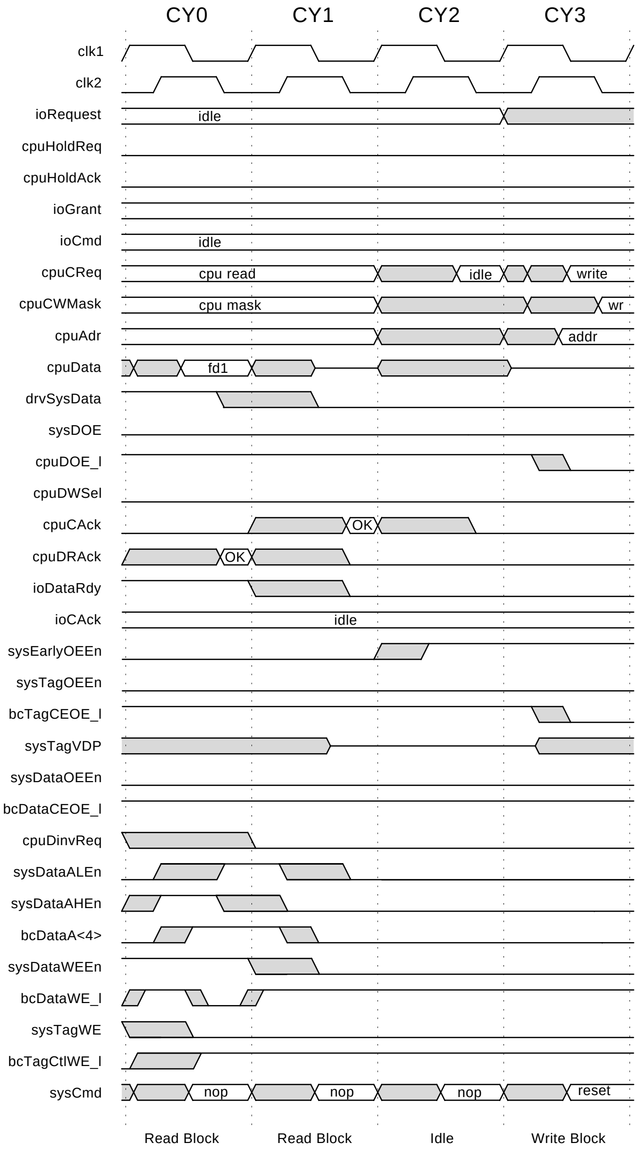
This section describes the CPU-to-CPU and DMA-to-DMA transactions.

5.1.3.1.1 CPU-to-CPU Figure 5–20 shows the actions between two back-to-back CPU transactions. It shows a CPU cacheable read followed by a CPU write, although this description is applicable to any transaction.

The following table describes the cycles for back-to-back transactions, as shown in Figure 5–20.

Cycle	Description
0	A cacheable read block transaction is in progress, as described in cycle 6 of Section 5.1.1.2.1.
1	In the cycle that cpuCAck<2:0> is sent, the cache controls are set inactive, with sysEarlyOEEn, sysTagOEEn, and sysDataOEEn all deasserted.
2	The previous transaction is done. To prepare for the next CPU transaction, sysEarlyOEEn is asserted.
3	cpuDOE_l is asserted. A CPU write transaction is next, as described in cycle 1 of Section 5.1.1.3.1.

Figure 5–20 Switch From CPU Read to CPU Write



LJ-03145-TI0

5.1.3.1.2 DMA-to-DMA

The actions between two back-to-back DMA transactions are shown in Figures 5–21 and 5–22. Figure 5–21 shows a DMA read hit followed by a DMA write, and Figure 5–22 shows a DMA write hit followed by a second DMA write. This description applies to any back-to-back DMA transaction.

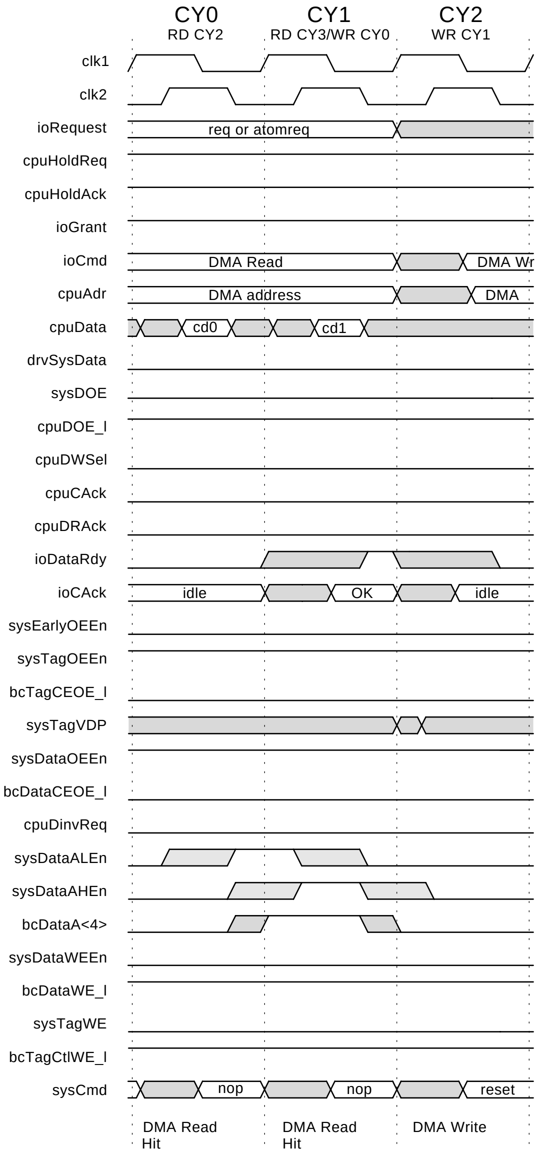
The following table describes the cycles for back-to-back DMA transactions shown in Figure 5–21.

Cycle	Description
0	A DMA read miss transaction is in progress, as described in cycle 5 of Section 5.1.2.2.2. If not already in the proper state, sysDataOEEn and bcDataA<4> are deasserted.
1	The DMA read miss transaction is finished with ioCAck<1:0> being sent.
2	A DMA write transaction is next, as described in cycle 1 of Section 5.1.2.6.2.

The following table describes the cycles for back-to-back transactions shown in Figure 5–22.

Cycle	Description
0	A DMA write hit transaction is in progress, as described in cycle 3 of Section 5.1.2.6.1.
1	The DMA write hit transaction is still in progress.
2	The DMA write hit transaction is finished. ioCAck<1:0> = OK is sent.
3	A DMA write transaction is next, as described in cycle 1 of Section 5.1.2.6.2.

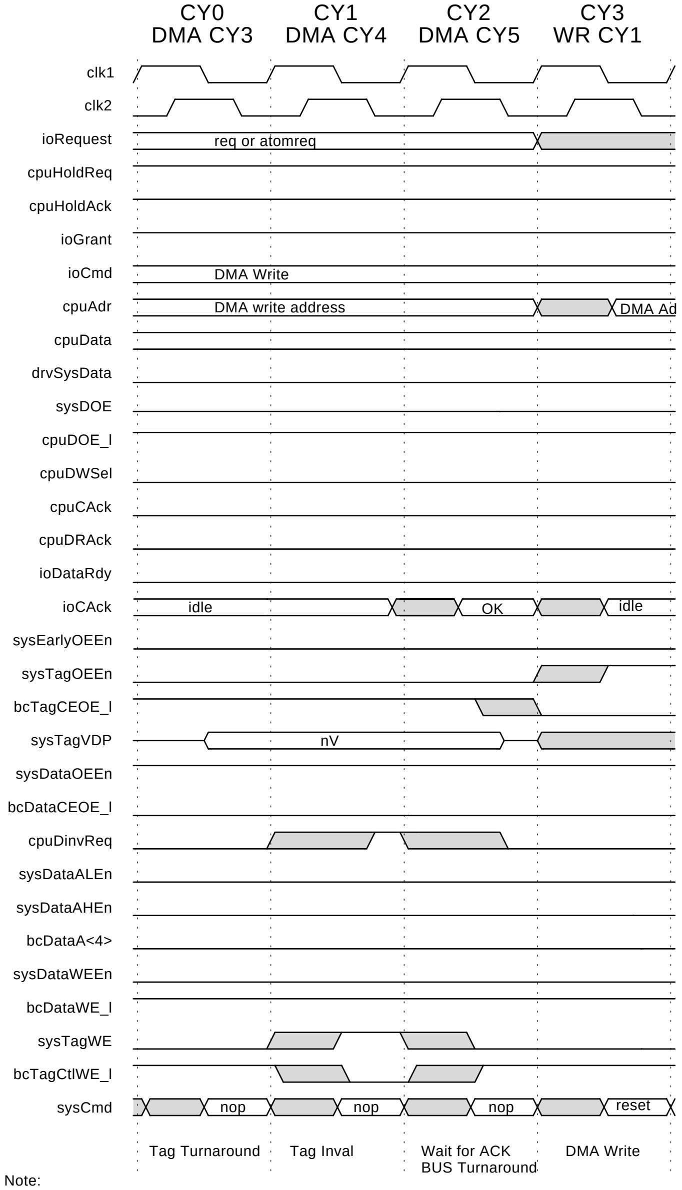
Figure 5–21 Switch From DMA Read Hit to DMA Write



Note:
cpuCReq and cpuCWMask are not important during this transaction.

LJ-03151-TI0

Figure 5–22 Switch from DMA Write Hit to DMA Write



Note:
cpuCReq and cpuCWMask are not important during this transaction.

5.1.3.2 Transitions

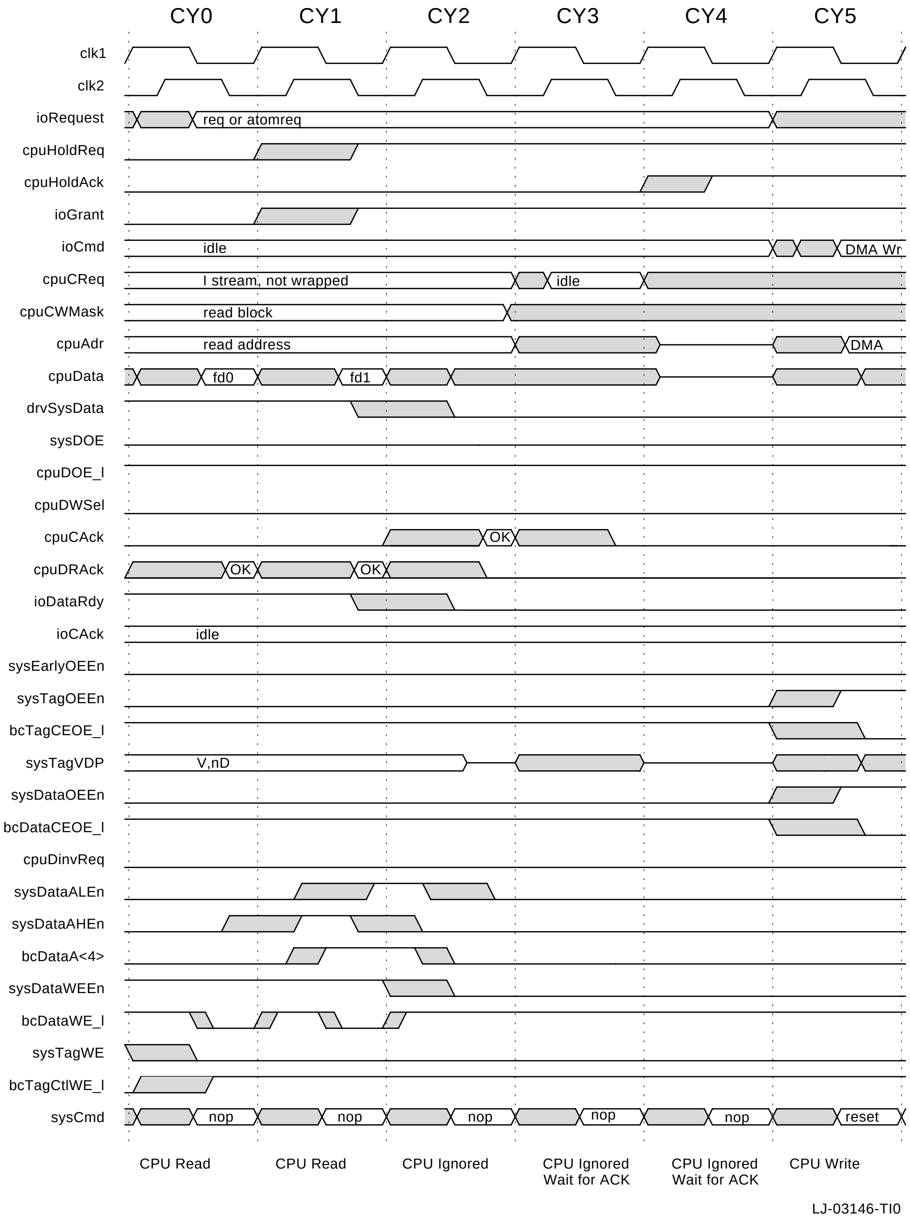
This section describes the transition transactions.

5.1.3.2.1 CPU-to-DMA Figure 5–23 shows when the arbiter decides that the sysBus will be granted to DMA and several signals must change their default states in preparation for the DMA transaction.

The following table describes the cycles for a CPU read to DMA write transactions, as shown in Figure 5–23.

Cycle	Description
0	A CPU read block cacheable with victim transaction is in progress, as described in cycle 5 of Section 5.1.1.2.1. The 21071-CA chip samples the ioRequest<1:0> signals for a request in this cycle. (This figure represents the earliest possible sampling, two cycles before a transaction is acknowledged on ioCAck<1:0> or cpuCAck<2:0>.)
1	The arbiter decides that the 21071-DA chip will be granted the bus. While the read is finishing, cpuHoldReq and ioGrant are asserted.
2	The 21071-DA chip detects the assertion of ioGrant, ignores any CPU transaction to its space, and waits for cpuHoldAck to assert.
3	The 21071-CA and 21071-DA chips wait for cpuHoldAck to assert. In the fastest case, cpuHoldAck asserts this cycle.
4	In this cycle, the CPU issues cpuHoldAck and tristates its buses.
5	The cache tags and data are enabled with sysTagOEEn and sysDataOEEn. The 21071-DA chip drives sysAdr<33:5> and places a DMA command request on ioCmd<2:0>. The details of the DMA write transaction are described in cycle 1 of Section 5.1.2.6.2. The 21071-CA chip receives the command and processes it.

Figure 5–23 Switch from CPU Read to DMA Write



5.1.3.2.2 DMA to CPU, Cache Not Released When the 21071-DA chip owns the sysBus and cache, and the arbiter is ready to grant the bus back to the CPU, the cache and CPU controls must switch back to their CPU defaults.

The descriptions in the following tables apply to any back-to-back DMA transaction.

The following table describes the cycles for a DMA write hit followed by a CPU write, as shown in Figure 5–24.

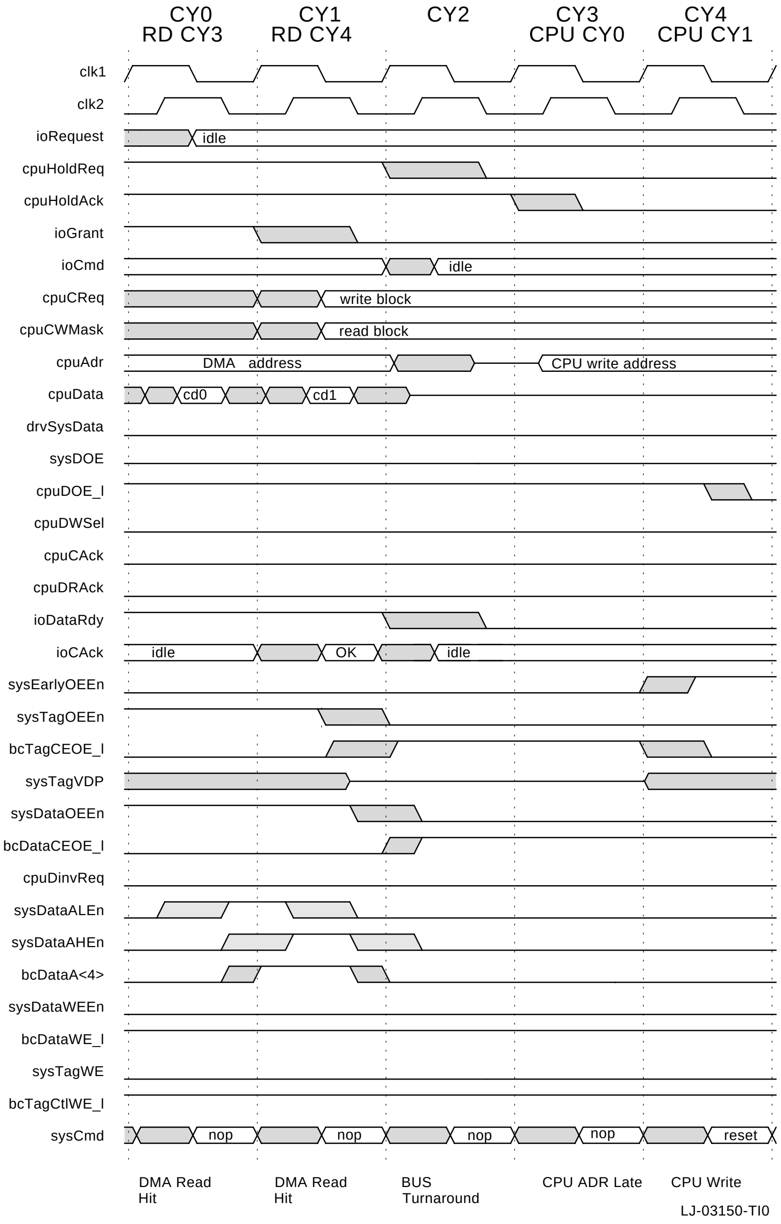
Cycle	Description
0	A DMA write hit transaction is in progress, as described in cycle 4 of Section 5.1.2.6.1. The 21071-DA chip indicates that it does not have additional DMA transactions to complete by sending idle on ioRequest<1:0>. (Or the CPU has priority and is requesting a cycle on cpuCReq<2:0>.)
1	One cycle before the cycle ioCAck<1:0> asserts, the 21071-CA chip deasserts ioGrant and sysDataOEEn.
2	The 21071-CA chip deasserts cpuHoldReq and sysTagOEEn. (In the figure, sysTagOEEn was already deasserted for the invalidate.) The 21071-DA chip detects the deassertion of ioGrant, tristates its address buffers, and waits for cpuHoldAck to deassert.
3	CpuHoldAck deasserts in this cycle.
4	The 21071-CA chip asserts sysEarlyOEEn and may begin processing the CPU transaction.
5	The CPU write appears on sysData.

The following table describes the cycles for a DMA read hit followed by a CPU write, as shown in Figure 5–25.

Cycle	Description
0	A DMA read hit transaction is in progress, as described in cycle 2 of Section 5.1.2.2.1.
1	One cycle before the cycle ioCAck<1:0> asserts, the 21071-CA chip deasserts ioGrant and sysDataOEEn.
2	The 21071-CA chip deasserts cpuHoldReq and sysTagOEEn. (In the figure, sysTagOEEn was already deasserted for the invalidate.) The 21071-DA chip detects the deassertion of ioGrant, tristates its address buffers, and waits for cpuHoldAck to deassert.
3	CpuHoldAck deasserts in this cycle.
4	The 21071-CA chip asserts sysEarlyOEEn and may begin processing the CPU transaction.

Figure 5–24 Switch from DMA Write Hit to CPU Write

Figure 5–25 Switch from DMA Read to CPU Write



5.1.3.2.3 DMA to CPU, Cache Previously Released

If the arbitration allows cache releases, the 21071-CA chip may have released the cache to the CPU after a DMA read or write. This is indicated by ioGrant and cpuHoldReq being deasserted during a DMA transaction. To grant the sysBus to the CPU, additional signals must be changed. This is shown in Figure 5–26 and is described in the following table.

Cycle	Description
0	A DMA read miss transaction is in progress, as described in cycle 6 of Section 5.1.2.2.2. One cycle before ioCAck<1:0> asserts, the 21071-CA chip decides that the CPU has won arbitration.
1	After the cycle ioCAck<1:0> asserts, the 21071-CA chip may begin processing the CPU transaction.
2	The CPU transaction begins.
3	The CPU transaction continues.

5.1.3.2.4 DMA to DMA, Cache Previously Released

To grant the cache back to the 21071-DA chip after a release, the CPU must be forced off the cache. This is shown in Figure 5–27 and is described in the following table.

Cycle	Description
0	A DMA read miss transaction is in progress, as described in cycle 6 of Section 5.1.2.2.2. The 21071-CA chip decides that the 21071-DA chip has won arbitration and asserts cpuHoldReq and ioGrant.
1	The 21071-DA chip sees ioGrant asserted and waits for cpuHoldAck.
2	The 21071-CA and 21071-DA chips wait for cpuHoldAck. In the fastest case, cpuHoldAck asserts in this cycle.
3	The 21071-CA chip asserts sysDataOEEn and sysTagOEEn. The 21071-DA chip sees cpuHoldAck asserted and begins the DMA write. The 21071-CA chip may start the cache probe for the DMA write.
4	The DMA transaction continues.

Figure 5–26 Switch from CPU Released to CPU Write

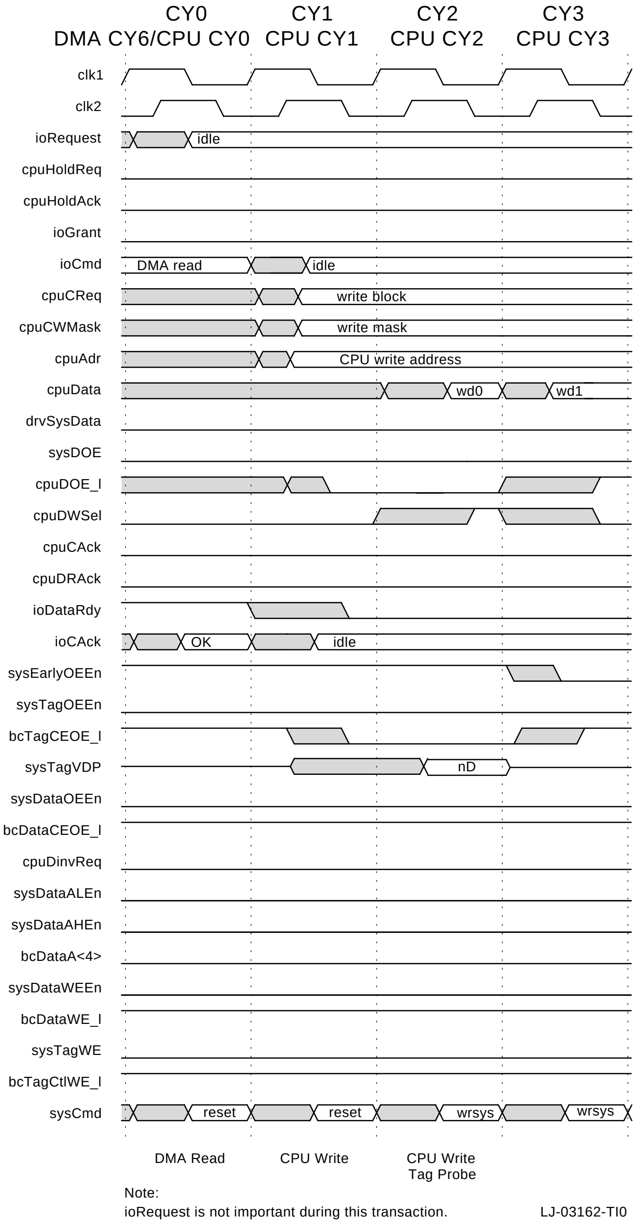
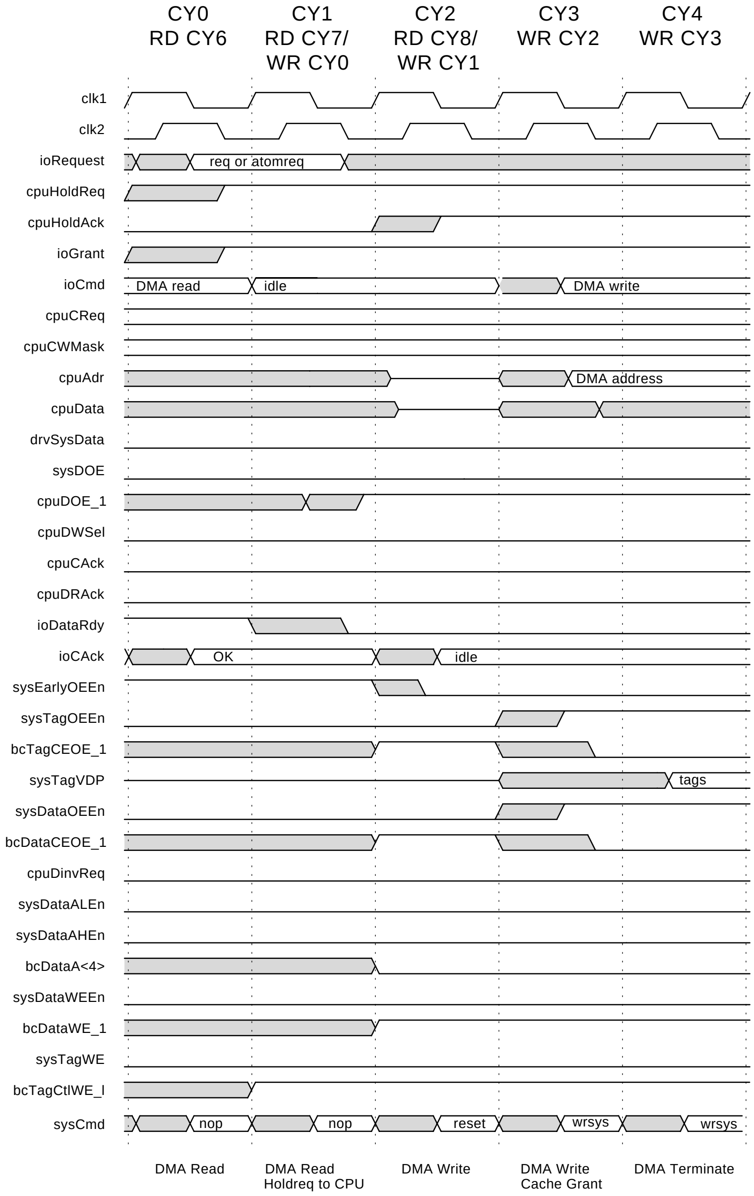


Figure 5–27 Switch from CPU Released to DMA Write



LJ-03163-TI0

5.1.3.3 Preemption

Reads and writes to I/O space, and all barriers may be preempted by the 21071-DA chip. A preemption causes the current CPU transaction to be suspended and causes DMA transactions to be performed. After the DMA transactions are complete, the suspended CPU transaction is resumed.

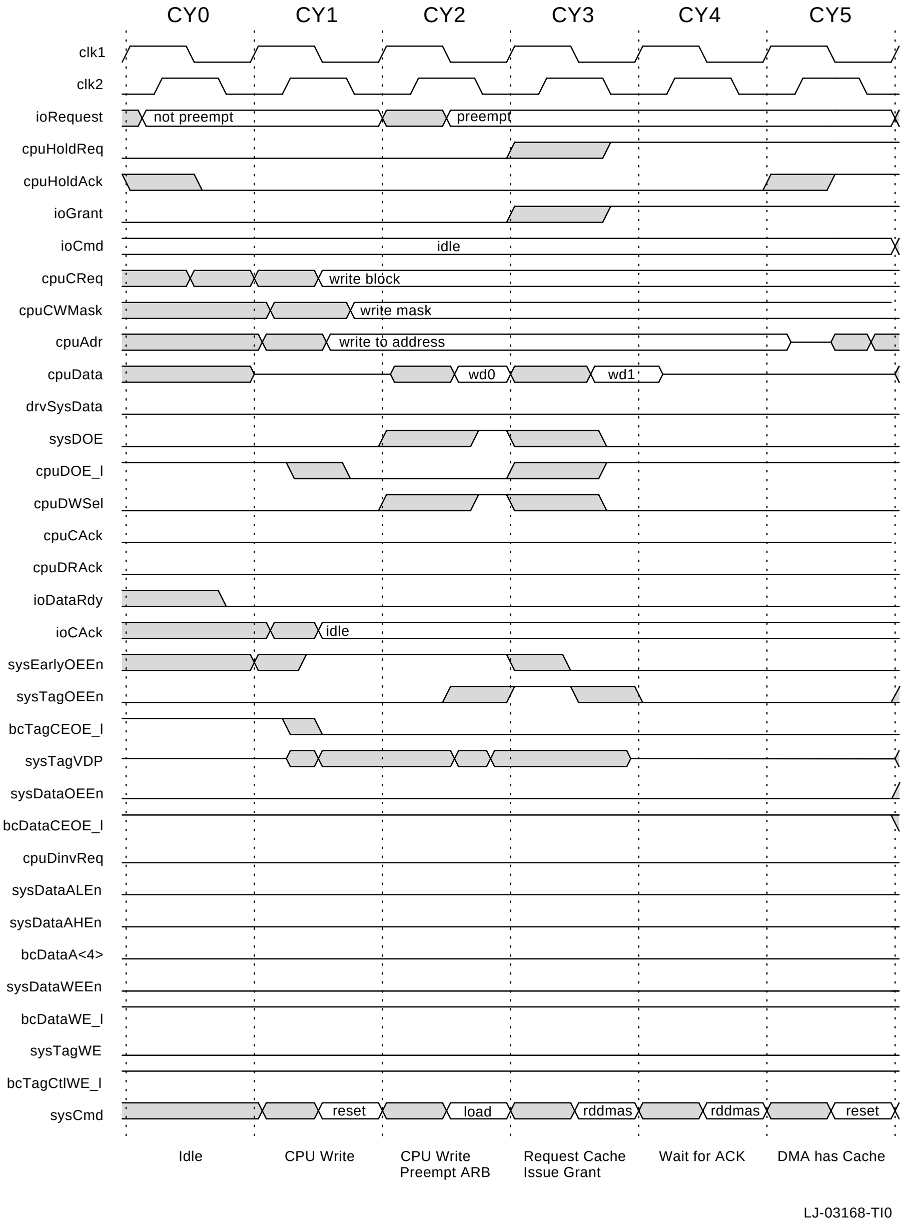
5.1.3.3.1 I/O Write Preempted for DMA Write The following table describes the cycles for a write block transaction to remote I/O space that requires preemption, as shown in Figure 5–28.

This section describes the details of the preemption. For details about the write block to I/O space, see Section 5.1.1.3.5; for details about the DMA read, see Figure 5–13.

Cycle	Description
0	The bus is idle and is owned by the CPU.
1	The CPU requests a read block to I/O space with cpuCReq<2:0>.
2	The 21071-DA chip determines that the I/O read creates a deadlock condition and requests a preempt using ioRequest<1:0>. **Note** Preempt cannot be requested during an I/O write until the CPU data has been latched, otherwise that data will be lost.
3	The 21071-CA chip receives the preempt and asserts cpuHoldReq and ioGrant.
4	The 21071-DA chip receives ioGrant and waits for cpuHoldAck.
5	The CPU happens to assert cpuHoldAck this cycle.
6	The 21071-CA chip receives cpuHoldAck and turns the cache on with sysDataOEEn and sysTagOEEn. The 21071-DA chip places its transaction on the bus. It also determines that another DMA transaction will not be required inside the preempt, and it returns ioRequest<1:0> to idle (or request if a regular DMA is desired after the I/O write).
7	The 21071-CA chip detects a cache hit and loads the DMA read and I/O write buffer with the data.

Cycle	Description
8	The 21071-CA chip loads the second octaword of data and acknowledges the DMA transaction on ioCAck<1:0>. It samples ioRequest<1:0> and finds that the preempt no longer exists, and it deasserts ioGrant, sysTagOEEn, and sysDataOEEn.
9	The 21071-DA chip sees the deassertion of ioGrant and tristates its drivers. It also sees ioCAck<1:0> and knows that the DMA transaction is complete. The 21071-CA chip deasserts cpuHoldAck. The 21071-DA chip sees that the DMA transaction was complete last cycle, and no more preemption is required, so it may request a cpuCAck on ioCmd<2:0> in this cycle. As the sysData drivers have not been enabled yet, a cpuDRAck<2:0> request on ioCmd<2:0> may not be sent until the next cycle.
10	CpuHoldAck deasserts, and the 21071-CA chip sees cpuHoldAck deasserted. The 21071-CA chip enables its data bus drivers if an I/O read was preempted. The 21071-DA chip detects ioGrant and cpuHoldAck both deasserted and continues the preempted CPU transaction. The remaining cycles are the same as the regular non-preempted transaction, resuming where the preempt interrupted it.

Figure 5–28 Timing of CPU Write Block to I/O Space, Preempted by a DMA Read Hit




CY6
CY7
CY8
CY9
CY10
clk1
clk2
ioRequest
idle (only one dma)
cpuHoldReq
cpuHoldAck
ioGrant
ioCmd
DMA Read
cack cpu
idle
cpuCReq
write block
cpuCWMask
write mask
cpuAdr
DMA cac address
cpu adr
cpuData
cd0
cd1
drvSysData
sysDOE
cpuDOE_l
cpuDWSel
cpuCAck
OK
cpuDRAck
ioDataRdy
ioCAck
idle
OK
idle
sysEarlyOEEn
sysTagOEEn
bcTagCEOE_l
sysTagVDP
hit
sysDataOEEn
bcDataCEOE_l
cpuDinvReq
sysDataALEn
sysDataAHEn
bcDataA<4>
sysDataWEEn
bcDataWE_l
sysTagWE
bcTagCtlWE_l
sysCmd
reset
load
rddmas
rddmas
nop
DMA Address
Tag Result
Cache Read
DMA Read Hit
BUS Turnaround
CPU ADR Late
CPU CY0
LJ-03169-TI0

5.1.4 Write Speed

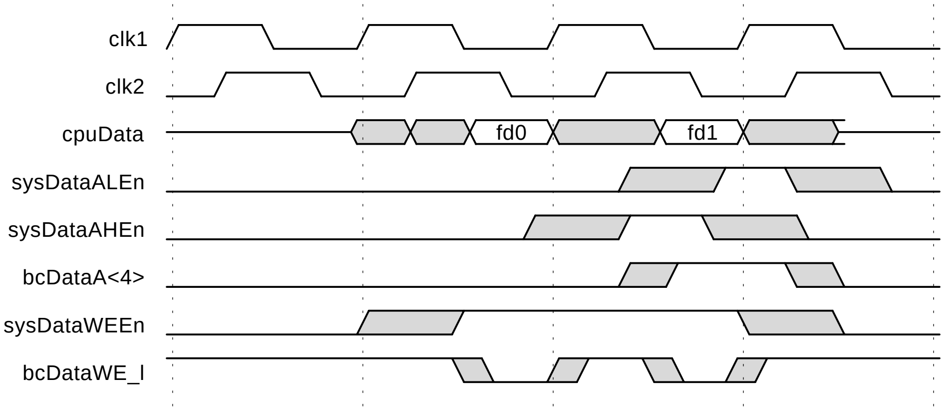
The 21071-CA chip supports two different speeds for writing the cache. A system must determine which speed is required based on the RAM setup, hold, and pulse width constraints.

Note

Different PAL equations are required for each mode.

The normal speed allows one octaword of data to be written each cycle. It is the default and is indicated by the bc_LongWr bit in the general control register being clear. Figure 5–29 shows the timing of two back-to-back writes. This mode is also used as the base for all of the transaction timing diagrams in this chapter.

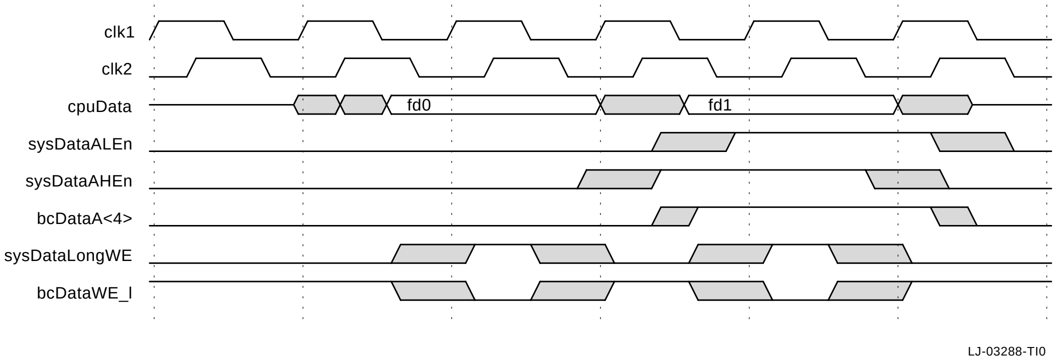
Figure 5–29 Timing of Regular Writes



LJ-03287-TI0

Long writes allow one octaword of data to be written in two cycles. It is indicated by the bc_LongWr bit in the general control register being set. Figure 5–30 shows the timing of two back-to-back writes. All transactions that are limited by the write speed and not by memory or I/O read throughput will be two cycles longer (except for DMA write invalidate, which is not affected by bc_LongWr). bcDataA<4> will be stable for both cycles. SysDataLongWE will pulse for one cycle and may be delayed to generate the write pulse.

Figure 5–30 Timing of Long Writes



5.2 Memory Transactions

This section describes the transaction timing on the memory interface.

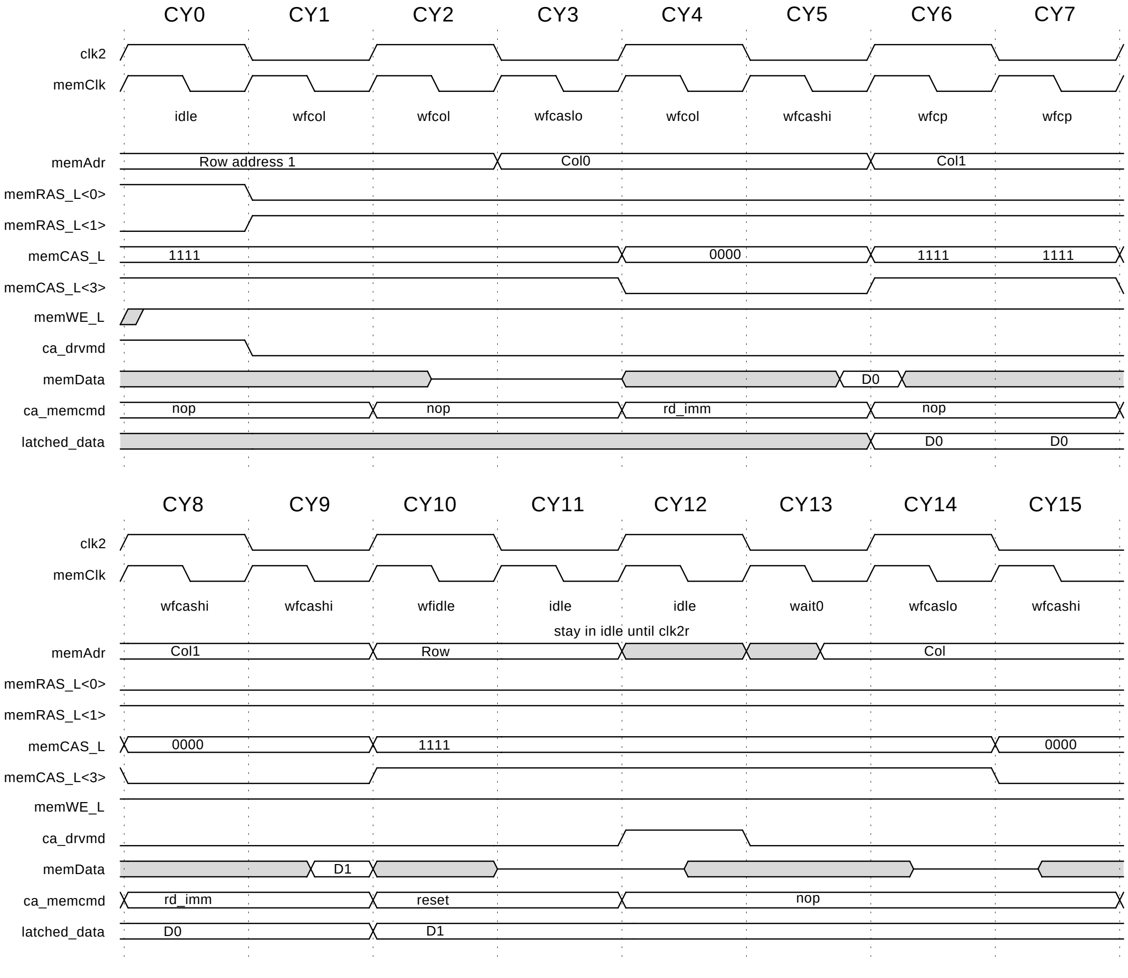
5.2.1 Memory Read Followed by a Page Mode Memory Read

The following table describes a memory read followed by a page mode read, as shown in Figure 5–31.

Cycle	Description
0	The transaction starts when memclkR coincides with clk2R. The 21071-CA has started driving the row address on the memAdr<11:0> pins, one-quarter cycle earlier.
1	The 20171-CA asserts the appropriate memRAS_l<8:0> or memRASB_l<8:0> lines after waiting for the row address setup time. This example uses a ROWSETUP value of 0. The memData drivers are turned off in cycle 1 because the current transaction is a read.
2	The 20171-CA waits for the row address hold time to be satisfied.
3	The 20171-CA commences driving the column address after waiting for the row address hold time. This example uses a ROWHOLD value of 1.
4	The 20171-CA asserts memCAS_l<3:0> after waiting for the column address setup time. This example uses a COLSETUP value of 0. The 21071-CA changes the memCmd<3:1> from NOP to RDIMM, indicating to the 21071-BA chips that memory data should be latched on the rising edge of memClk in cycle 6. This example uses a RDlyRow value of 2. The 21071-CA will change the column address to point to the next column after the column address hold time has been satisfied. This example uses a ColHold value of 1, causing the next column address to be driven in cycle 6.
5	The memCAS_l<3:0> pins remain asserted.
6	The 20171-CA deasserts memCAS_l<3:0>, because the value of RTCAS is 0 in this example. Data is latched into the 21071-BA chips due to the command driven by the 21071-CA chip in cycle 4.
7	The memCAS_l<3:0> pins remain deasserted until the CAS precharge time is satisfied. This example uses a TCP value of 1.

Cycle	Description
8	The 21071-CA chip asserts memCAS_l<8:0> because the CAS precharge time has been satisfied. The 21071-CA changes the memCmd<3:1> from NOP to RDIMM, indicating to the 21071-BA chips that memory data should be latched on the rising edge of memClk of cycle 10. The delay between latching successive read data is internally calculated from other programmed parameters. The 21071-CA will change the column address to point to the next column after the Column address hold time has been satisfied. This example uses a ColHold value of 1.
9	This cycle is similar to cycle 5.
10	The 20171-CA deasserts memCAS_l<3:0>. Data is latched into the 21071-BA chips due to the command driven by the 21071-CA chip in cycle 8. The 21071-CA keeps memRAS_l asserted because it decides to perform the next read in page mode. If the 20171-CA chip were to decide not to remain in page mode, it would deassert memRAS_l in this cycle. The 20171-CA begins driving the next row address on memAdr<11:0> because the read transfers have ended. The 21071-CA drives the memCmd<3:1> to RESET, indicating to the 21071-BA chips that their internal counters and pointers should be reset.
11	The state machine is in idle, but clk2 is low in this cycle. The state machine waits until the next cycle to begin, where clk2 is high. This is required to synchronize two state machines inside the 21071-CA chip.
12	The next transaction is begun and is confirmed to be a page mode read. The 21071-CA switches the drvMemData pin to its default asserted value, causing the 21071-BA chips to turn on their memData drivers.
13	The memData drivers are turned off in this cycle because the current transaction is a read. The 21071-CA begins to drive the first column address. This is a wait cycle for the state machine in the 21071-CA.
14	The column setup time counter is started in this cycle and the 21071-CA waits to assert memCAS_l<3:0>.
15	This cycle is similar to cycle 4.

Figure 5–31 Memory Read Followed by a Page Mode Memory Read



Note:
All signals except memData are drawn at DECchip 21071-CA driver pin with zero delay.
Nonpage Memory Read to Bank 0 is followed by page mode Read to Bank 0

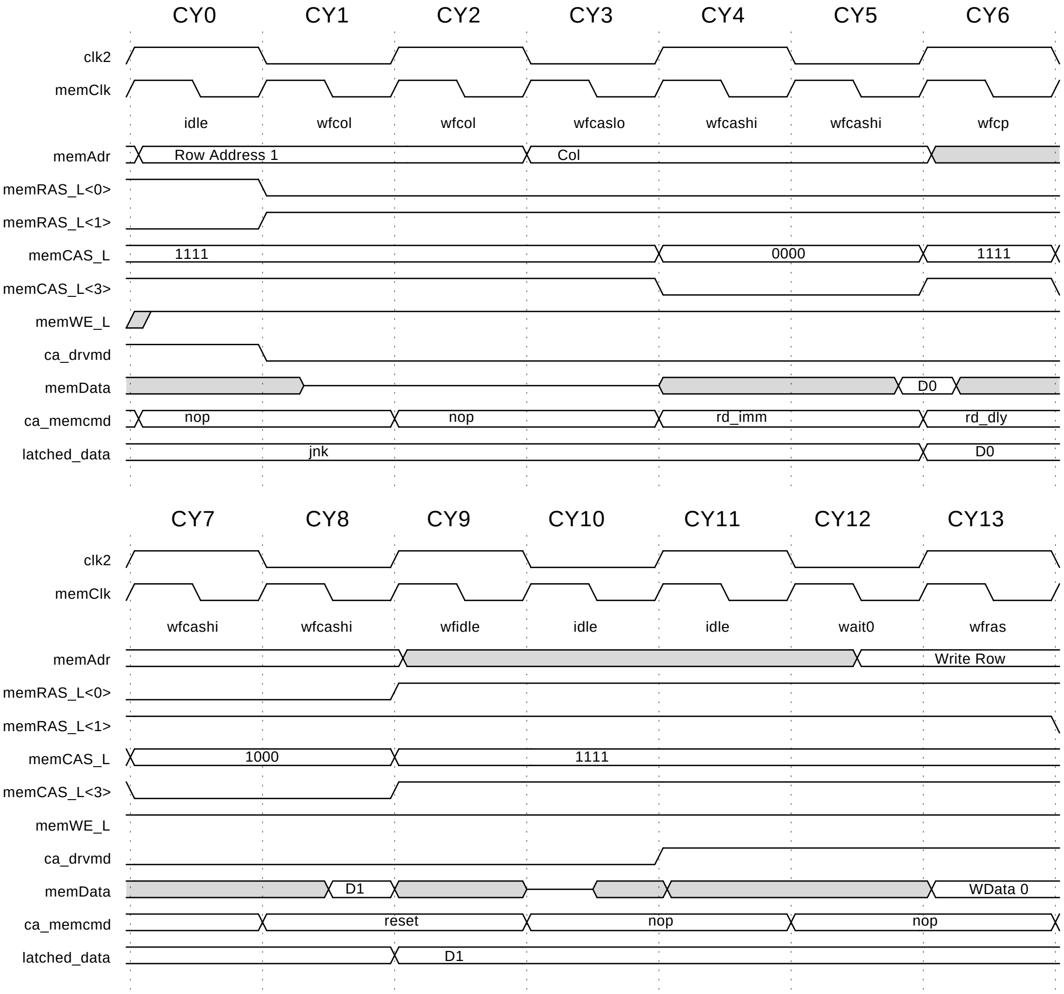
LJ-03172-TI0

5.2.2 Memory Read Followed by a Non-Page Mode Memory Write

Cycles 0 through 5 are the same as in Section 5.2.1. In cycle 6, memCmd<3:1> is RDDly, because the read data must be latched after 3 memClk cycles.

Figure 5–32 shows a memory read followed by a non-page mode memory write.

Figure 5–32 Memory Read Followed by a Non-Page Mode Memory Write



Note:
All signals except memData are drawn at DECchip 21071-CA driver pin with zero delay.
Non page Memory Read to Bank 0 is followed by Write to Bank 1.

LJ-03173-TI0

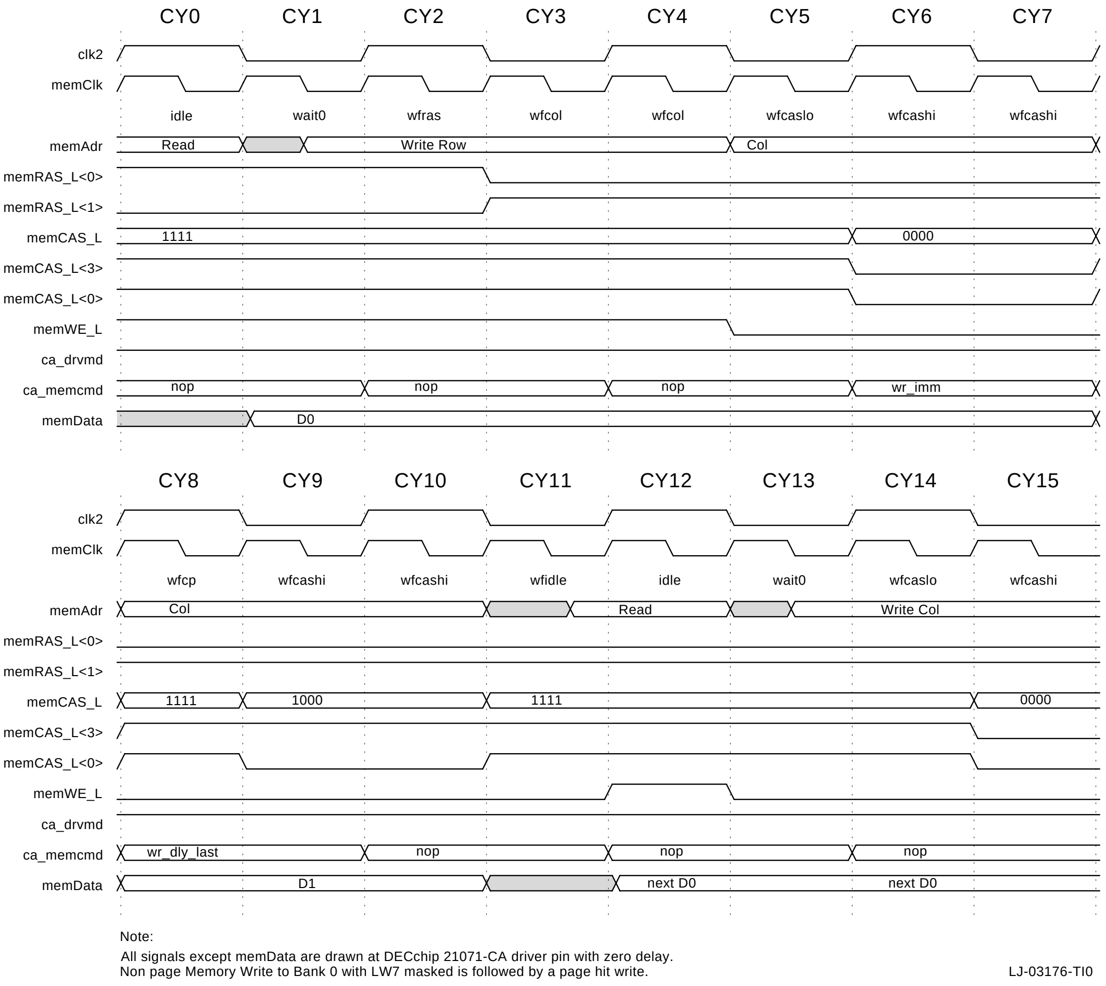
5.2.3 Memory Write Followed by a Page Mode Memory Write

The following table describes the cycles for a memory write followed by a page mode memory write transaction, as shown in Figure 5–33.

Cycle	Description
0	The 21071-CA decides to begin a non-page-mode write in this cycle.
1	The 21071-CA switches the address on memAdr<11:0> from the default of read row address to write row address. The 21071-BA chips are already driving the first write data on the memData pins. The 21071-CA waits in this cycle to retrieve programmed information to do the write.
2	The 21071-CA waits for the address setup time to memRAS_l<8:0> (or memRASB_l<8:0>) to be satisfied.
3	The 21071-CA asserts the appropriate memRAS_l<8:0> (or memRASB_l<8:0>) after waiting for the row address setup time. This example uses a RowSetUp value of 0.
4	The 20171-CA waits for row address hold time to be satisfied.
5	The 20171-CA commences driving the column address after waiting for the row address hold time. This example uses a ROWHOLD value of 1. The 21071-CA asserts memWE_l<1:0> in the same cycle as switching from row to column address. memWE_l<1:0> is held asserted until the end of the transaction.
6	The 20171-CA asserts memCAS_l<3:0> after waiting for the column address setup time. This example uses a COLSETUP value of 0. The 21071-CA changes the memCmd<3:1> from NOP to WRIMM, indicating to the 21071-BA chips that the next memory data should be driven on the rising edge of memClk of cycle 8. This example uses a WHold0Row value of 4. The 21071-CA will change the column address to point to the next column after the column address hold time has been satisfied. This example uses a ColHold value of 1, causing the next column address to be driven in cycle 8.
7	The memCAS_l<3:0> pins remain asserted.
8	The 20171-CA deasserts memCAS_l<3:0> after waiting for the CAS assertion time. This example uses a WTCAS value of 0. The 21071-CA changes memCmd<3:1> to WRDLY_LAST, indicating to the 21071-BA chips that this is the last memData transfer and that the 21071-BA should increment its write buffer cache line pointer after 3 cycles.
9	The 20171-CA asserts memCAS_l<3:0> after waiting for CAS precharge time. This example uses a TCP value of 0.
10	This cycle is similar to cycle 7.

Cycle	Description
11	The 20171-CA deasserts memCAS_l<3:0>. The 21071-CA keeps memRAS_l asserted because it decides to do the next write in page mode. If the 20171-CA chip were to decide not to remain in page mode, it would deassert memRAS_l (or memRASB_l) in this cycle. The 20171-CA begins driving the default read row address on memAdr<11:0> because the write transfers have ended. The 21071-CA deasserts memWE_l<1:0> in this cycle.
12	A page mode write is selected in this cycle.
13	The 21071-CA switches the address on memAdr<11:0> from the default of read row address to write column address. The 21071-BA chips are already driving the first write data on the memData pins. The 21071-CA waits in this cycle to retrieve programmed information to do the write. The 21071-CA asserts memWE_l<1:0> in this cycle.
14	The 21071-CA waits for the address setup time to memCAS_l to be satisfied.
15	This cycle is similar to cycle 6.

Figure 5–33 Memory Write Followed by a Page Mode Memory Write

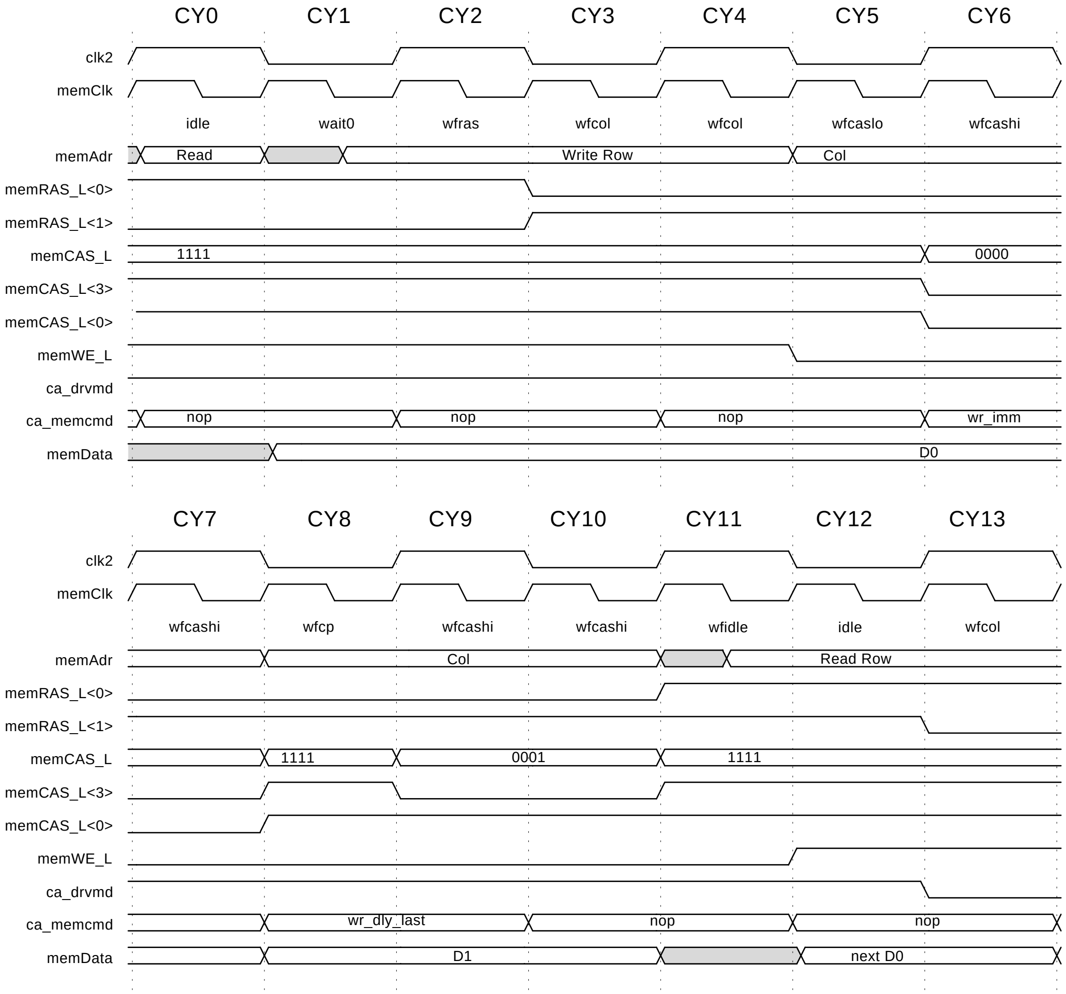


5.2.4 Memory Write Followed by a Non-Page Mode Memory Read

The write portion of the transaction is the same as in Section 5.2.3. The difference is in cycle 12 when the write is completed. Because the default address sent out on memAdr<11:0> is the read address, no extra cycles are required to switch the address multiplexer (mux), when a read is selected.

The memRAS_l<8:0> for the read can assert as early as cycle 13. Figure 5–34 shows a memory write followed by a non-page mode memory read.

Figure 5–34 Memory Write Followed by a Non-Page Mode Memory Read



Note:

All signals except mem Data are drawn at DECchip 21071-CA driver pin with zero delay.
Non Page Memory Write to Bank 0 with LW4 masked is followed by a Read.

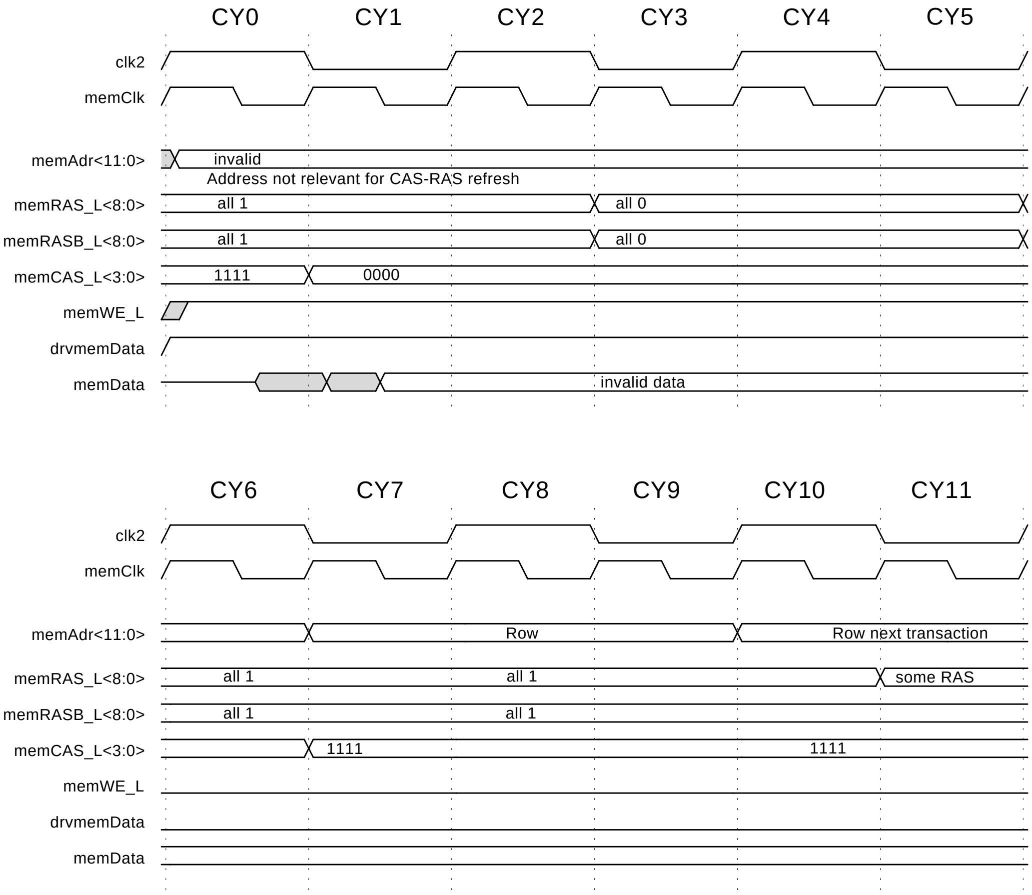
LJ-03175-TI0

5.2.5 Memory Refresh

The following table describes the cycles for a memory refresh transaction, as shown in Figure 5–35.

Cycle	Description
0	The 21071-CA decides to do a CAS-before-RAS refresh. The address is a don't-care during the refresh and continues to point to the default read row address.
1	The 21071-CA asserts all memCAS_l<3:0> signals.
2	The 21071-CA waits to assert memRAS_l and memRASB_l. This example uses a Ref_Cas2Ras value of 1.
3	The 21071-CA asserts all memRAS_l<8:0> and memRASB_l<8:0> signals.
4	The 21071-CA waits to deassert memRAS_l and memRASB_l. This example uses a Ref_RasWidth value of 1.
5	This cycle is the same as cycle 4.
6	The 21071-CA deasserts all memRAS_L and memRASB_l signals. This example uses a Ref_RasWidth value of 1. The RAS precharge count starts in this cycle. This example uses a RAS precharge value (gtr_rp) of 3.
7	The 21071-CA deasserts memCAS_l<3:0> signals in this cycle.
8–10	The 21071-CA waits for the RAS precharge count (in this example, 3) to complete.
11	The 21071-CA asserts RAS for the next transaction.

Figure 5–35 Memory Refresh



Note:
All signals except memData are drawn at DECchip 21071-CA driver pin with zero delay.

LJ-03174-TI0

6

DECchip 21071-CA Electrical Data

This chapter includes the following information about the DECchip 21071-CA chip:

- DC Electrical Data
- AC Electrical Data

6.1 DC Electrical Data

This section describes the dc characteristics of the DECchip 21071-CA chip.

6.1.1 Absolute Maximum Ratings

Table 6–1 lists the maximum ratings of the DECchip 21071-CA chip.

Table 6–1 DECchip 21071-CA Maximum Ratings

Characteristics	Minimum	Maximum
Storage temperature	–55°C (–67°F)	125°C (257°F)
Operating ambient temperature	0°C (32°F)	40°C (104°F)
Air flow	0 lfpm[1]	—
Junction temperature	25°C (77°F)	85°C (185°F)
Supply voltage with respect to Vss, with reset_l asserted	–0.5 V	+6.5 V
Supply voltage with respect to Vss, with reset_l deasserted	4.75 V	5.25 V
Voltage on any pin with respect to Vss	–0.5 V	Vdd + 0.5 V
Maximum power:		
@Vdd = 5.25 V @Cycle = 30 ns		1 W

[1] lfpm = linear feet per minute

Table 6–2 lists the dc parametric values of the DECchip 21071-CA chip.

Table 6–2 DC Parametric Values

Symbol	Description	Minimum	Maximum	Units	Test Conditions
V_{ih}	Input high voltage	2.0	–	V	–
V_{il}	Input low voltage	–	0.8	V	–
V_{oh}	Output high voltage	2.4	–	V	–
V_{ol}	Output low voltage	–	0.4	V	–
I_{il}	Input leakage current[1]	–5	5	µA	0V < Vin < Vdd
I_{ilpu}	Input leakage current[2]	–15	–100	µA	0V < Vin < Vdd
I_{ilpd}	Input leakage current[3]	15	100	µA	0V < Vin < Vdd
I_{ol}	Output leakage current (tristated)	–10	10	µA	0V < Vin < Vdd

[1] Excluding memPDDIn, scanEnable, vFrame, vRefresh, wideMem, testMode and tristateL.

[2] For tristateL, vFrame, and vRefresh.

[3] For memPDDIn, scanEnable, testMode, and wideMem.

6.2 AC Electrical Data

This section describes the ac characteristics of the DECchip 21071-CA chip.

6.2.1 Clocks

The DECchip 21071-CA uses one clock (running at twice the nominal system frequency) plus a synchronous phase reference signal to generate five internal clock edges. See Figures 6–1 and 6–2, and Tables 6–3 and 6–4 for details about DECchip 21071-CA external clock requirements and internal clock phase relationships.

A clock system must meet the requirements in Figure 6–1 and Table 6–4 to guarantee the proper behavior of the 21071-CA chip's internal logic. The 21071-CA chip does not specify the maximum skew allowed for external transfers to or from the CPU, Bcache PALs, Bcache, 21071-BA chips, or 21071-DA chip because these skew limits are dependent on module placement and routing. A system designer must examine external transfers to determine the maximum clock skews allowed between chips.

The skew numbers shown in Figure 6–1 and Table 6–4 are given for a 30.0 ns cycle time. At a longer cycle time, the allowable skew may be increased, as long as the given minimum times between clock edges are not violated. These skew limits assume that the 21071-CA chip adds another 0.1 ns of uncertainty between rising and falling edges due to non-ideal input buffer switching thresholds.

Table 6–3 DECchip 21071-CA Clock AC Characteristics

Parameter	Minimum	Maximum	Unit	Note
System cycle time	30	—	ns	*c* in Figure 6–1
clk1x2 period	15	—	ns	—
clk1x2 frequency	—	66	MHz	—
clk1x2 rise time	—	1	ns	—
clk1x2 fall time	—	1	ns	—
clk2ref setup to clk1x2 rising	0.43	—	ns	*Tsu* in Figure 6–1
clk2ref hold from clk1x2 rising	2.32	—	ns	*Th* in Figure 6–1

Figure 6–1 DECchip 21071-CA Clock Skew Requirements

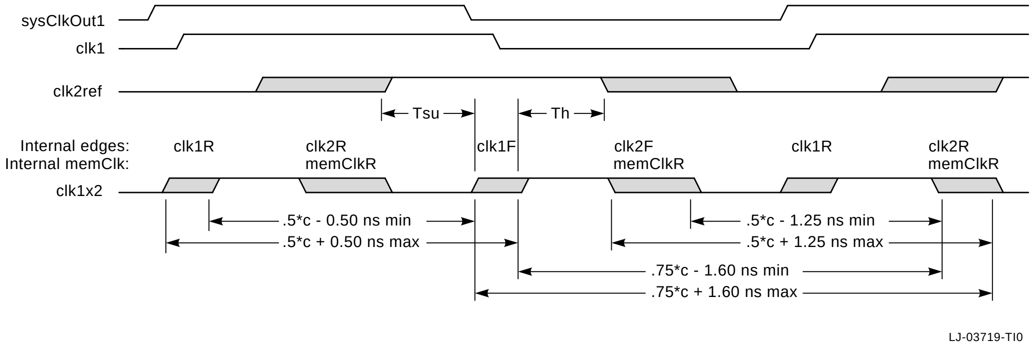
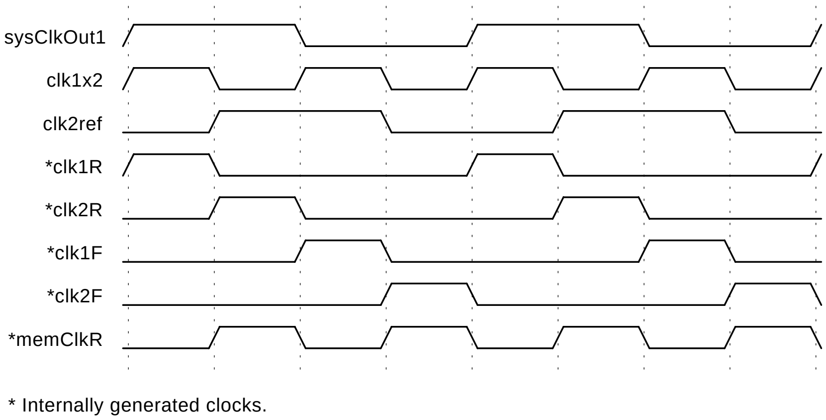


Table 6–4 DECchip 21071-CA Clock Skew Limits at clk1x2 Pin

Parameter	Example Transfers	Maximum	Unit	Note
clk1x2 rising edge to rising edge	clk1R to clk1R, clk1R to clk1F, clk1F to clk1R, clk1F to clk1F	0.50	ns	@ Cycle = 30 ns
clk1x2 falling edge to falling edge	clk2R to clk2R, clk2R to clk2F, clk2F to clk2R, clk2F to clk2F	1.25	ns	@ Cycle = 30 ns
clk1x2 rising edge to falling edge	clk1R to clk2R, clk1R to clk2F, clk1F to clk2R, clk1F to clk2F	1.60	ns	@ Cycle = 30 ns
clk1x2 falling edge to rising edge	clk2R to clk1R, clk2R to clk1F, clk2F to clk1R, clk2F to clk1F	1.60	ns	@ Cycle = 30 ns

Figure 6–2 DECchip 21071-CA Clock Signals



The 21071-CA imposes no requirements on clk1 or sysClkOut1. Skew on clk1 will be constrained by limits imposed by external paths to or from the Bcache control PALs. The phase error between sysClkOut1 and clk1x2 will be constrained by limits imposed by external paths to or from the CPU chip.

6.2.2 Signals

Figures 6–3 and 6–4 demonstrate the timing measurements specified in Tables 6–6 and 6–7.

Figure 6–3 DECchip 21071-CA Output Delay Measurement

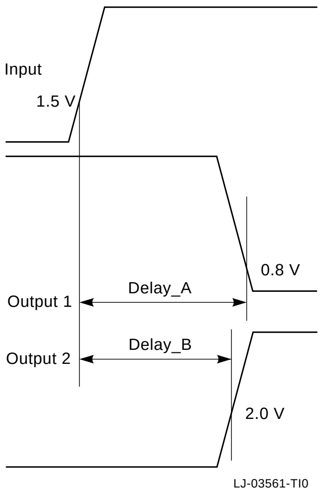
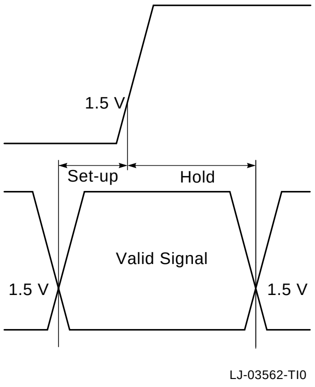


Figure 6–4 DECchip 21071-CA Setup and Hold Time Measurement



The following ac electrical data is specified with respect to the appropriate edge at the clk1x2 pin. Both the output delay table and the setup/hold time table assume a 1 ns edge rate at the clk1x2 pin.

All outputs drive a 50 pF load. When estimating module delays, you may need to replace the 50 pF load delay with a simulated (or calculated) delay. The delays for 4 mA and 8 mA drivers driving a 50 pF load are provided in Table 6–5. See Table 2–1 for information about the buffer size of every output pin.

Table 6–5 DECchip 21071-CA Output Buffer Delays into a 50 pF Load

Type	Minimum	Maximum	Unit
4 mA	3.5	7.6	ns
8 mA	2.3	5.0	ns

Table 6–6 DECchip 21071-CA AC Characteristics (Valid Delay into a 50 pF Load)

Signal	Minimum	Maximum	Unit	Reference Edge
sysData<15:0>[1]	5.9	19.1	ns	clk1R
tagAdr<31:17>, tagAdr P, tagCtlVDP	6.0	21.3	ns	clk1F
cpuCAck<2:0>, cpuDRAck<2:0>, cpuDWSel<1>, cpuDInvReq, cpuHoldReq	4.8	14.6	ns	clk1R
sysDOE, sysEarlyOEEn	4.5	11.6	ns	clk1R
sysTagOEEn[2]	4.8	11.5	ns	clk1F
sysTagOEEn[2]	4.9	11.7	ns	clk1R
sysDataOEEn[3]	4.9	12.3	ns	clk2F
sysDataOEEn[3]	4.9	12.0	ns	clk1F
sysDataOEEn[3]	4.9	11.8	ns	clk1R
sysTagWE, sysDataWEEn	4.3	11.6	ns	clk1R
sysDataLongWE	4.5	11.6	ns	clk1F
sysDataALEn	4.6	12.0	ns	clk2R
sysDataAHEn	4.7	12.0	ns	clk2F
ioGrant, ioCAck<1:0>, ioDataRdy	3.3	12.1	ns	clk1R

---

[1] Two cycles are allocated for returning CSR read data.

[2] See Section 2.2.2.2 to determine which measurement is relevant.

[3] See Section 2.2.2.3 to determine which measurement is relevant.

(continued on next page)

Table 6–6 (Cont.) DECchip 21071-CA AC Characteristics (Valid Delay into a 50 pF Load)

Signal	Minimum	Maximum	Unit	Reference Edge
memAdr<11:0>[4]	7.4	16.1	ns	clk1R
memAdr<11:0>[5]	9.1	24.3	ns	clk1R
memAdr<11:0>[6]	5.4	12.3	ns	memClkR
memAdr<11:0>[7]	5.4	13.7	ns	memClkR
memRAS_l<8:0>[8], memRASB_l<8:0>[8]	4.7	11.8	ns	memClkR
memRAS_l<8:0>[9], memRASB_l<8:0>[9]	4.0	10.3	ns	memClkR
memCAS_l<3:0>[8]	4.9	12.5	ns	memClkR
memCAS_l<3:0>[9]	4.1	12.5	ns	memClkR
memCAS_l<3:0>[10]	15.0	—	ns	memClkR
memWE_l<1:0>	4.9	11.9	ns	memClkR
memPDClk, memPDLoad_l	5.1	15.3	ns	clk2R
memDTOE_l	4.8	11.7	ns	memClkR
memDSF	5.0	12.5	ns	memClkR
sysCmd<2:0>	3.3	11.7	ns	clk1R

---

[4] Delay to valid row address for banksets 0 through 7. Row addresses transition on clk1R, one quarter-cycle before memClkR. Subtract (system cycle / 4) from the numbers in this row to calculate the row address delay from memClkR.

[5] Delay to valid row address for bankset 8. Row addresses transition on clk1R, one quarter-cycle before memClkR. Subtract (system cycle / 4) from the numbers in this row to calculate the row address delay from memClkR.

[6] Delay on transition from row address to column address.

[7] Delay on transition from column address to subsequent column address.

[8] Delay for falling edge of signal.

[9] Delay for rising edge of signal.

[10] Pulse width from rising to falling edge of signal.

(continued on next page)

Table 6–6 (Cont.) DECchip 21071-CA AC Characteristics (Valid Delay into a 50 pF Load)

Signal	Minimum	Maximum	Unit	Reference Edge
subCmdA<1:0>, subCmdB<1:0>	3.3	14.1	ns	clk1R
subCmdCommon	3.3	11.8	ns	clk1R
sysIORead	3.3	12.0	ns	clk1R
sysReadOW	3.3	12.6	ns	clk1R
drvSysData[9]	3.3	13.4	ns	clk2R
drvSysData[8]	3.3	13.4	ns	clk2F
drvSysCSR	3.3	14.3	ns	clk2R
drvMemData	4.8	11.7	ns	memClkR
memCmd<3:1>	3.3	12.4	ns	clk2R

---

[8] Delay for falling edge of signal.

[9] Delay for rising edge of signal.

Table 6–7 DECchip 21071-CA AC Characteristics (Setup/Hold Time)

Signal	Setup	Hold	Unit	Reference Edge
sysData<15:0>	0.4	4.4	ns	clk2F
sysAdr<33:5>[1]	12.2	4.3	ns	clk1R
sysAdr<33:5>[2]	9.7	4.3	ns	clk1F
tagAdr<31:17>, tagAdrP, tagCtlVDP	–0.4	4.4	ns	clk1F
cpuCWMask<7:0>	7.7	—	ns	clk1R
	—	2.2	ns	clk1F

---

[1] For CPU transactions only.

[2] For DMA transactions only.

(continued on next page)

Table 6–7 (Cont.) DECchip 21071-CA AC Characteristics (Setup/Hold Time)

Signal	Setup	Hold	Unit	Reference Edge
cpuCReq<2:0>	1.8	3.4	ns	clk1F
cpuCReq<2:0>[3]	12.0	—	ns	clk1R
cpuHoldAck	–0.8	3.9	ns	clk1F
ioRequest<1:0>, ioCmd<2:0>	–0.1	3.4	ns	clk1F
memPDDIn	–0.8	4.4	ns	clk2R

[3] In initial cycle of transaction; referenced to the clk1R which receives sysAdr.

7

DECchip 21071-CA Power-Up and Initialization

This chapter describes the behavior of the DECchip 21071-CA chip on power-up and assertion of reset_l. It also describes the system level requirements and the various registers that have to be initialized after reset_l is deasserted.

7.1 Power-Up

On power-up, the reset_l input of the DECchip 21071-CA chip should be asserted. It should be kept asserted until the system clocks are up and running for 20 cycles.

7.2 Internal Reset

The assertion and deassertion of the reset_l pin on the module is asynchronous to the DECchip 21071-CA. An internal reset signal is generated from reset_l which asserts asynchronously as soon as reset_l is asserted, but deasserts synchronously. Due to the synchronous deassertion of the internal reset, the DECchip 21071-CA requires that no external transaction should start until 10 system clock cycles after the deassertion of reset_l.

7.3 State of Pins on Reset Assertion

The following are general rules and requirements for the behavior of the DECchip 21071-CA chip pins during reset:

- All input only control signals (except the clocks and reset_l) must be in the deasserted state as long as reset is asserted.
- All output only signals are deasserted.
- All bidirectional signals are tristated.

The exceptions to these rules are as follows:

- sysDataOEEn and sysTagOEEn are asserted synchronously after the assertion of reset_l and are deasserted as soon as reset_l deasserts (without waiting for the deassertion of synchronous internal reset). These signals keep sysData<127:0>, sysCheck<7:0>, tagAdr<32:17>, and the tag control signals driven during reset.
- The presence detect logic activates on the deassertion of internal reset. For details of the operation, refer to Section 3.2.7.
- drvMemData is asserted by the DECchip 21071-CA so that memData<127:0> are driven by the 21071-BA during reset.

Note

In all cases, the assertion of tristate_l will override the assertion of reset_l. That is, if tristate_l is asserted during reset, all the outputs of the DECchip 21071-CA go to their High-Z state.

If reset_l is still asserted when tristate_l deasserts, the signals return to the normal reset state described previously.

7.4 Configuration after Reset Deassertion

Software must initialize the following registers in the DECchip 21071-CA after the deassertion of reset_l:

- General control register
- Tag enable register
- Bankset configuration registers

  To determine memory configuration, see Section 4.5.
- Bankset base address registers
- Bankset timing registers A and B

  To determine the programmed values of these registers, see Section 4.6.
- Global timing register
- Refresh timing register

The deassertion of internal reset causes the DECchip 21071-CA to commence doing refreshes. Most DRAMs require that they be refreshed 8 times before any write or read transactions are addressed to them. The DECchip 21071-CA does not guarantee this. Software has to ensure that memory reads and writes are not performed until the eight refreshes are completed. The refresh rate can be increased using two mechanisms:

1. Software can use the force_Ref bit in the refresh timing register to generate back-to-back refreshes. In this case, software has to write the force_Ref bit, wait 10 cycles for it to be cleared (indicating that one refresh has been completed), and then set it again for the next refresh.
2. Software can also choose to set ref_Interval in the refresh timing register at its minimum value of 64 memClk cycles (ref_Interval = 1). This will cause refreshes to happen every 32 system clock cycles.

After initialization of the registers, the Bcache and memory must be written with good parity or ECC, otherwise errors may prevent correct operation.

Part II

Part II contains six chapters that provide information about the DECchip 21071-DA chip. The following table provides a brief description of each chapter:

Chapter	Description
8	Describes the DECchip 21071-DA pin signals.
9	Describes the DECchip 21071-DA architecture.
10	Describes the DECchip 21071-DA control and status registers.
11	Describes the transaction flows for the 21071-DA chip.
12	Describes the DECchip 21071-DA electrical requirements.
13	Describes the behavior of the DECchip 21071-DA chip during power-up.

8

DECchip 21071-DA Pin Descriptions

This section provides a listing and description of pin signals for the DECchip 21071-DA chip.

The 21071-DA chip has three major bus interfaces:

- sysBus
- Peripheral Component Interconnect (PCI)
- epiBus interface

8.1 DECchip 21071-DA Pin List

Table 8–1 lists the pin signals grouped by function. The information in the Type column identifies a signal as input (I), output (O), or bidirectional (B). The Buffer Strength column indicates the buffer drive strength.

All output and bidirectional pins, except pTestout, can be tristated.

Table 8–1 DECchip 21071-DA Pin List

Signal Name	Quantity	Type	Buffer Strength	Function
sysBus Signals (52 Total)				
sysAdr<33:5>	29	B	4 ma	Address bus
cpuCReq<2:0>	3	I	—	Cycle request
cpuCWMask<7:0>	8	I	—	Cycle write mask
cpuHoldAck	1	I	—	Hold acknowledge
ioCmd<2:0>	3	O	8 ma	Command for DMA transactions; acknowledgment for I/O transactions.
ioCAck<1:0>	2	I	—	Acknowledgment from the 21071-CA chip on DMA transactions
ioDataRdy	1	I	—	Indicates that the requested data is loaded into the 21071-BA chips and can be extracted
ioLineSel<1:0>	2	O	4 ma	Selects which cache line should be read/written from the sysBus
ioRequest<1:0>	2	O	8 ma	Request for DMA transactions on sysBus
ioGrant	1	I	—	Indicates that the sysBus has been granted to the 21071-DA chip

(continued on next page)

Table 8–1 (Cont.) DECchip 21071-DA Pin List

Signal Name	Quantity	Type	Buffer Strength	Function
PCI Signals (47 Total)				
AD<31:0>	32	B	12/16 ma	PCI data and address lines
CBE_l<3:0>	4	B	12/16 ma	Bus command and byte enable
FrameL	1	B	12/16 ma	Cycle frame
TrdyL	1	B	12/16 ma	Target ready
IrdyL	1	B	12/16 ma	Initiator ready
StopL	1	B	12/16 ma	Stop the current transaction
LockL	1	I	—	Indicates an atomic operation that may take multiple transactions to complete
DevselL	1	B	12/16 ma	Device select
Par	1	B	12/16 ma	Parity bit
PerrL	1	B	12/16 ma	Parity error
ReqL	1	O	12/16 ma	Bus request
GntL	1	I	—	Bus grant
pClk	1	I	—	PCI clock
PCI Sideband Signals (2 Total)				
MemReql	1	I	—	Clears path from PCI to memory
MemAckl	1	O	12/16 ma	Acknowledgment that path for PCI to memory has been cleared by the 21071-DA chip

(continued on next page)

Table 8–1 (Cont.) DECchip 21071-DA Pin List

Signal Name	Quantity	Type	Buffer Strength	Function
epiBus Signals (46 Total)				
epiData<31:0>	32	B	4 ma	Interchip data for both DMA and I/O operations
epiBEnErr<3:0>	4	B	4 ma	epiData byte enable
epiOWSel	1	O	4 ma	Selects which octaword of the cache line will be transferred on the epiData bus
epiLineSel<1:0>	2	O	4 ma	Selects which cache line will be transferred on the epiData bus
epiSelDMA	1	O	4 ma	Selects which buffer (I/O or DMA) will be transferred on the epiData bus
epiFromIOB	1	O	4 ma	Selects the next epiData transfer from the 21071-DA chip to the 21071-BA chips
epiEnable<3:0>	4	O	4 ma	Qualifies epiData control signals and enables output drivers
epiLineInval	1	O	4 ma	Clears all byte valid bits in the current line of the DMA write buffer

(continued on next page)

Table 8–1 (Cont.) DECchip 21071-DA Pin List

Signal Name	Quantity	Type	Buffer Strength	Function
Miscellaneous/Clock Signals (4 Total)				
intHw0	1	O	4 ma	Interrupt to the DECchip 21064 microprocessor indicating that the 21071-DA chip has detected an abnormal condition
resetL	1	I	—	21071-DA chip reset
clk1x2	1	I	—	Clock input
clk2ref	1	I	—	Phase reference for clk1x2
Test Signals (4 Total)				
testMode	1	I	—	Test mode select
scanEn	1	I	—	Scan Enable for chip testing
tristate_l	1	I	—	Tristates all output and bidirectional pins for chip and module testing
pTestout	1	O	4 ma	Parametric NAND tree output

Pin Totals

Total signal pins:	155
Total power and ground pins:	53
Total pins:	208

8.2 DECchip 21071-DA Signal Descriptions

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

Signal descriptions are grouped by function and correspond to the pin list (Table 8–1).

8.2.1 sysBus Signals

This section describes the sysBus signals.

8.2.1.1 sysAdr<33:5>

Signal Type: 21071-CA Input, CPU output, 21071-DA bidirectional
Input Sampling Clock Edge: clk1R
Output Clock Edge: clk1R

sysAdr<33:5> signals contain the cache line address of sysBus transactions; sysAdr<33:32> indicates the address quadrant.

sysAdr<33:5> are driven by the CPU on CPU-initiated transactions and by the 21071-DA chip on DMA transactions.

- On CPU-initiated transactions, the cache line address is expected to be held on the bus from the command cycle through the terminate/acknowledge cycle.
- On DMA transactions, the 21071-DA chip drives the address from the time cpuHoldAck and ioGrant are asserted until ioGrant or cpuHoldAck is deasserted. sysAdr<33:5> are valid when ioCmd<1:0> carry a valid DMA command.

8.2.1.2 cpuCReq<2:0>

Signal Type: 21071-DA Input
Signal Source: CPU
Output Clock Edge: clk1F

Whenever the DECchip 21064 microprocessor wants to initiate an external transaction, it puts a transaction type code onto cpuCReq<2:0>. Table 8–2 lists the encodings for the different transaction types.

Table 8–2 CPU-Initiated Transaction Encodings

cpuCReq<2:0>	Transaction
000	Idle
001	Barrier
010	Fetch
011	FetchM
100	Read block
101	Write block
110	LD*x*_L
111	ST*x*_C

The transaction types must be held on cpuCReq<2:0> until the end of the transaction. The 21071-DA chip does not latch these signals.

Transactions on cpuCReq<2:0> are ignored by the 21071-DA chip when the bus is granted to 21071-DA chip. That is, from the cycle following ioGrant assertion to the cycle after cpuHoldAck and ioGrant deassertion at the end of the DMA transaction.

8.2.1.3 cpuCWMask<7:0>

Signal Type: 21071-DA Input
Signal Source: CPU
Input Sampling Clock Edge: clk1F

cpuCWMask<7:0> signals are used on CPU-initiated read block and write block transactions. These signals carry different information on both read and write block transactions.

On CPU write block and CPU ST*x*_C transactions, cpuCWMask<7:0> carry the longword mask for the whole cache line. An asserted cpuCWMask signal indicates that the corresponding longword from the cache line is valid and should be written.

On CPU read block and CPU LD*x*_L transactions, the cpuCWMask<7:0> signals carry additional information about the read transaction. cpuCWMask<1:0> carry address bits <4:3>, thereby indicating the address of the actual quadword to be returned. This information is used to implement quadword granularity to I/O space.

8.2.1.4 cpuHoldAck

Signal Type: 21071-DA Input
Signal Source: CPU
Input Sampling Clock Edge: clk1F

When cpuHoldAck is asserted in conjunction with ioGrant, the 21071-DA chip drives sysAdr<33:5> in the following cycle and may send out a valid DMA command on ioCmd<2:0>.

8.2.1.5 ioCmd<2:0>

Signal Type: 21071-DA Output
Signal Destination: 21071-CA
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1R

The 21071-DA chip asserts ioCmd<2:0> to request an action by the 21071-CA chip. When the 21071-DA chip owns the sysBus, ioCmd<2:0> signals are used to request a DMA transaction. When the CPU owns the bus, ioCmd<2:0> is used to request assertion of the cpuCAck and cpuDRAck signals. Table 8–3 lists the encodings for ioCmd<2:0>.

Table 8–3 ioCmd<2:0> Encodings

ioCmd<2:0>	CPU Owns sysBus	21071-DA Chip Owns sysBus
000	Idle	Idle
001	ClrLock	Flush
010	cpuDRAck OK_NCACHE_NCHK	Write
011	cpuDRAck OK_NCACHE	Write masked
100	cpuCAck OK	Read
101	cpuCAck HARD_ERROR	Read burst
110	cpuCAck SOFT_ERROR	Read wrapped
111	cpuCAck ST*x*C_FAIL	Read burst wrapped

8.2.1.6 ioCAck<1:0>

Signal Type: 21071-DA Input
Signal Source: 21071-CA
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1R

The 21071-CA chip asserts ioCAck<1:0> to acknowledge a DMA transaction. ioCAck<1:0> indicates that the DMA transaction has been completed. If any error occurs during the transaction, an error response is sent. Table 8–4 lists the encodings for ioCAck<1:0>.

Table 8–4 ioCAck<1:0> Encodings

ioCAck<1:0>	Function
00	Idle
01	Reserved/unused
10	DMA cycle acknowledge
11	DMA cycle error

8.2.1.7 ioDataRdy

Signal Type: 21071-DA Input
Signal Source: 21071-CA
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1R

When ioDataRdy is sampled asserted, the 21071-DA chip assumes that read data is available on epiData<31:7> in the following cycle.

If the 21071-DA chip samples ioCAck<1:0> = DMA cycle acknowledge without a prior assertion of ioDataRdy, it assumes that all the data will be available in the 21071-BA chips on the second subsequent cycle, and it does not wait for ioDataRdy to assert.

8.2.1.8 ioLineSel<1:0>

Signal Type: 21071-DA Output
Signal Destination: 21071-BA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

ioLineSel<1:0> is driven by the 21071-DA chip to the 21071-BA chips. During DMA read transactions, ioLineSel<1:0> indicates the DMA read buffer line that should be loaded. During DMA write transactions, ioLineSel<1:0> indicates the DMA write buffer line that has to be written to memory.

When the 21071-DA chip does not own the sysBus, the 21071-DA chip uses ioLineSel<1:0> to select the cache line of the I/O write buffer that should be loaded with CPU I/O write data.

8.2.1.9 ioRequest<1:0>

Signal Type: 21071-DA Output
Signal Destination: 21071-CA
Input Sampling Clock Edge: clk1F
Output Clock Edge: clk1R

The 21071-DA chip asserts ioRequest<1:0> to request ownership of sysAdr<33:5> to perform a DMA transaction. ioRequest<1:0> is acknowledged by the 21071-CA, using ioGrant. When a DMA transaction is started, ioRequest<1:0> is returned to idle in the cycle after ioCmd, if no further DMA transactions are required.

The 21071-DA chip uses the DMA request encoding on most DMA read and write transactions except in the following situations:

- The 21071-DA uses an atomic request to perform a DMA read prefetch.
- The 21071-DA uses an atomic request to perform a DMA read or write transaction following a scatter/gather map read.
- The 21071-DA chip uses the preempt request in order to flush the DMA write buffer on memory barriers.
- The 21071-DA chip uses the preempt request to prevent deadlock situations when an I/O transaction is stalled on the sysBus and a memory read targeted to the 21071-DA happens on the PCI, or when the write buffer is full.

Table 8–5 lists the encodings for ioRequest<1:0>.

Table 8–5 ioRequest<1:0> Encodings

ioRequest<1:0>	Function
00	Idle
01	DMA preempt request
10	DMA request
11	DMA atomic request

8.2.1.10 ioGrant

Signal Type: 21071-DA Input
Signal Source: 21071-CA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk1F

The assertion of ioGrant indicates to the 21071-DA chip that it has won ownership of the sysBus. The 21071-DA chip does not begin any new CPU transactions unless both ioGrant and cpuHoldAck are asserted. If the 21071-DA chip samples ioGrant deasserted in any cycle, it turns off its sysAdr drivers in the next clk1R.

The 21071-DA chip uses the ioGrant in combination with cpuHoldAck to determine if cpuCReq<2:0> should be ignored.

8.2.2 PCI Signals

For a detailed description of PCI interface pins, see the *PCI Local Bus Specification 2.0*. Table 8–6 provides a translation between the 21071-DA chip pin names and PCI specification signal names.

Table 8–6 Translation of 21071-DA Pin Names to PCI Signal Names

21071-DA Pin Name	PCI Signal Name
AD<31:0>	AD <31:0>
CBE_l<3:0>	C/BE#<3:0>
FrameL	FRAME#
TrdyL	TRDY#
IrdyL	IRDY#
StopL	STOP#
LockL	LOCK#
DevselL	DEVSEL#
Par	PAR
PerrL	PERR#
ReqL	REQ#
GntL	GNT#
pClk	CLK

8.2.2.1 AD<31:0>

Signal Type: Bidirectional (21071-DA, PCI devices)
Input Sampling Clock Edge: pClkR
Output Clock Edge: pClkR

This signal indicates multiplexed PCI address and data bus. During an address phase of a transaction, AD<31:0> contains a physical byte address. During subsequent data phases, AD<31:0> contains data.

A PCI bus transaction consists of one or two address phases followed by one or more data phases. The 21071-DA chip supports reads and writes and may act as initiator or target of a transaction on the bus.

8.2.2.2 CBE_l<3:0>

Signal Type: Bidirectional (21071-DA, PCI devices)
Input Sampling Clock Edge: pClkR
Output Clock Edge: pClkR

This signal communicates multiplexed bus command and byte enables. During an address phase of a transaction, CBE_l<3:0> contains the bus command that defines the type of PCI transaction. During data phases, CBE_l<3:0> contains byte enables dictating which byte lanes carry valid data. CBE_l<0> applies to byte 0, CBE_l<3> applies to byte 3.

8.2.2.3 FrameL

Signal Type: Bidirectional (21071-DA, PCI devices)
Input Sampling Clock Edge: pClkR
Output Clock Edge: pClkR

FrameL is driven by the initiator of the transaction to indicate the beginning and duration of an access on the PCI bus. FrameL assertion indicates the beginning of an access. While FrameL is asserted, data transfers continue. FrameL deassertion indicates the final data phase. The 21071-DA chip samples FrameL as an input and also drives FrameL when acting as the initiator of a transaction on the PCI bus.

8.2.2.4 TrdyL

Signal Type: Bidirectional (21071-DA, PCI devices)
Input Sampling Clock Edge: pClkR
Output Clock Edge: pClkR

This signal indicates the target agent's ability to complete the current data phase of a transaction on the PCI bus. The 21071-DA chip drives TrdyL when acting as a target on the PCI bus and samples TrdyL when acting as an initiator on the PCI bus.

8.2.2.5 IrdyL

Signal Type: Bidirectional (21071-DA, PCI devices)
Input Sampling Clock Edge: pClkR
Output Clock Edge: pClkR

This signal indicates the initiator's ability to complete the current data phase of a transaction on the PCI bus. The 21071-DA chip drives IrdyL when acting as an initiator on the PCI bus and samples IrdyL when acting as a target on the PCI bus.

8.2.2.6 StopL

Signal Type: Bidirectional (21071-DA, PCI devices)
Input Sampling Clock Edge: pClkR
Output Clock Edge: pClkR

This signal indicates that the current target is requesting the bus initiator to stop the current transaction on the PCI bus. The 21071-DA chip may drive StopL when acting as a target on the PCI bus, and it samples StopL when acting as an initiator on the PCI bus.

8.2.2.7 LockL

Signal Type: 21071-DA Input
Signal Source: PCI Devices
Input Sampling Clock Edge: pClkR

LockL indicates an atomic operation that may require multiple transactions to complete. The 21071-DA may be locked, but it will never request a lock. The 21071-DA treats the entirety of system memory as a single resource for the purposes of PCI exclusive access. The 21071-DA chip samples LockL when acting as a target on the PCI bus.

8.2.2.8 DevselL

Signal Type: Bidirectional (21071-DA, PCI devices)
Input Sampling Clock Edge: pClkR
Output Clock Edge: pClkR

The 21071-DA chip asserts DevselL through positive decoding of the address on AD<31:0>. The 21071-DA chip asserts DevselL when it is accepting a transaction for system memory. The 21071-DA chip samples DevselL when it is acting as an initiator on the PCI bus, and it expects DevselL to be asserted within five cycles of FrameL assertion. Otherwise, the transaction is terminated with an initiator abort.

8.2.2.9 Par

Signal Type: Bidirectional (21071-DA, PCI devices)
Input Sampling Clock Edge: pClkR
Output Clock Edge: pClkR

The Par signal is even parity, calculated on 36 bits comprised of AD<31:0> and CBE_l<3:0>. The Par signal is generated for all address and data phases and is valid one clock cycle after valid data or address is driven on AD<31:0>. The Par signal is driven and tristated identically to AD<31:0>, except that it is delayed one clock cycle. The Par signal is driven by the 21071-DA chip when acting as an initiator during address phases and write data phases. The Par signal is driven by the 21071-DA chip when acting as a target during read data phases. The Par signal is sampled as an input during all address phases and when acting as a target during write data phases.

8.2.2.10 PerrL

Signal Type: Bidirectional (21071-DA, PCI devices)
Input Sampling Clock Edge: pClkR
Output Clock Edge: pClkR

The PerrL signal is asserted when a data parity error is detected, and it corresponds to Par driven one clock cycle earlier. The 21071-DA chip may assert PerrL when it detects a write data parity error when acting as a target, or when it detects a read data parity error when acting as an initiator.

8.2.2.11 ReqL

Signal Type: 21071-DA Output
Signal Destination: PCI Arbiter
Output Clock Edge: pClkR

The 21071-DA chip asserts ReqL to indicate to the bus arbiter that it wants to use the PCI bus.

8.2.2.12 GntL

Signal Type: 21071-DA Input
Signal Source: PCI Devices
Input Sampling Clock Edge: pClkR

When GntL is asserted, it indicates to the 21071-DA chip that access to the PCI bus is granted. The 21071-DA chip may start a transaction as soon as GntL is asserted and the bus is idle.

8.2.2.13 pClk

Signal Type: 21071-DA Input
Signal Source: External Logic

The pClk signal provides timing for all transactions on the PCI bus. All PCI bus inputs are sampled on the rising edge of pClk, and all PCI bus outputs are driven from the rising edge of pClk. Frequencies supported by the bridge range from 0 to 33 megahertz.

8.2.3 PCI Sideband Signals

This section describes the PCI sideband signals.

8.2.3.1 MemReql

Signal Type: 21071-DA Input
Signal Source: ISA/EISA bridge chip
Input Sampling Clock Edge: pClkR

This signal is asserted by ISA/EISA bridge chips to indicate that an ISA/EISA device requires guaranteed access time (2.1 μs) to main memory. Refer to Section 9.4.3 for details. This is a PCI sideband signal.

8.2.3.2 MemAckl

Signal Type: 21071-DA Output
Signal Destination: External logic
Input Clock Edge: pClkR

This signal is asserted by the 21071-DA chip to indicate that guaranteed access time can be achieved on each subsequent PCI transaction directed toward main memory which is not retried by the 21071-DA chip. This is a PCI sideband signal.

8.2.4 epiBus Signals

This section describes the epiBus signals.

8.2.4.1 epiData<31:0>

Signal Type: Bidirectional (21071-BA, 21071-DA)
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

epiData<31:0> is a 32-bit bidirectional bus which connects the 21071-DA and 21071-BA chips. epiData<31:0> are driven on clk1R and is tristated on clk2F.

8.2.4.2 epiBEnErr<3:0>

Signal Type: Bidirectional (21071-BA, 21071-DA)
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

epiBEnErr<3:0> is timed with epiData<31:0>. During epiBus transfers to the 21071-BA chips, this field indicates which bytes of the longword on the epiData bus are valid. When an epiBEnErr bit is asserted, the corresponding byte is valid. The byte enable is used for DMA write transfers and is ignored on I/O read transfers.

During epiBus transfers from the 21071-BA chip DMA read and I/O write buffers, epiBEnErr<0> is asserted if the longword being sent on epiData contains a parity error or uncorrectable ECC error. epiBEnErr<1> is asserted if the longword being sent on epiData contained a correctable ECC error. Table 8–7 lists the epiBEnErr functions.

Table 8–7 epiBEnErr Functions

Signal	Transfers to 21071-BA	Transfers from 21071-BA
epiBEnErr<0>	epiData<7:0> byte enable	DMA read I/O write uncorrectable error (this longword)
epiBEnErr<1>	epiData<15:8> byte enable	DMA read I/O write corrected error (this longword)
epiBEnErr<2>	epiData<23:16> byte enable	Reserved
epiBEnErr<3>	epiData<31:24> byte enable	Reserved

8.2.4.3 epiAdr Signals

epiOWSel, epiLineSel<1:0>, epiSelDMA, epiFromIOB, epiEnable<3:0>, and epiLineInval are collectively referred to as the *epiAdr bus.* All these signals are set up one cycle prior to each epiData transfer to address a particular longword within the 21071-BA chip. A detailed description of each signal follows.

Note

epiEnable<3:0>, epiOWSel, epiLineSel, epiFromIOB, and epiSelDMA collectively address the contents of the 21071-BA chips. In a synchronous fashion, these address signals select data to be transferred in the subsequent cycle.

8.2.4.3.1 epiOWSel

Signal Type: 21071-DA Output
Signal Destination: 21071-DA, 21071-BA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

epiOWSel is driven by the 21071-DA chip to the 21071-BA chips on epiBus transfers. It is asserted to select the upper octaword within the current hexaword cache line and is to be read or written using the epiData bus. Table 8–8 lists the longword selection.

Table 8–8 Longword Selection

Longword Desired	21071-BA Chip Number	epiOWSel	epiEnable<3:0>
LW 0	0	0	0001
LW 1	1	0	0010
LW 2	0(2)[1]	0	0100
LW 3	1(3)[1]	0	1000
LW 4	0	1	0001
LW 5	1	1	0010
LW 6	0(2)[1]	1	0100
LW 7	1(3)[1]	1	1000

---

[1] The number in parenthesis indicates the 21071-BA chip number when four 21071-BA chips are used in the system.

8.2.4.3.2 epiLineSel<1:0>

Signal Type: 21071-DA Output
Signal Destination: 21071-BA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

epiLineSel<1:0> is driven by the 21071-DA chip to the 21071-BA chips. This field selects which cache line is sent from the DMA read and I/O write buffer to the 21071-DA chip or from the 21071-DA chip to the DMA write buffer using the epiData bus. This signal is ignored on 21071-DA to I/O read buffer transfers.

8.2.4.3.3 epiSelDMA

Signal Type: 21071-DA Output
Signal Destination: 21071-BA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

The epiSelDMA signal is asserted by the 21071-DA chip to indicate to the 21071-BA chips that the 21071-DA chip is performing a DMA transfer (to the DMA write buffer). When epiSelDMA is deasserted, the 21071-DA chip is performing an I/O transfer (to the I/O read buffer). epiSelDMA is used to select the transfer, as shown in Table 8–9.

8.2.4.3.4 epiFromIOB

Signal Type: 21071-DA Output
Signal Destination: 21071-BA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

The epiFromIOB signal is asserted by the 21071-DA chip to the 21071-BA chips to indicate that the 21071-DA chip is performing a transfer from the 21071-DA chip and to the 21071-BA chips. When epiFromIOB is driven low, the 21071-DA chip is performing transfer from the 21071-BA chips to the 21071-DA chip. epiFromIOB is used to select the transfer, as shown in Table 8–9.

8.2.4.3.5 epiEnable<3:0>

Signal Type: 21071-DA Output
Signal Destination: 21071-BA
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

The epiEnable<3:0> signals are asserted by the 21071-DA to the 21071-BA to indicate that the 21071-DA is performing an epiBus transfer. When epiEnable is driven low, the epiData and epiBus control signals are ignored by the 21071-BA chips. Each bit of epiEnable<3:0> corresponds to one of four longwords. Table 8–9 lists the epiBus interface functions.

Table 8–9 21071-BA epiBus Interface Function

epiEnable	epiFromIOB	epiSelDMA	Function
0	X	X	No action except for possible line invalidate; epiData tristated.
1	0	X	The DMA read and I/O write buffer is driven onto epiData.
1	1	0	epiData is loaded into the I/O read buffer.
1	1	1	epiData is loaded into the DMA write buffer.

8.2.4.3.6 epiLineInval

Signal Type: 21071-DA Output
Signal Destination: 21071-BA
Input Sampling Clock Edge: clk2F
Output Clock Edge: clk1R

epiLineInval is asserted during 21071-DA to 21071-BA transfers to indicate that the cache line being loaded should be invalidated. All byte enables for that line must be cleared. For the invalidate to take place, epiFromIOB is asserted. (epiEnable must be ignored.) epiLineInval is asserted by the 21071-DA chip when the first longword of data is loaded into a new cache line from epiData.

8.2.4.4 Miscellaneous/Clock Signals

This section describes the miscellaneous and clock signals.

8.2.4.4.1 intHw0

Signal Type: 21071-DA Output
Signal Destination: External logic
Output Clock Edge: clk1F

The intHw0 interrupt pin is an output from the 21071-DA chip and is connected to one of the irq<5:0> pins of the DECchip 21064 microprocessor through the interrupt control/configuration PAL. This signal is asserted when the 21071-DA chip detects certain errors in the transactions it processes. intHw0 is kept asserted until all such error conditions are cleared.

8.2.4.4.2 resetL

Signal Type: 21071-DA Input
Signal Source: External logic
Input Clock Edge: Asynchronous on assertion, clk1R on deassertion

Assertion of resetL sets all internal logic and state machines in the 21071-DA chip to their initialized states.

8.2.4.4.3 clk1x2

Signal Type: 21071-DA Input
Signal Source: Clock generator

clk1x2 is a clock input which supplies a clock at twice the frequency of the DECchip 21064 sysClkOut1, with a minimum period of 15 ns, and a 50% duty cycle.

8.2.4.4.4 clk2ref

Signal Type: 21071-DA Input
Signal Source: Clock generator

clk2ref is a signal input which is low when the assertion of clk1x2 corresponds to the assertion of sysClkOut1. The received signal must be set up to the assertion of clk1x2.

8.2.4.5 Test Signals

This section describes the test signals.

8.2.4.5.1 testMode

Signal Type: 21071-DA Input
Signal Source: Test logic
Input Clock Edge: Asynchronous

Assertion of testMode places the chip into a mode for chip testing. testMode is intended to be used only during chip testing and must be tied low during normal system operation.

testMode has a weak internal pull down and a Schmitt trigger input.

8.2.4.5.2 scanEn

Signal Type: 21071-DA Input
Signal Source: Test logic

Assertion of scanEn places all internal flops in their scan state. scanEn is intended to be used only during chip testing and must be tied low during normal system operation.

scanEn has a weak internal pull down and a Schmitt trigger input.

8.2.4.5.3 tristate_l

Signal Type: 21071-DA Input
Signal Source: External logic
Input Clock Edge: Asynchronous

Assertion of this signal tristates all output and bidirectional drivers. tristate_l is intended to be used only during chip testing and power-up.

tristate_l has a weak internal pull up and a Schmitt trigger input.

8.2.4.5.4 pTestout

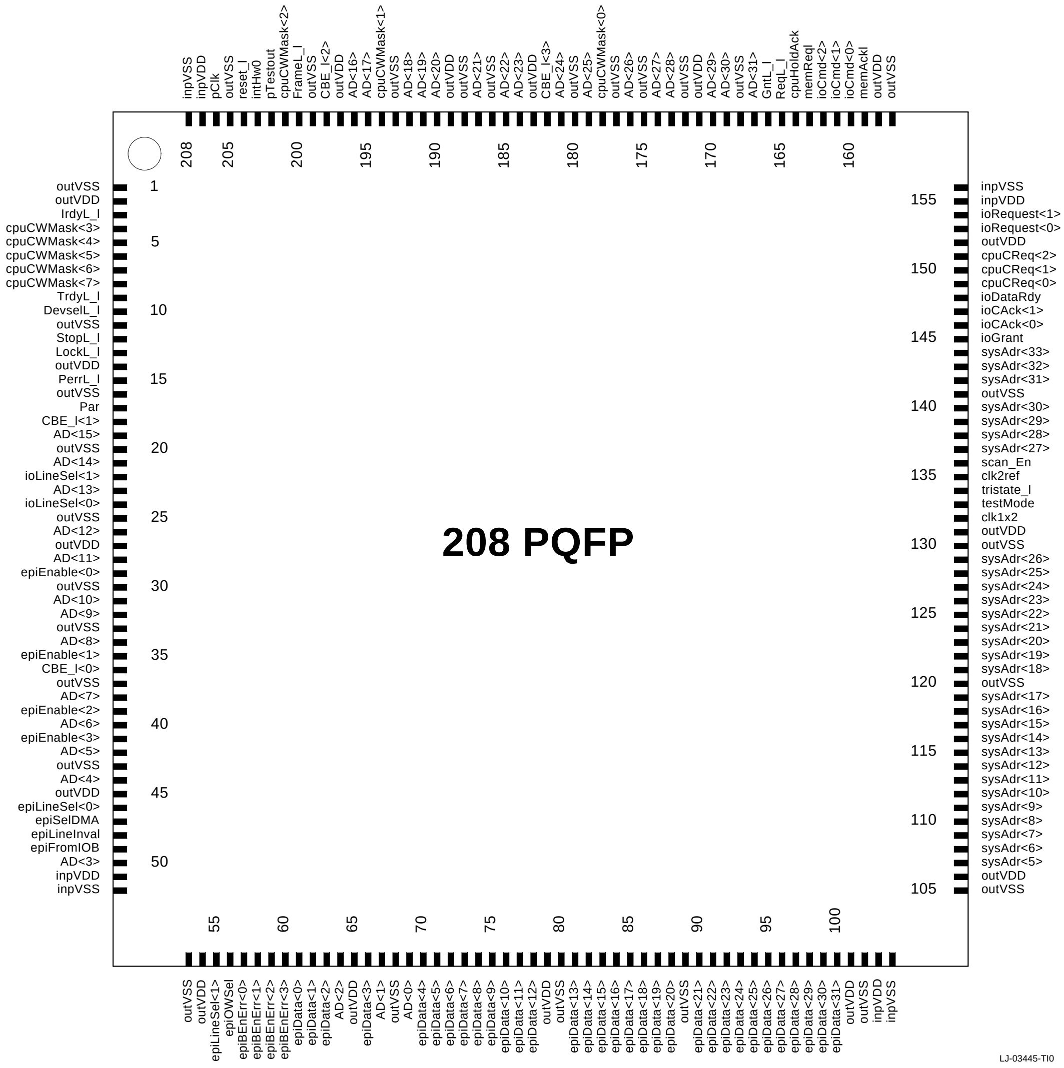
Signal Type: 21071-DA Output
Signal Source: Test logic
Output Clock Edge: Flow through

The pTestout signal contains the output from the parametric NAND tree, as required for testability. The testMode signal must be asserted for pTestout to be valid. pTestout is intended for use only during chip testing.

8.3 DECchip 21071-DA Pin Assignment

The DECchip 21071-DA chip is a 208-pin plastic quad flat pack (PQFP). Figure 8–1 shows the signal assignments. Sections 8.3.1 and 8.3.2 provide alphabetical and numerical pin listings.

Figure 8–1 DECchip 21071-DA Pinout Diagram



8.3.1 DECchip 21071-DA Alphabetical Pin Assignment List

Table 8–10 lists the DECchip 21071-DA pins in alphabetical order. The following list describes the abbreviations used in the Type column of the table.

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

Table 8–10 DECchip 21071-DA Alphabetical Pin Assignment List

Pin Name	Pin	Type	Pin Name	Pin	Type
AD<0>	69	B	**AD<26>**	176	B
AD<1>	67	B	**AD<27>**	174	B
AD<2>	64	B	**AD<28>**	173	B
AD<3>	50	B	**AD<29>**	170	B
AD<4>	44	B	**AD<30>**	169	B
AD<5>	42	B	**AD<31>**	167	B
AD<6>	40	B	**CBE<0>**	36	B
AD<7>	38	B	**CBE<1>**	18	B
AD<8>	34	B	**CBE<2>**	198	B
AD<9>	32	B	**CBE<3>**	182	B
AD<10>	31	B	**clk1x2**	132	I
AD<11>	28	B	**clk2Ref**	135	I
AD<12>	26	B	**cpuCReq<0>**	149	I
AD<13>	23	B	**cpuCReq<1>**	150	I
AD<14>	21	B	**cpuCReq<2>**	151	I
AD<15>	19	B	**cpuCWMask<0>**	178	I
AD<16>	196	B	**cpuCWMask<1>**	194	I
AD<17>	195	B	**cpuCWMask<2>**	201	I
AD<18>	192	B	**cpuCWMask<3>**	4	I
AD<19>	191	B	**cpuCWMask<4>**	5	I
AD<20>	190	B	**cpuCWMask<5>**	6	I
AD<21>	187	B	**cpuCWMask<6>**	7	I
AD<22>	185	B	**cpuCWMask<7>**	8	I
AD<23>	184	B	**cpuHoldAck**	164	I
AD<24>	181	B	**DevselL**	10	B
AD<25>	179	B	**epiBEnErr<0>**	57	B

Pin Name	Pin	Type	Pin Name	Pin	Type
epiBEnErr<1>	58	B	**epiData<26>**	95	B
epiBEnErr<2>	59	B	**epiData<27>**	96	B
epiBEnErr<3>	60	B	**epiData<28>**	97	B
epiData<0>	61	B	**epiData<29>**	98	B
epiData<1>	62	B	**epiData<30>**	99	B
epiData<2>	63	B	**epiData<31>**	100	B
epiData<3>	66	B	**epiEnable<0>**	29	O
epiData<4>	70	B	**epiEnable<1>**	35	O
epiData<5>	71	B	**epiEnable<2>**	39	O
epiData<6>	72	B	**epiEnable<3>**	41	O
epiData<7>	73	B	**epiFromIOB**	49	O
epiData<8>	74	B	**epiLineInval**	48	O
epiData<9>	75	B	**epiLineSel<0>**	46	O
epiData<10>	76	B	**epiLineSel<1>**	55	O
epiData<11>	77	B	**epiOWSel**	56	O
epiData<12>	78	B	**epiSelDMA**	47	O
epiData<13>	81	B	**FrameL**	200	B
epiData<14>	82	B	**Gntl**	166	I
epiData<15>	83	B	**inpVdd**	51	P
epiData<16>	84	B	**inpVdd**	103	P
epiData<17>	85	B	**inpVdd**	155	P
epiData<18>	86	B	**inpVdd**	207	P
epiData<19>	87	B	**inpVss**	104	P
epiData<20>	88	B	**inpVss**	52	P
epiData<21>	90	B	**inpVss**	156	P
epiData<22>	91	B	**inpVss**	208	P
epiData<23>	92	B	**intHw0**	203	O
epiData<24>	93	B	**ioCAck<0>**	146	I
epiData<25>	94	B	**ioCAck<1>**	147	I

Pin Name	Pin	Type	Pin Name	Pin	Type
ioCmd<0>	160	O	**outVss**	199	P
ioCmd<1>	161	O	**outVss**	180	P
ioCmd<2>	162	O	**outVss**	188	P
ioDataRdy	148	I	**outVss**	11	P
ioGrant	145	I	**outVss**	193	P
ioLineSel<0>	24	O	**outVss**	168	P
ioLineSel<1>	22	O	**outVss**	172	P
ioRequest<0>	153	O	**outVss**	102	P
ioRequest<1>	154	O	**outVss**	43	P
IrdyL	3	B	**outVss**	30	P
LockL	13	B	**outVss**	89	P
MemAckl	159	O	**outVss**	177	P
MemReql	163	I	**outVss**	120	P
outVdd	45	P	**outVss**	186	P
outVdd	183	P	**outVss**	105	P
outVdd	106	P	**outVss**	141	P
outVdd	27	P	**outVss**	53	P
outVdd	171	P	**outVss**	37	P
outVdd	14	P	**outVss**	16	P
outVdd	131	P	**outVss**	1	P
outVdd	79	P	**outVss**	25	P
outVdd	54	P	**outVss**	20	P
outVdd	65	P	**outVss**	68	P
outVdd	158	P	**outVss**	157	P
outVdd	152	P	**outVss**	33	P
outVdd	101	P	**outVss**	175	P
outVdd	2	P	**outVss**	130	P
outVdd	189	P	**outVss**	205	P
outVdd	197	P	**Par**	17	B
outVss	80	P	**pClk**	206	I

Pin Name	Pin	Type	Pin Name	Pin	Type
PerrL	15	B	**sysAdr<24>**	127	B
pTestOut	202	O	**sysAdr<25>**	128	B
ReqL	165	O	**sysAdr<26>**	129	B
resetL	204	I	**sysAdr<27>**	137	B
scanEn	136	I	**sysAdr<28>**	138	B
StopL	12	B	**sysAdr<29>**	139	B
sysAdr<5>	107	B	**sysAdr<30>**	140	B
sysAdr<6>	108	B	**sysAdr<31>**	142	B
sysAdr<7>	109	B	**sysAdr<32>**	143	B
sysAdr<8>	110	B	**sysAdr<33>**	144	B
sysAdr<9>	111	B	**testMode**	133	I
sysAdr<10>	112	B	**TrdyL**	9	B
sysAdr<11>	113	B	**triState_l**	134	I
sysAdr<12>	114	B			
sysAdr<13>	115	B			
sysAdr<14>	116	B			
sysAdr<15>	117	B			
sysAdr<16>	118	B			
sysAdr<17>	119	B			
sysAdr<18>	121	B			
sysAdr<18>	121	B			
sysAdr<19>	122	B			
sysAdr<20>	123	B			
sysAdr<21>	124	B			
sysAdr<22>	125	B			
sysAdr<23>	126	B			

8.3.2 Numerical DECchip 21071-DA Pin Assignment List

Table 8–11 lists the DECchip 21071-DA pins in numerical order. The following list describes the abbreviations used in the Type column of the table.

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

Table 8–11 DECchip 21071-DA Numerical Pin Assignment List

Pin Name	Pin	Type	Pin Name	Pin	Type
outVss	1	P	**AD<12>**	26	B
outVdd	2	P	**outVdd**	27	P
IrdyL	3	B	**AD<11>**	28	B
cpuCWMask<3>	4	I	**epiEnable<0>**	29	O
cpuCWMask<4>	5	I	**outVss**	30	P
cpuCWMask<5>	6	I	**AD<10>**	31	B
cpuCWMask<6>	7	I	**AD<9>**	32	B
cpuCWMask<7>	8	I	**outVss**	33	P
TrdyL	9	B	**AD<8>**	34	B
DevSelL	10	B	**epiEnable<1>**	35	O
outVss	11	P	**CBE<0>**	36	B
StopL	12	B	**outVss**	37	P
LockL	13	B	**AD<7>**	38	B
outVdd	14	P	**epiEnable<2>**	39	O
PerrL	15	B	**AD<6>**	40	B
outVss	16	P	**epiEnable<3>**	41	O
Par	17	B	**AD<5>**	42	B
CBE<1>	18	B	**outVss**	43	P
AD<15>	19	B	**AD<4>**	44	B
outVss	20	P	**outVdd**	45	P
AD<14>	21	B	**epiLineSel<0>**	46	O
ioLineSel<1>	22	O	**epiSelDMA**	47	O
AD<13>	23	B	**epiLineInval**	48	O
ioLineSel<0>	24	O	**epiFromIOB**	49	O
outVss	25	P	**AD<3>**	50	B

Pin Name	Pin	Type	Pin Name	Pin	Type
inpVdd	51	P	**outVdd**	79	P
inpVss	52	P	**outVss**	80	P
outVss	53	P	**epiData<13>**	81	B
outVdd	54	P	**epiData<14>**	82	B
epiLineSel<1>	55	O	**epiData<15>**	83	B
epiOWSel	56	O	**epiData<16>**	84	B
epiBEnErr<0>	57	B	**epiData<17>**	85	B
epiBEnErr<1>	58	B	**epiData<18>**	86	B
epiBEnErr<2>	59	B	**epiData<19>**	87	B
epiBEnErr<3>	60	B	**epiData<20>**	88	B
epiData<0>	61	B	**outVss**	89	P
epiData<1>	62	B	**epiData<21>**	90	B
epiData<2>	63	B	**epiData<22>**	91	B
AD<2>	64	B	**epiData<23>**	92	B
outVdd	65	P	**epiData<24>**	93	B
epiData<3>	66	B	**epiData<25>**	94	B
AD<1>	67	B	**epiData<26>**	95	B
outVss	68	P	**epiData<27>**	96	B
AD<0>	69	B	**epiData<28>**	97	B
epiData<4>	70	B	**epiData<29>**	98	B
epiData<5>	71	B	**epiData<30>**	99	B
epiData<6>	72	B	**epiData<31>**	100	B
epiData<7>	73	B	**outVdd**	101	P
epiData<8>	74	B	**outVss**	102	P
epiData<9>	75	B	**inpVdd**	103	P
epiData<10>	76	B	**inpVss**	104	P
epiData<11>	77	B	**outVss**	105	P
epiData<12>	78	B	**outVdd**	106	P

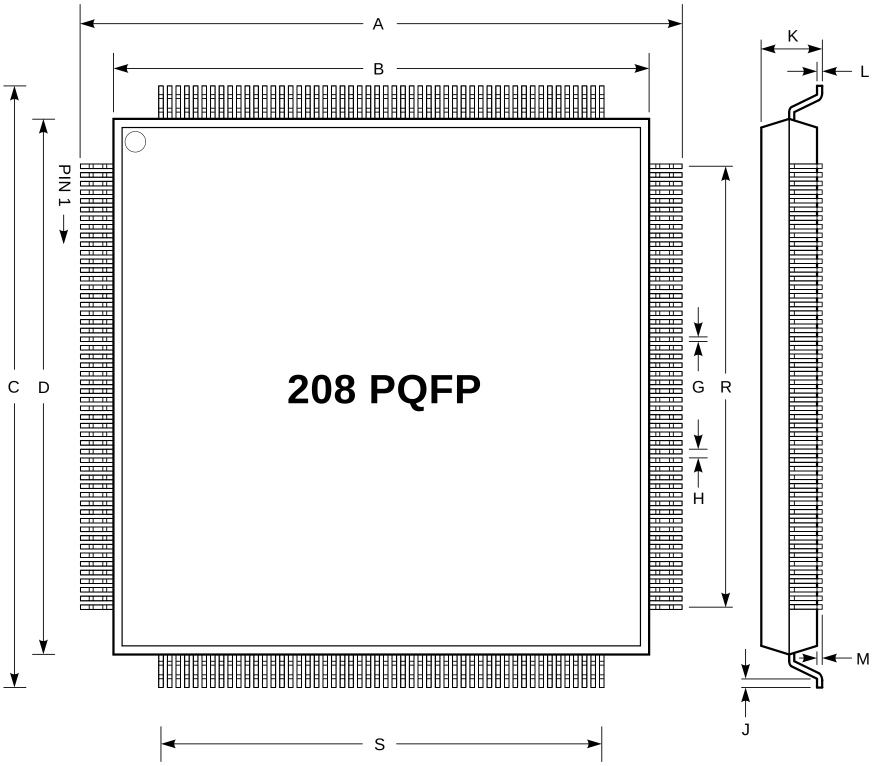
Pin Name	Pin	Type	Pin Name	Pin	Type
sysAdr<5>	107	B	**sysAdr<28>**	138	B
sysAdr<6>	108	B	**sysAdr<29>**	139	B
sysAdr<7>	109	B	**sysAdr<30>**	140	B
sysAdr<8>	110	B	**outVss**	141	P
sysAdr<9>	111	B	**sysAdr<31>**	142	B
sysAdr<10>	112	B	**sysAdr<32>**	143	B
sysAdr<11>	113	B	**sysAdr<33>**	144	B
sysAdr<12>	114	B	**ioGrant**	145	I
sysAdr<13>	115	B	**ioCAck<0>**	146	I
sysAdr<14>	116	B	**ioCAck<1>**	147	I
sysAdr<15>	117	B	**ioDataRdy**	148	I
sysAdr<16>	118	B	**cpuCReq<0>**	149	I
sysAdr<17>	119	B	**cpuCReq<1>**	150	I
outVss	120	P	**cpuCReq<2>**	151	I
sysAdr<18>	121	B	**outVdd**	152	P
sysAdr<19>	122	B	**ioRequest<0>**	153	O
sysAdr<20>	123	B	**ioRequest<1>**	154	O
sysAdr<21>	124	B	**inpVdd**	155	P
sysAdr<22>	125	B	**inpVss**	156	P
sysAdr<23>	126	B	**outVss**	157	P
sysAdr<24	127	B	**outVdd**	158	P
sysAdr<25>	128	B	**MemAckl**	159	O
sysAdr<26>	129	B	**ioCmd<0>**	160	O
outVss	130	P	**ioCmd<1>**	161	O
outVdd	131	P	**ioCmd<2>**	162	O
clk1x2	132	I	**MemReql**	163	I
testMode	133	I	**cpuHoldAck**	164	I
tristate_l	134	I	**ReqL**	165	O
clk2Ref	135	I	**GntL**	166	I
scanEn	136	I	**AD<31>**	167	B
sysAdr<27>	137	B	**outVss**	168	P

Pin Name	Pin	Type	Pin Name	Pin	Type
AD<30>	169	B	**AD<16>**	196	B
AD<29>	170	B	**outVdd**	197	P
outVdd	171	P	**CBE<2>**	198	B
outVss	172	P	**outVss**	199	P
AD<28>	173	B	**FrameL**	200	B
AD<27>	174	B	**cpuCWMask<2>**	201	I
outVss	175	P	**pTestOut**	202	O
AD<26>	176	B	**intHw0**	203	O
outVss	177	P	**resetL**	204	I
cpuCWMask<0>	178	I	**outVss**	205	P
AD<25>	179	B	**pClk**	206	I
outVss	180	P	**inpVdd**	207	P
AD<24>	181	B	**inpVss**	208	P
CBE<3>	182	B			
outVdd	183	P			
AD<23>	184	B			
AD<22>	185	B			
outVss	186	P			
AD<21>	187	B			
outVss	188	P			
outVdd	189	P			
AD<20>	190	B			
AD<19>	191	B			
AD<18>	192	B			
outVss	193	P			
cpuCWMask<1>	194	I			
AD<17>	195	B			

8.4 DECchip 21071-DA Mechanical Specifications

Figure 8–2 shows DECchip 21071-DA package dimensions.

Figure 8–2 DECchip 21071-DA Package Dimensions



DIM	Millimeters		Inches	
	MIN	MAX	MIN	MAX
A	30.50	30.77	1.201	1.211
B	27.90	28.10	1.098	1.106
C	30.50	30.77	1.201	1.211
D	27.90	28.10	1.098	1.106
G	0.23	0.33	0.009	0.013
H	.500 BSC		0.0197 BSC	
J	0.45	0.62	0.018	0.024
K	3.45	3.85	0.136	0.152
L	0.13	0.23	0.005	0.009
M	0.25	0.35	0.010	0.012
R	25.5 REF		1.004 REF	
S	25.5 REF		1.004 REF	

LJ-03666-TI0

9

DECchip 21071-DA Architecture Overview

This chapter describes the 21071-DA architecture. The 21071-DA chip is a bridge between the PCI local bus and the Alpha 21064 microprocessor, its Bcache, and memory. The 21071-DA chip contains all the control functions of the bridge, as well as some data path functions. Other data path functions reside within the 21071-BA chip.

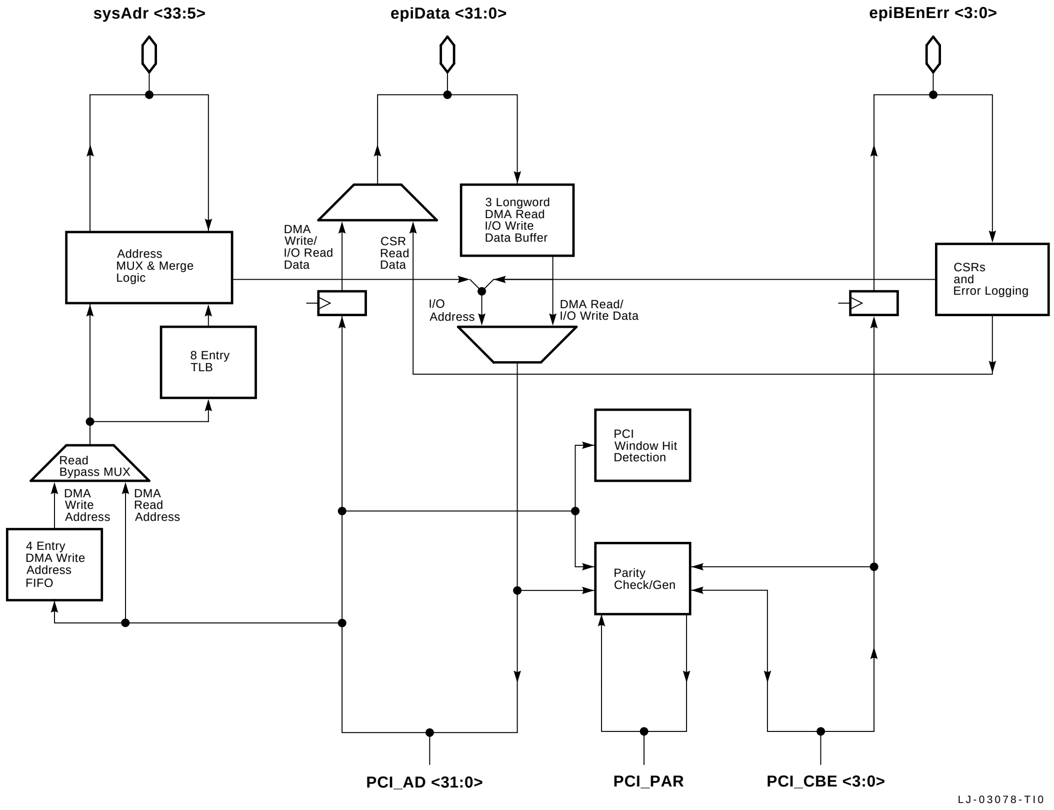
The 21071-DA chip can be divided into two major sections:

- sysBus (processor, memory) interface
- PCI interface

The following sections provide an overview of the architectural features of the sysBus and PCI interfaces.

Figure 9–1 shows a block diagram of the DECchip 21071-DA chip.

Figure 9–1 DECchip 21071-DA Block Diagram



9.1 sysBus Interface Architecture

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

9.1.1 Address Decode

The 21071-DA chip provides logic for translating and extending the DECchip 21064 34-bit physical address space into 32-bit PCI address space and vice versa. The address decode in the 21071-DA chip uses the address mapping and translation scheme described in Section 10.1 to generate PCI addresses on CPU-initiated transactions. All systems using the 21071-DA chip are required to follow this address mapping scheme.

9.1.2 Buffering for I/O Write Transactions

The 21071-DA chip supports *write-and-run* I/O write transactions using a 1-entry deep write buffer. The address and control mechanism are in the 21071-DA chip; the corresponding data is stored in the 21071-BA chip.

As soon as an I/O write transaction is received on the sysBus, the data and address is loaded in the write buffer and the transaction is acknowledged on the sysBus. Subsequent I/O transactions to the 21071-DA chip are not acknowledged until the previous I/O write transaction is completed. The I/O write could be directed towards the 21071-DA CSRs or the PCI bus.

The 21071-BA chip provides a holding buffer to store write data for one subsequent write transaction. If the I/O write buffer is occupied, and another I/O write to the 21071-DA chip appears on the sysBus, the data of that write is captured from the sysBus and is loaded into the holding buffer. Even though the data is loaded into the holding buffer, the sysBus transaction is stalled until the I/O write buffer is free. The holding buffer is required so that all the write data can be captured before suspending the write transaction for deadlock resolution. See Section 9.4.2 for details.

The description of the holding buffer and I/O write buffer is a conceptual one. In the actual implementation there are two data buffers, and they alternate as I/O write and holding buffers.

9.1.3 Buffering for I/O Read Data

The 21071-DA chip provides data buffering for one I/O read transaction initiated by the CPU. The I/O read buffer resides in the 21071-BA chip, but is controlled by the 21071-DA chip. The I/O read buffer is only a temporary holding buffer, and is invalidated at the end of every I/O read transaction. The I/O read buffer is loaded with data received from the PCI or the 21071-DA CSRs depending on whether the transaction is addressed to the PCI or the CSRs. The I/O read buffer is necessary to make the sysBus interface and PCI interfaces independent of each other. An I/O read may complete on the PCI, while the sysBus interface is busy flushing DMA writes to memory. (This is done by suspending the I/O read transaction using a preempt DMA request to the sysBus arbiter; see Sections 9.4.2 and 9.4.1 for details.) The I/O read buffer allows the PCI transaction to terminate without waiting for the read data to be returned to the CPU.

9.1.4 Wrapping for I/O Transactions

The CPU must be configured in wrap mode for I/O reads to function correctly. The requested quadword is the only one that is returned on I/O read transactions.

9.2 PCI Interface Architecture

The PCI interface of the 21071-DA chip is a fully compliant PCI host bridge. It behaves as a master on the PCI on CPU-initiated transactions and is a target on memory space transactions initiated by other PCI masters. The architectural features of the PCI interface are described in the following sections.

9.2.1 DMA Address Translation

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

If the address region is direct mapped, then the PCI address is directly sent out on the sysBus. Higher order sysBus address bits have to be obtained from the PCI base address registers in the 21071-DA chip.

If the address region is scatter/gather mapped, the PCI address indicates the address of a page table entry, which contains the physical address of that page. Thus, there is a virtual (PCI address) to physical translation involved. The actual scatter/gather map is stored in memory. The 21071-DA chip accesses the map in memory to do all the required translation. To improve the performance of scatter/gather mapped DMA transactions, the 21071-DA chip implements an 8-entry translation lookaside buffer (TLB). Incoming PCI addresses to scatter/gather regions are looked up in the TLB. If there is a hit, the translation is done within the 21071-DA chip. If there is a miss, then the 21071-DA chip reads memory (through the sysBus) to obtain the required page table entry. The entry is then loaded into the TLB; a round-robin replacement scheme is used. The translation is done by the 21071-DA chip and the transaction is completed on the sysBus.

For details about the actual mapping scheme and the page table entry format, see Chapter 10.

Note

The slave machine of the DECchip 21071-DA PCI interface will not respond to a CPU-initiated address that has been driven onto the PCI by the master machine of the PCI interface, even if the address hits the programmed PCI DMA window. That is, the DECchip 21071-DA chip does not support loopback mode on the PCI.

9.2.2 DMA Write Buffer

The PCI interface has a write buffer for buffering DMA write data. The DMA write buffer is made up of four entries; each entry contains a cache line address, 8 longwords of data, the byte enables corresponding to each longword, a lock bit, a mask bit, a flush bit, and a valid bit for the entry. The untranslated PCI address is stored in the DMA write buffer. Address translation is performed when the particular entry is unloaded from the DMA write buffer. The address and control bits are stored in the 21071-DA chip, and corresponding data and byte enables are stored in the 21071-BA chip.

Data is received on the PCI and is transferred to the 21071-BA chip over the epiData bus. When the transaction is completed on the PCI, the entry is marked valid and is available for unloading. A subsequent PCI write transaction to the same cache line will consume a separate write buffer entry. The 21071-DA chip does not support merging of write transactions.

DMA reads are allowed to bypass the writes in the DMA write buffer, depending on the state of the dByp<1:0> bits from the DCSR. This improves average DMA read latency considerably, because most DMA reads are not expected to match addresses in the write buffer. When the dByp<1:0> mode indicates full bypass, read address bits <31:6> are compared with those of the buffered writes. If there is no match, the read is serviced ahead of the writes. When the dByp<1:0> mode indicates partial bypass, read bypass happens only if the read page offset does not match the write page offset; only address bits <12:6> are compared. This mode can be used if comparing virtual (PCI) addresses between reads and writes is not desirable or could lead to coherency problems. In the No_Bypass mode, DMA reads are stalled until all the DMA writes have been flushed out of the write buffer.

There are two situations when read bypassing is disabled independent of the programmed value of dByp<1:0>:

- The 21071-DA chip does not allow DMA reads to bypass buffered DMA writes if any of the buffered writes were locked by a PCI master.
- The DMA write buffer has to be flushed to memory on memory barriers from the DECchip 21064 microprocessor to ensure data coherency. The 21071-DA chip does not permit DMA reads to bypass buffered DMA writes while flushing the write buffer. See Section 9.4.1.

If the DMA write buffer is full and a DMA write to memory is initiated on the PCI, the transaction is disconnected by the 21071-DA PCI interface without accepting any data. If the buffer is filled during a PCI DMA write transaction, the transaction is disconnected, and no more data is accepted by the 21071-DA PCI interface.

9.2.3 DMA Read Buffer

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

The DMA read buffer does not require an address to be stored, because the contents of the buffer are invalidated at the end of the current PCI read transaction. There is *never* any stale data in the DMA read buffer.

9.2.4 PCI Burst Length and Prefetching

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

- When a burst order other than linear incrementing is specified by the master, the transaction length is kept to one transfer. See Section 9.2.5.
- When the transaction starts on an even cache line boundary, but PCI address <4:2> are non-zero. In this case the first cache line is a partial write.

- When the transaction starts on an odd cache line boundary, PCI address <5> = 1. The burst length is this case is ≤ 8 longwords.
- If there is only one cache line entry available in the DMA write buffer, the burst is terminated after ≤ 8 longwords of data have been transferred, even if the transfer started on an even cache line boundary. This is because after that cache line has been loaded into the write buffer, the buffer is full.

On DMA read transactions, a maximum burst length of 8 longwords is supported if DMA prefetching is not enabled in the 21071-DA chip, and a PCI read multiple command was not used by the requesting device. A maximum burst length of 16 longwords is supported if DMA prefetching is enabled in the 21071-DA chip, or a PCI read multiple command was used by the requesting device. The following describes the various cases of DMA read burst transactions and indicates the burst length.

- When a burst order other than linear incrementing is specified by the master, the burst length is kept to 1 longword. See Section 9.2.5. Prefetching is performed if prefetching is enabled or a read multiple command is specified on the PCI, and the transaction starts on an even cache line.
- When prefetching is not enabled and the incoming PCI command is not a read multiple, and when the PCI transaction starts with PCI address<4:2> = 0, the PCI interface disconnects the transaction after 8 longwords have been transferred on the PCI. No prefetching is performed.
- When prefetching is not enabled and the incoming PCI command is not a read multiple, and when the PCI transaction starts with a non-zero value on PCI address<4:2>, the PCI interface disconnects the transaction after ≤ 7 longwords have been transferred on the PCI. No prefetching is performed.
- When prefetching is enabled or a read multiple command is specified on the PCI, and the transaction starts on an even cache line boundary with PCI address<4:2> = 0, the PCI interface disconnects the transaction after 16 longwords have been transferred on the PCI. The odd cache line is prefetched.
- When prefetching is enabled or a read multiple command is specified on the PCI, and the transaction starts on an even cache line boundary with a non-zero value on PCI address<4:2>, the PCI interface disconnects the transaction after ≤ 15 longwords have been transferred on the PCI. The odd cache line is prefetched.

- When prefetching is enabled or a read multiple command is specified on the PCI, and the transaction starts on an odd cache line boundary, the PCI interface disconnects the transaction after ≤ 8 longwords have been transferred on the PCI. No prefetching is performed.

On CPU-initiated read transactions, when the 21071-DA chip is a master on the PCI, a maximum burst length of 2 is supported.

On CPU-initiated write transactions, when the 21071-DA chip is a master on the PCI, a maximum burst length of 2 is supported in sparse memory and I/O spaces, and a maximum burst length of 8 is supported in dense memory space.

9.2.5 PCI Burst Order

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

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

9.2.6 PCI Parity Support

All PCI devices are required to generate parity across AD<31:0> (data and address lines) and C/BE#<3:0> (command/byte enables). The 21071-DA chip complies with this specification.

When it is master of the PCI, it also checks the incoming parity on I/O reads, interrupt vector reads, and configuration reads during data phases.

When it is a target on the PCI, it checks parity during the address phase, and during data phases on memory write transactions.

9.2.7 PCI Exclusive Access

The 21071-DA chip supports the PCI Exclusive Access protocol using the LockL signal. A locked transaction to main memory on the PCI causes the PCI interface to lock out all non-exclusive main memory accesses initiated by PCI masters. This is done by disconnecting the PCI transaction without completing any data transfers. Until the Lock is cleared on the PCI, only the PCI master that locked main memory is allowed to complete transactions to main memory. Refer to the *PCI Local Bus Specification* for details.

On the sysBus side, the PCI lock causes the system lock flag to be cleared by using the ioClrLock command encoding on the ioCmd signals. The system lock flag is held cleared until all locked DMA reads and locked DMA writes to memory have been completed on the sysBus, and the Lock is cleared on the PCI.

As a master on the PCI, the 21071-DA chip does not initiate locked transactions. See Section 11.2.3 for a detailed description of the 21071-DA response to locked transactions initiated by other master devices.

9.2.8 PCI Bus Parking

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

The 21071-DA chip also supports PCI bus parking during reset. If the GntL signal is asserted by the PCI arbiter (ReqL is always tristated by the 21071-DA chip during reset), the 21071-DA chip will drive AD<31:0>, BE<3:0>#, and (one clock cycle later) PAR. When GntL is deasserted, the 21071-DA chip tristates these signals.

9.2.9 PCI Retry Timeout

The 21071-DA chip implements a timeout mechanism to terminate CPU-initiated transactions that do not complete on the PCI because of too many disconnects or retries. When it initiates a CPU transaction on the PCI, the 21071-DA chip counts the number of times it gets retried or disconnected, and if the number exceeds 2^{24} it flags an error to the CPU and aborts the transaction. The 21071-DA chip considers losing GntL during address stepping (for configuration cycles) as a disconnect for the purposes of this timeout mechanism.

9.2.10 PCI Master Timeout

The PCI protocol specifies a mechanism to limit the duration of a master's burst sequence. The mechanism requires a PCI master to implement a latency timer that counts the number of cycles since the assertion of FRAME#. If the master latency timer has expired and the master's grant has been taken away, the master is required to surrender the bus. This mechanism is intended to prevent masters from holding bus ownership for extended periods of time, trading off high throughput for low latency. The 21071-DA implements a programmable master latency timer.

9.2.11 Address Stepping in Configuration Cycles

The 21071-DA chip does not have dedicated IDSEL# pins for use in PCI configuration cycles. Because AD<31:11> are not used during configuration cycles, they are connected to the IDSEL# pins of the various PCI devices. These devices can then uniquely be selected during configuration cycles by using addresses which assert only one bit of AD<31:11> at a time. By doing this, an added load is presented to those address lines that are connected to the IDSEL# pins of PCI devices. This load can be reduced by resistively coupling the line to the pin; however, the time for the signal to become valid at the IDSEL# pin is then increased.

In order to provide flexibility and reduce design complexity when using resistive coupling to IDSEL# pins, the 21071-DA chip performs address stepping on configuration reads and write transactions. For these transactions, the 21071-DA chip will drive the PCI bus for two clock cycles during the address phase in order for the IDSEL# pins of all the PCI devices to reach a valid logic level.

The 21071-DA chip does not perform address or data stepping in any other case.

9.3 Transactions

This section describes the transactions performed by the 21071-DA chip.

9.3.1 sysBus Transactions

The 21071-DA chip is a master and a slave on the sysBus. When it is a master, it performs DMA transactions on the sysBus. When it is a slave, it responds to I/O transactions initiated by the CPU.

9.3.1.1 CPU-Initiated Transactions

When the CPU is master of the sysBus, the sysBus interface monitors the commands and addresses sent out by the CPU. If the addresses are within the 21071-DA chip address range, and the command is valid, the 21071-DA chip responds to the transaction. The 21071-DA chip does not acknowledge the CPU directly. Acknowledgments are communicated to the CPU through the 21071-CA chip, and data is communicated through the 21071-BA chips.

The following transactions are supported by the 21071-DA sysBus interface:

- Read Block to Remote (PCI) Space

The 21071-DA chip responds to the transaction by notifying the 21071-CA chip when data is ready in the I/O read data buffer. The 21071-DA chip may choose to preempt this transaction if a DMA read transaction is in progress on the PCI, or if the DMA write buffer is full, and the DMA transaction needs to get on to the sysBus (deadlock resolution).

The 21071-DA chip supports longword or quadword reads in PCI space.

- Read Block to Local (CSR) Space

  This is treated similarly to the read block to remote space. The only difference is in the conditions for preemption. This transaction is preempted only if it is stalled on the sysBus and queued behind an I/O write transaction that cannot be completed until a DMA transaction on the PCI can access the sysBus (deadlock resolution).

  The 21071-DA chip supports only longword reads aligned on cache line boundaries in CSR space.

- Write Block to Remote or Local Space

  The 21071-DA chip acknowledges the transaction when all previous I/O writes have been completed on the PCI. This transaction is preempted only if it is stalled on the sysBus and queued behind an I/O write transaction that cannot be completed until a DMA transaction on the PCI can get onto the sysBus (deadlock resolution).

  The 21071-DA chip supports only longword writes in CSR space, up to quadword writes in sparse PCI space, and up to 8 longword writes in dense PCI memory space.

- LD*x*_L to I/O Space

  This is treated just like a read block to I/O space.

- ST*x*_C to I/O Space

  This is treated just like a write block to I/O space.

- Barrier

  The 21071-DA chip uses the barrier command to ensure synchronization between the CPU and DMA devices on the PCI. It does not acknowledge the command until the I/O write buffer has been flushed, and any writes that were present in the DMA write buffer when the barrier command was received have been flushed to memory.

- Fetch, FetchM to 21071-DA Space

  The 21071-DA chip does not do anything special on a fetch, fetchM transaction to its address space. It sends an acknowledgment to the 21071-CA chip as soon as it sees the command on the bus.

9.3.1.2 PCI-Initiated Transactions

Transactions from PCI devices to main memory cause the 21071-DA chip to arbitrate for the sysBus and perform DMA transactions to memory.

DMA transactions on the sysBus always start with an arbitration cycle where the 21071-DA chip asserts one of the three possible request codes to the arbiter. When the grant is received, the address is driven on to the sysAdr bus, which is common to the CPU, the Bcache, the 21071-CA, and the 21071-DA chips. Data is transferred between the 21071-BA and the 21071-DA chips using the epiBus prior to the start of a DMA write or as data is returned on DMA reads.

The sysBus interface uses the atomic request when it has to prefetch read data or when it needs to perform a scatter/gather lookup. It does at most two memory read transactions during such a request. It uses the preempt request, when it has to suspend the current CPU transaction, which is targeted towards it, in order to let a DMA transaction complete (Section 9.4.2). At all other times, it uses the normal DMA request.

The following DMA transactions are performed by the 21071-DA chip on the sysBus:

- PCI DMA Read

  On a PCI DMA read transaction, the 21071-DA chip could use one of four sysBus DMA read commands:

  - DMA read
  - DMA read wrapped
  - DMA read burst
  - DMA read burst wrapped

The wrapped qualifier is used to indicate whether the lower octaword of data from the cache line is requested or the upper octaword is requested. The wrapped command is used when the upper octaword is requested.

The burst qualifier is a hint to the memory controller that the following read transaction is likely to be in the same page. The burst command is used when the 21071-DA chip is likely to prefetch data from memory. The maximum data that can be prefetched is a cache line; a memory read on the PCI can be at most 16 longwords (Section 9.2.4). The 21071-DA chip uses DMA read burst on the first cache line read indicating that it is going to follow it up with another read. The second read uses the DMA read command because that cache line is the end of the burst.

- Scatter/Gather Read

  21071-DA chip performs a DMA read or DMA read wrapped transaction on the sysBus when it needs to read the scatter/gather map.

- PCI DMA Write

  A PCI DMA write transaction causes a DMA write full or a DMA write masked command on the sysBus. A DMA write full command is used when the whole cache line will be written to memory, and a DMA write masked command is used when the cache line has to be written partially. The byte masks for the data are transferred to the 21071-BA chip along with the data.

9.3.2 PCI Transactions

The 21071-DA chip supports the following transactions on the PCI:

- Interrupt acknowledge: PCI master.
- Special cycle: PCI master.
- I/O read: PCI master.
- I/O write: PCI master.
- Memory read: A slave, when the transaction is initiated by another PCI device accessing system memory. A master, when the CPU is accessing an address in PCI memory space.
- Memory write: A slave, when the transaction is initiated by another PCI device accessing system memory. A master, when the CPU is accessing an address in PCI memory space.
- Configuration read: PCI master.
- Configuration write: PCI master.

- Memory write and invalidate: PCI slave; treated just like a memory write.
- Memory read line: PCI slave; treated just like a memory read.
- Memory read multiple: PCI slave; cache line read prefetch is performed irrespective of the state of the prefetch enable bit.
- Dual address cycles: Ignored.

9.4 Miscellaneous Architectural Issues

This section describes the miscellaneous architectural issues, including:

- Data coherency
- Deadlock resolution
- Guaranteed access time mode support

9.4.1 Data Coherency

There are generally two agents in the system whose data transfer actions need to be synchronized:

- The CPU
- A remote PCI device

The 21071-DA chip maintains data coherency and synchronization between these two agents using the following mechanisms:

- The 21071-DA chip preserves strict ordering of DMA writes initiated on the PCI.
- DMA reads can bypass writes that are not to the same address (double cache line). Strict ordering is maintained between reads and writes to the same address.
- I/O transfers from the CPU to the PCI or to 21071-DA CSRs are performed in order. This policy guarantees a coherent view of PCI I/O space from the CPU viewpoint.
- The 21071-DA chip flushes DMA write data to memory prior to acknowledging a barrier command from the CPU. Because explicit ordering commands are absent on the PCI, the software MB instruction is used to order CPU and DMA accesses.

- The 21071-DA chip also flushes the I/O write buffer to the PCI before acknowledging a barrier command. This preserves the order between CPU I/O accesses and CPU memory accesses.
- The 21071-DA chip clears the system lock flag on PCI exclusive reads and writes to system memory.

9.4.2 Deadlock Resolution

In a 21071 or 21072 system, two major buses are allocated for use during data transfers—the sysBus and the PCI. Some data transfers require the use of both of these buses to complete. In particular, CPU I/O transfers to or from the PCI require ownership of the sysBus followed by ownership of the PCI. Similarly, PCI DMA transfers to or from the memory subsystem require ownership of the PCI followed by ownership of the sysBus.

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

For any transfer requiring the use of both buses, the 21071-DA chip is responsible for acquiring the second level bus on behalf of the initiator. Deadlocks can occur when the CPU and a remote PCI agent have both initiated transfers requiring the use of both the sysBus and the PCI bus. If the CPU has already acquired the sysBus, and the PCI agent has already acquired the PCI bus, the 21071-DA chip would be unable to complete either transaction without resolving the deadlock.

The 21071-DA chip resolves deadlock by forcing the CPU to relinquish ownership of the sysBus thereby giving priority to the PCI agent. By giving priority to the PCI agent, the 21071-DA chip gives the system designer more flexibility when choosing PCI devices. In particular, this flexibility allows designers to choose devices that resort to using PCI disconnect when handling deadlock situations that arise at their end. The 21071-DA chip forces the CPU to relinquish the sysBus by using a preempt request while arbitrating for the sysBus.

9.4.3 Guaranteed Access Time Mode Support for Intel 82375EB and 82378IB ISA/EISA Bridges

The Intel 82375EB and 82378IB EISA/ISA bridges (EIB) provide three sideband signals to provide mechanisms for flushing system write buffers and to allow a guaranteed access time of 2.1 μs to a master on the ISA/EISA bus. The three signals are:

- FLUSHREQ#
- MEMREQ#
- MEMACK#

The first two are outputs from the EIB, and the last one is an input to the EIB.

- The EIB asserts MEMREQ# and FLUSHREQ# when it requires guaranteed access from memory. It expects the host bridge to assert MEMACK# when it has cleared the path to memory. This is accomplished by flushing any posted writes and disabling the posting of any further writes, thereby guaranteeing an access time of 2.1 μs on the bus.
- The EIB keeps MEMREQ# deasserted and asserts FLUSHREQ# when it requires that posted writes from the CPU to the PCI and from the PCI to the CPU be flushed to prevent deadlocks between a DMA request from an ISA master and an ISA bus access from the host bridge. In this case too, it expects to see MEMACK# asserted when the appropriate buffers have been flushed.

The 21071-DA chip provides its own mechanism for deadlock prevention, by preempting CPU transactions to allow DMA transactions to complete. It therefore does not need to support the deadlock prevention mechanism of the EIB, and it does not implement the FLUSHREQ# protocol.

Note

Because the 21071-DA chip does not implement FLUSHREQ#, external logic must force the assertion of MEMACK# to the EIB upon the assertion of FLUSHREQ#. The equation for MEMACK# going to the EIB should be as follows:

EIB_MEMACK# = NOT ((MEMREQ# AND (NOT FLUSHREQ#)) OR (NOT DA_MEMACK#))

To meet MEMACK# timing requirements, the output of this equation may need to be clocked through a flip-flop. The equation and output flip-flop could both be implemented in a single PAL.

9.4.3.1 DECchip 21071-DA GAT Mode Operation

The 21071-DA chip takes the following action when it sees MemReql asserted:

1. The 21071-DA chip turns down its DMA *write-and-run* buffering capacity to a single burst of up to eight longwords. Any PCI write transaction directed toward the 21071-DA chip will be retried by the 21071-DA chip unless its DMA write buffer is empty. Read-bypass-write flows remain enabled.
2. The 21071-DA chip requests the sysBus (ioRequest = regular or preempt).
3. Once granted the sysBus, the 21071-DA chip holds the sysBus grant (ioRequest = atomic) until MemReql is deasserted.
4. With the sysBus grant, the 21071-DA chip flushes all DMA write buffers (if non-empty) and then performs a flush transaction (ioCmd = Flush) to ensure that posted writes in the 21071-CA chip have completed. At the end of the flush transaction, the 21071-DA chip asserts MemAckl.
5. While MemReql continues to be asserted, the 21071-DA chip will continue to service PCI transactions with *write-and-run* buffering capacity set to a single hexaword. The 21071-DA chip performs a flush transaction on the sysBus atomically following each DMA write transaction. Read-bypass-write flows will not be exercised at this point because the 21071-DA chip is holding the sysBus grant. (DMA writes will always start on the sysBus before a DMA read is far enough along on the PCI to bypass the DMA write.)
6. Upon deassertion of MemReql, the 21071-DA chip deasserts MemAckl, returns DMA *write-and-run* buffering capacity to four hexawords, releases the sysBus grant (ioRequest = regular, preempt, or idle), and no longer performs flush transactions following DMA writes.

9.4.3.2 GAT Mode System Requirements

Although the 21071 and 21072 chipsets provide the functionality described here in order to help guarantee access time, the system designer must ensure that worst case latencies are not excessive. The following system parameters must be considered:

- DRAM width
- DRAM access time
- sysClk speed
- Scatter/gather mapping of GAT mode devices
- Scatter/gather mapping of all other PCI devices

Analysis shows that 30 ns (PCI cycle time) systems with any one of the following characteristics will be able to meet GAT mode latency requirements:

- Systems with 128-bit wide memory interfaces.
- Systems that do not scatter/gather map GAT mode devices.
- Systems that do not scatter/gather map any other PCI devices.
- Systems with 64-bit wide memory interfaces that utilize 60 ns DRAMs with appropriately programmed memory timing (assuming that either the refresh time does not exceed 180 ns, or the video support feature of the 21071-CA chip is not utilized).
- Systems that implement PCI arbiters that do not allow third-party scatter/gather-mapped writes to sneak in ahead of the GAT mode read.

Systems that do not conform to any of the above specifications (including all systems with 40 ns PCI cycle times) will require further analysis to determine if GAT mode latency requirements can be met. The following sequence illustrates the worst case scenario and should be used as a guideline for further analysis.

1. In the time between the 21071-DA chip's assertion of MemAckl and when the EISA/ISA bridge (EIB) acquires the PCI bus to perform the GAT mode read, a third-party PCI device acquires the PCI bus and performs a scatter/gather-mapped partial write to main memory.
2. The scatter/gather-mapped write is posted to the 21071 (or 21072) chipset, misses in the TLB, and therefore results in a scatter/gather read from memory followed by a masked write to memory.

3. As soon as the scatter/gather-mapped write completes on the PCI bus, the EIB can acquire the PCI bus, grant the EISA/ISA bus to the GAT mode device, and start the GAT mode read. Our analysis assumes that the GAT interval begins four PCI cycles (one EISA/ISA cycle) after the PCI bus is detected as idle following the third-party scatter/gather-mapped write. The GAT mode read will never bypass the third-party DMA write inside the 21071-DA because the DMA write starts on the sysBus immediately after it completes on the PCI (sysBus is held atomically after MemAckl is asserted). The GAT mode read will start on the PCI while the scatter/gather read for the third-party DMA write is in progress on the sysBus.
4. If the read from the GAT mode device is also scatter/gather mapped (and also misses in the TLB), the total latency from the start of the GAT interval to the return of requested read data will include the time required to perform:
    1. Part of the scatter/gather read for third-party masked DMA write
    2. The third-party masked DMA write
    3. The scatter/gather read for the GAT mode DMA read
    4. The GAT mode DMA read

    Our analysis assumes that the GAT interval ends eight PCI cycles (two EISA/ISA cycles) after the first requested longword is transferred on the PCI bus.

The total latency of these four transactions, plus the time required for one memory refresh transaction, plus the time for a 21071-CA video transaction (if this feature is in use), plus the eight tail-end cycles, must not exceed the GAT latency limit of 2.1 µs.

Note that a major portion of the latency can be eliminated if one of the following occurs:

- The GAT mode device, all other PCI devices, or both, are not scatter/gather mapped.
- The PCI arbiter prevents third-party writes from sneaking in before the GAT mode read.

The previous measures allow more systems, including certain 40 ns systems, to meet the GAT mode latency requirements; however, system designers must analyze their implementations in order to accurately estimate worst-case latencies.

9.5 Interrupts

The 21071-DA chip interrupts the CPU using the intHw0 signal when it has errors to report. The 21071-DA chip does not distinguish between hard and soft errors when asserting this interrupt signal. However, the software can mask the assertion of the interrupt signal on soft (correctable) errors by disabling error correction reporting using the dCEI bit in the DCSR register.

The 21071-DA chip does not provide an interval timer interrupt. This functionality is expected to be provided to the CPU by some other device in the system. In addition, interrupts from other PCI devices or from a PCI interrupt controller must be sent directly to the CPU.

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

9.6 Error Handling

This section describes how errors are handled by the 21071-DA chip. The following descriptions assume that the 21071-DA error registers are not already locked by a previous error condition. If the 21071-DA errors registers are locked by an earlier error, then additional errors merely set the lost error bit, and, if appropriate, cause the 21071-DA chip to assert intHw0.

IntHw0 is kept asserted as long as the corresponding error bit is set.

The PCI error address register (PEAR) logs addresses sent out or received on the PCI. The sysBus error address register (SEAR) logs addresses sent out or received on the sysBus. The error logging CSRs are described in further detail in Chapter 10.

9.6.1 CPU-Initiated Transactions

The 21071-DA chip always returns HARD_ERROR on ioCmd<2:0> field on I/O read transactions that have errors. No interrupt is asserted in this case, because the microprocessor has been notified that the read had an error. In no situation does the 21071-DA chip assert SOFT_ERROR on I/O read transactions because the microprocessor would interpret it as a failure that occurred during the transaction, but was corrected.

I/O writes are always acknowledged with OK (100 on cpuCAck<2:0>). Because of the *write-and-run* feature for I/O writes in the 21071-DA chip, the transaction is always acknowledged on the sysBus before it is initiated on the PCI. An interrupt (intHw0) will assert to notify the microprocessor if an error occurs on the PCI during the I/O write.

The actions taken on the various errors that can occur on CPU-initiated transactions are described in the following sections.

9.6.1.1 No Device Error

On an I/O transaction initiated by the 21071-DA chip, if DEVSEL# is not asserted within 5 cycles, the 21071-DA chip assumes that no PCI device is going to respond to this transaction. The following action is taken:

- The 21071-DA chip terminates the PCI transaction using the master-abort protocol.
- The nDev bit is set in the DCSR. The pci_Cmd field is set to the appropriate value depending upon the transaction.
- The PCI error address register (PEAR) contains the address sent out at the beginning of the PCI transaction and is locked.
- On writes, intHw0 signal is asserted to interrupt the processor.
- On reads, the 21071-DA chip forces the value 101 (cpuCAck HARD_ERROR) on ioCmd<2:0> to end the sysBus transaction.
- To clear the error, a logic 1 must be written to the nDev bit in the DCSR.

9.6.1.2 Target Abort Errors

On an I/O transaction initiated by the 21071-DA chip, if the target device terminates the PCI transaction using the target-abort protocol, the following action is taken:

- The 21071-DA chip, as master, terminates the PCI transaction in accordance with the target-abort protocol.
- The tAbt bit is set in the DCSR, and the pci_Cmd field is set to the appropriate value depending upon the transaction.
- The PCI error address register (PEAR) contains the address sent out at the beginning of the PCI transaction and is locked.
- On writes, intHw0 signal is asserted to interrupt the processor.
- On reads, the 21071-DA chip forces the value 101 (cpuCAck HARD_ERROR) on ioCmd<2:0> to end the sysBus transaction.
- To clear the error, a logic 1 must be written to the tAbt bit in the DCSR.

9.6.1.3 Address Parity Errors

On any I/O transaction there is no way for the 21071-DA chip to determine that a parity error occurred in the address phase of the transaction because the 21071-DA chip does not have a SERR# pin. (PERR# is not used to convey address parity error information.) As a result, the 21071-DA chip can take no action.

9.6.1.4 Read Data Parity Errors

On an I/O read transaction initiated by the 21071-DA chip, if the parity generated off the incoming data sampled from the PCI AD lines (data), and the byte enables driven by the 21071-DA chip are different from the value sampled from PAR, a read data parity error condition has occurred. The following action is taken:

- The transaction continues normally.
- The 21071-DA chip asserts PERR# on the PCI.
- The ioPE bit is set in the DCSR, and the pci_Cmd field is set to the appropriate value depending upon the transaction.
- The PCI error address register (PEAR) contains the address sent out at the beginning of the PCI transaction and is locked. (Note: If an error occurs on both longwords of a quadword transaction, then the lost bit will be set.)

- The 21071-DA chip forces the value 101 (cpuCAck HARD_ERROR) on ioCmd<2:0> to end the sysBus transaction.
- To clear the error, a logic 1 must be written to ioPE bit in the DCSR.

9.6.1.5 Write Data Parity Errors

On an I/O write transaction initiated by the 21071-DA chip, if PERR# is asserted by the slave device for any longword of data, a write data parity error condition has occurred. The following action is taken:

- The transaction completes normally on the PCI.
- The ioPE bit is set in the DCSR, and the pci_Cmd field is set to the appropriate value depending upon the transaction.
- The PCI error address register (PEAR) contains the address sent out at the beginning of the PCI transaction and is locked. (Note: If an error occurs on more than 1 longword of a single write burst the lost bit will be set.)
- intHw0 signal is asserted to interrupt the processor.
- To clear the error, a logic 1 must be written to the ioPE bit in the DCSR.

9.6.1.6 Retry Timeout

On an I/O transaction initiated by the 21071-DA chip, if the retry timeout counter overflows (this happens when the 21071-DA chip has been retried 2^{24} times by the target), the 21071-DA chip does the following:

- The 21071-DA chip does not retry the transaction on the PCI again.
- The ioRT bit is set in the DCSR and the pci_Cmd field is set to the appropriate value depending upon the transaction.
- The PCI error address register (PEAR) contains the address sent out at the beginning of the PCI transaction and is locked.
- intHw0 signal is asserted to interrupt the processor.
- On reads, the 21071-DA chip forces the value 101 (cpuCAck HARD_ERROR) on ioCmd<2:0> to end the sysBus transaction.
- To clear the error, a logic 1 must be written to the ioRT bit in the DCSR.

9.6.2 DMA Transactions

All DMA transaction errors will be flagged by interrupting the processor (intHw0 asserted) when the error occurs, except where noted.

9.6.2.1 Address Parity Errors

On any DMA (PCI-initiated) transaction address phase, if the generated parity of the incoming address and command sampled from the PCI AD and C/BE# lines is different from the value sampled from PAR, an address parity error condition has occurred for that transaction. The following action is taken:

The 21071-DA chip does not respond to the transaction. Due to the parity error, the 21071-DA chip is not certain if the command was correct (read or write) and is not sure what the intended address for that transaction was. (PERR# is not asserted because it is only intended for data parity errors on the PCI.) intHw0 is not asserted.

9.6.2.2 Read Data Parity Errors

On a DMA read transaction data phase, if there is a parity error it might be detected by the PCI master device. Even if this device asserts PERR# and the 21071-DA chip takes no action, it is the PCI master's responsibility to handle the error condition. intHw0 is not asserted.

9.6.2.3 Write Data Parity Errors

On any DMA write transaction data phase, if the generated parity of the incoming data and byte enables sampled from the PCI AD and C/BE# lines is different from the value sampled from PAR, a data parity error condition has occurred. The following action is taken:

- The 21071-DA chip asserts PERR# pin for one cycle on the PCI two cycles after the data was transferred on the bus, to indicate the condition.
- The dDPE bit is set in the DCSR.
- The PCI error address register (PEAR) contains the address that came off the PCI bus at the beginning of the transaction and is locked.
- intHw0 signal is asserted to interrupt the processor.
- The write will continue normally on the PCI.
- That particular cache line entry will not be written to memory.
- To clear the error, a logic 1 must be written to the dDPE bit in the DCSR.

9.6.2.4 Memory Errors

On a DMA transaction, if the 21071-CA chip detects an error (non-existent memory address, tag address parity error, or tag control parity error), it will log the address and the specific error bit, and terminate the sysBus transaction by driving 11 (DMA cycle error) on ioCAck<1:0>. The following action is taken if data was going to be transferred on the PCI.

Note

> Prefetched cache line data may not be required by the PCI device. If there is a tag address parity error or tag control parity error on an unused prefetched cache line, the error will be logged by the 21071-CA chip, but the CPU will not be interrupted by the 21071-DA chip.

- The mErr bit is set.
- The sysBus error address register (SEAR) contains the address that caused the memory error and is locked.
- intHw0 signal is asserted to interrupt the processor.
- On reads, the 21071-DA chip terminates the PCI transaction using the target-abort protocol.
- On writes, the 21071-DA chip dismisses the write buffer entry (single cache line). Note that if a single PCI write burst crossed a cache line boundary and therefore filled two write buffer entries (two cache lines), each entry is handled separately on the sysBus.
- To clear the error, a logic 1 must be written to the mErr bit in the DCSR.

9.6.2.5 Read Correctable Data Error

On a DMA read transaction, if there is a correctable error in memory or Bcache and the 21071-BA chips are configured in ECC mode, the 21071-BA chips will correct the longword with the single-bit error before sending it to the 21071-DA chip.

If and when this longword is sent to the 21071-DA chip, along with the data on the epiData bus, epiBEnErr<1> will contain information whether or not this longword had a correctable error. The following action is taken if the longword was going to be transferred on the PCI.

- If the Disable Correctable Error Interrupt bit (dCEI) is set, the information on epiBEnErr<1> is ignored. As a result, intHw0 will not be asserted, the cMRD bit will not be set, and the error address will not be logged in SEAR.
- intHw0 signal is asserted to interrupt the microprocessor.
- No error occurs on the PCI and the transaction completes normally.
- The cMRD bit is set in the DCSR.
- The sysBus error address register (SEAR) contains the address that caused the correctable error and is locked.
- To clear the error, a logic 1 must be written to the cMRD bit in the DCSR.

9.6.2.6 Read Uncorrectable Data Error

On a DMA read transaction, if there is an uncorrectable error (parity error or double-bit ECC error) in memory or Bcache, the 21071-BA chips will inform the 21071-DA chip when they send the bad data over the epiData bus.

If and when this longword is sent to the 21071-DA chip, along with the data on the epiData bus, epiBEnErr<0> will contain information whether this longword had an uncorrectable error or not. The following action is taken if the longword was going to be transferred on the PCI. (Note: In some cases, not all longwords of a cache line will be transferred.)

- intHw0 is asserted to interrupt the microprocessor.
- The uMRD bit is set.
- The sysBus error address register (SEAR) contains the address that caused the uncorrectable error and is locked.
- The 21071-DA chip terminates the PCI transaction using the target-abort mechanism.
- To clear the error, a logic 1 must be written to the uMRD bit in the DCSR.

9.6.2.7 Scatter/Gather Entry Invalid Errors

On scatter/gather mapped DMA transactions, the scatter/gather entry being accessed might be invalid. The actual write to or read from memory will not occur. The following action is taken:

- iPTL bit is set in the DCSR.
- The PCI error address register (PEAR) contains the address that caused the error and is locked.
- intHw0 signal is asserted to interrupt the processor.

- If the scatter/gather read was for a DMA read, the 21071-DA terminates the PCI transaction using the target abort protocol.
- If the scatter/gather read was for a DMA write, the 21071-DA dismisses the write buffer entry (single cache line). Note that if a single PCI write burst crossed a cache line boundary and therefore filled two write buffer entries (two cache lines), each entry is handled separately on the sysBus.
- To clear the error, a logic 1 must be written to the iPTL bit in the DCSR.

9.6.2.8 Write Correctable and Uncorrectable Data Errors

If a DMA write is not a full hexaword, the 21071-CA chip performs a read-modify-write. If an error is detected on the read from memory before the write is done, the 21071-DA chip does not perform any action. See Section 16.2.2 for details about how the 21071-BA chip handles data errors on DMA write transactions.

9.6.2.9 Scatter/Gather Correctable Data Error

On a scatter/gather read transaction, if there is a correctable error in memory or Bcache and the 21071-BA chips are configured in ECC mode, the 21071-BA chips will correct the longword with the single-bit error before sending it to the 21071-DA chip. When this longword is sent to the 21071-DA chip, along with the data on the epiData bus, epiBEnErr<1> will contain information whether or not this longword had a correctable error. The following action is taken:

- If the Disable Correctable Error Interrupt bit (dCEI) is set, the information on epiBEnErr<1> is ignored. As a result, intHw0 will not be asserted, the cMRD bit will not be set, and the error address will not be logged in SEAR.
- intHw0 signal is asserted to interrupt the microprocessor.
- No error occurs on the PCI and the transaction completes normally.
- The cMRD bit is set in the DCSR.
- The sysBus error address register (SEAR) contains the address that caused the correctable error and is locked.
- To clear the error, a logic 1 must be written to the cMRD bit in the DCSR.

9.6.2.10 Scatter/Gather Uncorrectable Data Error

On a scatter/gather read transaction, if there is an uncorrectable error (parity error or double-bit ECC error) in memory or Bcache, the 21071-BA chips will inform the 21071-DA chip when it sends this data over the epiData bus.

When this longword is sent to the 21071-DA chip, along with the data on the epiData bus, epiBEnErr<0> will contain information whether or not this longword had an uncorrectable error. The following action is taken:

- intHw0 is asserted to interrupt the microprocessor.
- The uMRD bit is set.
- The sysBus error address register (SEAR) contains the address that caused the uncorrectable error and is locked.
- If the scatter/gather read was for a DMA read, the 21071-DA terminates the PCI transaction using the target abort protocol.
- If the scatter/gather read was for a DMA write, the 21071-DA dismisses the write buffer entry (single cache line). Note that if a single PCI write burst crossed a cache line boundary and therefore filled two write buffer entries (two cache lines), each entry is handled separately on the sysBus.
- To clear the error, a logic 1 must be written to the mErr bit in the DCSR.

9.6.2.11 Scatter/Gather Memory Errors

On a DMA transaction, if the 21071-CA detects an error (non-existent memory address, tag address parity error, or tag control parity error) during a scatter/gather read transaction, it will terminate the sysBus transaction. The 21071-CA will log the address and the specific error bit, and terminate the sysBus transaction by driving 10 (DMA cycle error) on ioCAck<1:0>. The following actions are taken by the 21071-DA:

- The mErr bit is set in the DCSR.
- The sysBus Error Address Register (SEAR) contains the address that caused the memory error and is locked.
- intHw0 signal is asserted to interrupt the processor.
- If the scatter/gather read was for a DMA read, the 21071-DA terminates the PCI transaction using the target abort protocol.

- If the scatter/gather read was for a DMA write, the 21071-DA dismisses the write buffer entry (single cache line). Note that if a single PCI write burst crossed a cache line boundary and therefore filled two write buffer entries (two cache lines), each entry is handled separately on the sysBus.
- To clear the error, a logic 1 must be written to the mErr bit in the DCSR.

10

DECchip 21071-DA Programmer's Reference

This chapter provides information about DECchip 21071-DA address translation. It also describes the DECchip 21071-DA internal registers.

10.1 Address Translation

This section describes the mapping of the 34-bit processor physical address space to 32-bit PCI address space, and the translation of the 32-bit PCI addresses to 34-bit physical memory space.

Note

The slave machine of the DECchip 21071-DA PCI interface will not respond to a CPU-initiated address that has been driven onto the PCI by the master machine of the PCI interface, even if the address hits the programmed PCI DMA window. That is, the DECchip 21071-DA chip does not support loopback mode on the PCI.

10.1.1 CPU Address Mapping to PCI Space

The 34-bit physical sysBus address space is divided to form:

- Memory address space
- Local I/O space (local I/O space is used for CSRs in the 21071-CA and 21071-DA chips)

- PCI space

  The PCI defines three physical address spaces:

  - PCI memory (for memory residing on the PCI)
  - PCI I/O space
  - PCI configuration space

In addition to these three address spaces on the PCI, the sysBus I/O space is also used to generate PCI interrupt acknowledge cycles and PCI special cycles. Table 10–1 shows the sysBus address mapping required to generate these address spaces.

Table 10–1 sysBus Address Map

sysAdr<33:32>	sysAdr<31:28>	Address Space	Notes
00	XXXX	Cacheable memory space	The 21071-DA chip does not respond to addresses in this space.
01	0XXX	Noncacheable memory space	The 21071-DA chip does not respond to addresses in this space.
01	100X	21071-CA CSRs	The 21071-DA chip does not respond to addresses in this space.
01	1010	21071-DA CSRs	The 21071-DA chip will respond to all addresses in this space. Dstream access only.
01	1011	PCI interrupt acknowledge or PCI special cycle	A read causes a PCI interrupt acknowledge cycle; a write causes a special cycle. Dstream access only.

(continued on next page)

Table 10–1 (Cont.) sysBus Address Map

sysAdr<33:32>	sysAdr<31:28>	Address Space	Notes
01	110X	PCI sparse I/O space	16 MB of PCI space. Lower 256 KB of this space must be used for addressing PCI, EISA, and ISA devices. The rest of the space can be used for other devices. Dstream access only.
01	111X	PCI configuration space	Refer to Section 10.1.1.6 for details. Dstream access only.
10	XXXX	PCI sparse memory space	128 MB of PCI space addressable. The lower address bits are used to determine byte masks and transaction length information, hence the 4 GB space is reduced to a 128 MB sparse space. Must use this space when byte or word access granularity is required. Read or write length is no more than a quadword. Reading more than the requested data is harmful. Prefetching read data is prohibited. Dstream access only.
11	XXXX	PCI dense memory space	4 GB of PCI space. Used for devices with access granularity greater than a longword. Reads do not have side effects; prefetching of data from PCI devices is allowed. Typically used for data buffers. Dstream access only.

10.1.1.1 PCI Sparse Memory Space

2 0000 0000 .. 2 FFFF FFFF

Accesses to this space can have byte, word, tribyte, longword, or quadword granularity. The Alpha architecture does not provide byte, word, or tribyte granularity, which the PCI requires. Therefore to provide this granularity, the byte enable and byte length information are encoded in the lower address bits in this space. Address bits <7:3> are used for this purpose. Bits <31:8> are used to generate quadword addresses on the PCI, thus resulting in a sparse 4 GB space that maps to 128 MB of address space on the PCI. An access to this space causes a memory read or memory write access on the PCI.

The mapping is as follows:

Address <33:32> are used to identify the various address spaces on the sysBus. Address <7:3> are used to generate the length of the PCI transaction in bytes, the byte enables, and address <2:0>. Refer to Table 10–2. Address <31:8> correspond to the quadword PCI addresses and are sent out on AD<26:3> during the address phase on the PCI. AD<31:27> are obtained from one of two host address extension registers—HAXR0 and HAXR1. HAXR0 (which is hard coded as 0) is used for sysBus addresses between 2 0000 0000 .. 2 1FFF FFFF, that is, when sysBus address <31:29> is 0. HAXR1 is used to map sysBus addresses between 2 2000 0000 .. 2 FFFF FFFF, that is, when sysBus address <31:29> is non-zero anywhere in the PCI address space. HAXR1 is a CSR in the 21071-DA chip and is fully programmable. This allows EISA/ISA devices that require memory to be mapped in the lower 16 MB to coexist with other devices that do not have that restriction. The lower 16 MB have a fixed mapping (HAXR0) to 0, and the remaining 112 MB can be programmed anywhere in PCI space.

Figure 10–1 shows the sysBus to PCI memory address translation. Table 10–2 shows the generation of the byte enables and the PCI address <2:0> from sysBus address <6:3>.

Table 10–2 PCI Sparse Memory Space Byte Enable Generation

Length	CPU Address <6:5>	CPU Address <4:3>	PCI Byte Enable[1]	PCI Address<2:0>[2]
Byte	00	00	1110	CPU Address<7>, 00
	01	00	1101	CPU Address<7>, 00
	10	00	1011	CPU Address<7>, 00
	11	00	0111	CPU Address<7>, 00
Word	00	01	1100	CPU Address<7>, 00
	01	01	1001	CPU Address<7>, 00
	10	01	0011	CPU Address<7>, 00
	11	01	Illegal[3]	—
Tribyte	00	10	1000	CPU Address<7>, 00
	01	10	0001	CPU Address<7>, 00
	10	10	Illegal[3]	—
	11	10	Illegal[3]	—
Longword	00	11	0000	CPU Address<7>, 00
Longword	01	11	Illegal[3]	—
Longword	10	11	Illegal[3]	—
Quadword	11	11	0000	000

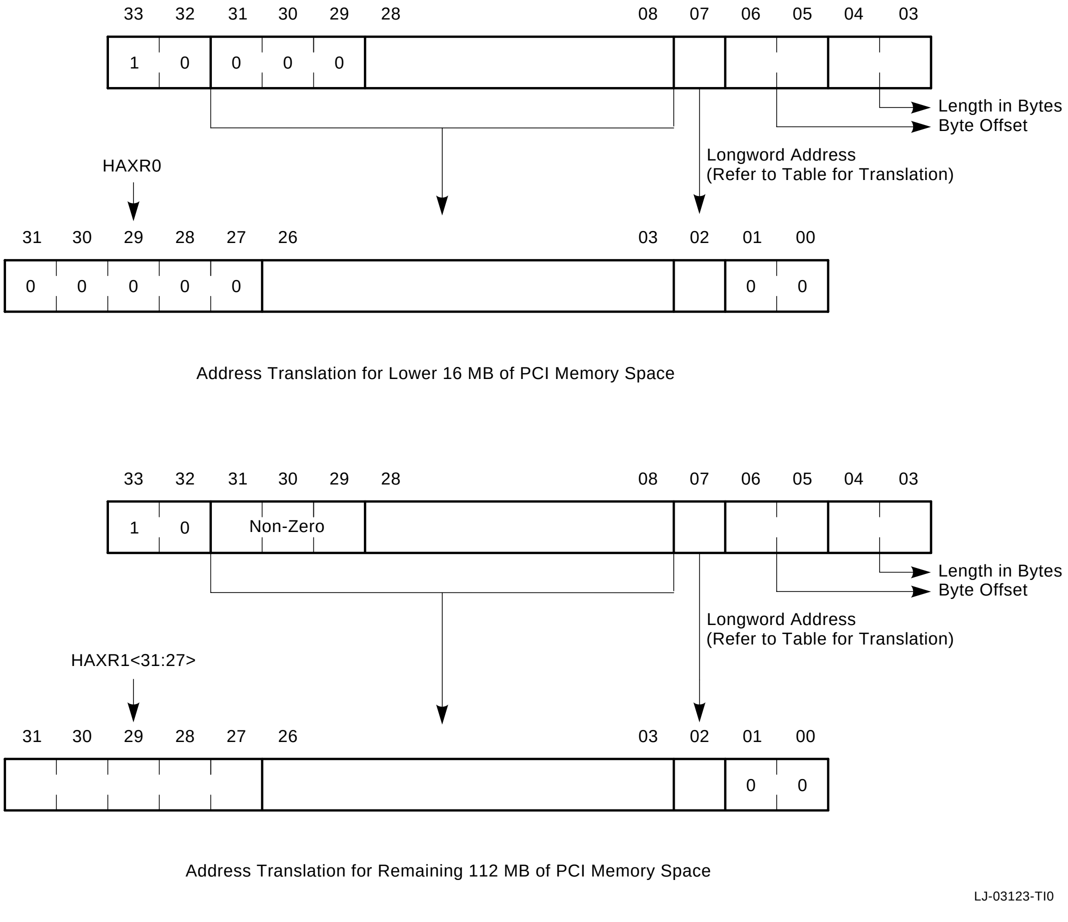
---

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

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

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

Figure 10–1 PCI Memory Space Address Translation



It is important to note that sysBus address<33:5> are directly available from the Alpha 21064 microprocessor. sysBus address<4:3> have to be derived from the longword masks, cpuCWMask<7:0>. On read transactions, the DECchip 21064 sends out address bits <4:3> on cpuCWMask<1:0>. On write transactions, the relationship between cpuCWMask<7:0> and address bits <4:3> is as follows:

If cpuCWMask<1:0> is non-zero, then address <4:3> is 00.
If cpuCWMask<3:2> is non-zero, then address <4:3> is 01.
If cpuCWMask<5:4> is non-zero, then address <4:3> is 10.
If cpuCWMask<7:6> is non-zero, then address <4:3> is 11.

Note

Accesses in this space are no longer than a quadword. Software must ensure that the processor does not merge consecutive writes in its write buffers by using memory barriers after each write.

Architecturally, if a byte, word, tribyte, or longword must be written on the PCI, an STL instruction must be done to the lower longword in the corresponding quadword address. An STQ or an STL instruction to the upper longword is not allowed. One bit-pair among cpuCWMask<1:0>, <3:2>, <5:4>, and <7:6> must have a value of 01 (binary); the other bit fields must all be 00 (binary). The location of the 01 (binary) field indicates whether the reference is byte, word, tribyte, or longword (respectively) in length.

Similarly, if a quadword has to be written to the PCI, software must do an STQ instruction to the corresponding address; the only legal value on cpuCWMask<7:0> in sparse space is 11000000 (binary).

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

10.1.1.2 PCI Dense Memory Space

3 0000 0000 .. 3 FFFF FFFF

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

- There is a one-to-one mapping between CPU addresses and PCI addresses. A longword address from the CPU maps to a longword on the PCI. Hence the name *dense space* (as opposed to PCI sparse memory space).
- Byte or word accesses are not allowed in this space. Minimum access granularity is a longword. The maximum transfer length implemented by the 21071 and 21072 chipsets is a cache line (32 bytes) on writes and a quadword on reads.

- Read prefetching is allowed in this space; extra reads have no side effects. The DECchip 21064 microprocessor does not specify a longword address on read transactions; it only specifies a quadword address. Therefore, reads in this space will always be done as a quadword read with a burst length of two on the PCI.
- Writes to addresses in this space can be buffered in the DECchip 21064 microprocessor. The 21071 and 21072 chipsets support a maximum burst length of 8 on the PCI corresponding to a cache line of data.

The address generation in dense space is as follows:

CPU address <31:5> is directly sent out on PCI address <31:5>. On read transactions, PCI address <4:3> is generated from cpuCWMask<1:0>, PCI address <2> is always 0.

On write transactions, PCI address <4:2> is generated from cpuCWMask<7:0>. If the lower longword is to be written, PCI address <2> is 0; if the lower longword is masked out and the upper longword is to be written, PCI address <2> is 1. The number of longwords written on the PCI is directly obtained from cpuCWMask<7:0>. Any combination of cpuCWMask<7:0> is allowed by the 21071 or 21072 chipsets.

Note

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

10.1.1.3 PCI Sparse I/O Space

1 C000 0000 .. 1 DFFF FFFF

The PCI sparse I/O space is sparse and has similar characteristics to the PCI sparse memory space. This 512 MB sysBus address space maps to 16 MB of PCI I/O address space. A read or write to this space causes a PCI I/O read or PCI I/O write command respectively. The address generation is as follows:

Address <33:29> are used to identify the various address spaces on the sysBus. Address <7:3> are used to generate the length of the PCI transaction in bytes, the byte enables, and address <2:0> on the PCI (Table 10–3). Address <28:8> correspond to the quadword PCI addresses and are sent out on AD<23:3> during the address phase on the PCI. AD<31:24> are obtained from one of two host address extension registers—HAXR0 and HAXR2.

HAXR0 (which is hard coded as 0) is used for sysBus addresses between 1 C000 0000 .. 1 C07F FFFF, that is, when sysBus address <28:23> is 0. HAXR2 is used to map sysBus addresses between 1 C080 0000 .. 1 DFFF FFFF, that is, when sysBus address <28:23> is non-zero, anywhere in the PCI address space. HAXR2 is a CSR in the 21071-DA chip and is fully programmable. This allows EISA/ISA devices that require their I/O space to be in the lower 256 KB, to coexist with other devices that do not have that restriction. The lower 256 KB have a fixed mapping (HAXR0) to 0, and the remaining 64 MB – 256 KB can be programmed anywhere in PCI space. Figure 10–2 shows the sysBus to PCI I/O address translation. Table 10–3 describes the generation of the byte enables, and the PCI address<2:0> from sysBus address <6:3>.

Table 10–3 PCI Sparse I/O Space Byte Enable Generation

Length	CPU Address <6:5>	CPU Address <4:3>	PCI Byte Enable[1]	PCI Address <2:0>
Byte	00	00	1110	CPU Address<7>, 00
	01	00	1101	CPU Address<7>, 01
	10	00	1011	CPU Address<7>, 10
	11	00	0111	CPU Address<7>, 11
Word	00	01	1100	CPU Address<7>, 00
	01	01	1001	CPU Address<7>, 01
	10	01	0011	CPU Address<7>, 10
	11	01	Illegal[2]	—
Tribyte	00	10	1000	CPU Address<7>, 00
	01	10	0001	CPU Address<7>, 01
	10	10	Illegal[2]	—
	11	10	Illegal[2]	—
Longword	00	11	0000	CPU Address<7>, 00
Longword	01	11	Illegal[2]	—
Longword	10	11	Illegal[2]	—
Quadword	11	11	0000	000

---

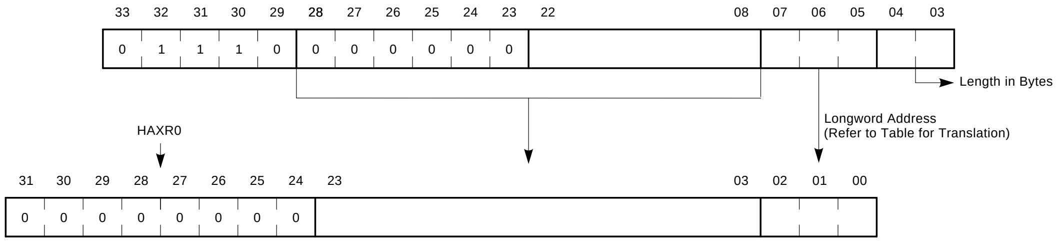
[1] Byte enable set to 0 indicates that byte lane carries meaningful data.

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

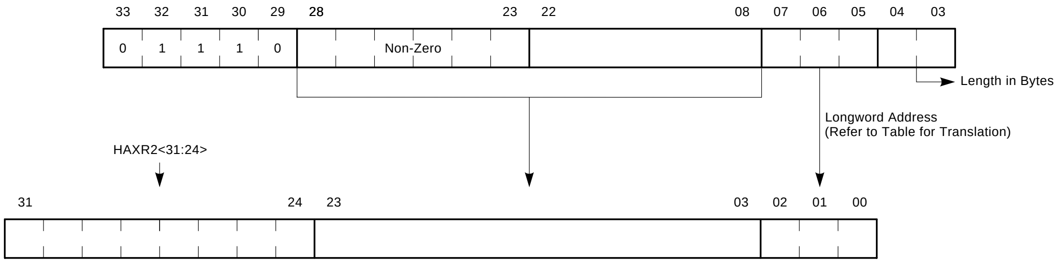
Warning

Quadword accesses to this PCI sparse I/O space will cause a two longword burst on the PCI. PCI devices cannot support bursting in I/O space.

Figure 10–2 PCI I/O Space Address Translation



Address Translation for Lower 256 KB of PCI I/O Space



Address Translation for Remaining 16 MB of PCI I/O Space

LJ-03124-TI0

10.1.1.4 DECchip 21071-DA CSR Space

1 A000 0000 .. 1 AFFF FFFF

All the 21071-DA CSRs are mapped in the DECchip 21071-DA CSR space. The 21071-DA chip responds to all accesses in this space.

10.1.1.5 PCI Interrupt Acknowledge/Special Cycle Space

1 B000 0000 .. 1 BFFF FFFF

A read access to this address space causes an interrupt acknowledge cycle on the PCI. The byte enable generation mechanism is based on address<6:3> and is the same as that of the PCI sparse I/O space. See Table 10–3. The address is a don't care during this transaction.

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

Note

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

10.1.1.6 PCI Configuration Space

1 E000 0000 .. 1FFF FFFF

A read or write access to this space causes a configuration read or write cycle on the PCI. There are two classes of targets—devices on the primary PCI bus and devices on secondary PCI buses that are accessed through PCI-to-PCI bridge chips. During PCI configuration cycles, the meanings of the address fields vary depending on the intended target of the configuration cycle. AD<1:0>, which are supplied by the HAXR2 register, indicate the target bus:

AD<1:0> equal to 00 indicates the primary PCI bus.
AD<1:0> equal to 01 indicates a secondary PCI bus.

Table 10–4 defines the various fields of AD during the address phase of a configuration read or write cycle.

Table 10–4 PCI Configuration Space Definition

Target Bus	AD Bits	Definition
Primary PCI Bus		
	<31:11>	Decoded from sysAdr<20:16> according to Table 10–5. Can be used for IDSEL# or don't cares. Typically, the IDSEL# pin of each device is connected to a unique AD line.
	<10:8>	Function select (1 of 8), from sysAdr<15:13>.
	<7:2>	Register select, from sysAdr<12:7>.
	<1:0>	00, from HAXR2<1:0>
Secondary PCI Buses (Must pass through a PCI-to-PCI bridge)		
	<31:24>	Forced to 0 by the 21071-DA chip.
	<23:16>	Secondary bus number, from sysAdr<28:21>.
	<15:11>	Device number, from sysAdr<20:16>.
	<10:8>	Function select (1 of 8), from sysAdr<15:13>.
	<7:2>	Register select, from sysAdr<12:7>.
	<1:0>	01, from HAXR2<1:0>

10.1.1.6.1 PCI Configuration Cycles to Primary Bus Targets Primary PCI bus devices are selected during a PCI configuration cycle if their IDSEL# pin is asserted, the PCI bus command indicates a configuration read or write, and AD<1:0> are 00. AD<7:2>, which are taken from sysAdr<12:7>, select a longword register in the device's 256-byte configuration address space. Configuration accesses can use byte masks, which may be derived by following the method shown in Table 10–3.

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

AD<31:11> are used to generate the IDSEL# signals. Typically, the IDSEL# pin of each PCI peripheral is connected to a unique address line. AD<31:11> are decoded from sysAdr<20:16> according to Table 10–5, ensuring that only one bit of AD<31:11> is asserted for any given configuration space transaction on the primary PCI bus. sysAdr<28:21> are ignored.

Table 10–5 PCI Address Decoding for Primary Bus Configuration Accesses

Device Number (sysAdr<20:16>)	PCI AD<31:11>
00000	0000 0000 0000 0000 0000 1
00001	0000 0000 0000 0000 0001 0
00010	0000 0000 0000 0000 0010 0
00011	0000 0000 0000 0000 0100 0
00100	0000 0000 0000 0000 1000 0
00101	0000 0000 0000 0001 0000 0
00110	0000 0000 0000 0010 0000 0
00111	0000 0000 0000 0100 0000 0
01000	0000 0000 0000 1000 0000 0
01001	0000 0000 0001 0000 0000 0
01010	0000 0000 0010 0000 0000 0
01011	0000 0000 0100 0000 0000 0
01100	0000 0000 1000 0000 0000 0
01101	0000 0001 0000 0000 0000 0
01110	0000 0010 0000 0000 0000 0
01111	0000 0100 0000 0000 0000 0
10000	0000 1000 0000 0000 0000 0
10001	0001 0000 0000 0000 0000 0
10010	0010 0000 0000 0000 0000 0
10011	0100 0000 0000 0000 0000 0
10100	1000 0000 0000 0000 0000 0
10101	0000 0000 0000 0000 0000 0
10110	0000 0000 0000 0000 0000 0
10111	0000 0000 0000 0000 0000 0
11000	0000 0000 0000 0000 0000 0
11001	0000 0000 0000 0000 0000 0
11010	0000 0000 0000 0000 0000 0
11011	0000 0000 0000 0000 0000 0
11100	0000 0000 0000 0000 0000 0
11101	0000 0000 0000 0000 0000 0
11110	0000 0000 0000 0000 0000 0
11111	0000 0000 0000 0000 0000 0

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

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

If the bus number of the configuration cycle matches the bus number of the bridge chip's secondary PCI interface, then it will intercept the configuration cycle, decode it, and generate a PCI configuration cycle with AD<1:0> equal to 00 on its secondary PCI interface. If the bus number is within the range of bus numbers that may exist hierarchically behind its secondary PCI interface, the PCI configuration cycle passes, unmodified (leaving AD<1:0> = 01), through the bridge. The configuration cycle will be intercepted and decoded by a downstream bridge.

10.1.2 PCI To Physical Memory Addressing

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

- PCI base register
- PCI mask register
- Translated base register

The PCI mask register provides a mask corresponding to bits <31:20> of an incoming PCI address. The size of each window can be programmed to be from 1 MB to 4 GB, in powers of two, by masking bits of the incoming PCI address using the PCI mask register. Table 10–6 shows an example of this.

Table 10–6 PCI Target Window Enables

pci_Mask<31:20>[1]	Size of Window	Value of n[2]
0000 0000 0000	1 MB	20
0000 0000 0001	2 MB	21
0000 0000 0011	4 MB	22
0000 0000 0111	8 MB	23
0000 0000 1111	16 MB	24
0000 0001 1111	32 MB	25
0000 0011 1111	64 MB	26
0000 0111 1111	128 MB	27
0000 1111 1111	256 MB	28
0001 1111 1111	512 MB	29
0011 1111 1111	1 GB	30
0111 1111 1111	2 GB	31
1111 1111 1111	4 GB[3]	32

[1] Combinations of bits in pci_Mask<31:20> that are not shown in Table 10–6 are not supported.

[2] Depending upon the target window size, only the incoming address bits <31:*n*> are compared with bits <31:*n*> of the PCI base registers, as shown in Figure 10–3 (n = 20 to 32). If n=32, no comparison is performed. n is also used in Figure 10–5.

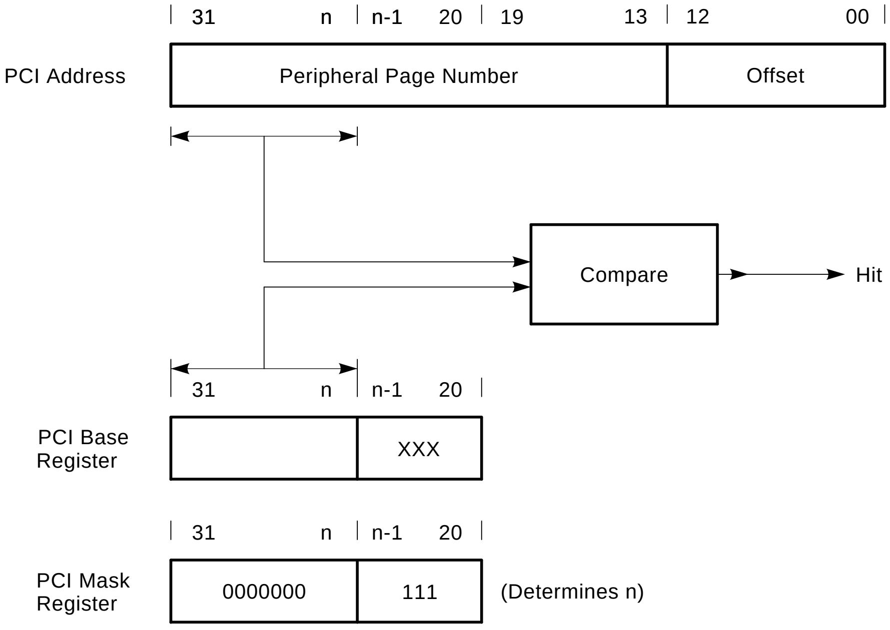
[3] When this combination is chosen, the wEnb bit in the other PCI base register has to be cleared, otherwise the two windows will overlap.

Based on the value of the PCI mask register, the unmasked bits of the incoming PCI address are compared with the corresponding bits of each window's PCI base register. If the base registers and the incoming PCI address match, then the incoming PCI address has hit in that PCI target window; otherwise, the incoming PCI address has missed in that window. A window enable bit, wEnb, is provided in each window's PCI base register to allow windows to be independently enabled or disabled. If a window's wEnb bit is set, then the window is enabled. The PCI target windows must be programmed so that the PCI address ranges that each one responds to do not overlap. The compare scheme between the incoming PCI address and the PCI base register (along with the PCI mask register) described previously is shown in Figure 10–3.

Note

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

Figure 10–3 PCI Target Window Compare



LJ-03126-TI0

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

If the scatter/gather bit is cleared, the DMA address is direct mapped, and the translated address is generated by concatenating bits from the matching window's translated base register with bits from the incoming PCI address. The PCI mask register determines which bits of the translated base register and PCI address are used to generate the translated address, as shown in Table 10–7.

Note

The unused bits of the translated base register indicated in Table 10–7 must be cleared for proper operation. Because system memory is located in the lower half of the CPU address space, address <33> is always 0. Address <32:5> is obtained from the translated base register.

Table 10–7 PCI Target Address Translation—Direct Mapped (Scatter/Gather Mapping Disabled)

pci_Mask<31:20>	Translated Address<32:5>
0000 0000 0000	t_Base<32:20> :pci_Address<19:5>
0000 0000 0001	t_Base<32:21> :pci_Address<20:5>
0000 0000 0011	t_Base<32:22> :pci_Address<21:5>
0000 0000 0111	t_Base<32:23> :pci_Address<22:5>
0000 0000 1111	t_Base<32:24> :pci_Address<23:5>
0000 0001 1111	t_Base<32:25> :pci_Address<24:5>
0000 0011 1111	t_Base<32:26> :pci_Address<25:5>
0000 0111 1111	t_Base<32:27> :pci_Address<26:5>
0000 1111 1111	t_Base<32:28> :pci_Address<27:5>
0001 1111 1111	t_Base<32:29> :pci_Address<28:5>
0011 1111 1111	t_Base<32:30> :pci_Address<29:5>
0111 1111 1111	t_Base<32:31> :pci_Address<30:5>
1111 1111 1111	t_Base<32>: pci_Address<31:5>

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

Each scatter/gather map entry maps an 8 KB page of PCI address space into an 8 KB page of the processor's address space. Each scatter/gather map entry is a quadword. Each entry has a valid bit in bit position 0. Address bit <13> is at bit position 1 of the map entry. Because the DECchip 21071 and DECchip 21072 chipsets implement only valid memory addresses up to 6 GB, bits <63:21> of the scatter/gather map entry should be programmed to 0. Bits <20:1> of the scatter/gather entry are used to generate the physical page address. This is appended to bits <12:5> of the incoming PCI address to generate the memory address that needs to go out on the sysBus. Figure 10–4 shows the scatter/gather map entry.

The size of the scatter/gather map table is determined by the size of the PCI target window as defined by the PCI mask register (Table 10–8). Because the scatter/gather map is located in system memory, translated address <33> is always 0. Address<32:3> are obtained from translated base register and the PCI address, as shown in Table 10–8.

Table 10–8 Scatter/Gather Map Address

pci_Mask<31:20>	Scatter/Gather Map Table Size	Scatter/Gather Map Address<32:3>
0000 0000 0000	1 KB	t_Base<32:10> :pci_Address<19:13>
0000 0000 0001	2 KB	t_Base<32:11> :pci_Address<20:13>
0000 0000 0011	4 KB	t_Base<32:12> :pci_Address<21:13>
0000 0000 0111	8 KB	t_Base<32:13> :pci_Address<22:13>
0000 0000 1111	16 KB	t_Base<32:14> :pci_Address<23:13>
0000 0001 1111	32 KB	t_Base<32:15> :pci_Address<24:13>
0000 0011 1111	64 KB	t_Base<32:16> :pci_Address<25:13>
0000 0111 1111	128 KB	t_Base<32:17> :pci_Address<26:13>
0000 1111 1111	256 KB	t_Base<32:18> :pci_Address<27:13>
0001 1111 1111	512 KB	t_Base<32:19> :pci_Address<28:13>
0011 1111 1111	1 MB	t_Base<32:20> :pci_Address<29:13>
0111 1111 1111	2 MB	t_Base<32:21> :pci_Address<30:13>
1111 1111 1111	4 MB	t_Base<32:22> :pci_Address<31:13>

Figure 10–4 Scatter/Gather Map Page Table Entry in Memory

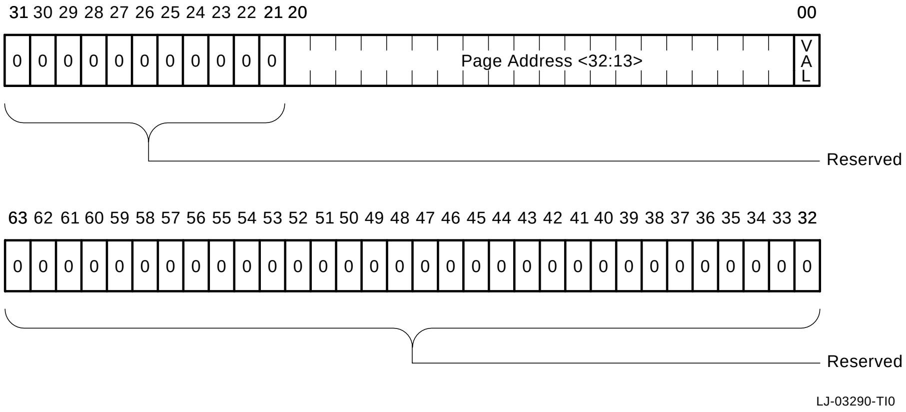
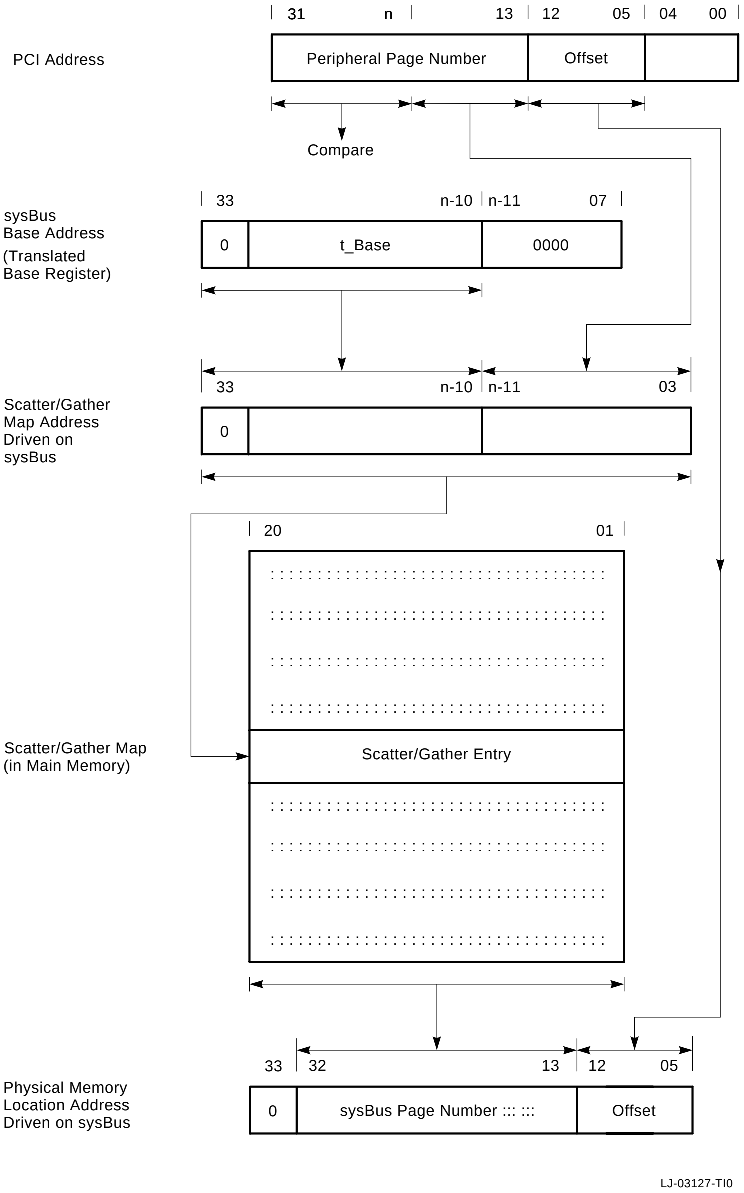


Figure 10–5 shows the entire translation from PCI address to physical address on a window that implements scatter/gather mapping.

The process is as follows:

- Bits <12:2> of the PCI address are used directly to generate the page offset.
- The relevant bits of the PCI address (as specified by the window mask register, depending on the size of the window) are used to generate the offset into the scatter/gather map.
- The relevant bits of the translated base register indicate the base address of the scatter/gather map.
- The map base is appended to the map offset to generate the address of the corresponding scatter/gather entry.
- Bits <20:1> of the map entry are used to generate the physical page address, which is appended to the page offset to generate the PCI address.
- Bit <0> is the valid bit for the Page Table Entry.

Figure 10–5 Scatter/Gather Map Translation of PCI to sysBus Address



10.2 DECchip 21071-DA Internal Registers

This section provides a summary of the DECchip 21071-DA internal registers, and it describes each register.

10.2.1 Register Overview

The control and status register (CSR) addresses are listed in Table 10–9. All registers are longword and are addressed on cache line boundaries (address <4:2> must be 0). Writes to read-only registers do not cause errors and are acknowledged as normal. Only 0's should be written to unspecified bits within a register. Registers are initialized as specified in the detailed descriptions in this chapter. Addresses in CSR space that are not specified here should not be read or written.

Table 10–9 DECchip 21071-DA Register Summary

Address (hex)	Register Name
1 A000 0000	Diagnostic control and status register (DCSR)
1 A000 0020	PCI error address register (PEAR)
1 A000 0040	sysBus error address register (SEAR)
1 A000 0060	Dummy register1
1 A000 0080	Dummy register2
1 A000 00A0	Dummy register3
1 A000 00C0	Translated base 1 register
1 A000 00E0	Translated base 2 register
1 A000 0100	PCI base 1 register
1 A000 0120	PCI base 2 register
1 A000 0140	PCI mask 1 register
1 A000 0160	PCI mask 2 register
1 A000 0180	Host address extension register 0 (HAXR0)
1 A000 01A0	Host address extension register 1 (HAXR1)
1 A000 01C0	Host address extension register 2 (HAXR2)
1 A000 01E0	PCI master latency timer (PMLT)
1 A000 0200	TLB tag 0 register
1 A000 0220	TLB tag 1 register

(continued on next page)

Table 10–9 (Cont.) DECchip 21071-DA Register Summary

Address (hex)	Register Name
1 A000 0240	TLB tag 2 register
1 A000 0260	TLB tag 3 register
1 A000 0280	TLB tag 4 register
1 A000 02A0	TLB tag 5 register
1 A000 02C0	TLB tag 6 register
1 A000 02E0	TLB tag 7 register
1 A000 0300	TLB data 0 register
1 A000 0320	TLB data 1 register
1 A000 0340	TLB data 2 register
1 A000 0360	TLB data 3 register
1 A000 0380	TLB data 4 register
1 A000 03A0	TLB data 5 register
1 A000 03C0	TLB data 6 register
1 A000 03E0	TLB data 7 register
1 A000 0400	Translation buffer invalidate all register (TBIA)

10.2.2 Register Descriptions

This section provides registers descriptions.

10.2.2.1 Dummy Registers 1–3

Dummy registers are CSRs that have no side effects on writes, and return 0's on reads. Writes to these registers can be used to pack the DECchip 21064 write buffers to prevent merging of sparse space I/O writes. Software does not have to use memory barrier instructions between writes if this mechanism is used.

10.2.2.2 Diagnostic Control and Status Register (DCSR)

The DCSR provides control of operational and diagnostic modes, and reports status and error conditions. Figure 10–6 shows the register bit assignments, and Table 10–10 provides the bit descriptions for the diagnostic control and status register.

Figure 10–6 Diagnostic Control and Status Register (DCSR)

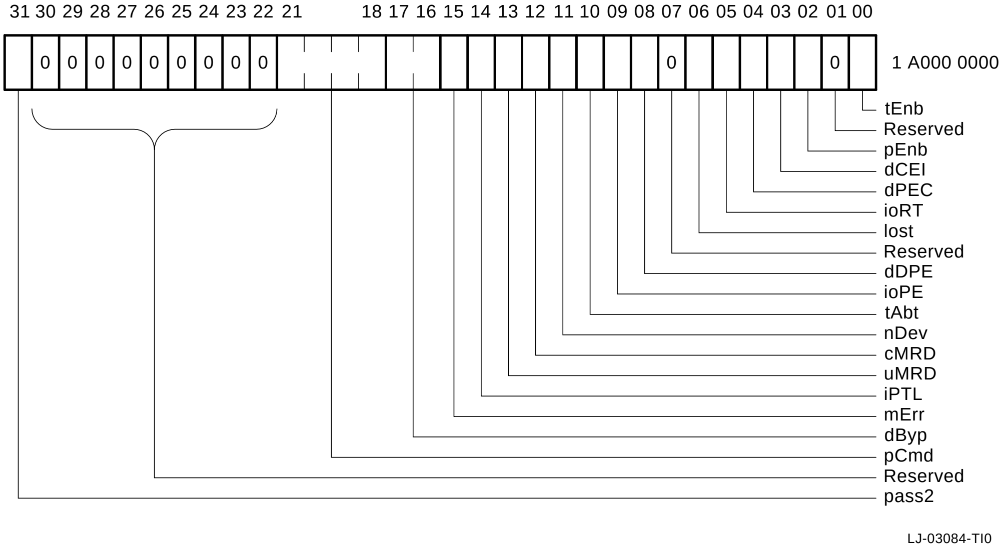


Table 10–10 Diagnostic Control and Status Register

Field	Extent	Type, Reset	Description
pass2	<31>	RO,–	Chip version reads low on pass1 and reads high on pass2.
Reserved	<30:22>	MBZ	
pCmd	<21:18>	RO,–	The pCmd field indicates the PCI cycle type when a PCI-initiated error is logged in the DCSR. This field is only valid when iPTL, nDev, tAbt, or ioPE errors are set.

(continued on next page)

Table 10–10 (Cont.) Diagnostic Control and Status Register

<table>
<tr><th>Field</th><th>Extent</th><th>Type, Reset</th><th>Description</th></tr>
<tr><td>dByp<1:0></td><td><17:16></td><td>RW, 0</td><td>The disable read bypass bits are used to control the ordering of PCI-initiated memory reads with respect to PCI-initiated memory writes. This field has three modes:
<table>
<tr><th>Value</th><th>Mode</th><th>Description</th></tr>
<tr><td>00</td><td>Full Bypass</td><td>In this mode, PCI-initiated memory reads will bypass buffered DMA writes if the double-hexaword address of the read does not match that of the buffered writes. The address comparison is done across address bits <31:6>.</td></tr>
<tr><td>01</td><td>Reserved</td><td>—</td></tr>
<tr><td>10</td><td>Partial Bypass</td><td>In this mode, DMA reads will bypass buffered memory writes if the address within the page does not match that of the buffered DMA writes. The address comparison is done across bits <12:6>.</td></tr>
<tr><td>11</td><td>No Bypass</td><td>In this mode, DMA read bypassing is completely disabled. DMA reads will be ordered with respect to DMA writes originating on the PCI.</td></tr>
</table>
</td></tr>
</table>

(continued on next page)

Table 10–10 (Cont.) Diagnostic Control and Status Register

Field	Extent	Type, Reset	Description
mErr	<15>	RWC,0	The memory error (mErr) bit is set when the 21071-DA chip receives an error code in the ioCAck<1:0> field in response to a memory access. sysAdr<33:5> for this transaction is logged in the sysBus error address register<31:4>. This bit is not logged if the sysBus error address register is locked by a previous error. The lost error bit is set instead. If the mErr bit and either the cMRD or the uMRD bits are set, this indicates that the address for the mErr is lost.
iPTL	<14>	RWC,0	The invalid page table lookup (iPTL) bit is set when the longword scatter/gather map entry being accessed is invalid. (See Figure 10–4.) AD<31:0> is logged in the PCI error address register, if it is not already locked. If the iPTL bit and any of the ioRT, ioPE, nDev, tAbt, and dDPE bits are set, this indicates that the address for the iPTL is lost.
uMRD	<13>	RWC,0	The uncorrectable memory read data (uMRD) bit is set when an uncorrectable error is encountered by the 21071-DA chip. The error is encountered when the data read from the DMA read buffer in the 21071-BA chip reaches the 21071-DA chip on a DMA read or a scatter/gather read transaction. sysAdr<33:6> for this transaction is logged in the sysBus error address register<31:4>, if the SEAR is not locked.

(continued on next page)

Table 10–10 (Cont.) Diagnostic Control and Status Register

Field	Extent	Type, Reset	Description
cMRD	<12>	RWC,0	The correctable memory read data (cMRD) bit is set when a correctable error is encountered by the 21071-DA chip. The error is encountered when the data read from the DMA read buffer in the 21071-BA reaches the 21071-DA on a DMA read or a scatter/gather read transaction. sysAdr<33:6> for this transaction is logged in the sysBus error address register<31:4> if the SEAR is not locked. The logging of this error can be prevented by setting the disable correctable error (dCEI) in this register.
nDev	<11>	RWC,0	The no device (nDev) bit is set when DEVSEL# is not asserted in response to an I/O read or write transaction initiated on the PCI by the 21071-DA chip. AD<31:0> for this transaction is logged in the PCI error address register<31:0>.
tAbt	<10>	RWC,0	The target abort (tAbt) bit is set when a PCI slave device ends an I/O read or write transaction using the PCI target abort protocol. AD<31:0> for this transaction is logged in the PCI error address register<31:0>.
ioPE	<9>	RWC,0	The I/O parity error (ioPE) bit is set when a parity error occurs in the data phase of an I/O read or I/O write transaction. AD<31:0> for this transaction is logged in the PCI error address register<31:0>.
dDPE	<8>	RWC,0	The DMA data parity error (dDPE) bit is set when a parity error occurs in the data phase of a DMA transaction. AD<31:0> for this transaction is logged in the PCI error address register<31:0>.
Reserved	<7>	MBZ	—

(continued on next page)

Table 10–10 (Cont.) Diagnostic Control and Status Register

Field	Extent	Type, Reset	Description
lost	<6>	RWC,0	The lost error (lost) bit is set by the occurrence of an 21071-DA chip error condition when the address register corresponding to that error is locked because of a previous error. Under those circumstances, error information pertaining to the second error is lost. The logged address information in the sysBus error address register or the PCI error address register still remains valid for the initial error condition indicated by the error bit already set.
ioRT	<5>	RWC, 0	This bit is set when a retry timeout error occurs on CPU-initiated write or read transactions on the PCI. AD<31:0> is logged in the PCI error address register. This bit is also set in the event that the 21071-DA chip sees GntL deassert during the address portion of a configuration transaction 2^{24} consecutive times.
dPEC	<4>	RW,0	When the disable parity error checking (dPEC) bit is set, parity checking will not be performed on the PCI bus (address and data cycles, DMA and I/O transactions). Parity generation is not affected in any way.
dCEI	<3>	RW,0	When the disable correctable error interrupt (dCEI) bit is set, correctable errors on DMA read data are not logged in the cMRD bit (DCSR12), and the address is not updated in the sysBus error address register. This bit only determines whether the error is logged and if the processor is interrupted.
pEnb	<2>	RW,0	If the prefetch enable (pEnb) bit is set, the sysBus master machine will enable prefetching on DMA reads. This bit should be self cleared following system reset and should not be changed while DMA operations are going on.
Reserved	<1>	MBZ	—

(continued on next page)

Table 10–10 (Cont.) Diagnostic Control and Status Register

Field	Extent	Type, Reset	Description
tEnb	<0>	RW,0	When the TLB enable (tEnb) bit is set, the entire translation buffer (TLB) is enabled. When this bit is cleared, the TLB will be turned off and subsequent scatter/gather reads will not result in allocation of TLB entries. Entries that were valid when the tEnb bit was cleared will remain valid. To invalidate valid entries, software must write to the TBIA register.

10.2.2.3 PCI Error Address Register

Figure 10–7 shows the register bit assignments, and Table 10–11 provides the bit descriptions for the PCI error address register.

Figure 10–7 PCI Error Address Register

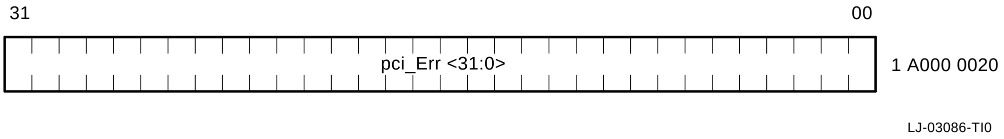


Table 10–11 PCI Error Address Register

Field	Extent	Type, Reset	Description
pci_Err<31:0>	<31:0>	RO,—	The address sent out on the PCI bus (AD<31:0>) as a result of an I/O transaction is stored here. This field logs the address of the errors indicated by the nDev, tAbt, ioPE, dDPE, iPTL, and ioRT bits in the DCSR. This register is valid only when one of these six error bits is set. If one of these six error bits is set, then a subsequent nDev, tAbt, ioPE, dDPE, iPTL, or ioRT error will not update the address logged in this register, and the lost bit in DCSR is set. pci_Err<31:0> are valid for nDev and iPTL. Only pci_Err<31:5> are valid for ioRT, tAbt, and ioPE errors that occur during dense memory writes. For ioRT, tAbt, and ioPE errors on any other transaction, pci_Err<31:3> are valid. pci_Err<31:6> are valid for dDPE errors.

10.2.2.4 sysBus Error Address Register

Figure 10–8 shows the register bit assignments, and Table 10–12 provides the bit descriptions for the sysBus error address register.

Figure 10–8 sysBus Error Address Register

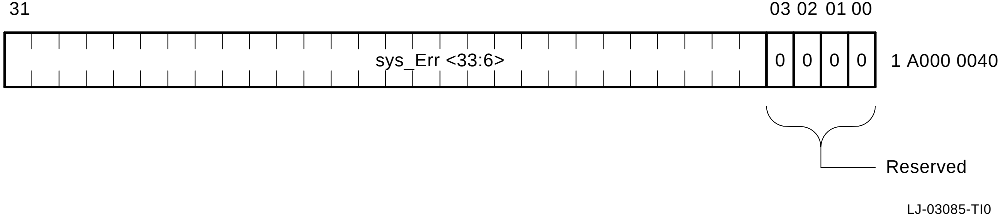


Table 10–12 sysBus Error Address Register

Field	Extent	Type, Reset	Description
sys_Err<33:6>	<31:4>	RO,—	The address sent out on the sysBus. (sysAdr<33:6> as a result of a DMA transaction is stored here.) This field logs the address of the errors indicated by the mErr, uMRD, or cMRD bits in the DCSR. This register is valid only when one of these three error bits is set. If one of these three error bits is set, a subsequent mErr, uMRD, or cMRD error will not update the address logged in this register, and the lost bit in DCSR is set.
Reserved	<3:0>	MBZ	—

10.2.2.5 Translated Base Registers 1–2

Figure 10–9 shows the register bit assignments, and Table 10–13 provides the bit descriptions for the translated base registers 1 through 2.

Figure 10–9 Translated Base Registers 1–2

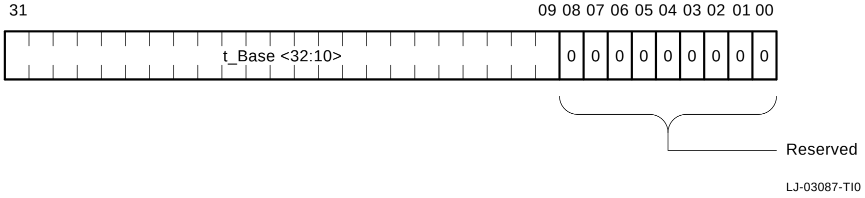


Table 10–13 Translated Base Registers 1–2

Field	Extent	Type, Reset	Description
t_Base<32:10>	<31:9>	RW,—	If scatter/gather mapping is disabled t_Base specifies the base CPU address of the translated PCI address for the PCI target window. If scatter/gather mapping is enabled t_Base specifies the base CPU address for the scatter/gather map table for the PCI target window.
Reserved	<8:0>	MBZ	—

10.2.2.6 PCI Base Registers 1–2

Figure 10–10 shows the register bit assignments, and Table 10–14 provides the bit descriptions for the PCI base registers 1 through 2.

Figure 10–10 PCI Base Registers 1–2

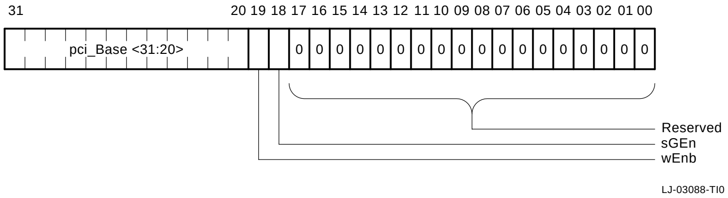


Table 10–14 PCI Base Registers 1–2

Field	Extent	Type, Reset	Description
pci_Base<31:20>	<31:20>	RW,—	pci_Base specifies the base address of the PCI target window.
wEnb	<19>	RW,0	When the window enable (wEnb) bit is cleared, this PCI target window is disabled and will not respond to PCI-initiated transfers. When wEnb is set, this PCI target window is enabled and will respond to PCI-initiated transfers that hit in the address range of the target window. This bit should be disabled by the processor (software) when modifying any of the PCI target window registers (base, mask, or translated base).
sGEn	<18>	RW,0	When the scatter/gather enable (sGEn) bit is cleared, the PCI target window uses direct mapping to translate a PCI address to a CPU address. When this bit is set, the PCI target window uses scatter/gather mapping to translate a PCI address to a CPU address.
Reserved	<17:0>	MBZ	—

10.2.2.7 PCI Mask Registers 1–2

Figure 10–11 shows the register bit assignments, and Table 10–15 provides the bit descriptions for the PCI mask registers 1 through 2.

Figure 10–11 PCI Mask Registers 1–2

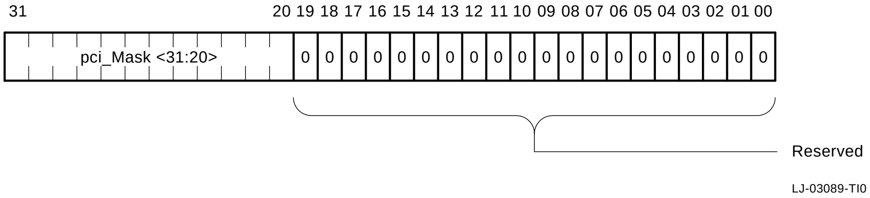


Table 10–15 PCI Mask Registers 1–2

Field	Extent	Type, Reset	Description
pci_Mask<31:20>	<31:20>	RW,—	pci_Mask specifies the size of the PCI target window. It is also used in the translation of the PCI address to the CPU address.
Reserved	<19:0>	MBZ	—

10.2.2.8 Host Address Extension Register 0 (HAXR0)

This register is hardcoded to 0. A read from this register returns a 0; a write does nothing.

10.2.2.9 Host Address Extension Register 1 (HAXR1)

This register is used to generate AD<31:27> on CPU-initiated transactions that address PCI memory space. Figure 10–12 shows the register bit assignments, and Table 10–16 provides the bit descriptions for host address extension register 1.

Figure 10–12 Host Address Extension Register 1 (HAXR1)

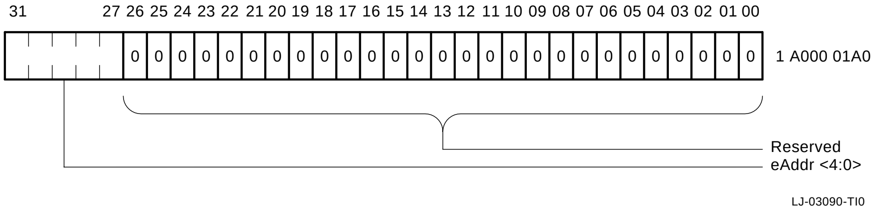


Table 10–16 Host Address Extension Register 1

Field	Extent	Type, Reset	Description
eAddr<4:0>	<31:27>	RW,0	For CPU-initiated transactions to PCI memory, eAddr<4:0> are used as the upper five PCI address bits (AD<31:27>).
Reserved	<26:0>	MBZ	—

10.2.2.10 Host Address Extension Register 2 (HAXR2)

This register is used to generate AD<31:24> on CPU-initiated transactions that address PCI I/O space. It is also used to generate AD<1:0> during PCI configuration reads and writes. Figure 10–13 shows the register bit assignments, and Table 10–17 provides the bit descriptions for host address extension register 2.

Figure 10–13 Host Address Extension Register 2 (HAXR2)

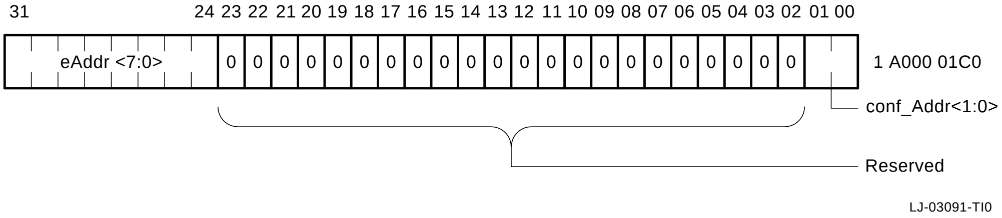


Table 10–17 Host Address Extension Register 2

Field	Extent	Type, Reset	Description
eAddr<7:0>	<31:24>	RW,0	For CPU-initiated transactions to PCI I/O space, eAddr<7:0> are used as the upper eight PCI address bits (AD<31:24>).
Reserved	<23:2>	MBZ	—
conf_Addr<1:0>	<1:0>	RW,0	For CPU-initiated transactions to PCI configuration space, conf_Addr<1:0> are used as the lower two PCI address bits (AD<1:0>).

10.2.2.11 PCI Master Latency Timer Register

Figure 10–14 shows the register bit assignments, and Table 10–18 provides the bit descriptions for the PCI master latency timer register..

Figure 10–14 PCI Master Latency Timer Register

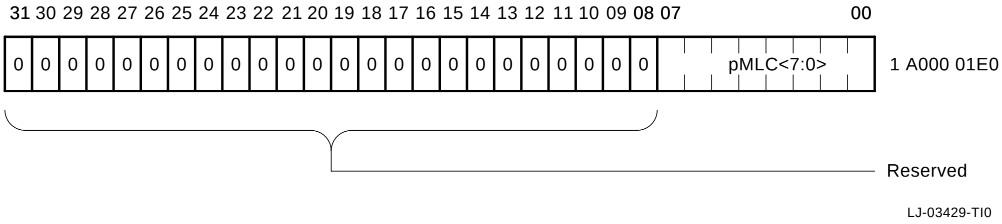


Table 10–18 PCI Master Latency Timer Register

Field	Extent	Type, Reset	Description
Reserved	<31:8>	MBZ	—
pMLC<7:0>	<7:0>	RW,0	pMLC<7:0> is loaded into the master latency timer register at the start of a PCI master transaction initiated by the 21071-DA chip.[1]

[1] This value should be programmed to be non-zero during system configuration.

10.2.2.12 TLB Tag Registers 0–7

These registers are read only. TLB tag registers 0 through 7 have identical formats.

Figure 10–15 shows the register bit assignments, and Table 10–19 provides the bit descriptions for TLB tag registers 0 through 7.

Figure 10–15 TLB Tag Registers 0–7

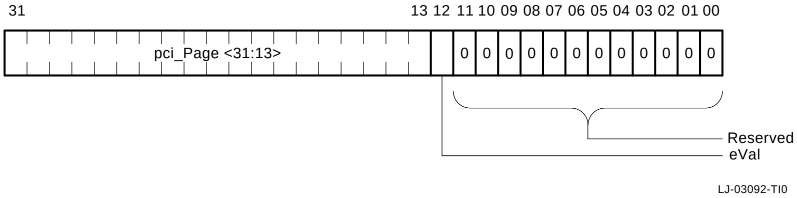


Table 10–19 TLB Tag Registers 0–7

Field	Extent	Type, Reset	Description
pci_Page<31:13>	<31:13>	RO,—	The pci_Page bit field specifies the PCI page address (TAG) corresponding to the translated CPU page address in the associated TLB data register.
eVal	<12>	RO,0	The entry valid (eVal) bit can be read through this bit position. Normally, the invalid bit contains the value read during a page table entry read.
Reserved	<11:0>	MBZ	—

10.2.2.13 TLB Data Registers 0–7

TLB data registers 0 through 7 have identical formats. These registers are read only. Figure 10–16 shows the register bit assignments, and Table 10–20 provides the bit descriptions for TLB data registers 0 through 7.

Figure 10–16 TLB Data Registers 0–7

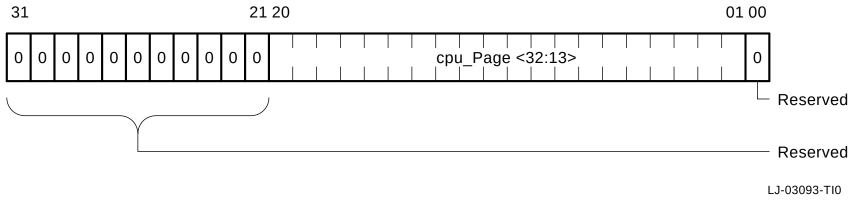


Table 10–20 TLB Data Registers 0–7

Field	Extent	Type, Reset	Description
Reserved	<31:21>	MBZ	—
cpu_Page<32:13>	<20:1>	RO,—	Bits <32:13> of the translated CPU address can be read through this bit field.
Reserved	<0>	MBZ	—

10.2.2.14 Translation Buffer Invalidate All (TBIA)

This register is a write-only register. A write to this register causes all the valid entries in the scatter/gather map TLB to be invalidated.

11

DECchip 21071-DA Transactions

This chapter describes the transaction flows for the 21071-DA chip, from the sysBus to the PCI and vice versa.

Throughout this chapter, the terms *transaction* and *command* are used interchangeably. In general, higher level transactions are composed of lower level transactions, or bus commands. For example, a DMA write transaction consists of a PCI memory write transaction (command) and a sysBus DMA write transaction (command).

11.1 CPU-Initiated Transactions

The 21071-DA chip responds to CPU-initiated transactions that address PCI space or the 21071-DA CSR space. In addition to this, it also responds to barrier, fetch, and fetchM. fetch and fetchM transactions are acknowledged immediately by sending cpuCAck OK on ioCmd<2:0>; no further action is taken.

11.1.1 Remote (PCI) Space I/O Read

This section describes the 21071-DA chip response to CPU-initiated remote (PCI) space I/O read transactions.

- The sysBus interface continuously monitors the command and address on the sysBus. When it detects a read block command and the address is in PCI space, it generates the PCI address, byte enables, and the PCI command (PCI memory read, PCI I/O read, PCI configuration read, and PCI interrupt acknowledge), and it notifies the PCI master state machine.
- The PCI master state machine asserts the ReqL pin on the PCI and waits for the bus to be granted to it. If the bus is parked with the 21071-DA chip, that is, GntL is already asserted, then the 21071-DA chip does not assert ReqL.

- If a read from system memory happens on the PCI before the PCI master machine has gained ownership of the PCI, the I/O read on the sysBus is preempted by the 21071-DA chip. This is done to prevent deadlocks from occurring.
- When a grant is received on the PCI, the address and command are sent out on the PCI, and a request is sent to the epiBus arbiter to set the direction of the epiBus from the 21071-DA chip to the 21071-BA chips. The epiBus arbiter is described in Section 11.3.
- The PCI master state machine waits for a response from the PCI target device.
- When the target responds, the transaction can complete in different ways:
  - If the target successfully returns data, the PCI master terminates the transaction and releases the PCI.
  - If the target aborts the transaction with a error, or if a parity error is found by the 21071-DA chip on the read data, the PCI master terminates the transaction and releases the PCI. When the data is returned to the CPU, an error is flagged.
  - If the target disconnects the transaction, indicating that the master should retry the transaction at a later time, the PCI master machine terminates the transaction, gives up the request for the epiBus, goes back to idle, and retries the transaction after the PCI bus becomes idle.
- If, when data is available from the PCI device, the epiBus has been granted to the PCI master machine, data is sent across the epiBus into the I/O read buffer and the request for the epiBus is cleared. If the epiBus has not been granted to the PCI master state machine, data cannot be sent to the I/O read buffer in the 21071-BA chip and is temporarily held in the 21071-DA chip. A subsequent PCI transaction addressed to the 21071-DA chip may stall TrdyL until the I/O read data has been moved.
- The transaction completes when the read data has been returned to the CPU.

11.1.2 Remote (PCI) Space I/O Write

This section describes the 21071-DA chip response to CPU-initiated remote (PCI) space I/O write transactions.

- When an I/O write to PCI space is detected on the sysBus, the data is loaded into the I/O write buffer by the 21071-CA chip. The 21071-BA chip always has room to accommodate the data for an I/O write transaction. One transaction is queued to go out on the PCI, and the second transaction stalls on the sysBus until the first is completed. The data for the second write is loaded into the second entry of the I/O write buffer (which acts as a holding buffer).
- The address is loaded into the I/O write address buffer, an I/O write request is posted to the PCI master state machine, and the transaction is acknowledged on the sysBus. A second I/O write will stall on the sysBus, until the first one completes on the PCI.
- The PCI master state machine asserts the ReqL pin on the PCI and waits for the bus to be granted to it. At the same time, a request is sent to the epiBus arbiter to set the direction of the epiBus from the 21071-BA chips to the 21071-DA chip.
- DMA transactions are serviced until the PCI master machine gets ownership of the PCI. If a second I/O write, CSR write, I/O read, or CSR read is stalled on the sysBus behind this write, it will be preempted to allow DMA read transactions or one DMA write transaction (if the DMA write buffer is full) to complete.
- When the PCI master acquires ownership of the epiBus, two longwords of data are then loaded into PCI output latches (temporary holding latches only). If a DMA read happens before the I/O write has been able to get out on the PCI, the data in the holding latches is lost and the arbitration has to be performed again.
- When the PCI master receives the grant on the PCI, it drives the address and command, and it asserts FrameL on the PCI if the I/O write data is ready to go out on the PCI in the following cycle (if the epiBus has been granted to the PCI master for transferring I/O write data).
- The PCI master drives AD<31:0> and CBE_l<3:0>, and it asserts FrameL. This allows the target device to decode and acknowledge (by asserting DevselL) the address. Data should be ready to be driven on the PCI by that time.

- As write data is sent out on the PCI, subsequent longwords are unloaded from the I/O write buffer into the PCI output latches through the epiData bus, until the transaction is terminated.
- It is possible that the target of the I/O transaction retries or disconnects the transaction before all the data has been transferred. The 21071-DA chip waits for the PCI to become idle and performs another I/O write transaction with the unwritten data.
- If the entire data transfer completes, the transaction is terminated on the PCI.

11.1.3 CSR Space I/O Read

A read from the 21071-DA CSRs behaves similarly to the remote read, except that the transaction does not go out on the PCI. Instead, the data is read from the 21071-DA CSRs.

Because CSR reads complete with a fixed known latency, a CSR read transaction is not preempted unless it is queued behind an I/O write to the PCI, which cannot complete because of DMA transactions on the PCI directed toward system memory. No errors are detected during this transaction.

11.1.4 CSR Space I/O Write

A write to the 21071-DA CSRs behaves similarly to the remote write, except that the transaction does not go out on the PCI. Instead, the data is written to the 21071-DA CSRs. Data from the 21071-BA chip still has to be transferred to the 21071-DA chip.

CSR writes, like I/O writes, are only preempted if they are queued behind an I/O write to the PCI, which cannot complete because of DMA transactions on the PCI.

11.1.5 Memory Barrier

The 21071-DA chip uses the memory barrier transaction as a means of synchronization between the DMA stream and the CPU stream.

On a memory barrier transaction, the 21071-DA chip flushes the I/O write buffer and those entries of the DMA write buffer that were valid when the memory barrier transaction was recognized on the sysBus. The 21071-DA chip preempts the memory barrier in order to flush the DMA writes. A memory barrier is also preempted if a DMA read transaction is directed toward the 21071-DA chip, or if the DMA write buffer becomes full while the 21071-DA chip is waiting to unload its I/O write buffer on the PCI.

11.2 PCI-Initiated Transactions

The 21071-DA chip supports PCI-initiated transactions directed towards system memory only. System memory can be mapped to two regions in PCI space. The PCI slave is always monitoring FrameL and the PCI address and command to determine if there is a transaction targeted towards system memory. All PCI memory write commands are treated the same, and all the PCI memory read commands are treated the same, except for read multiple commands (causes start of prefetch sequence).

11.2.1 PCI Memory Read, Read Line, and Read Multiple

This section describes the 21071-DA chip response to PCI memory read, read line, and read multiple transactions.

- Whenever the slave machine sees FrameL asserted, it checks for a valid PCI command and for a hit in one of its address windows. If the address or command is a hit, it asserts DevselL and proceeds with the transaction. If the address or command is a miss, it does not do anything.
- The slave machine posts a DMA read request to the sysBus master machine.
- A DMA request is posted to the sysBus arbiter.
- The sysBus master compares the read address with addresses queued in the DMA write buffer. If there is a hit, writes are serviced until the write that matches the read has been retried on the sysBus. If bypass mode is turned off, the DMA read does not proceed until all buffered DMA writes are completed.

Note

The comparison is on untranslated PCI addresses, not on physical memory addresses.

- At the same time, the sysBus master does a lookup in the TLB in order to determine if a scatter/gather map read is necessary. A scatter/gather map read is performed if the PCI window being addressed had scatter/gather mapping enabled, and there was a miss in the TLB.
- The scatter/gather read is performed (described later) or the TLB is read, and a translated physical memory address is generated.

- When grant is received on the sysBus, the sysBus master will perform a fetch. The 21071-DA will also perform a prefetch, if the prefetch enable bit is set in the DCSR and the transaction is addressed to the even cache line, or if the PCI command is a read multiple and the transaction is addressed to the even cache line. If prefetching will be performed, an atomic request is posted on ioRequest<1:0>. The sysBus master arbitrates for the epiBus so that the direction of the epiBus is from the 21071-BA chips to the 21071-DA chip.
- The address and command are sent out on the sysBus. If the sysBus master is prefetching, two DMA read transactions are done one after the other on the sysBus (guaranteed by the atomic request). A DMA read burst command is used on the first read, and a DMA read command is used on the prefetched read. If the requested data is the second octaword in the cache line, the wrapped command is used; if prefetching is enabled, the first command used is DMA read burst wrapped, but the second command is always DMA read.
- The read data (either from the cache or memory) is loaded into the DMA read buffer of the 21071-BA chip by the 21071-CA chip. The control signals are used to access the appropriate cache line, and longwords within it are set up when the sysBus master receives ownership of the epiBus. As the data is loaded into the DMA read buffer, valid bits are set to indicate which longwords are ready to be returned on the PCI.
- The termination conditions of a PCI memory read transaction are as follows:
  - The initiator terminates the transaction.
  - Prefetching is not enabled and a cache line boundary crossing is attempted.
  - A burst attempts to cross an odd-even cache line boundary, even if prefetching is enabled.
  - An uncorrectable memory data error is detected on the requested data.
  - The sysBus transaction is acknowledged with an error by the 21071-CA chip.
- If the sysBus transaction completes before the PCI transaction (this will usually happen on a long burst), the sysBus is released.

- If the PCI transaction completes before the sysBus transaction (this will usually happen on a short burst when prefetching is enabled), data remaining in the DMA read buffer is discarded. If the transaction completes before the prefetch has started on the sysBus, the prefetch transaction will not be performed.

11.2.2 PCI Memory Write/Write and Invalidate

This section describes the 21071-DA chip response to PCI memory write/write and invalidate transactions.

- The transaction begins just like a read. If the address is a hit, DevselL is asserted and the transaction continues.
- The PCI slave machine requests arbitration of the epiBus. The default path of the epiBus is in the direction of the DMA write. If the PCI slave machine has the epiBus, the write data is transferred to the 21071-BA chips. If the epiBus is busy doing a CSR read, a CSR write, or a scatter/gather read, the PCI slave will stall the first data transfer.
- After the DMA write path is set up between the 21071-DA chip and the DMA write buffer through the epiBus, the 21071-DA chip does not stall data transfers. If the transfer is stalled by the PCI master, the corresponding epiBus transfer is also stalled.
- The termination conditions of the PCI memory write transactions are as follows:
  - The initiator terminates the transaction.
  - The write burst attempts to cross an odd-even cache line boundary.
  - Only one DMA write buffer entry was available at the beginning of the transaction, and a cache line boundary crossing is attempted.
- If the write buffer was full at the beginning of the transaction or if the 21071-DA chip is locked by a different PCI bus master, the PCI slave disconnects without any data transfers.
- When a cache line boundary is crossed, and there were no data parity errors, a valid bit is set, and the corresponding cache line entry is ready to go out on the sysBus. If a write data parity error is detected on any longword of that cache line, the valid bit is not set and data is not written to memory. The PCI burst continues normally.
- The sysBus master is always monitoring the state of the DMA write buffer. When it sees a valid write, it performs the address translation (doing a scatter/gather read if necessary) and requests the sysBus using DMA request.

- The address of the transaction is sent out on the sysBus along with a DMA write full or DMA write masked command, depending on whether the entire cache line has valid data.
- The transaction completes when the 21071-CA chip returns an OK on ioCAck<1:0> to the 21071-DA chip.

11.2.3 PCI Exclusive Access to System Memory

This section describes the 21071-DA chip response to PCI exclusive access to system memory transactions.

- The PCI slave machine monitors the LockL signal along with FrameL. It uses the value of LockL in the cycle of FrameL assertion and in the cycle following the assertion of FrameL to determine whether or not the access is locked.
- If LockL is asserted during the cycle of FrameL, the PCI slave machine will not accept the transaction and will terminate it with a target disconnect (retry, no data transfers).
- If LockL is deasserted during the cycle of FrameL and is deasserted in the following cycle, the transaction proceeds normally as described in the previous sections.
- If LockL is deasserted during the cycle of FrameL, and is asserted in the following cycle, the transaction is treated as locked.
- Locked transactions are not treated specially on the PCI by the PCI slave machine. They clear the system lock flag. The system lock flag is held clear until the PCI LockL signal is released by the locking master. DMA read bypass is disabled as long as there are locked writes in the DMA write buffer.
- The system lock flag is cleared by sending an ioClrLock encoding on ioCmd<2:0> instead of an Idle encoding when the 21071-DA chip does not own the sysBus.

11.2.4 Scatter/Gather Map Read

A scatter/gather read is similar to a PCI-initiated DMA read on the sysBus. Data has to be loaded from the DMA read buffer into the TLB.

If errors (uncorrectable data errors, memory errors, and invalid scatter/gather entry errors) are found on a scatter/gather read, the transaction that caused the scatter/gather read is not performed. On PCI read transactions, the transaction is aborted by the 21071-DA chip; on writes an interrupt is posted.

11.3 epiBus Arbitration

At any given time, the 21071-DA chip could be servicing multiple transactions (CPU-initiated and PCI-initiated), all of which have to use the epiBus. The 21071-DA chip contains a central epiBus arbiter, which arbitrates for the bus and appropriately sets the direction of the epiBus. The PCI master and slave, as well as the sysBus master and slave, all request the bus for various transactions. Table 11–1 lists the priority of the various requests and the direction of the epiBus.

Table 11–1 epiBus Arbitration Priority

Priority	Transaction	Direction
1	PCI I/O reads	21071-DA to 21071-BA
2	DMA writes (default)	21071-DA to 21071-BA
3	DMA reads	21071-BA to 21071-DA
4	PCI I/O writes	21071-BA to 21071-DA
5	CSR writes	21071-BA to 21071-DA
6	CSR reads	21071-DA to 21071-BA
7	Scatter/gather reads	21071-BA to 21071-DA

12

DECchip 21071-DA Electrical Data

This chapter includes the following information about the DECchip 21071-DA chip:

- DC Electrical Data
- AC Electrical Data

12.1 DC Electrical Data

This section describes the dc characteristics of the DECchip 21071-DA chip.

12.1.1 Absolute Maximum Ratings

Table 12–1 lists the maximum ratings of the DECchip 21071-DA chip.

Table 12–1 DECchip 21071-DA Maximum Ratings

Characteristics	Minimum	Maximum
Storage temperature	–55°C (–67°F)	125°C (257°F)
Operating ambient temperature	0°C (32°F)	40°C (104°F)
Air flow	100 lfpm[1]	—
Junction temperature	25°C (77°F)	85°C (185°F)
Supply voltage with respect to Vss, with reset_l asserted	–0.5 V	+6.5 V
Supply voltage with respect to Vss, with reset_l deasserted	4.75 V	5.25 V
Voltage on any pin with respect to Vss	–0.5 V	Vdd + 0.5 V
Maximum power:		
@Vdd = 5.25 V @Cycle = 30 ns		1.5 W

---

[1] lfpm = linear feet per minute

Table 12–2 lists the dc parametric values of the DECchip 21071-DA chip.

Table 12–2 DC Parametric Values

Symbol	Description	Minimum	Maximum	Units	Test Conditions
^{V}ih	Input high voltage	2.0	–	V	–
^{V}il	Input low voltage	–	0.8	V	–
^{V}oh	Output high voltage	2.4	–	V	–
^{V}ol	Output low voltage	–	0.4	V	–
^{I}il	Input leakage current[1]	–5	5	µA	0V < Vin < Vdd
^{I}ilpu	Input leakage current[2]	–15	–100	µA	0V < Vin < Vdd
^{I}ilpd	Input leakage current[3]	15	100	µA	0V < Vin < Vdd
^{I}ol	Output leakage current (tristated)	–10	10	µA	0V < Vin < Vdd

[1] Excluding scanEn, testMode, and tristateL.

[2] For tristateL.

[3] For scanEn and testMode.

12.2 AC Electrical Data

This section describes the ac characteristics of the DECchip 21071-DA chip.

12.2.1 Clocks

The DECchip 21071-DA chip uses one clock (running at twice the nominal system frequency) plus a synchronous phase reference signal to generate four internal clock edges. An additional clock input is used to generate two internal clocks for the PCI logic. See Figures 12–1 and 12–2, and Tables 12–3 and 12–4 for details about DECchip 21071-DA external clock requirements and internal clock phase relationships.

A clock system must meet the requirements shown in Figure 12–1 and Table 12–4 to guarantee the proper behavior of the 21071-DA chip's internal logic. The 21071-DA chip does not specify the maximum skew allowed for external transfers to or from the CPU, Bcache PALs, Bcache, 21071-BA chips, or 21071-CA chip because these skew limits are dependent on module placement and routing. A system designer must examine external transfers to determine the maximum clock skews allowed between chips.

The skew numbers shown in Figure 12–1 and Table 12–4 are given for a 30.0 ns cycle time. At a longer cycle time, the allowable skew may be increased, as long as the given minimum times between clock edges are not violated. These skew limits assume that the 21071-DA chip adds another 0.1 ns of uncertainty between rising and falling edges due to non-ideal input buffer switching thresholds.

Table 12–3 DECchip 21071-DA Clock AC Characteristics

Parameter	Minimum	Maximum	Unit	Note
System cycle time	30	—	ns	*c* in Figure 12–1
clk1x2 period	15	—	ns	—
clk1x2 frequency	—	66	MHz	—
clk1x2 rise time	—	1	ns	—
clk1x2 fall time	—	1	ns	—
pClk period	30	—	ns	—
pClk rise time	—	1	ns	—
clk2ref setup to clk1x2 rising	0.4	—	ns	*Tsu* in Figure 12–1
clk2ref hold from clk1x2 rising	2.3	—	ns	*Th* in Figure 12–1

Figure 12–1 DECchip 21071-DA Clock Skew Requirements

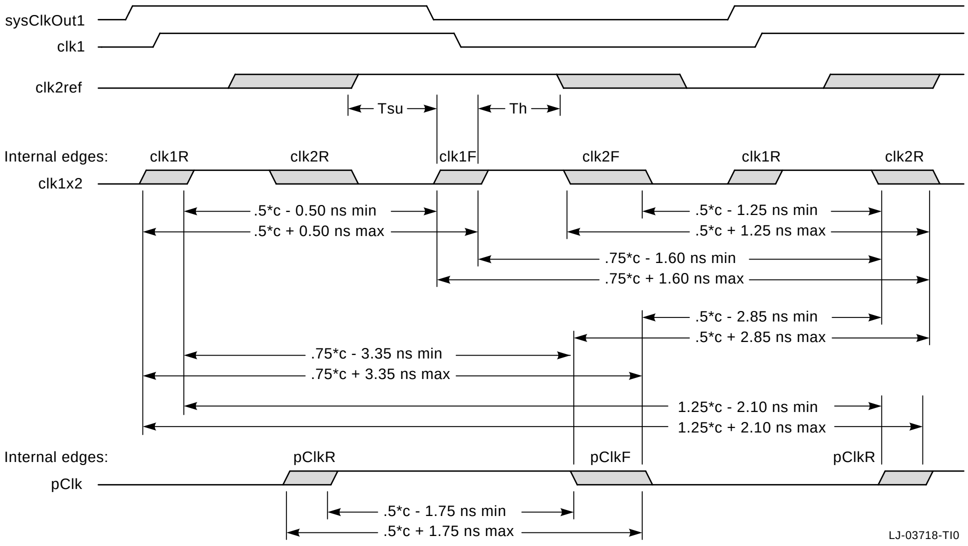
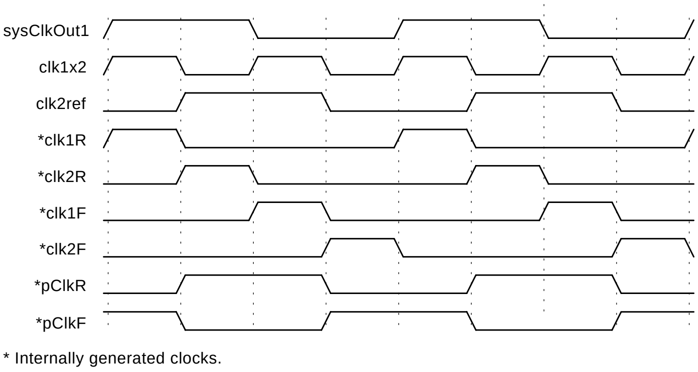


Table 12–4 DECchip 21071-DA Clock Skew Limits at clk1x2 Pin

Parameter	Example Transfers	Maximum	Unit	Note
clk1x2 or pClk rising edge to same clock rising edge	clk1R to clk1R, clk1R to clk1F, clk1F to clk1R, clk1F to clk1F, pClkR to pClkR	0.50	ns	@ Cycle = 30 ns
clk1x2 or pClk falling edge to same clock falling edge	clk2R to clk2R, clk2R to clk2F, clk2F to clk2R, clk2F to clk2F, pClkF to pClkF	1.25	ns	@ Cycle = 30 ns
clk1x2 rising edge to falling edge	clk1R to clk2R, clk1R to clk2F, clk1F to clk2R, clk1F to clk2F	1.60	ns	@ Cycle = 30 ns
clk1x2 falling edge to rising edge	clk2R to clk1R, clk2R to clk1F, clk2F to clk1R, clk2F to clk1F	1.60	ns	@ Cycle = 30 ns
clk1x2 rising edge to pClk rising edge, pClk rising edge to clk1x2 rising edge	clk1R to pClkR, clk1F to pClkR, pClkR to clk1R, pClkR to clk1F	2.10	ns	@ Cycle = 30 ns
clk1x2 falling edge to pClk falling edge, pClk falling edge to clk1x2 falling edge	clk2R to pClkF, clk2F to pClkF, pClkF to clk2R, pClkF to clk2F	2.85	ns	@ Cycle = 30 ns
clk1x2 rising edge to pClk falling edge, clk1x2 falling edge to pClk rising edge, pClk rising edge to clk1x2 falling edge, pClk falling edge to clk1x2 rising edge	clk1R to pClkF, clk1F to pClkF, clk2R to pClkR, clk2F to pClkR, pClkR to clk2R, pClkR to clk2F, pClkF to clk1R, pClkF to clk1F	3.35	ns	@ Cycle = 30 ns
pClk rising edge to falling edge, pClk falling edge to rising edge	pclkR to pClkF, pclkF to pClkR	1.75	ns	@ Cycle = 30 ns

Figure 12–2 DECchip 21071-DA Clock Signals



LJ-03456-TI0

The 21071-DA imposes no requirements on clk1 or sysClkOut1. Skew on clk1 will be constrained by limits imposed by external paths to or from the Bcache control PALs. The phase error between sysClkOut1 and clk1x2 will be constrained by limits imposed by external paths to or from the CPU chip.

12.2.2 Signals

Figures 12–3 and 12–4 demonstrate the timing measurements specified in Tables 12–6 and 12–7.

Figure 12–3 DECchip 21071-DA Output Delay Measurement

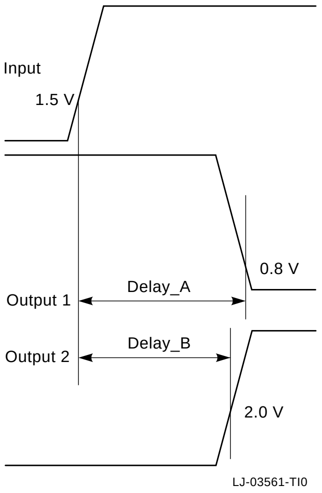
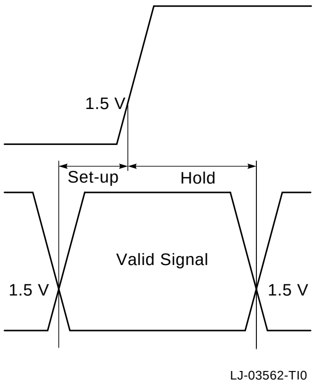


Figure 12–4 DECchip 21071-DA Setup and Hold Time Measurement



The following ac electrical data is specified with respect to the appropriate edge at the clk1x2 or pClk pins. Both the output delay table and the setup/hold time table assume a 1 ns edge rate at the clk1x2 and pClk pins.

All outputs drive a 50 pF load. When estimating module delays, you may need to replace the 50 pF load delay with a simulated (or calculated) delay. The delays for 4 mA and 8 mA drivers that drive a 50 pF load are provided in Table 12–5. See Table 8–1 for information about the buffer size of every output pin.

Table 12–5 DECchip 21071-DA Output Buffer Delays into a 50 pF Load

Type	Minimum	Maximum	Unit
4 mA	3.5	7.6	ns
8 mA	2.3	5.0	ns

Table 12–6 DECchip 21071-DA AC Characteristics (Valid Delay into a 50 pF Load)

Signal	Minimum	Maximum	Unit	Reference Edge
sysAdr<33:5>	4.8	14.2	ns	clk1R
ioRequest<1:0>, ioCmd<2:0>	4.6	11.8	ns	clk1R
AD<31:0>, CBE_l<31:0>, Par, FrameL, TrdyL, IrdyL, StopL, PerrL, LockL, DevselL	2.0	11.0	ns	pClkR
MemAckl, ReqL	2.0	12.0	ns	pClkR
epiData<31:0>, epiBEnErr<3:0>	4.8	16.1	ns	clk1R
epiSelDMA, epiFromIOB, epiOWSel, epiLineSel<1:0>, epiEnable<3:0>, ioLineSel<1:0>, epiLineInval	4.8	14.9	ns	clk1R
intHw0	7.5	20.1	ns	clk1F

Table 12–7 DECchip 21071-DA AC Characteristics (Setup/Hold Time)

Signal	Setup	Hold	Unit	Reference Edge
sysAdr<33:5>	13.1	3.7	ns	clk1R
cpuCWMask<7:0>	7.1	3.8	ns	clk1R
cpuCReq<2:0>	0.0	3.2	ns	clk1F
cpuCReq<2:0>	15.1	0.0	ns	clk1R
cpuHoldAck	–0.3	3.0	ns	clk1F
ioGrant, ioCAck<1:0>, ioDataRdy	–0.3	3.2	ns	clk1F
AD<31:0>, CBE_l<31:0>, Par, FrameL, TrdyL, IrdyL, StopL, PerrL, LockL, DevselL, MemReql	7.0	0.0	ns	pClkR
GntL	10.0	0.0	ns	pClkR
epiData<31:0> epiBEnErr<3:0>	0.4	5.2	ns	clk2F

13

DECchip 21071-DA Power-Up and Initialization

This chapter describes the behavior of the DECchip 21071-DA on power-up and assertion of reset_l. It also describes the system level requirements and the various registers that have to be initialized after reset_l is deasserted.

13.1 Power-Up

On power-up, the reset_l input of the DECchip 21071-DA should be asserted. It should be kept asserted until the system clocks are up and running for 20 cycles.

13.2 Internal Reset

The assertion and deassertion of the reset_l pin on the module is asynchronous to the DECchip 21071-DA chip. An internal reset signal is generated from reset_l, which asserts asynchronously as soon as reset_l is asserted but deasserts synchronously. Due to the synchronous deassertion of the internal reset, the DECchip 21071-DA requires that no external transaction should start until 10 system clock cycles after the deassertion of reset_l.

13.3 State of Pins on Reset Assertion

The following are general rules and requirements for the behavior of DECchip 21071-DA pins during reset:

- All input only control signals (except the clocks and reset_l) should be in the deasserted state as long as reset is asserted.
- All output only signals are deasserted.
- All bidirectional signals are tristated.

The exceptions to these rules are as follows:

- sysAdr<33:5> are driven synchronously with the assertion of reset and are tristated as soon as reset_l deasserts (without waiting for the deassertion of synchronous internal reset).
- epiData<31:0> and epiBEnErr<3:0> are driven as long as reset is asserted, and they continue to be driven after reset_l deassertion.
- ReqL is tristated on the assertion of reset_l and remains tristated until the deassertion of reset_l.
- If the PCI is not parked (that is, GntL is deasserted during reset) with the DECchip 21071-DA, then AD<31:0> and CBE_l<3:0> are tristated immediately on the assertion of reset_l, and Par is tristated a cycle later. If the PCI is parked with the DECchip 21071-DA (that is, GntL is asserted during reset), then AD<31:0>, CBE_l<3:0>, and Par are driven to 0.
- memAckl is tristated on the assertion of reset_l and remains tristated until the deassertion of reset_l.

Note

In all cases, the assertion of tristate_l overrides the assertion of reset_l. That is, if tristate_l is asserted during reset, all the outputs of the chip go to their High-Z state. If reset_l is still asserted when tristate_l deasserts, the signals return to the normal reset state described previously.

13.4 Configuration after Reset Deassertion

The following states must be initialized by software in the DECchip 21071-DA chip after the deassertion of reset_l.

- Diagnostic control and status register (DCSR)
- PCI base address registers
- PCI mask registers
- Translated base address registers
- Host address extension registers
- PCI master latency timer register

Part III

Part III contains five chapters that provide information about the DECchip 21071-BA chip. The following table provides a brief description of each chapter:

Chapter	Description
14	Describes the DECchip 21071-BA pin signals.
15	Describes the DECchip 21071-BA architecture.
16	Describes the flow of data within the DECchip 21071-BA for various transactions on the sysBus, memory data bus, and PCI bus.
17	Describes the DECchip 21071-BA electrical requirements.
18	Describes the behavior of the DECchip 21071-BA chip during power-up.

14

DECchip 21071-BA Pin Descriptions

The 21071-BA chip interfaces to three major buses:

- sysBus
- Memory data bus
- epiBus

This chapter provides a brief description of the pin signals for the 21071-BA data chip followed by detailed description of the 21071-BA data chip interfaces. This chapter also provides pin connection tables for the 21071-BA data chips in different bus width modes and for each 21071-BA instance (21071-BA 0,1,2,3).

14.1 DECchip 21071-BA Pin List

Table 14–1 lists the pin signals grouped by function. The information in the Type column identifies a signal as input (I), output (O), or bidirectional (B). The Buffer Strength column indicates the buffer drive strength.

All output and bidirectional pins, except pTestout, can be tristated.

Table 14–1 DECchip 21071-BA Pin List

Signals	Quantity	Type	Buffer Strength	Function
CPU/Bcache Interface Signals (66 Total)				
sysData<63:0>	64	B	4 ma	sysBus Data. In ECC mode, sysCheck<6:0> appears on sysData<38:32>, and memCheck<6:0> appears on sysData<57:63>.
sysPar<1:0>	2	B	4 ma	Parity pins for sysBus data.
Cache/Memory Data Path Control Signals (13 Total)				
drvSysData	1	I	—	Turns on 21071-BA sysData<63:16> drivers.
drvSysCSR	1	I	—	Turns off 21071-BA sysData<15:0> drivers.
drvMemData	1	I	—	Turns on 21071-BA memData and memPar drivers.
sysIORead	1	I	—	Selects I/O read buffer to sysBus.
sysReadOW	1	I	—	Selects octaword to be read.
subCmd<1:0>	2	I	—	Sub-commands for sysBus side of the 21071-BA.
sysCmd<2:0>	3	I	—	Commands for sysBus side of the 21071-BA.
memCmd<3:1>	3	I	—	Commands for memory side of chip.

(continued on next page)

Table 14–1 (Cont.) DECchip 21071-BA Pin List

Signals	Quantity	Type	Buffer Strength	Function
epiBus Signals (46 Total)				
epiData<31:0>	32	B	4 ma	Interchip data for both DMA and I/O operations.
epiBEnErr<3:0>	4	B	4 ma	epiData byte enables for epiBus from the 21071-DA operations and error/corrected status for epiBus to 21071-DA operations.
epiFromIOB	1	I	—	Selects the next epiBus transfer from the 21071-DA to the data chip.
epiSelDMA	1	I	—	Selects which buffer (I/O or DMA) will be transferred on the epiData bus.
epiEnable<1:0>	2	I	—	Qualifies epiData control signals and enables output drivers.
epiOWSel	1	I	—	Selects which octaword of the cache line will be transferred on the epiData bus.
epiLineSel<1:0>	2	I	—	Selects which cache line will be transferred on the epiData bus.
ioLineSel<1:0>	2	I	—	Selects which cache line should be read or written from the sysBus.
epiLineInval	1	I	—	Clears all byte valid bits in the current line of the DMA write buffer.

(continued on next page)

Table 14–1 (Cont.) DECchip 21071-BA Pin List

Signals	Quantity	Type	Buffer Strength	Function
Memory Signals (33 Total)				
memData<31:0>	32	B	4 ma	Memory data.
memPar	1	B	4 ma	Memory parity pins.
Miscellaneous/Clock Signals (8 Total)				
clk1x2	1	I	—	Clock input.
clk2ref	1	I	—	Phase reference for clk1x2.
reset_l	1	I	—	Reset.
testMode	1	I	—	Test mode select.
tristate_l	1	I	—	Tristate.
pTestout	1	O	4 ma	Parametric NAND tree output.
eccMode	1	I	—	True indicates ECC enabled.
wideMem	1	I	—	True indicates 128-bit wide memory.

Pin Totals

Total signal pins:	166
Total power and ground pins:	42
Total pins:	208

14.2 DECchip 21071-BA Signal Descriptions

This section provides signal descriptions of the 21071-BA data chip, the clock edges at which they can change, and rules about their usage during various transactions.

For simplicity, sysClkOut1_h is treated as clk1R.

Signal descriptions are grouped by function and correspond to the pin list provided in Section 14.1.

Note

The DECchip 21064 microprocessor does not use clk1R, but it uses sysClkOut1_h to generate and sample signals.

14.2.1 CPU/Bcache Interface Signals

See Section 2.2.4 for descriptions of 21071-CA signals that control 21071-BA data chip functions.

This section describes the CPU/Bcache signals.

14.2.1.1 sysData<63:0>, sysPar<1:0>

Signal Type: Bidirectional (21071-BA, CPU, Bcache)
Input Sampling Clock Edge: clk2F
Output Clock Edge: clk1R

sysData<63:0> is a bidirectional bus which provides data to and from the 21071-CA chip and the CPU. sysPar<1:0> are the parity bits for sysData<63:0>.

The CPU is the default driver of sysData.

When the system is configured in longword parity mode:

- sysPar<0> is the even parity across sysData<31:0> and is connected to check<0> of the processor.
- sysPar<1> is the even parity across sysData<63:32> and is connected to check<7> of the processor.

When the system is configured in longword ECC mode:

- sysData<38:32> is the ECC across sysData<31:0> and is connected to check<6:0> of the processor.
- sysData<57:63> (*note reversed order*) is the ECC across memData<31:0>, and is connected to check<6:0> of the memory bus.

14.2.2 Cache/Memory Data Path Control

This section describes the cache/memory data path control signals.

14.2.2.1 drvSysData

Signal Type: 21071-BA Input
Signal Source: 21071-CA
Input Sampling Clock Edge: clk1R assertion, clk1F deassertion

When drvSysData is sampled asserted, the 21071-BA chips drive sysData<63:16> and sysData<15:0> (only if drvSysCSR is deasserted) on this clk1R.

14.2.2.2 drvSysCSR

Signal Type: 21071-BA Input
Signal Source: 21071-CA
Input Sampling Clock Edge: clk1R

When drvSysCSR is asserted, the 21071-BA chips will not drive sysData<15:0> on the next clk1R.

14.2.2.3 drvMemData

Signal Type: 21071-BA Input
Signal Source: 21071-CA
Input Clock Edge: Flow through

drvMemData directly controls the memData drivers on the 21071-BA chips. When drvMemData is asserted, memData is driven; when drvMemData is deasserted, memData is tristated.

14.2.2.4 sysIORead

Signal Type: 21071-BA Input
Signal Source: 21071-CA
Input Sampling Clock Edge: clk2F

sysIORead is asserted by the 21071-CA chip along with drvSysData to indicate that the contents of the I/O read buffer should be driven onto the sysBus.

sysIORead is used by the 21071-BA chips to drive the contents of the I/O read buffer onto the sysBus.

14.2.2.5 sysReadOW

Signal Type: 21071-BA Input
Signal Source: 21071-CA
Input Sampling Clock Edge: clk2F

sysReadOW is asserted by the 21071-CA chip to indicate to the 21071-BA chips that the upper octaword of data should be taken from the memory read, merge, and I/O read buffers.

14.2.2.6 subCmd<1:0>

Signal Type: 21071-BA Input
Signal Source: 21071-CA
Input Sampling Clock Edge: clk2F

The subCmd<1:0> signals are asserted to further qualify the sysCmd<2:0> signals, as described in Table 14–2.

The subCmd<1:0> signals, in conjunction with sysCmd<2:0> signals, are used by the 21071-BA chips as commands for operations on the sysBus data buffers.

14.2.2.7 sysCmd<2:0>

Signal Type: 21071-BA Input
Signal Source: 21071-CA
Input Sampling Clock Edge: clk2F

The sysCmd<2:0> signals, in combination with the subCmd<1:0> signals, indicate to the 21071-BA chip the action to take on the sysData bus. In general, they echo the actions that took place on the sysBus during the previous cycle. The bits are decoded into various actions. Table 14–2 describes the sysCmd<2:0> and subCmd<1:0> encodings.

Table 14–2 sysCmd<2:0> and subCmd<1:0> Encodings

sysCmd	subCmd	Mnemonic	Function
000	0X	RESET	The merge bits in the merge buffer are cleared. All sysBus counters are reset. The data in the pad latches is held (to save power).
000	1X	NOP	The data in the pad latches is held in the latches, and new data will not be clocked into them. Used during reads or to hold the first transfer of write data due to a full write buffer.
001	XX	LOAD	No write action is performed. Sent when waiting for write data to be ready. Data from the sysData bus is loaded into the pad flops.
010	XX	RDDMAS WRIO	Data in the sysData pad latches is loaded into the DMA read buffer, which also serves as the I/O write buffer. A counter is incremented so that the next RDDMAS will load data into the next sub-cache line of the buffer.
011	XX	RDDMAM	Data in the memory read buffer is loaded into the DMA read buffer. A counter is incremented so that the next RDDMAM will load data into the next sub-cache line of the buffer.

(continued on next page)

Table 14–2 (Cont.) sysCmd<2:0> and subCmd<1:0> Encodings

sysCmd	subCmd	Mnemonic	Function
100	00	MERGE00	Nothing is loaded into the merge buffer. A counter is incremented so that the next MERGEnn will load data into the next sub-cache line of the buffer. During ST*x*_C transactions that hit in the cache, each sub-cache line of the merge buffer is loaded twice: once with the CPU write data using MERGE (that is, MERGE01) and once with the cache data using MERGE with inverted enables, called an overlay (that is, OVLY10).
100	01	MERGE01	Same as MERGE00, but longword 0's data in the sysData pad latches is loaded into the read/merge buffer, and longword 0's merge bit is set.
100	10	MERGE10	Same as MERGE00, but longword 1's data in the sysData pad latches is loaded into the read/merge buffer, and longword 1's merge bit is set.
100	11	MERGE11	Same as MERGE00, but longword 0 and 1's data in the sysData pad latches is loaded into the read /merge buffer, and longword 0 and 1's merge bits are set.
101	00	WRSYS0	Data in the sysData pad latches is loaded into the memory write buffer that represents cache line 0. A counter is incremented so that the next WRSYS0 will load data into the next sub-cache line of cache line 0.
101	01	WRSYS1	Same as WRSYS0, but for cache line 1.

(continued on next page)

Table 14–2 (Cont.) sysCmd<2:0> and subCmd<1:0> Encodings

sysCmd	subCmd	Mnemonic	Function
101	10	WRSYS2	Same as WRSYS0, but for cache line 2.
101	11	WRSYS3	Same as WRSYS0, but for cache line 3.
110	00	WRDMAS0	Data in the sysData pad latches is merged with the DMA write buffers and is loaded into the memory write buffer that represents cache line 0. A counter is incremented so that the next WRDMAS0 will load data into the next sub-cache line of cache line 0.
110	01	WRDMAS1	Same as WRDMAS0, but for cache line 1.
110	10	WRDMAS2	Same as WRDMAS0, but for cache line 2.
110	11	WRDMAS3	Same as WRDMAS0, but for cache line 3.
111	00	WRDMAM0	Data in the memory read buffer is merged with the DMA write buffers and is loaded into the memory write buffer that represents cache line 0. A counter is incremented so that the next WRDMAM0 will load data into the next sub-cache line of cache line 0.
111	01	WRDMAM1	Same as WRDMAM0, but for cache line 1.
111	10	WRDMAM2	Same as WRDMAM0, but for cache line 2.
111	11	WRDMAM3	Same as WRDMAM0, but for cache line 3.

14.2.2.8 memCmd<3:1>

Signal Type: 21071-BA Input
Signal Source: 21071-CA
Input Sampling Clock Edge: clk1R

The memCmd<3:1> signals indicate to the 21071-BA chips which action to take on the memData bus. For the encodings of memCmd<3:1>, see Table 14–3.

Table 14–3 memCmd<3:1> Encodings

memCmd	Mnemonic	Function
000	RDIMM	Read data is loaded into the read/merge buffer on the next memClkR. A counter is incremented so that the next RD*xxx* will load data into the next available sub-cache line of the read buffer.
001	RDDLY	Read data is loaded into the read/merge buffer on the memClkR after the next memClkR. A counter is incremented so that the next RD*xxx* will load data into the next available sub-cache line of the read buffer.
010	NOP	No operation.
011	RESET	All memory counters are reset.
100	WRIMM	Data from the memory write buffer is driven to memory on the next memClkR. A counter is incremented so that the next WR*xxx* will drive the next sub-cache line to memory.
101	WRDLY	Data from the memory write buffer is driven to memory on the memClkR after the next memClkR. A counter is incremented so that the next WR*xxx* will drive the next sub-cache line to memory.
110	WRIMML	Data from the memory write buffer is driven to memory on the next memClkR. After the write, the quadword pointer is reset to 0, and the cache line pointer is incremented so that the next WR*xxx* will drive the first sub-cache line of the next line to memory.

(continued on next page)

Table 14–3 (Cont.) memCmd<3:1> Encodings

memCmd	Mnemonic	Function
111	WRDLYL	Data from the memory write buffer is driven to memory on the memClkR after the next memClkR. After the write, the quadword pointer is reset to 0, and the cache line pointer is incremented so that the next WR*xxx* will drive the first sub-cache line of the next line to memory.

14.2.3 epiBus Signals

This section describes the epiBus signals.

14.2.3.1 epiData<31:0>

Signal Type: Bidirectional (21071-BA, 21071-DA)
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

epiData is a 32-bit bidirectional bus which connects the 21071-DA and the 21071-BA chips.

14.2.3.2 epiBEnErr<3:0>

Signal Type: Bidirectional (21071-BA, 21071-DA)
Output Clock Edge: clk1R
Input Sampling Clock Edge: clk2F

epiBEnErr<3:0> is timed with epiData. During epiBus transfers from the 21071-DA chip to the 21071-BA chips, this field indicates which bytes of the longword on the epiData bus are valid. When an epiBEnErr<3:0> bit is set (high), the corresponding byte is valid. The byte enable is used for DMA write transfers and ignored on I/O read transfers.

During epiBus transfers from the 21071-BA data chips to the 21071-DA chip, epiBEnErr<0> is asserted if the longword being sent on epiData contains a parity error or uncorrectable ECC error. epiBEnErr<1> is asserted if the longword being sent on epiData contained a correctable ECC error. Table 14–4 lists the epiBEnErr functions.

Table 14–4 epiBEnErr Functions

Signal	Transfers to 21071-BA	Transfers from 21071-BA
epiBEnErr<0>	epiData<7:0> byte enable	DMA read or I/O write uncorrectable error (this longword)
epiBEnErr<1>	epiData<15:8> byte enable	DMA read or I/O write corrected error (this longword)
epiBEnErr<2>	epiData<23:16> byte enable	Reserved
epiBEnErr<3>	epiData<31:24> byte enable	Reserved

14.2.3.3 epiFromIOB

Signal Type: 21071-BA Input
Signal Source: 21071-DA
Input Sampling Clock Edge: clk2F

epiFromIOB indicates the direction of epiData to the 21071-BA chips. When epiFromIOB is deasserted, only the 21071-BA chip selected with epiEnable drives epiData<31:0> and epiBEnErr<3:0>. When epiFromIOB is asserted, the 21071-BA chips receive data on epiData<31:0> and epiBEnErr<3:0>.

14.2.3.4 epiSelDMA

Signal Type: 21071-BA Input
Signal Source: 21071-DA
Input Sampling Clock Edge: clk2F

epiSelDMA is used by the 21071-BA chips when epiFromIOB is asserted, to determine whether the destination of epiData is the DMA write buffer (epiSelDMA = high) or the I/O read buffer (epiSelDMA = low).

14.2.3.5 epiEnable<1:0>

Signal Type: 21071-BA Input
Signal Source: 21071-DA
Input Sampling Clock Edge: clk2F

The epiEnable<1:0> signals are asserted by the 21071-DA chip to the 21071-BA chip to indicate that the 21071-DA is performing an epiBus transfer. When epiEnable is driven low, the epiData and epiBus control signals are ignored.

epiEnable is used to determine which longword within the octaword has to be driven onto and received from the epiData bus in the following cycle.

The *command* is always sent 1 cycle prior to the corresponding data.

Table 14–5 indicates the function performed by the 21071-BA chips based on the values of epiEnable, epiFromIOB, and epiSelDMA.

Table 14–5 21071-BA epiBus Interface Function

epiEnable	epiFromIOB	epiSelDMA	Function
0	X	X	No action, except for possible line invalidate; epiData is tristated.
1	0	X	The DMA read or I/O write buffer is driven onto epiData.
1	1	0	epiData is loaded into the I/O read buffer.
1	1	1	epiData is loaded into the DMA write buffer.

14.2.3.6 epiOWSel

Signal Type: 21071-BA Input
Signal Source: 21071-DA
Input Sampling Clock Edge: clk2F

epiOWSel is used by the 21071-BA chips to select the octaword within the cache line that has to be written or read using the epiData bus. When epiOWSel is 0, the lower octaword is selected. When epiOWSel is 1, the upper octaword is selected.

14.2.3.7 epiLineSel<1:0>

Signal Type: 21071-BA Input
Signal Source: 21071-DA
Input Sampling Clock Edge: clk2F

epiLineSel<1:0> is used to select the cache line of the DMA read or I/O write buffer that has to be read onto the epiBus. epiLineSel<1:0> is also used to select the cache line of the DMA write buffer to be loaded from the epiBus.

14.2.3.8 ioLineSel<1:0>

Signal Type: 21071-BA Input
Signal Source: 21071-DA
Input Sampling Clock Edge: clk2F

ioLineSel<1:0> is used to select the cache line of the DMA read or I/O write buffer that has to be loaded from the sysBus.

14.2.3.9 epiLineInval

Signal Type: 21071-BA Input
Signal Source: 21071-DA
Input Sampling Clock Edge: clk2F

When epiLineInval is asserted, all byte enables for the selected cache line will be cleared in the DMA write buffer.

14.2.4 Memory Signals

This section describes the memory signals.

14.2.4.1 memData<31:0>, memPar<0>

Signal Type: Bidirectional (21071-BA, Memory)
Input Sampling Clock Edge: memClkR
Output Clock Edge: memClkR

memData<31:0> is a bidirectional bus which provides data to and from the 21071-BA chip and memory. memPar<0> is the corresponding parity bit.

The 21071-BA chip is the default driver of memData<31:0>. memData<31:0> is driven during all transactions except memory reads. During reads, memData<31:0> is tristated on memClkR. The 21071-CA chip controls the turn-on and turn-off of the memData bus with drvMemData. The timing for driving out write data or latching in read data is controlled by the 21071-CA chip using memCmd<3:1>.

14.2.5 Miscellaneous/Clock Signals

This section describes the miscellaneous and clock signals.

14.2.5.1 clk1x2

Signal Type: 21071-BA Input
Signal Source: Clock generator

clk1x2 is a clock input which supplies a clock at twice the frequency of the DECchip 21064 sysClkOut1, with a minimum period of 15 ns and a 50% duty cycle.

14.2.5.2 clk2ref

Signal Type: 21071-BA Input
Signal Source: Clock generator

clk2ref is a signal input which is low when the assertion of clk1x2 corresponds to the assertion of sysClkOut1. The received signal must be setup to the assertion of clk1x2.

14.2.5.3 reset_l

Signal Type: 21071-BA Input
Signal Source: External logic
Input Clock Edge: Asynchronous on assertion, clk1R on deassertion

Assertion of reset_l sets all internal logic and state machines to their initialized states.

14.2.5.4 testMode

Signal Type: 21071-BA Input
Signal Source: Test logic
Input Clock Edge: Asynchronous

Assertion of testMode places the chip into a mode for chip testing. testMode is intended to be used only during chip testing, and it must be tied low during normal system operation.

testMode has a weak internal pull-down and a Schmitt trigger input.

14.2.5.5 tristate_l

Signal Type: 21071-BA Input
Signal Source: External logic
Input Clock Edge: Asynchronous

Assertion of this signal tristates all output and bidirectional drivers. tristate_l is intended for use during chip testing and power-up.

tristate_l has a weak internal pull-up and a Schmitt trigger input.

14.2.5.6 pTestout

Signal Type: 21071-BA Output
Signal Source: Test logic
Output Clock Edge: Flow through

The pTestout signal contains the output from the Parametric NAND tree, as required for testability. The testMode signal must be asserted for pTestout to be valid. pTestout is intended for use only during chip testing.

14.2.5.7 eccMode

Signal Type: 21071-BA
Input Clock Edge: Static

The eccMode signal is an input to the 21071-BA chip which indicates the type of error-checking used on the module. eccMode tied high indicates that the 21071-BA chip must use the 7-bit ECC code used by the DECchip 21064; eccMode tied low indicates that the 21071-BA chip must use longword parity checking. See Section 15.2.6 for a description of how and when the 21071-BA chip performs data checks and corrections.

eccMode should be used only in conjunction with a 128-bit memory data bus (using four 21071-BA chips).

Caution

eccMode tied high with wideMem tied low will result in UNDEFINED behavior and may cause damage to system hardware.

eccMode has a weak internal pull-down and a Schmitt trigger input buffer.

Note

Changing eccMode after reset is deasserted may result in UNDEFINED behavior.

14.2.5.8 wideMem

Signal Type: 21071-BA Input
Input Clock Edge: Static

The wideMem signal is an input to the 21071-BA chip that indicates the width of the memory data bus. wideMem tied high indicates a 128-bit wide memory data bus (DECchip 21072); wideMem tied low indicates a 64-bit wide memory data bus (DECchip 21071).

wideMem has a weak internal pull-down and a Schmitt trigger input buffer.

Note

Changing wideMem after reset is deasserted may result in UNDEFINED behavior.

14.3 DECchip 21071-BA Pin Connection Table

This section includes DECchip 21071-BA pin connection tables.

Table 14–6 DECchip 21071-BA Pin Assignments for DECchip 21072 with Parity

	Module Trace Name			
21071-BA Pin Name	**21071-BA Chip #3**	**21071-BA Chip #2**	**21071-BA Chip #1**	**21071-BA Chip #0**
eccMode	VSS	VSS	VSS	VSS
wideMem	VCC	VCC	VCC	VCC
epiBEnErr<3:0>	epiBEnErr<3:0>	epiBEnErr<3:0>	epiBEnErr<3:0>	epiBEnErr<3:0>
epiData<31:0>	epiData<31:0>	epiData<31:0>	epiData<31:0>	epiData<31:0>
epiEnable<1>	VSS	VSS	VSS	VSS
epiEnable<0>	epiEnable<3>	epiEnable<2>	epiEnable<1>	epiEnable<0>
memData<31:0>	memData<127:96>	memData<95:64>	memData<63:32>	memData<31:0>
memPar<0>	memPar<3>	memPar<2>	memPar<1>	memPar<0>
drvSysCSR	VSS	VSS	VSS	drvSysCSR
drvSysData	drvSysData	drvSysData	drvSysData	drvSysData
subCmd<1>	subCmdCommon	subCmdCommon	subCmdCommon	subCmdCommon
subCmd<0>	subCmdB<1>	subCmdA<1>	subCmdB<0>	subCmdA<0>
sysData<63:32>	—[1]	—[1]	—[1]	—[1]
sysData<31:0>	sysData<127:96>	sysData<95:64>	sysData<63:32>	sysData<31:0>
sysPar<1>	—[1]	—[1]	—[1]	—[1]
sysPar<0>	sysCheck<21>	sysCheck<14>	sysCheck<7>	sysCheck<0>

[1] *Tie off to* VCC *or* VSS *with resistor*

Table 14–7 DECchip Pin Assignments for DECchip 21072 with ECC

	Module Trace Name			
21071-BA Pin Name	**21071-BA Chip #3**	**21071-BA Chip #2**	**21071-BA Chip #1**	**21071-BA Chip #0**
eccMode	vcc	vcc	vcc	vcc
wideMem	vcc	vcc	vcc	vcc
epiBEnErr<3:0>	epiBEnErr<3:0>	epiBEnErr<3:0>	epiBEnErr<3:0>	epiBEnErr<3:0>
epiData<31:0>	epiData<31:0>	epiData<31:0>	epiData<31:0>	epiData<31:0>
epiEnable<1>	vss	vss	vss	vss
epiEnable<0>	epiEnable<3>	epiEnable<2>	epiEnable<1>	epiEnable<0>
memData<31:0>	memData<127:96>	memData<95:64>	memData<63:32>	memData<31:0>
memPar<0>	N/C[2]	N/C[2]	N/C[2]	N/C[2]
drvSysCSR	vss	vss	vss	drvSysCSR
drvSysData	drvSysData	drvSysData	drvSysData	drvSysData
subCmd<1>	subCmdCommon	subCmdCommon	subCmdCommon	subCmdCommon
subCmd<0>	subCmdB<1>	subCmdA<1>	subCmdB<0>	subCmdA<0>
sysData<63:57>	memCheck<21:27>	memCheck<14:20>	memCheck<7:13>	memCheck<0:6>
sysData<56:39>	—[1]	—[1]	—[1]	—[1]
sysData<38:32>	sysCheck<27:21>	sysCheck<20:14>	sysCheck<13:7>	sysCheck<6:0>
sysData<31:0>	sysData<127:96>	sysData<95:64>	sysData<63:32	sysData<31:0>
sysPar<1:0>	—[1]	—[1]	—[1]	—[1]

[1] *Tie off to* vcc *or* vss *with resistor*

[2] N/C = not connected

Table 14–8 DECchip 21071-BA Pin Assignments for DECchip 21071 With Parity[1]

	Module Trace Name	
21071-BA Pin Name	**21071-BA Chip #1**	**21071-BA Chip #0**
eccMode	vss	vss
wideMem	vss	vss
epiBEnErr<3:0>	epiBEnErr<3:0>	epiBEnErr<3:0>
epiData<31:0>	epiData<31:0>	epiData<31:0>
epiEnable<1>	epiEnable<3>	epiEnable<2>
epiEnable<0>	epiEnable<1>	epiEnable<0>
memData<31:0>	memData<63:32>	memData<31:0>
memPar<0>	memPar<1>	memPar<0>
drvSysCSR	vss	drvSysCSR
drvSysData	drvSysData	drvSysData
subCmd<1>	subCmdB<1>	subCmdA<1>
subCmd<0>	subCmdB<0>	subCmdA<0>
sysData<63:32>	sysData<127:96>	sysData<95:64>
sysData<31:0>	sysData<63:32>	sysData<31:0>
sysPar<1>	sysCheck<21>	sysCheck<14>
sysPar<0>	sysCheck<7>	sysCheck<0>

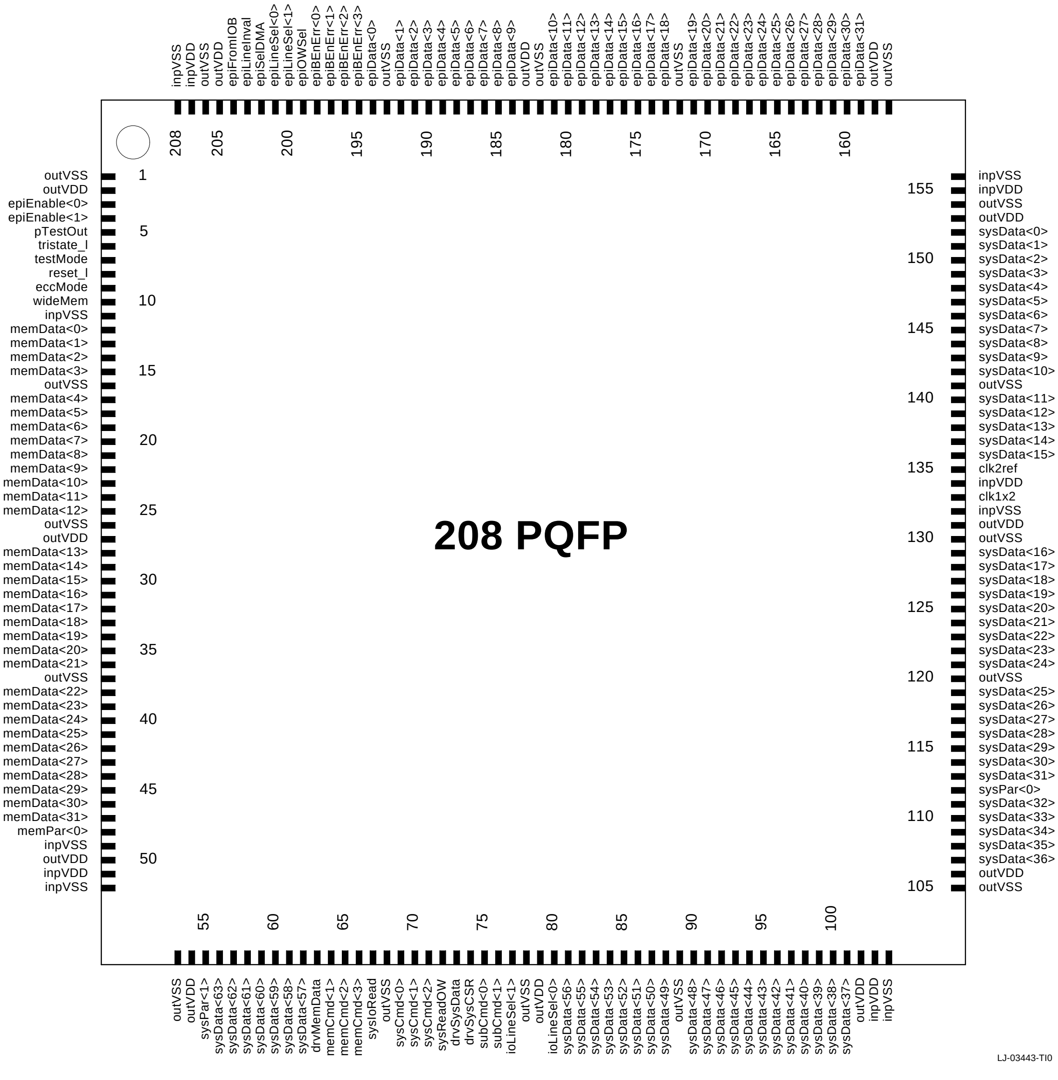
14.4 DECchip 21071-BA Pin Assignment

The DECchip 21071-BA is a 208-pin plastic quad flat pack (PQFP). Figure 14–1 shows the signal assignments. Sections 14.4.1 and 14.4.2 provide alphabetical and numerical pin listings.

---

[1] DECchip 21071-BA does not support ECC with 64-bit memory.

Figure 14–1 DECchip 21071-BA Pinout Diagram



14.4.1 DECchip 21071-BA Alphabetical Pin Assignment List

Table 14–9 lists the DECchip 21071-BA pins in alphabetical order. The following list describes the abbreviations used in the Type column of the table.

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

Table 14–9 Alphabetical Pin Assignment List

Pin Name	Pin	Type	Pin Name	Pin	Type
clk1x2	133	I	**epiData<16>**	175	B
clk2ref	135	I	**epiData<17>**	174	B
drvMemData	63	I	**epiData<18>**	173	B
drvSysCSR	74	I	**epiData<19>**	171	B
drvSysData	73	I	**epiData<20>**	170	B
eccMode	9	I	**epiData<21>**	169	B
epiBEnErr<0>	198	B	**epiData<22>**	168	B
epiBEnErr<1>	197	B	**epiData<23>**	167	B
epiBEnErr<2>	196	B	**epiData<24>**	166	B
epiBEnErr<3>	195	B	**epiData<25>**	165	B
epiData<0>	194	B	**epiData<26>**	164	B
epiData<1>	192	B	**epiData<27>**	163	B
epiData<2>	191	B	**epiData<28>**	162	B
epiData<3>	190	B	**epiData<29>**	161	B
epiData<4>	189	B	**epiData<30>**	160	B
epiData<5>	188	B	**epiData<31>**	159	B
epiData<6>	187	B	**epiEnable<0>**	3	I
epiData<7>	186	B	**epiEnable<1>**	4	I
epiData<8>	185	B	**epiFromIOB**	204	I
epiData<9>	184	B	**epiLineInval**	203	I
epiData<10>	181	B	**epiLineSel<0>**	201	I
epiData<11>	180	B	**epiLineSel<1>**	200	I
epiData<12>	179	B	**epiOWSel**	199	I
epiData<13>	178	B	**epiSelDMA**	202	I
epiData<14>	177	B	**inpVdd**	51	P
epiData<15>	176	B	**inpVdd**	103	P

Pin Name	Pin	Type	Pin Name	Pin	Type
inpVdd	134	P	**memData<14>**	29	B
inpVdd	155	P	**memData<15>**	30	B
inpVdd	207	P	**memData<16>**	31	B
inpVss	11	P	**memData<17>**	32	B
inpVss	49	P	**memData<18>**	33	B
inpVss	52	P	**memData<19>**	34	B
inpVss	104	P	**memData<20>**	35	B
inpVss	132	P	**memData<21>**	36	B
inpVss	156	P	**memData<22>**	38	B
inpVss	208	P	**memData<23>**	39	B
ioLineSel<0>	80	I	**memData<24>**	40	B
ioLineSel<1>	77	I	**memData<25>**	41	B
memCmd<1>	64	I	**memData<26>**	42	B
memCmd<2>	65	I	**memData<27>**	43	B
memCmd<3>	66	I	**memData<28>**	44	B
memData<0>	12	B	**memData<29>**	45	B
memData<1>	13	B	**memData<30>**	46	B
memData<2>	14	B	**memData<31>**	47	B
memData<3>	15	B	**memPar<0>**	48	B
memData<4>	17	B	**outVdd**	2	P
memData<5>	18	B	**outVdd**	27	P
memData<6>	19	B	**outVdd**	50	P
memData<7>	20	B	**outVdd**	54	P
memData<8>	21	B	**outVdd**	79	P
memData<9>	22	B	**outVdd**	102	P
memData<10>	23	B	**outVdd**	106	P
memData<11>	24	B	**outVdd**	131	P
memData<12>	25	B	**outVdd**	153	P
memData<13>	28	B	**outVdd**	158	P

Pin Name	Pin	Type	Pin Name	Pin	Type
outVdd	183	P	**sysData<5>**	147	B
outVdd	205	P	**sysData<6>**	146	B
outVss	1	P	**sysData<7>**	145	B
outVss	16	P	**sysData<8>**	144	B
outVss	26	P	**sysData<9>**	143	B
outVss	37	P	**sysData<10>**	142	B
outVss	53	P	**sysData<11>**	140	B
outVss	68	P	**sysData<12>**	139	B
outVss	78	P	**sysData<13>**	138	B
outVss	105	P	**sysData<14>**	137	B
outVss	89	P	**sysData<15>**	136	B
outVss	120	P	**sysData<16>**	129	B
outVss	130	P	**sysData<17>**	128	B
outVss	141	P	**sysData<18>**	127	B
outVss	154	P	**sysData<19>**	126	B
outVss	157	P	**sysData<20>**	125	B
outVss	172	P	**sysData<21>**	124	B
outVss	182	P	**sysData<22>**	123	B
outVss	193	P	**sysData<23>**	122	B
outVss	206	P	**sysData<24>**	121	B
pTestout	5	O	**sysData<25>**	119	B
reset_l	8	I	**sysData<26>**	118	B
sysCmd<0>	69	I	**sysData<27>**	117	B
sysCmd<1>	70	I	**sysData<28>**	116	B
sysCmd<2>	71	I	**sysData<29>**	115	B
sysData<0>	152	B	**sysData<30>**	114	B
sysData<1>	151	B	**sysData<31>**	113	B
sysData<2>	150	B	**sysData<32>**	111	B
sysData<3>	149	B	**sysData<33>**	110	B
sysData<4>	148	B	**sysData<34>**	109	B

Pin Name	Pin	Type	Pin Name	Pin	Type
sysData<35>	108	B	**sysData<60>**	59	B
sysData<36>	107	B	**sysData<61>**	58	B
sysData<37>	101	B	**sysData<62>**	57	B
sysData<38>	100	B	**sysData<63>**	56	B
sysData<39>	99	B	**sysIORead**	67	I
sysData<40>	98	B	**sysPar<0>**	112	B
sysData<41>	97	B	**sysPar<1>**	55	B
sysData<42>	96	B	**sysReadOW**	72	I
sysData<43>	95	B	**subCmd<0>**	75	I
sysData<44>	94	B	**subCmd<1>**	76	I
sysData<45>	93	B	**testMode**	7	I
sysData<46>	92	B	**tristate_l**	6	I
sysData<47>	91	B	**wideMem**	10	I
sysData<48>	90	B			
sysData<49>	88	B			
sysData<50>	87	B			
sysData<51>	86	B			
sysData<52>	85	B			
sysData<53>	84	B			
sysData<54>	83	B			
sysData<55>	82	B			
sysData<56>	81	B			
sysData<57>	62	B			
sysData<58>	61	B			
sysData<59>	60	B			

14.4.2 DECchip 21071-BA Numerical Pin Assignment List

Table 14–10 lists the DECchip 21071-BA pins in numerical order. The following list describes the abbreviations used in the Type column of the table.

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

Table 14–10 DECchip 21071-BA Numerical Pin Assignment List

Pin Name	Pin	Type	Pin Name	Pin	Type
outVss	1	P	**outVss**	26	P
outVdd	2	P	**outVdd**	27	P
epiEnable<0>	3	I	**memData<13>**	28	B
epiEnable<1>	4	I	**memData<14>**	29	B
pTestout	5	O	**memData<15>**	30	B
tristate_l	6	I	**memData<16>**	31	B
testMode	7	I	**memData<17>**	32	B
reset_l	8	I	**memData<18>**	33	B
eccMode	9	I	**memData<19>**	34	B
wideMem	10	I	**memData<20>**	35	B
inpVss	11	P	**memData<21>**	36	B
memData<0>	12	B	**outVss**	37	P
memData<1>	13	B	**memData<22>**	38	B
memData<2>	14	B	**memData<23>**	39	B
memData<3>	15	B	**memData<24>**	40	B
outVss	16	P	**memData<25>**	41	B
memData<4>	17	B	**memData<26>**	42	B
memData<5>	18	B	**memData<27>**	43	B
memData<6>	19	B	**memData<28>**	44	B
memData<7>	20	B	**memData<29>**	45	B
memData<8>	21	B	**memData<30>**	46	B
memData<9>	22	B	**memData<31>**	47	B
memData<10>	23	B	**memPar<0>**	48	B
memData<11>	24	B	**inpVss**	49	P
memData<12>	25	B	**outVdd**	50	P

Pin Name	Pin	Type	Pin Name	Pin	Type
inpVdd	51	P	**outVdd**	79	P
inpVss	52	P	**ioLineSel<0>**	80	I
outVss	53	P	**sysData<56>**	81	B
outVdd	54	P	**sysData<55>**	82	B
sysPar<1>	55	B	**sysData<54>**	83	B
sysData<63>	56	B	**sysData<53>**	84	B
sysData<62>	57	B	**sysData<52>**	85	B
sysData<61>	58	B	**sysData<51>**	86	B
sysData<60>	59	B	**sysData<50>**	87	B
sysData<59>	60	B	**sysData<49>**	88	B
sysData<58>	61	B	**outVss**	89	P
sysData<57>	62	B	**sysData<48>**	90	B
drvMemData	63	I	**sysData<47>**	91	B
memCmd<1>	64	I	**sysData<46>**	92	B
memCmd<2>	65	I	**sysData<45>**	93	B
memCmd<3>	66	I	**sysData<44>**	94	B
sysIORead	67	I	**sysData<43>**	95	B
outVss	68	P	**sysData<42>**	96	B
sysCmd<0>	69	I	**sysData<41>**	97	B
sysCmd<1>	70	I	**sysData<40>**	98	B
sysCmd<2>	71	I	**sysData<39>**	99	B
sysReadOW	72	I	**sysData<38>**	100	B
drvSysData	73	I	**sysData<37>**	101	B
drvSysCSR	74	I	**outVdd**	102	P
subCmd<0>	75	I	**inpVdd**	103	P
subCmd<1>	76	I	**inpVss**	104	P
ioLineSel<1>	77	I	**outVss**	105	P
outVss	78	P	**outVdd**	106	P

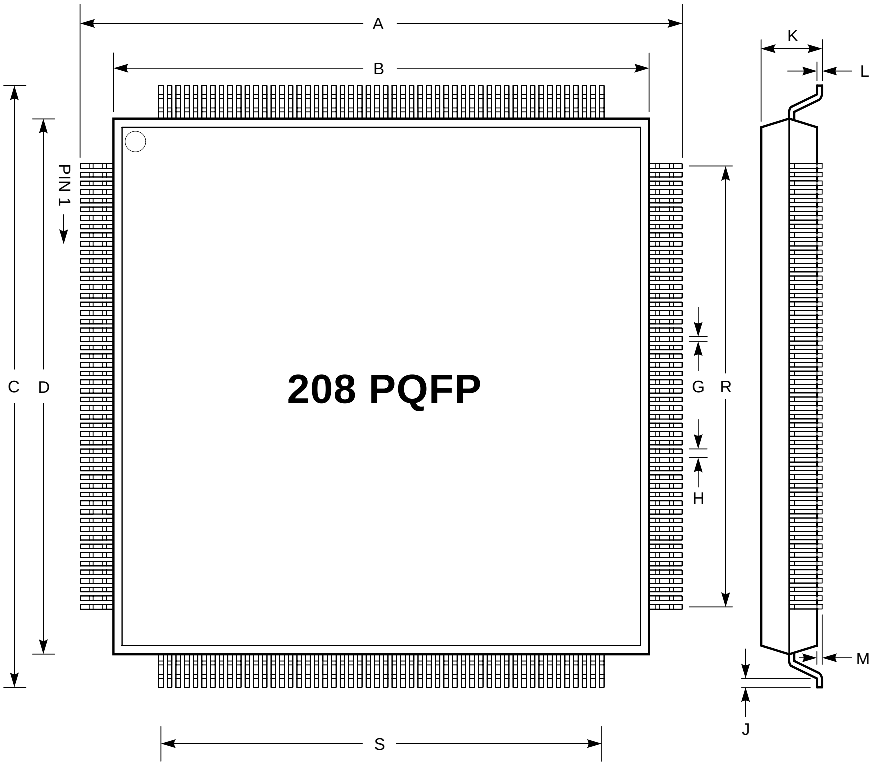
Pin Name	Pin	Type	Pin Name	Pin	Type
sysData<36>	107	B	**sysData<13>**	138	B
sysData<35>	108	B	**sysData<12>**	139	B
sysData<34>	109	B	**sysData<11>**	140	B
sysData<33>	110	B	**outVss**	141	P
sysData<32>	111	B	**sysData<10>**	142	B
sysPar<0>	112	B	**sysData<9>**	143	B
sysData<31>	113	B	**sysData<8>**	144	B
sysData<30>	114	B	**sysData<7>**	145	B
sysData<29>	115	B	**sysData<6>**	146	B
sysData<28>	116	B	**sysData<5>**	147	B
sysData<27>	117	B	**sysData<4>**	148	B
sysData<26>	118	B	**sysData<3>**	149	B
sysData<25>	119	B	**sysData<2>**	150	B
outVss	120	P	**sysData<1>**	151	B
sysData<24>	121	B	**sysData<0>**	152	B
sysData<23>	122	B	**outVdd**	153	P
sysData<22>	123	B	**outVss**	154	P
sysData<21>	124	B	**inpVdd**	155	P
sysData<20>	125	B	**inpVss**	156	P
sysData<19>	126	B	**outVss**	157	P
sysData<18>	127	B	**outVdd**	158	P
sysData<17>	128	B	**epiData<31>**	159	B
sysData<16>	129	B	**epiData<30>**	160	B
outVss	130	P	**epiData<29>**	161	B
outVdd	131	P	**epiData<28>**	162	B
inpVss	132	P	**epiData<27>**	163	B
clk1x2	133	I	**epiData<26>**	164	B
inpVdd	134	P	**epiData<25>**	165	B
clk2Ref	135	I	**epiData<24>**	166	B
sysData<15>	136	B	**epiData<23>**	167	B
sysData<14>	137	B	**epiData<22>**	168	B

Pin Name	Pin	Type	Pin Name	Pin	Type
epiData<21>	169	B	**epiBEnErr<2>**	196	B
epiData<20>	170	B	**epiBEnErr<1>**	197	B
epiData<19>	171	B	**epiBEnErr<0>**	198	B
outVss	172	P	**epiOWSel**	199	I
epiData<18>	173	B	**epiLineSel<1>**	200	I
epiData<17>	174	B	**epiLineSel<0>**	201	I
epiData<16>	175	B	**epiSelDMA**	202	I
epiData<15>	176	B	**epiLineInval**	203	I
epiData<14>	177	B	**epiFromIOB**	204	I
epiData<13>	178	B	**outVdd**	205	P
epiData<12>	179	B	**outVss**	206	P
epiData<11>	180	B	**inpVdd**	207	P
epiData<10>	181	B	**inpVss**	208	P
outVss	182	P			
outVdd	183	P			
epiData<9>	184	B			
epiData<8>	185	B			
epiData<7>	186	B			
epiData<6>	187	B			
epiData<5>	188	B			
epiData<4>	189	B			
epiData<3>	190	B			
epiData<2>	191	B			
epiData<1>	192	B			
outVss	193	P			
epiData<0>	194	B			
epiBEnErr<3>	195	B			

14.5 DECchip 21071-BA Mechanical Specifications

Figure 14–2 shows the package dimensions for the DECchip 21071-BA chip.

Figure 14–2 DECchip 21071-BA Package Dimensions



DIM	Millimeters		Inches	
	MIN	**MAX**	**MIN**	**MAX**
A	30.50	30.77	1.201	1.211
B	27.90	28.10	1.098	1.106
C	30.50	30.77	1.201	1.211
D	27.90	28.10	1.098	1.106
G	0.23	0.33	0.009	0.013
H	.500 BSC		0.0197 BSC	
J	0.45	0.62	0.018	0.024
K	3.45	3.85	0.136	0.152
L	0.13	0.23	0.005	0.009
M	0.25	0.35	0.010	0.012
R	25.5 REF		1.004 REF	
S	25.5 REF		1.004 REF	

LJ-03666-TI0

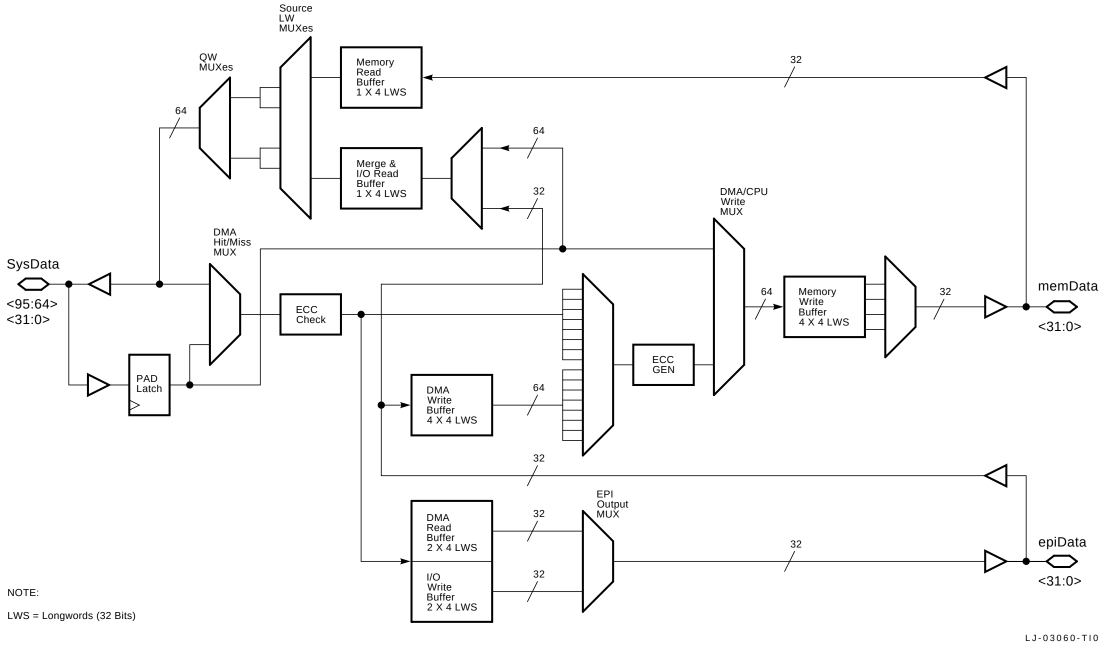
15

DECchip 21071-BA Architecture Overview

This chapter describes the architecture of the DECchip 21071-BA chip.

Figure 15–1 shows a block diagram of the DECchip 21071-BA chip.

Figure 15–1 DECchip 21071-BA Block Diagram



15.1 Bus Widths

This section describes the bus widths.

15.1.1 sysData Bus

Each 21071-BA data chip has 64 pins for the sysData bus. The DECchip 21071 and DECchip 21072 configured systems support only a 128-bit wide sysData bus; a 64-bit wide sysData bus is not supported.

In a DECchip 21071 configuration, sysData pins on each 21071-BA chip are connected to the sysBus:

- The lower 21071-BA data chip (#0) connects to sysData<31:0> (longword 0) and sysData<95:64> (longword 2).
- The upper 21071-BA data chip (#1) connects to sysData<63:32> (longword 1) and sysData<127:96> (longword 3).

In a DECchip 21072 configuration, only the lower 32-bits of the sysData bus of each 21071-BA chip are used:

- 21071-BA data chip #0 connects to longword 0 (sysData<31:0>).
- 21071-BA data chip #1 connects to longword 1 (sysData<63:32>).
- 21071-BA data chip #2 connects to longword 2 (sysData<95:64>).
- 21071-BA data chip #3 connects to longword 3 (sysData<127:96>).

15.1.2 memData Bus

The number of 21071-BA data chips used in a system depends on the width of the memData bus. If the width of the memData bus is 64-bits, two 21071-BA data chips are required (DECchip 21071). If the width of the memData bus is 128-bits, four 21071-BA chips are required (DECchip 21072).

Each 21071-BA data chip connects to 32 bits of memData. In all systems:

- 21071-BA #0 connects to longword 0 (memData<31:0>).
- 21071-BA #1 connects to longword 1 (memData<63:32>).

In a 4-chip configured system:

- 21071-BA #2 connects to longword 2 (memData<95:64>).
- 21071-BA #3 connects to longword 3 (memData<127:64>).

Each 21071-BA data chip needs to know the width of the memData bus for proper operation. This is obtained from the wideMem pin. The 21071-BA data chips do not need to know which longword they are connected to. The proper latching and driving of data is achieved by appropriately connecting the 21071-CA and 21071-DA command signals (Section 14.3).

15.1.3 epiData Bus

Each 21071-BA data chip has 32 epiData pins.

The epiData pins of all the 21071-BA data chips are tied together to form a 32-bit wide epiData bus.

15.2 Description of DECchip 21071-BA Architecture

This section describes the DECchip 21071-BA architecture.

15.2.1 Memory Read Buffer

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

Each 21071-BA data chip stores four longwords worth of data and corresponding check bits in the memory read buffer.

- In a two 21071-BA data chip designed system, the total storage is eight longwords or a cache line.
- A four 21071-BA data chip designed system contains an additional eight longwords of storage; however, this extra storage is not usable.

15.2.2 I/O Read Buffer and Merge Buffer

On CPU-initiated memory transactions, this buffer performs the merge buffer functions described in Section 3.1.7. On CPU-initiated I/O reads addressed to or through the 21071-DA chip, this buffer acts as the I/O read buffer. The loading of data into this buffer is therefore controlled by both the 21071-CA and 21071-DA chips.

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

15.2.3 I/O Write Buffer and DMA Read Buffer

This buffer can store up to four entries of data. Each entry has four longwords per 21071-BA data chip. Data from this buffer is sent out on the epiData bus. System designers may choose to allocate each entry of this buffer according to their needs. The 21071-DA chip may use the full cache line available in each entry.

In the 21071 or 21072 implementation, two entries of this buffer are allocated for I/O write data storage, and two entries are allocated for DMA read data storage.

In a two 21071-BA chip system, storing one cache line uses all four longwords of each DMA read buffer entry; in a four 21071-BA chip system it uses only two of the four longwords of each entry, but the extra storage is not accessible.

The loading of each entry can be controlled separately, thus allowing maximum flexibility in allocating the buffer entries to the 21071-DA.

The loading of this buffer is handled by the 21071-CA chip, with the address provided by the 21071-DA on ioLineSel<1:0>. The 21071-DA chip controls unloading of this buffer.

15.2.4 DMA Write Buffer

The DMA write buffer has four entries. Each entry contains four longwords per 21071-BA and corresponding byte masks. In a four 21071-BA data chip system, only half the storage per entry is used. The extra storage is not accessible.

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

15.2.5 Memory Write Buffer

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

15.2.6 Error Checking/Correction

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

If ECC is enabled, and the Bcache/memory data contains a correctable error, the 21071-BA data chip sends corrected data to its destination (DMA read buffer for DMA reads, memory write buffer for MUXing with DMA write data for DMA writes).

If the data contains an uncorrectable error (dual-bit ECC error or any parity error), then the 21071-DA is notified (for a DMA read), or bad ECC/parity is written back into memory (for a DMA write).

In case of a DMA write masked transaction, parity/ECC is calculated for the merged data going into the memory write buffer.

The 21071-BA data chip uses the same ECC code as the DECchip 21064 microprocessor. See the *DECchip 21064 Hardware Reference Manual* for details.

15.3 Data Path Logic

This section describes the data path logic.

15.3.1 epiBus

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

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

15.3.2 sysBus Output Selectors

Two levels of muxes select the output for the sysData bus. The first level selects the source for each longword of data and check bits, and the second level selects the 2 longwords to be driven on the sysData bus.

The source is described in Table 15–1. In 64-bit memory mode, the lower and upper mux work together to select longwords 0 and 2 in the first cycle (while the other 21071-BA data chip selects longwords 1 and 3) and to select 4 and 6 in the second cycle (while the other 21071-BA data chip selects longwords 5 and 7).

Table 15–1 sysBus Output Sources

Buffer	Function
Memory read	DMA and CPU read, DMA write masked
Merge	LD*x*_L, ST*x*_C
Merge and memory read	CPU write allocates
I/O read	CPU I/O space reads

The lower 16 bits of the sysData bus are controlled by a special signal to enable the 21071-CA chip to drive the lower 16 bits on CSR reads from the 21071-CA chip while the 21071-BA data chips drive the remaining data lines.

16

DECchip 21071-BA Transactions and Timing Diagrams

This chapter describes the flow of data within the 21071-BA chip on various transactions on the sysBus, memory data bus, and epiBus.

16.1 sysBus Transactions

This section describes the sysBus transactions.

16.1.1 CPU Memory Read

Read data from memory is loaded into the memory read buffer by the memory control machine in the 21071-CA. This data is available, by default, when the sysBus controller enables the 21071-BA chips to drive the sysData bus.

The sysBus controller sends sysReadOW to indicate when the 21071-BA chips must switch to the high-order octaword.

16.1.2 CPU Memory Read with Victim

The victim data is loaded from the sysBus into the memory write buffer through a holding latch. If the write buffer is full, the data is held in the holding latch until there is room for it in the write buffer. (The control for this is provided by the 21071-CA chip.)

Read data from memory can be loaded into the memory read buffer independent of the loading of the memory write buffer.

16.1.3 CPU Memory Write Allocate

The CPU write data is loaded by the 21071-CA chip into the merge buffer through the holding pad latch. The merge buffer can never be full, so this loading does not stall. If the write is partial, read data from memory is loaded into the memory read buffer. If there is a victim, the victim data is written into the memory write buffer through the holding pad latch.

When all the data is in place (memory read data, CPU write data, and victim data), the appropriate longwords of the memory read buffer and the merge buffer are merged and sent out on sysData.

16.1.4 CPU Memory Write Noncacheable/Noallocate

The data from the sysBus is loaded into the memory write buffer through the holding latch. If the memory write buffer is full, the data has to stall.

Data from the memory write buffer is unloaded by the memory control sequencer from the 21071-CA chip when it is ready to service the write.

16.1.5 ST*x*_C Hit

The write data from the CPU is loaded into the merge buffer. If the address is a hit in the cache, the remaining data is read from the cache and is loaded into the unwritten longwords of the merge buffer. Data from the merge buffer is then sent out on the sysBus.

16.1.6 ST*x*_C Miss

This is exactly like a CPU memory write.

16.1.7 LD*x*_L Hit

Data is read from the cache and is loaded into the merge buffer. It is sent out on the sysBus from there.

16.1.8 LD*x*_L Miss

This is exactly like a CPU memory read.

16.1.9 CPU Read From or Through the DECchip 21071-DA

The 21071-DA chip sets the direction of the epiData bus to be from the 21071-DA chip to the 21071-BA chips. It sets the epiBus controls to indicate the I/O read buffer as the destination of the data. When the I/O read data is available, it is loaded into the I/O read buffer. The I/O read buffer has already been selected as the source of sysBus data by 21071-CA. The I/O read data is thus returned to the CPU.

16.1.10 CPU Write To or Through the DECchip 21071-DA

When an I/O write transaction is detected on the sysBus, the 21071-DA chip is required to set up the controls for the I/O write buffer to point to the appropriate entry of the I/O write buffer. The loading of data is controlled by the 21071-CA chip.

The 21071-DA chip sets the direction of the epiData to point from the 21071-BA chip to the 21071-DA chip, and it extracts the data as needed by controlling the longword select bits and enabling the appropriate 21071-BA chips using epiEnable<3:0>.

16.2 PCI and Other I/O Bus Transactions

This section describes PCI and other I/O bus transactions.

16.2.1 PCI Read from System Memory

The 21071-DA chip performs a DMA read transaction on the sysBus and sets up the controls of the DMA read buffer to point to the appropriate entry of the DMA read buffer.

The 21071-CA chip gets the data from memory or Bcache. If the data is to be read from memory, the memory read buffer is loaded as data is received from memory. Data from the memory read buffer is loaded into the DMA read buffer after error checking has happened. If the data will be read from the Bcache, the data is loaded from the sysBus via the holding pad latch into the DMA read buffer, after error checking has happened.

The 21071-DA chip sets up the direction of the epiData bus to be from the 21071-BA chips to the 21071-DA chip whenever it is ready to receive data. As the data is loaded into the DMA read buffer, it is extracted by the 21071-DA chip.

16.2.2 PCI Write to System Memory

The direction of the epiData bus is set to be from the 21071-DA chip to the 21071-BA chip by the 21071-DA chip. The appropriate controls for loading the correct write buffer entry are set. The write data and the corresponding byte masks are loaded into the selected entry as it is available. If for some reason the write is not valid, the 21071-DA chip can overwrite that entry by using the epiLineInval signal. epiLineInval should be used at the start of any DMA write that does not use the full cache line.

Whenever the 21071-DA chip is ready to do the transaction on the sysBus, a DMA write is initiated. If the DMA write buffer contains completely unmasked data, the data from the DMA write buffer is moved to the memory write buffer after the proper error bits have been generated.

If the DMA write is partially masked, a read-modify-write is performed. Data is read from memory (cache miss) into the memory read buffer or from the sysBus (cache hit) and is merged with the data from the DMA write data based on the DMA write byte masks. Error checking is performed on the read data. If there is no error or a correctable error (error is corrected in this case), check bits are generated for the merged data and are written to the memory write buffer. If there is an uncorrectable error in the read data, the merge is performed but incorrect check bits are written into the memory write buffer. A read from this location will result in a hard error later.

16.3 epiBus Transactions

This section describes the epiBus transactions.

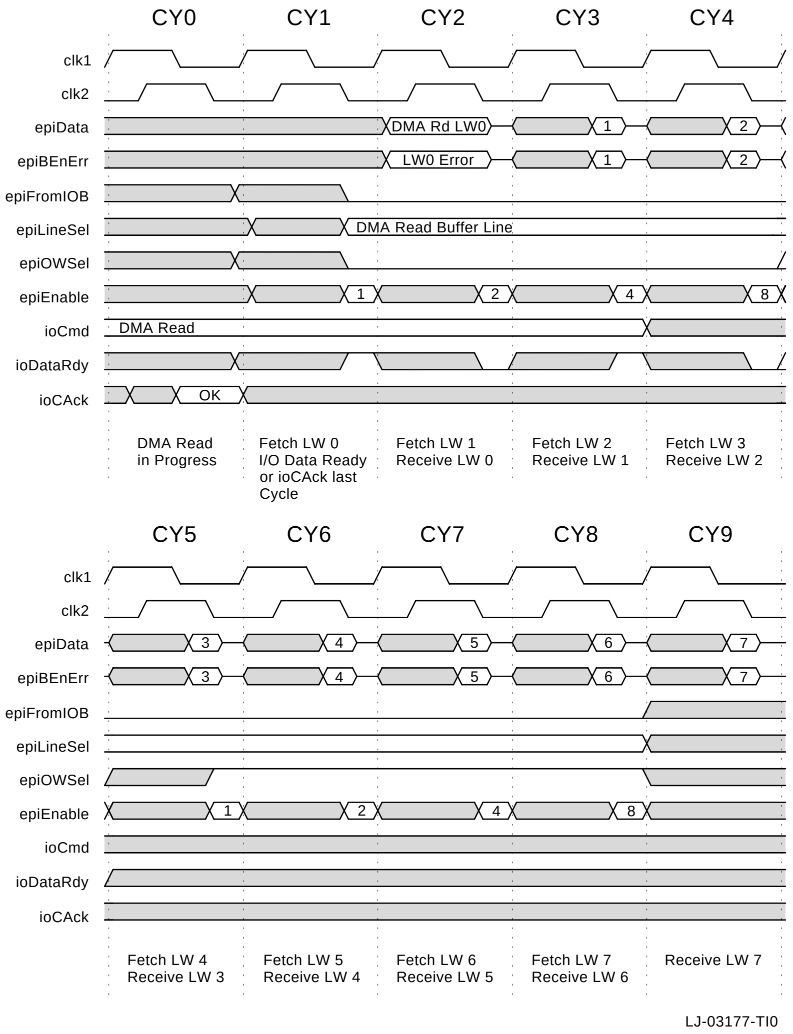
16.3.1 DMA Read Buffer to the 21071-DA

The following table describes the cycles for an epiBus transaction which transfers data from the DMA read buffer to the 21071-DA, as shown in Figure 16–1.

Cycle	Description
0	The 21071-DA chip may read data from the DMA read buffer after the data has been loaded by the 21071-CA chip. The earliest that data may be read out is two cycles after an ioCAck<2:0> for that read or one cycle after sysReadOW for the octaword to be read. ioCAck<2:0> in this cycle of the diagram indicates that data is ready by cycle 2.
1	If ioCAck<2:0> was not sent in cycle 0, a sysReadOW indicates that the first octaword of data may be read out in cycle 2. The 21071-DA chip recognizes that the data is going to be ready. It asserts the epiLineSel<1:0> lines to request a read of the DMA read buffer line which was indicated on ioLineSel when the read was started. The 21071-DA chip places a request for the first longword of read data by deasserting epiFromIOB (indicating a read), deasserting epiOWSel (indicating the first octaword), and asserting epiEnable<0> (LW 0 within first octaword). If the 21071-DA was driving epiData, it must tristate the bus by clk2F.
2	The 21071-BA chip receives the epiBus control signals and begins driving epiData with the first longword of data. The 21071-BA also drives error information on epiBENErr<3:0>. See Table 8–7. The 21071-DA chip requests LW 1 by changing to assert epiEnable<1>. (Shown in figure as a 2, because epiEnable<3:0> = 0010 = 2.) The 21071-DA chip receives and latches epiData<31:0> on clk2F. The 21071-BA receives epiEnable<3:0> and tristates epiData<31:0> and epiBENErr<3:0> on clk2F.

Cycle	Description
3	Similar to cycle 2, the 21071-DA chip requests LW 2, and another 21071-BA chip drives LW 1. EpiData<31:0> and epiBEnErr<3:0> are always one cycle behind the EPI control lines.
4	The 21071-DA chip requests LW 3; a 21071-BA chip drives LW 2.
5	The 21071-DA chip requests LW 4; a 21071-BA chip drives LW 3. Because LW 4 is in the second octaword, epiOWSel asserts and epiEnable<0> is used.
6	The read continues. There is no constraint on the order or number of times that a longword may be read out (as long as the LW is ready, as described in cycle 0).

Figure 16–1 Timing of DMA Read Buffer to the 21071-DA Transfer



16.3.2 I/O Write Buffer to 21071-DA

An epiBus transaction that transfers data from the I/O write buffer to the 21071-DA chip is identical to the previous case shown in Figure 16–1. Because the same buffer is used for both DMA reads and I/O writes, the only difference is that a different buffer line will be requested using epiLineSel<1:0>. (The line that was present on ioLineSel<1:0> when the I/O write occurred.)

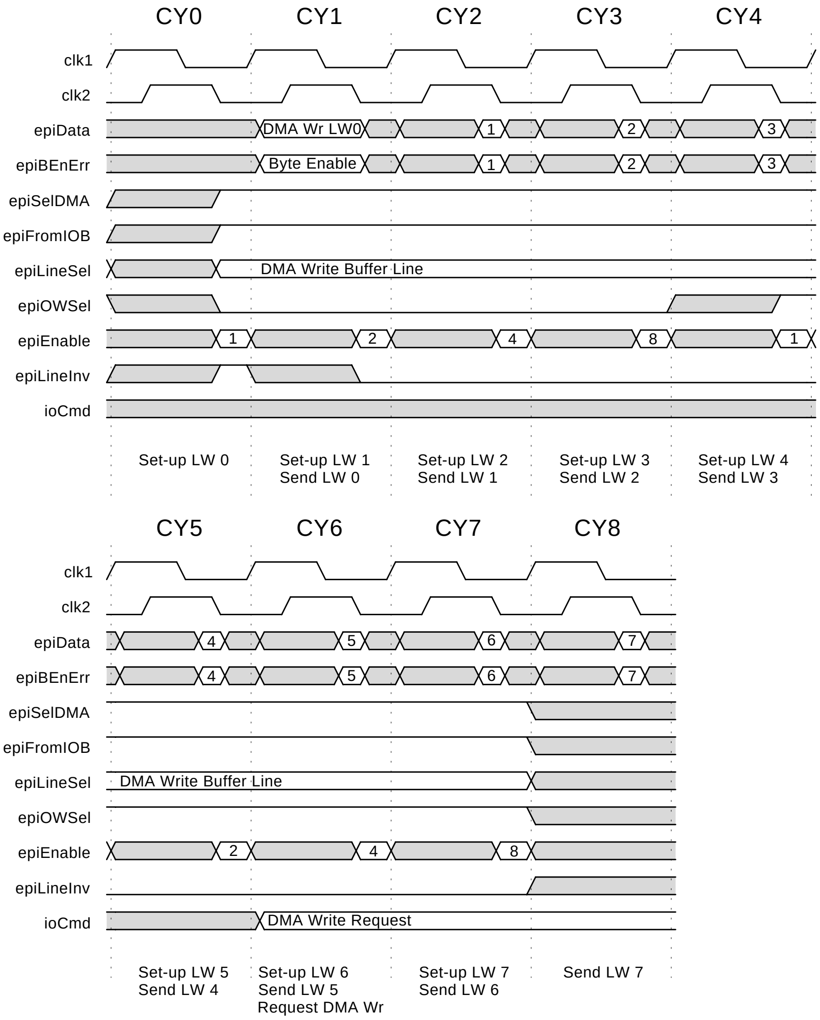
16.3.3 21071-DA to DMA Write Buffer

The following table describes the cycles for an epiBus transaction that transfers data from the 21071-DA chip into the DMA write buffer, as shown in Figure 16–2.

Cycle	Description
0	The 21071-DA chip places a request to store the first longword of DMA write data by asserting epiFromIOB (indicating a write into the 21071-BA), asserting epiSelDMA (indicating a DMA transfer), deasserting epiOWSel (indicating the first octaword), and asserting epiEnable<0> (LW 0 within first octaword). The 21071-DA chip asserts the epiLineSel<1:0> lines to point to an empty line in the DMA read buffer. Because this is the first store to this DMA write buffer line, epiLineInval is asserted to clear all of the byte enables left over from the previous usage of the cache line. If a 21071-BA chip was driving epiData<31:0>, then it will tristate the bus by clk2F.
1	The 21071-DA chip sends the data to be stored on epiData<31:0>. The 21071-DA chip drives epiBENErr<3:0> with the byte enables for the 4 bytes in the longword. (epiBEnErr<3:0> is on if the byte is valid.) The 21071-BA chip receives the epiBus control signals and latches LW 0 into the I/O read buffer. The 21071-DA chip requests that LW 1 be stored in the next cycle by changing to assert epiEnable<1>.
2	Similar to cycle 1, the 21071-DA chip requests storing LW 2 and drives data for LW 1, and the 21071-BA chip latches LW 1. epiData<31:0> and epiBEnErr<3:0> are always one cycle behind the epiBus control lines.
3	The 21071-DA chip requests storing LW 3; 21071-DA drives LW 2.
4	The 21071-DA chip requests storing LW 4; 21071-DA drives LW 3. Because LW 4 is in the second octaword, epiOWSel asserts and epiEnable<0> is used.
5	The 21071-DA chip requests storing LW 5; the 21071-DA drives LW 4.

Cycle	Description
6	The 21071-DA chip requests storing LW 6; the 21071-DA drives LW 5. If the 21071-DA can ensure that the last data will be sent by cycle 7, then it may request a DMA write transaction with the 21071-CA. By the time the 21071-CA requires the DMA write data, it will have been loaded into the DMA write buffer.
7	The stores continue. There is no constraint on the order or number of times that a longword may be stored. There is also no constraint that the entire cache line be loaded, because the epiLineInval will set all of the byte enables that were not loaded to off. (This functionality allows an 21071-DA chip aggregate writes.)

Figure 16–2 Timing of 21071-DA to DMA Write Buffer Transfer



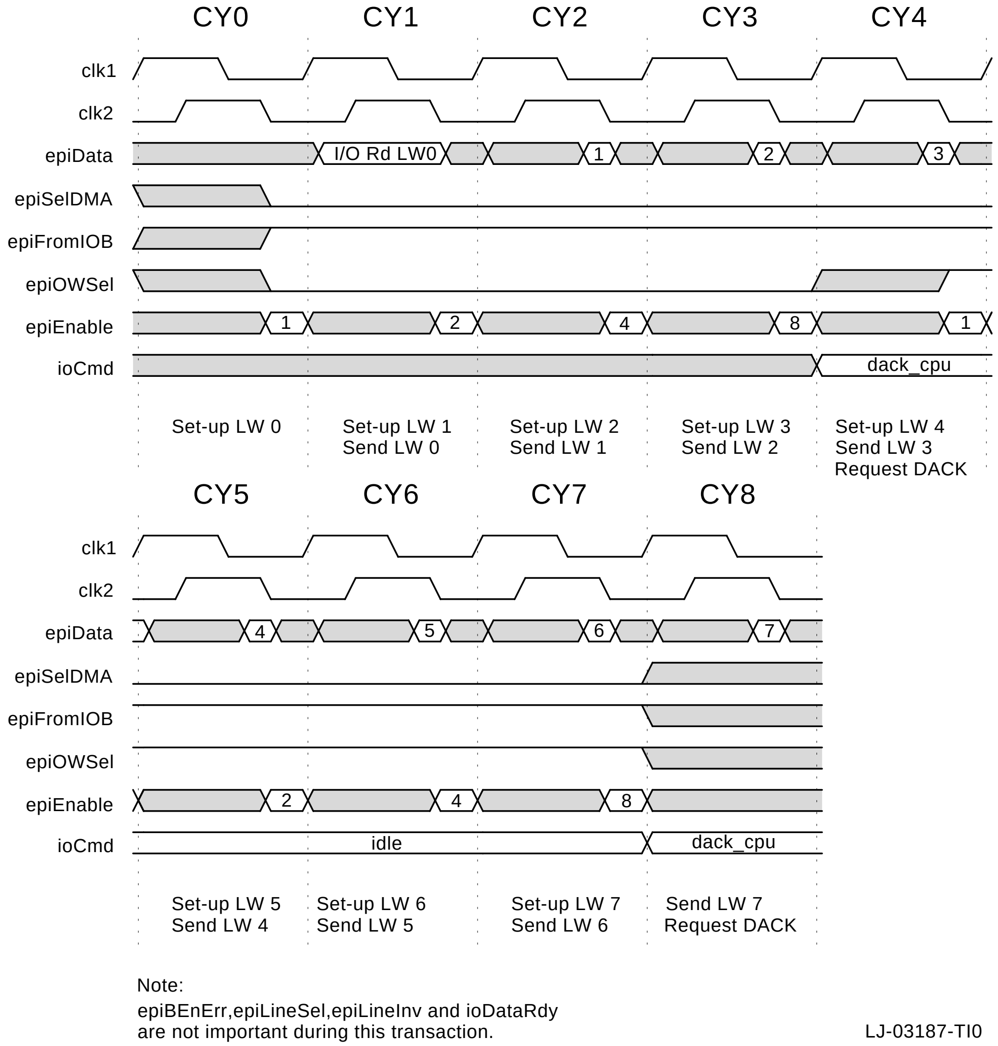
Note:
sysReadOW is not important during this transaction.

16.3.4 21071-DA to I/O Read Buffer

The following table describes the cycles for an epiBus transaction that transfers data from the 21071-DA chip into the CPU I/O read buffer, as shown in Figure 16–3.

Cycle	Description
0	It is presumed that a CPU read to I/O space has already begun, and that the 21071-DA chip recognizes that the read data is going to be ready. The 21071-DA chip places a request to store the first longword of read data by asserting epiFromIOB (indicating an write into the 21071-BA), deasserting epiSelDMA (indicating an I/O transfer), deasserting epiOWSel (indicating the first octaword), and asserting epiEnable<0> (LW 0 within first octaword). If the 21071-BA chip was driving epiData<31:0>, then it will tristate the bus by clk2F. The 21071-CA chip asserts sysIORead to select the I/O read buffer onto the sysData bus.
1	The 21071-DA chip sends the data to be stored on epiData<31:0>. The 21071-DA chip drives epiBENErr<3:0> with arbitrary values to prevent the bus from floating. The 21071-BA chip receives the epiBus control signals and latches LW 0 into the I/O read buffer. The 21071-DA chip requests that LW 1 be stored in the next cycle by changing to assert epiEnable<1>.
2	Similar to cycle 1, the 21071-DA chip requests storing LW 2, drives data for LW 1, and the 21071-BA chip latches LW 1. epiData<31:0> and epiBEnErr<3:0> are always one cycle behind the epiBus control lines.
3	The 21071-DA chip requests storing LW 3; 21071-DA drives LW 2.
4	The 21071-DA chip requests storing LW 4; 21071-DA drives LW 3. Because LW 4 is in the second octaword, epiOWSel asserts and epiEnable<0> is used. Because a full octaword of data will be stored in the 21071-BA chip by the end of cycle 4, the 21071-DA chip may send a cpuDRAck<2:0> OK_NCACHE_NCHK request through the 21071-BA chip to the CPU.
5	The 21071-DA chip requests storing LW 5; 21071-DA drives LW 4. The 21071-CA chip receives the cpuDRAck<2:0> OK_NCACHE_NCHK request and sends the OK on cpuDRAck<2:0>. The 21071-BA chip sends the first octaword on sysData<31:0> to the CPU.
6	The stores continue. There is no constraint on the order or number of times that a longword may be stored. There is also no constraint that an entire octaword be sent on epiData<31:0> before requesting a cpuDRAck<2:0>. (Of course, a cpuDRAck<2:0> cannot be requested before all of the data that needs to be sent has been transferred.)

Figure 16–3 Timing of 21071-DA to I/O Read Buffer Transfer



17

DECchip 21071-BA Electrical Data

This chapter includes the following information about the DECchip 21071-BA chip:

- DC Electrical Data
- AC Electrical Data

17.1 DC Electrical Data

This section describes the dc characteristics of the DECchip 21071-BA chip.

17.1.1 Absolute Maximum Ratings

Table 17–1 lists the maximum ratings of the DECchip 21071-BA chip.

Table 17–1 DECchip 21071-BA Maximum Ratings

Characteristics	Minimum	Maximum
Storage temperature	–55°C (–67°F)	125°C (257°F)
Operating ambient temperature	0°C (32°F)	40°C (104°F)
Air flow	100 lfpm[1]	—
Junction temperature	25°C (77°F)	100°C (212°F)
Supply voltage with respect to Vss, with reset_l asserted	–0.5 V	+6.5 V
Supply voltage with respect to Vss, with reset_l deasserted	4.75 V	5.25 V
Voltage on any pin with respect to Vss	–0.5 V	Vdd + 0.5 V
Maximum power:		
@Vdd = 5.25 V @Cycle = 30 ns		1.7 W

---

[1] lfpm = linear feet per minute

Table 17–2 lists the dc parametric values of the DECchip 21071-BA chip.

Table 17–2 DC Parametric Values

Symbol	Description	Minimum	Maximum	Units	Test Conditions
V_{ih}	Input high voltage	2.0	—	V	—
V_{il}	Input low voltage	—	0.8	V	—
V_{oh}	Output high voltage	2.4	—	V	—
V_{ol}	Output low voltage	—	0.4	V	—
I_{il}	Input leakage current[1]	–5	5	µA	0 V < Vin < Vdd
I_{ilpu}	Input leakage current[2]	–15	–100	µA	0 V < Vin < Vdd
I_{ilpd}	Input leakage current[3]	15	100	µA	0 V < Vin < Vdd
I_{ol}	Output leakage current (tristated)	–10	10	µA	0 V < Vin < Vdd

[1] Excluding drvSysCSR, eccMode, testMode, tristateL, and wideMem.

[2] For tristateL.

[3] For drvSysCSR, eccMode, testMode, and wideMem.

17.2 AC Electrical Data

This section describes the ac characteristics of the DECchip 21071-BA chip.

17.2.1 Clocks

The DECchip 21071-BA uses one clock (running at twice the nominal system frequency) plus a synchronous phase reference signal to generate five internal clock edges. See Figure 17–1, Figure 17–2, Table 17–3, and Table 17–4 for details about DECchip 21071-BA external clock requirements and internal clock phase relationships.

A clock system must meet the requirements in Figure 17–1 and Table 17–4 to guarantee the proper behavior of the 21071-BA chip's internal logic. The 21071-BA chip does not specify the maximum skew allowed for external transfers to or from the CPU, Bcache PALs, Bcache, 21071-CA chip, or 21071-DA chip because these skew limits are dependent on module placement and routing. A system designer must examine external transfers to determine the maximum clock skews allowed between chips.

The skew numbers shown in Figure 17–1 and Table 17–4 are given for a 30.0 ns cycle time. At a longer cycle time, the allowable skew may be increased, as long as the given minimum times between clock edges are not violated. These skew limits assume that the 21071-BA chip adds another 0.1 ns of uncertainty between rising and falling edges due to non-ideal input buffer switching thresholds.

Table 17–3 DECchip 21071-BA Clock AC Characteristics

Parameter	Minimum	Maximum	Unit	Note
System cycle time	30	—	ns	*c* in Figure 17–1
clk1x2 period	15	—	ns	—
clk1x2 frequency	—	66	MHz	—
clk1x2 rise time	—	1	ns	—
clk1x2 fall time	—	1	ns	—
clk2ref setup to clk1x2 rising	0.8	—	ns	*Tsu* in Figure 17–1
clk2ref hold from clk1x2 rising	1.8	—	ns	*Th* in Figure 17–1

Figure 17–1 DECchip 21071-BA Clock Skew Requirements

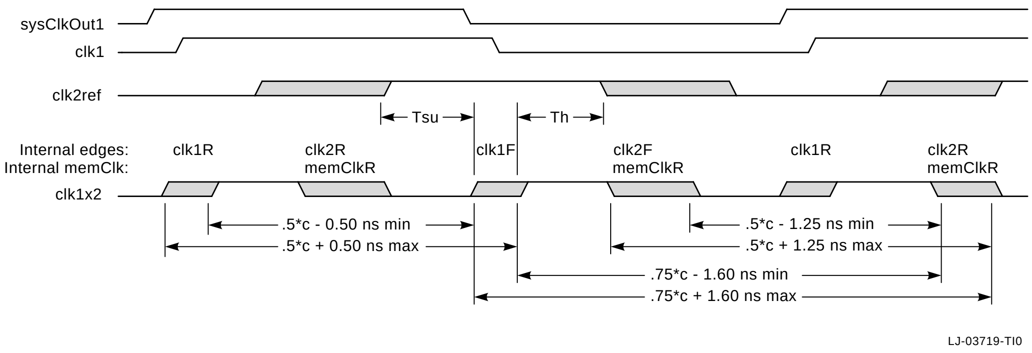
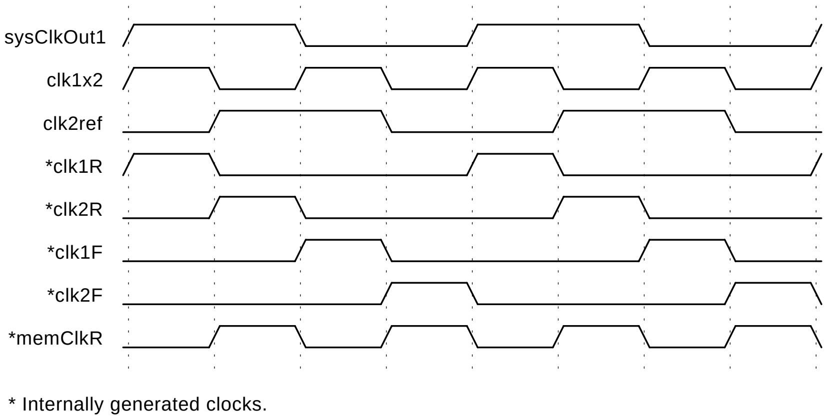


Table 17–4 DECchip 21071-BA Clock Skew Limits at clk1x2 Pin

Parameter	Example Transfers	Maximum	Unit	Note
clk1x2 rising edge to rising edge	clk1R to clk1R, clk1R to clk1F, clk1F to clk1R, clk1F to clk1F	0.50	ns	@ Cycle = 30 ns
clk1x2 falling edge to falling edge	clk2R to clk2R, clk2R to clk2F, clk2F to clk2R, clk2F to clk2F	1.25	ns	@ Cycle = 30 ns
clk1x2 rising edge to falling edge	clk1R to clk2R, clk1R to clk2F, clk1F to clk2R, clk1F to clk2F	1.60	ns	@ Cycle = 30 ns
clk1x2 falling edge to rising edge	clk2R to clk1R, clk2R to clk1F, clk2F to clk1R, clk2F to clk1F	1.60	ns	@ Cycle = 30 ns

Figure 17–2 DECchip 21071-BA Clock Signals



LJ-03455-TI0

The 21071-BA imposes no requirements on clk1 or sysClkOut1. Skew on clk1 will be constrained by limits imposed by external paths to or from the Bcache control PALs. The phase error between sysClkOut1 and clk1x2 will be constrained by limits imposed by external paths to or from the CPU chip.

17.2.2 Signals

Figure 17–3 and Figure 17–4 demonstrate the timing measurements specified in Table 17–6 and Table 17–7.

Figure 17–3 DECchip 21071-BA Output Delay Measurement

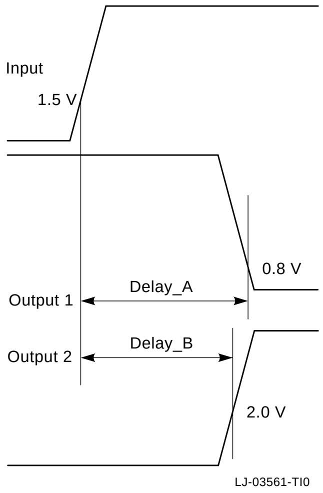
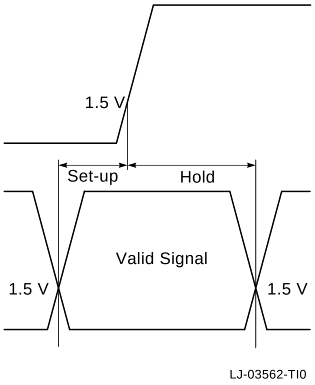


Figure 17–4 DECchip 21071-BA Setup and Hold Time Measurement



The following ac electrical data is specified with respect to the appropriate edge at the clk1x2 pin. Both the output delay table and the setup/hold time table assume a 1 ns edge rate at the clk1x2 pin.

All outputs drive a 50 pF load. When estimating module delays, you may need to replace the 50 pF load delay with a simulated (or calculated) delay. The delays for 4 mA and 8 mA drivers driving a 50 pF load are provided in Table 17–5. See Table 14–1 for information about the buffer size of every output pin.

Table 17–5 DECchip 21071-BA Output Buffer Delays into a 50 pF Load

Type	Minimum	Maximum	Unit
4 mA	3.5	7.6	ns
8 mA	2.3	5.0	ns

Table 17–6 DECchip 21071-BA AC Characteristics (Valid Delay into a 50 pF Load)

Signal	Minimum	Maximum	Unit	Reference Edge
sysData<63:0>, sysPar<1:0>, sysCheck<6:0>	5.9	18.5	ns	clk1R
memData<31:0>, memPar<0>, memCheck<6:0>	4.3	16.8	ns	memClkR
epiData<31:0>, epiBEnErr<3:0>	4.9	16.2	ns	clk1R

Table 17–7 DECchip 21071-BA AC Characteristics (Setup/Hold Time)

Signal	Setup	Hold	Unit	Reference Edge
sysData<63:0>, sysPar<1:0>, sysCheck<6:0>	2.4	2.9	ns	clk2F
memData<31:0>, memPar<0>, memCheck<6:0>	0.6	4.4	ns	memClkR
epiData<31:0>, epiBEnErr<3:0>	1.0	5.2	ns	clk2F
sysCmd<2:0>, subCmd<1:0>, sysIORead, sysReadOW	0.6	5.0	ns	clk2F
drvSysData[1], drvSysCSR	2.1	2.6	ns	clk1R
drvSysData[2]	2.1	2.6	ns	clk1F
memCmd<3:1>	2.1	5.2	ns	clk1R
epiFromIOB, epiSelDMA	1.0	5.2	ns	clk2F
ioLineSel, epiLineInval	1.0	5.2	ns	clk2F
epiOWSel[3], epiLineSel<1:0>[3]	1.0	5.2	ns	clk2F

---

[1] For dvrSysData asserting.

[2] For dvrSysData deasserting.

[3] These signals pass through a transparent latch (closing on clk2F) and then reach a clk1R flip-flop.

(continued on next page)

Table 17–7 (Cont.) DECchip 21071-BA AC Characteristics (Setup/Hold Time)

Signal	Setup	Hold	Unit	Reference Edge
epiOWSel[3], epiLineSel<1:0>[3]	9.7	—	ns	clk1R
epiEnable<1:0>[3]	1.0	5.2	ns	clk2F
epiEnable<1>[3]	6.0	—	ns	clk1R

---

[3] These signals pass through a transparent latch (closing on clk2F) and then reach a clk1R flip-flop.

18

DECchip 21071-BA Power-Up and Initialization

This chapter describes the behavior of the 21071-BA chip on power-up and assertion of reset_l.

18.1 Power-Up

On power-up, the reset_l input of the 21071-BA chip should be asserted. It should be kept asserted until the system clocks are up and running for 20 cycles.

18.2 Internal Reset

The assertion and deassertion of the reset_l pin on the module is asynchronous to the DECchip 21071-BA. An internal reset signal is generated from reset_l which asserts asynchronously as soon as reset_l is asserted, but which deasserts synchronously. Due to the synchronous deassertion of the internal reset, the DECchip 21071-BA requires that no external transaction should start until 10 system clock cycles after the deassertion of reset_l.

18.3 State of Pins on Reset Assertion

The following are general rules and requirements for the behavior of the 21071-BA at its pins during reset:

- All input only control signals (except the clocks and reset_l) should be in the deasserted state as long as reset is asserted.
- All output only signals are deasserted.
- All bidirectional signals are tristated.
- wideMem and eccMode should be stable before reset_l deasserts and should never change thereafter.

The exceptions to these rules are as follows:

- The value of memData<31:0> is unpredictable; the drive state depends on the state of the drvMemData input.
- drvMemData is asserted by the 21071-CA during reset so memData<31:0> are driven by the 21071-BA.

Note

In all cases, the assertion of tristate_l overrides the assertion of reset_l. That is, if tristate_l is asserted during reset, all the outputs of the chip go to their High-Z state. If reset_l is still asserted when tristate_l deasserts, the signals return to the normal reset state described previously.

A

Bcache PAL Equations

This appendix provides Bcache PAL equations.

Table A–1 provides cache data write enable equations.

Table A–1 Equations for Cache Data Write Enables

bcDataWE3[1] =	(sysDataLongWE & longWr) # (sysDataWEEn & !clk1 & !longWr) # cpuDataWE3;
bcDataWE3.OE =	((1));
bcDataWE2[1] =	(sysDataLongWE & longWr) # (sysDataWEEn & !clk1 & !longWr) # cpuDataWE2;
bcDataWE2.OE =	((1));
bcDataWE1[1] =	(sysDataLongWE & longWr) # (sysDataWEEn & !clk1 & !longWr) # cpuDataWE1;
bcDataWE1.OE =	((1));
bcDataWE0[1] =	(sysDataLongWE & longWr) # (sysDataWEEn & !clk1 & !longWr) # cpuDataWE0;
bcDataWE0.OE =	((1));
!bcDataA4_3[1] [2]=	(sysDataALEn & !clk2) # (sysDataAHEn & clk2) # (sysDataALEn & sysDataAHEn[3]) # cpuDataA4;
bcDataA4_3.OE =	((1));
!bcDataA4_2[1] [2] =	(sysDataALEn & !clk2) # (sysDataAHEn & clk2) # (sysDataALEn & sysDataAHEn[3]) # cpuDataA4;
bcDataA4_2.OE	= ((1));
!bcDataA4_1[1] [2] =	(sysDataALEn & !clk2) # (sysDataAHEn & clk2) # (sysDataALEn & sysDataAHEn[3]) # cpuDataA4;
bcDataA4_1.OE =	((1));

---

[1] # = OR, & = AND, ! = NOT

[2] Cache address bit 4; these are 4 identical copies.

[3] This term is logically redundant but must be included to prevent glitches in the output.

(continued on next page)

Table A–1 (Cont.) Equations for Cache Data Write Enables

!bcDataA4_0[1] [2] =	(sysDataALEn & !clk2) # (sysDataAHEn & clk2) # (sysDataALEn & sysDataAHEn[3]) # cpuDataA4;
bcDataA4_0.OE =	((1));

[1] # = OR, & = AND, ! = NOT

[2] Cache address bit 4; these are 4 identical copies.

[3] This term is logically redundant but must be included to prevent glitches in the output.

Table A–2 provides tag and data output enable equations.

Table A–2 Equations for the Tag and Data Output Enables

bcTagCEOE[1] =	(sysTagOEEn & !senseDis # cpuTagCEOE & !senseDis # sysEarlyOEEn & cpuCReq2 & !senseDis # sysEarlyOEEn & cpuCReq1 & !senseDis # sysEarlyOEEn & cpuCReq0 & !senseDis);
bcTagCEOE.OE =	((1));
bcDataCEOE0[1] =	(sysDataOEEn & !senseDis # cpuDataCEOE & !senseDis # sysEarlyOEEn & !cpuCReq2 & cpuCReq0 & !senseDis # sysEarlyOEEn & cpuCReq2 & !cpuCReq0 & !senseDis # sysEarlyOEEn & !cpuCReq2 & cpuCReq1 & !senseDis) ;
bcDataCEOE0.OE =	((1));
bcDataCEOE1[1] =	(sysDataOEEn & !senseDis # cpuDataCEOE & !senseDis # sysEarlyOEEn & !cpuCReq2 & cpuCReq0 & !senseDis # sysEarlyOEEn & cpuCReq2 & !cpuCReq0 & !senseDis # sysEarlyOEEn & !cpuCReq2 & cpuCReq1 & !senseDis) ;
bcDataCEOE1.OE =	((1));
bcDataCEOE2[1] =	(sysDataOEEn & !senseDis # cpuDataCEOE & !senseDis # sysEarlyOEEn & !cpuCReq2 & cpuCReq0 & !senseDis # sysEarlyOEEn & cpuCReq2 & !cpuCReq0 & !senseDis # sysEarlyOEEn & !cpuCReq2 & cpuCReq1 & !senseDis);
bcDataCEOE2.OE =	((1));
bcDataCEOE3[1] =	(sysDataOEEn & !senseDis # cpuDataCEOE & !senseDis # sysEarlyOEEn & !cpuCReq2 & cpuCReq0 & !senseDis # sysEarlyOEEn & cpuCReq2 & !cpuCReq0 & !senseDis # sysEarlyOEEn & !cpuCReq2 & cpuCReq1 & !senseDis;)
bcDataCEOE3.OE	((1));
cpuDOE[1] [2]=	(sysEarlyOEEn & cpuCReq2 & cpuCReq0 # sysDOE);
cpuDOE.OE[1] [2] =	!senseDis;

---

[1] # = OR, & = AND, ! = NOT

[2] CPU output enable; must be tristated when 3.3 V is not stable.

Note

In addition to the two PALs, the DECchip 21071 and DECchip 21072 chipsets also require two NOR gates to control the cache. These may be implemented using NORs or unused portions of the PALs.

Table A–3 provides Bcache and NOR gate equations.

Table A–3 Equations for Bcache and NOR Gates

tagCtlWE_l[1] =	! (cpuTagCtlWe + sysTagWE)
tagAdrWE_l[1] =	! (sysTagWE)

[1] # = OR, & = AND, ! = NOT

B

Technical Support and Ordering Information

B.1 Technical Support

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

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

B.2 Ordering Digital Semiconductor Products

To order the DECchip 21071 and DECchip 21072 core logic chipsets, contact your local distributor.

B.3 Ordering Associated Literature

For a complete list of available Digital Semiconductor literature contact the Digital Seminconductor Information Line.