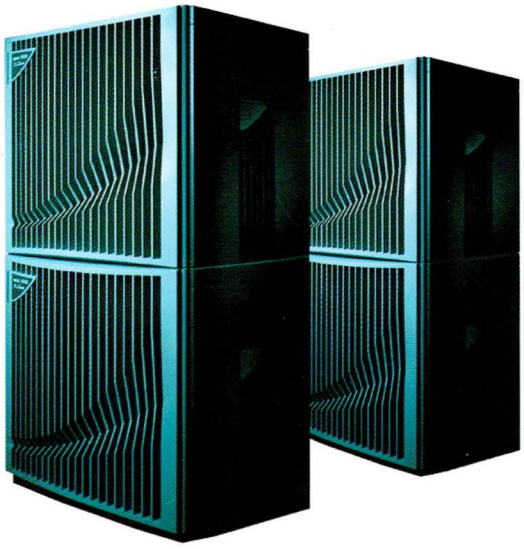
HEWLETT® PACKARD

S-Class Class
X-Class Servers



Exemplar Architecture

First Edition

Hewlett-Packard Company
Convex Division
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America

Exemplar Architecture

S-Class and X-Class Servers

A4716-90001

First Edition

January, 1997

Hewlett-Packard Company
Convex Division
Richardson, Texas
United States of America

Exemplar Architecture
S-Class and X-Class Servers

A4716-90001

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

This entire book is recyclable.

Printed in the United States of America

Revision Information for

Exemplar Architecture

S-Class and X-Class Servers

Edition	Document No.	Description
First	A4716-90001	Initial release January, 1997.

Contents

Figures

Tables

Preface

Purpose

This book provides technical information for the evaluation of the Exemplar S-Class and X-Class servers and the optimization of applications that may run on these servers.

A detailed description of the PA-RISC architecture is presented in the Hewlett-Packard *PA-RISC 2.0 Architecture* reference manual. This book does not attempt to duplicate information in that manual. Instead, it is a compliment to it and presents only S-Class and X-Class server specific information.

Using this book

This book has the following eleven chapters including:

- Chapter 1, "Overview," provides an introduction to the concepts used in S-Class and X-Class servers, such as hypernode, multiple hypernode configurations, and globally shared memory.
- Chapter 2, "Physical address space," discusses the address space, including coherent memory, core logic, local I/O CSR, and non-I/O CSR spaces.
- Chapter 3, "Cache management," discusses the management of data and instruction caches and CTI caches (used in X-Class servers).
- Chapter 4, "Data mover," describes the messaging mechanisms (including support for block data movement) and the low-level programming interface that accesses those mechanisms.
- Chapter 5, "Synchronization," defines mechanisms that enable S-Class and X-Class servers to operate efficiently as scalable, parallel processing systems.
- Chapter 6, "Interrupts," defines interrupt mechanisms.
- Chapter 7, "I/O subsystem," defines the I/O subsystem provided on the S-Class and X-Class servers.

- Chapter 8, "Performance monitors," defines performance monitoring mechanisms of S-Class and X-Class servers.
- Chapter 9, "Hypernode utilities," describes the hypernode utilities board, a crucial part of the hypernode. The utilities board contains the mechanism to initialize, boot, and test the hypernode.
- Chapter 10, "Booting and testing," defines procedures and mechanisms that support general system initialization, OS booting, and diagnostics.
- Chapter 11, "Error handling," describes how S-Class and X-Class servers handle exceptions and errors.

The glossary contains brief definitions of some of the terms used in this document.

Notational conventions

This section discusses notational conventions used in this book.

`Monospace`

The `monospace` font identifies command names, system calls, and data structures and types.

In command and code examples, `monospace` identifies command output, including error messages

Italic

In paragraph text, *italic* identifies new and important terms and titles of documents.

Bold

The **bold** character format indicates special emphasis.

Notes and cautions

This document presents notes and cautions in the following formats.

Note

A Note highlights supplemental information.

Caution

A Caution highlights information necessary to avoid damage to the system.

Associated documents

Using Exemplar S-Class or X-Class servers may require information not specific to the functionality described in this document.

- *PA-RISC 2.0 Architecture.* This manual contains generic information about the PA-RISC architecture used as a basis for the S-Class and X-Class servers.
- The *Exemplar Programming Guide: S-Class and X-Class Servers,* part number B5600-90001, provides a description of efficient programming methods in Convex C and Fortran.
- The *Exemplar Diagnostics Guide: S-Class and X-Class Servers,* part number A4716-90002, provides a reference to diagnostic tests and utilities.

Ordering documents

To order the current edition of these or any other Hewlett-Packard documents, send requests to:

Hewlett-Packard Company
Convex Division
Customer Service
P.O. Box 833851
Richardson TX 75083-3851 USA

Please include the order number (DSW or DHW number) or the exact title of the document.

Technical assistance

If you have questions that are not answered in this book, contact the Hewlett-Packard Convex Technical Assistance Center (TAC) at the following locations:

Within the continental U.S., call 1 (800) 952-0379.

From Canada, call 1 (800) 345-2384.

All other locations, contact the local Convex Division office.

You can also use the `contact` utility, if you would like to report any problems you may have.

Overview

1

This chapter presents an overview of S-Class and X-Class server architecture.

Exemplar S-Class and X-Class servers use the PA-8000, the latest in a line of high performance Precision Architecture - Reduced Instruction Set Computer (PA-RISC) processors from Hewlett-Packard Company.

In addition to multiple PA-RISC processors, these systems provide multipurpose, scalable computing resources through shared memory and I/O designed to increase throughput, thereby reducing the time to solution.

The basic building block of S-Class and X-Class servers is the hypernode, a self-contained, multiprocessor system with a common memory. The S-Class server is a single hypernode that can be upgraded to a multiple hypernode X-Class server. The X-Class server has one or more hypernodes connected by a high-performance interconnect that allows all processors in the system to share the memory of every hypernode.

The PA-8000 processor

The heart of the S-Class and X-Class server is the Hewlett-Packard PA-8000 processor, based on the concept of Reduced Instruction Set Computers (RISC). The PA-8000 was designed according to Hewlett-Packard's PA-RISC Architecture version 2.0 specifications.

Note

The PA-RISC architecture is presented in the *PA-RISC 2.0 Architecture* reference manual. Please refer to that document for detailed information about the features of the PA-8000. This document does not attempt to duplicate information in that manual. Instead, it compliments the document and presents only S-Class and X-Class server-specific information.

The processors of each hypernode are supported by several hardware controllers implemented using ASIC technology, an enhanced memory system, and a high-bandwidth I/O subsystem.

Special hardware and software allow them to perform both as conventional single-processors or together, in parallel, to solve more complex problems requiring parallel processing.

The hypernode

The fundamental building block of the Exemplar S-Class and X-Class servers is the hypernode.

A hypernode uses a symmetric multiprocessor (SMP) design that can exploit fine-grain parallelism. A hypernode can contain from four to 16 processors and can stand alone as a symmetric multiprocessor parallel computer.

The X-Class server can have from one to four hypernodes and up to 64 processors.

A conceptual block diagram of a single hypernode is shown in Figure 1. Centrally located in the diagram is the crossbar that is comprised of four routing array controllers (RACs). It connects the processor agent controllers (PACs) in the hypernode to all of the memory access controllers (MACs). Up to two processors are located on each processor agent. The memory controllers provide the crossbar access to both hypernode-local and off-hypernode memory (all memory is hypernode-local in S-Class servers).

Local memory interfaces directly to and is controlled by the memory controllers. For X-Class servers, off-hypernode memory transfers are routed through toroidal access controllers (TACs) that are connected to the memory controllers. The TACs control data transfers between hypernodes over rings that connect hypernodes together in two dimensions.

Input and output devices connect to the hypernode through a PCI-bus interface controller (PIC) connected to the processor agents.

The utility board in the hypernode contains a section of hardware called the core logic. It provides hypernode-specific interrupts to all of the processors in the hypernode through the core logic bus which connects to each processor agent.

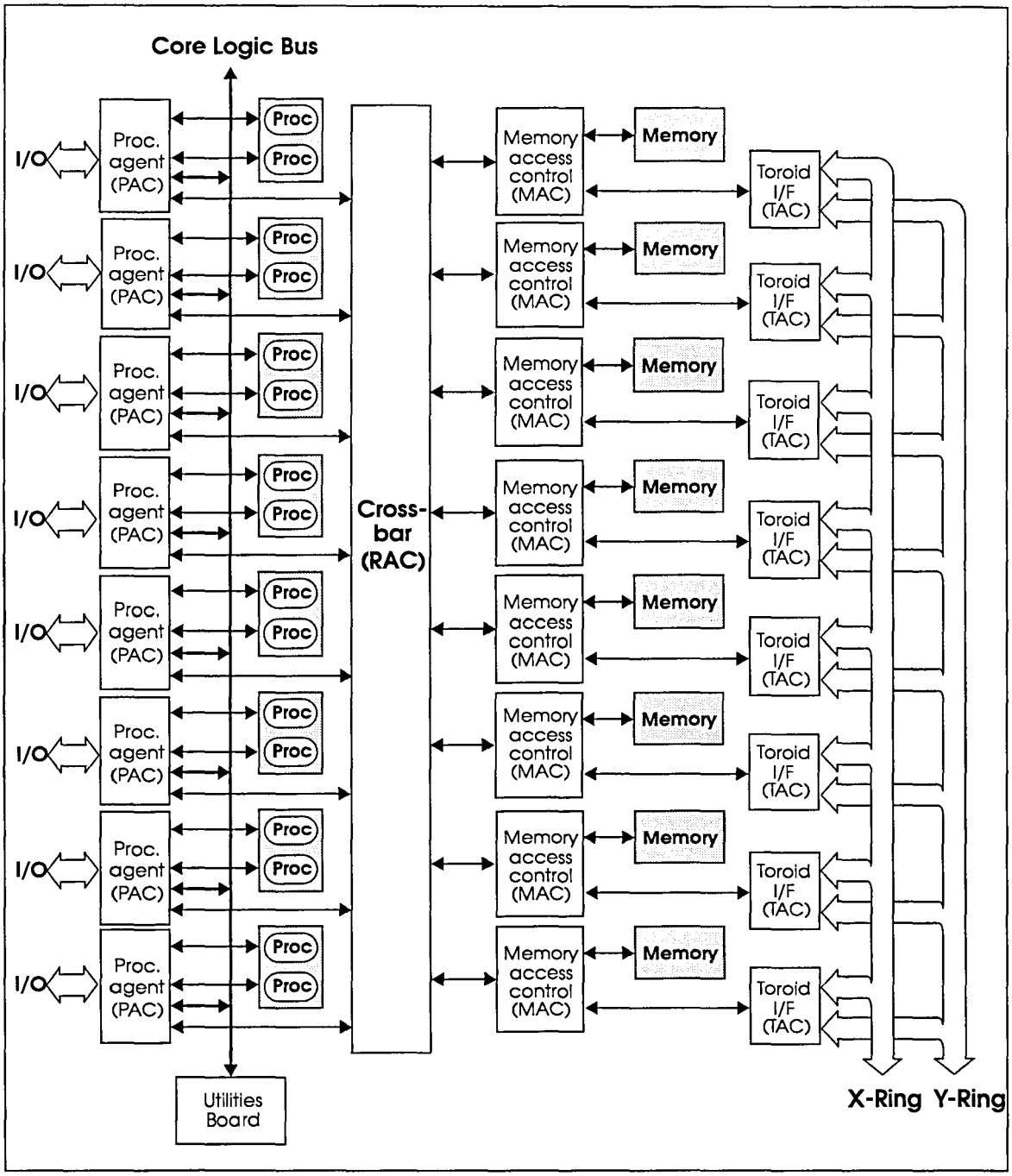


Figure 1 Functional block diagram of a full X-Class hypernode

Control and status registers (CSRs)

Hypernode hardware is manipulated by control and status registers, CSRs, located in each of the following controllers:

- Processor agent (PAC)
- PCI interface (PIC)
- Routing array (RAC)
- Memory access (MAC)
- Toroidal access (TAC)
- Processor utilities (PUC)
- Monitoring utilities (MUC)

Each CSR is memory mapped and is available to all processors in the hypernode. Many of the registers are described in detail throughout this book by functional groups, such as system configuration, messaging and data copy, error recovery, and so on. As the name implies, CSRs provide control, status, or both to the processors and other hardware in the hypernode.

Description of functional blocks

A hypernode can contain from four to 16 processors. See Table 1. There are two processors connected to each processor agent. The processor agents interface the processors to the I/O bus and the crossbar. The crossbar allows each of the processors access to memory, both hypernode-local and off-hypernode (for X-Class servers).

Processor agent controller

The PAC can contain zero, one, or two PA-8000 processors. It can also contain zero or one PIC. With no processors, the PAC serves as an I/O interface. The PAC has the following buses:

- Runway bus (0, 1)—Two each, 64-bit, bidirectional buses for processor 0 and processor 1, respectively. These buses have a raw bandwidth of 960 MBytes/sec.
- Crossbar port bus (0, 1)—Four 32-bit, unidirectional buses connected to two crossbar RACs, two for read and two for write. These buses have a total raw bandwidth of 1.9 GBytes/sec.
- I/O port—Two 16-bit or 32-bit, unidirectional interfaces to an I/O device, one for reading data and one for writing data. The width of the bus depends on the width of the I/O device connected. Each bus has a bandwidth of 120 MBytes/sec or 240 MBytes/sec, depending on the width of the interface.

- Core Logic Bus interface—Bidirectional bus that supports boot and support services.

The processor agent sends and receives transactions from the RACs using four unidirectional data paths. There are four RACs. However, each processor agent communicates with only two of the four.

The PAC includes special hardware called the *data mover* for rapid message and data movement between memory within a hypernode as well as with memory in other hypernodes (for X-Class servers). This dedicated hardware greatly improves file I/O and networking over software versions. The operating system and software compilers support the data mover through system calls, the Architectural Interface Library bcopy (`ail_bcopy`), Parallel Virtual Machine (PVM), and the Message Passing Interface (MPI). See Chapter 4, "Data mover," for more information.

Routing array controller—crossbar

S-Class and X-Class servers use a nonblocking crossbar to interconnect each processor and I/O device to memory. The crossbar consists of four routing array controllers.

Each of the four RACs has the following buses:

- PAC Port (A, B, C, D)—Eight 32-bit, unidirectional interfaces to four PAC ports, four for read and four for write. Each port has simultaneous (input and output) bandwidth of 960 MBytes/sec.
- MAC Port (A, B, C, D)—Eight 32-bit, unidirectional interfaces to four MAC ports, four for read and four for write. Each port has simultaneous (input and output) bandwidth of 960 MBytes/sec.

Memory access controller

The MAC controls all accesses to memory. These accesses can come from two directions: through the RAC via the PAC (processor or IO) or from another hypernode (in X-Class servers only) through the TAC. Each MAC controls four banks of memory, allowing up to 32 banks in an eight-MAC hypernode. Memory banks consist of Single Inline Memory Modules (SIMMs) of Synchronous Dynamic Random Access Memory (SDRAM). The memory controlled by a MAC is used for hypernode-local memory, remote accesses, and messaging.

The MAC has the following buses:

- RAC Port (A, B)—Four unidirectional, 32-bit interfaces to two of RAC ports, two for read and two for write. This interface

supports a total simultaneous read-write bandwidth of 1.9 GBytes/sec.

- TAC port—Two unidirectional, 32-bit interfaces to one of TAC ports. This interface supports a simultaneous read-write bandwidth of 960 MBytes/sec.
- Even Memory—Bidirectional, 88-bit interface to the even memory bank associated with the MAC.
- Odd Memory—Bidirectional, 88-bit interface to the odd memory bank associated with the MAC.

A processor accesses memory by sending a request to a RAC (crossbar). The request is then forwarded to one of the MACs. These requests are in the form of packets routed from the processor, through the PAC and the RAC.

The MAC queues requests into either the even or odd pending queue and arbitrates between them in a round robin fashion. Some packets are not destined for memory and are routed from processor to processor through the MAC. These packets are routed directly to the output ports.

In X-Class servers, request packets from other hypernodes are routed through the TAC to the MAC where they are queued for arbitration as discussed above.

The MAC accesses one of its four available memory banks, checking the Error Correction Code (ECC) and tagged coherency information. Coherency is discussed in several sections beginning with the "GSM and cache coherence" section on page 15. Provided no additional coherency operations are required, the data accessed from memory is returned to the processor by sending a response back to the crossbar. The crossbar forwards the response to the PAC.

Toroidal access controller

For X-Class servers, the TAC provides the hypernode an interface to other hypernodes across a *ring* interface referred to as the Coherent Toroidal Interconnect (CTI). The TAC receives incoming CTI requests and responses and routes them to the MAC. It also routes requests and responses for the hypernode to the CTI rings. The hypernode memory is made available for all other hypernodes in a multiple hypernode system through the CTI rings. See "Multiple hypernodes" section on page 10 for more information.

The TAC has the following buses:

- MAC port—Two 32-bit, unidirectional interfaces, one for inbound and one for outbound traffic.

- CTI ring (X, Y)—Two 32-bit unidirectional interfaces for each CTI ring (X and Y).

Utilities board and core logic bus

The utilities board connects to the core logic bus and contains two field-programmable gate arrays (FPGAs): the PUC and MUC. The PUC allows processors access to the system core logic and booting firmware, and the MUC processes the environmental state of the hypernode and interrupts the processors when appropriate. S-Class and X-Class servers use the core logic bus primarily to boot the system and to issue hypernode-local environmental interrupts.

The core logic bus is a low bandwidth, multidrop bus that connects each processor to the control and interface logic (both RS232 and ethernet) of other processors within a hypernode and other hypernodes. A processor can write to control and status registers (CSRs) accessed using the core logic bus to initialize and configure the RAC chips and utility board logic.

Hypernode configurations

Table 1 shows the available configurations for single hypernode.

Table 1 Hypernode configurations

Processors	Processor agents	Memory boards	Total memory (MB)	I/O chassis
4	2	2	256	1
4	2	2	512	1
4	2	2	1024	1
4	2	4	2048	1
8	4	4	512	1 or 2
8	4	4	1024	1 or 2
8	4	4	2048	1 or 2
8	4	8	4096	1 or 2
12	6	8	1024	1, 2, or 3
12	6	8	2048	1, 2, or 3
12	6	8	3072	1, 2, or 3
12	6	8	4096	1, 2, or 3
16	8	8	1024	1, 2, 3, or 4
16	8	8	2048	1, 2, 3, or 4
16	8	8	3072	1, 2, 3, or 4
16	8	8	4096	1, 2, 3, or 4

The maximum amount of memory per hypernode is currently 4 GBytes.

Multiple hypernodes

Up to 64 processors in four hypernodes can be connected together in the X-Class server. Hypernodes are tightly coupled by globally shared memory and CTI rings to provide minimum latency in global memory accesses. Figure 2 depicts how four hypernodes might be connected. The two Y CTI rings and two X CTI rings provide a low-latency interconnect that allows processors to access memory anywhere in the system and communicate with processors in other hypernodes. Each ring in the figure represents multiple interconnects (up to eight per hypernode in each dimension). One inherent advantage to this interconnect scheme is built-in redundancy afforded by multiple CTI rings.

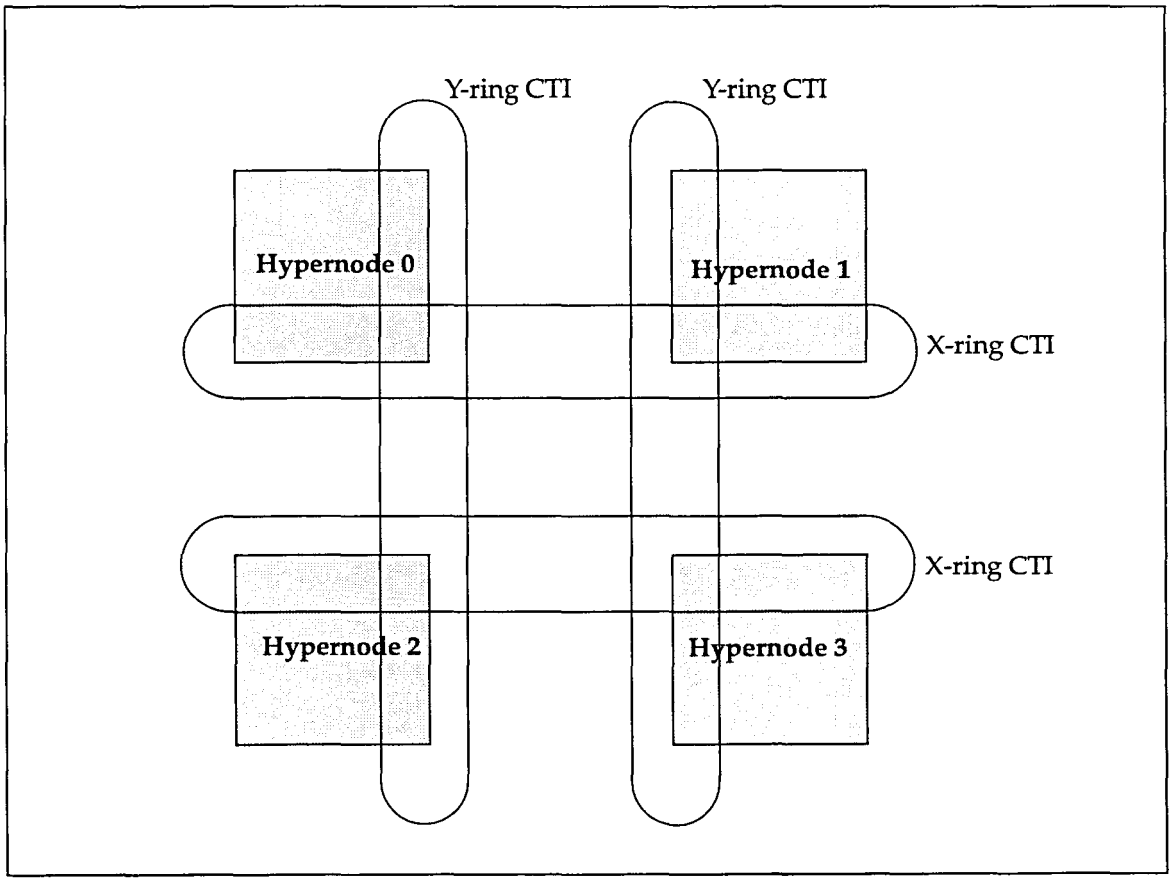


Figure 2 Four-hypernode interconnection

Coherent Toroidal Interconnect (CTI)

Multiple hypernodes are interconnected with one or more low-latency interconnections. The CTI refers to a collection of rings used by X-Class server hypernodes to access remote memory.

The CTI supports access to global or remote memory on a cache line basis. A 32-byte cache line is the amount of data moved over the CTI by hardware in response to load, store, or flush operations.

The CTI also supports messaging protocols. The operating system and distributed-memory applications use messages for interhypernode communication. Message data movement incurs somewhat less overhead than cache line movement. Cache lines migrate between hypernodes in X-Class servers without software intervention, resulting in lower overhead to programs using shared-memory.

X-Class servers use up to eight physical CTI rings per dimension between hypernodes. Eight rings provide higher interconnection bandwidth than a single ring and provide redundancy in case of ring failure.

Sequential memory references to global memory (by linearly ascending physical address) are interleaved across the eight rings. The eight CTI rings are interleaved on a 32-byte basis, the CTI cache line size. Ring interleaving tends to balance the traffic across all eight rings (global memory references from a processor to the memory on the same hypernode do not use the hypernode CTI).

The CTI implementation is a pair of 34-bit, differential CMOS, unidirectional links, clocked at 120 MHz. Each link provides 32 bits of data along with a clock and a framing delimiter flag for a total of 34 signals. Data is sampled on the rising edge of the clock for a peak raw rate of 480 Mbytes per second.

X-Class servers use an SCI-like coherency protocol. It is overlaid on a base protocol that supports forward progress, delivery, fairness, and basic error detection and recovery. The coherency protocol is based on a write-back and invalidate scheme.

Globally shared memory

All of the processors in the S-Class and X-Class servers share memory both within a hypernode (local memory) and across the entire array of hypernodes (remote memory), in the case of multiple hypernode X-Class servers. Shared memory makes the system easy to program while supporting message passing and shared-memory programming models.

The shared-memory programming model is that of a tightly coupled, shared-memory machine; it allows the developer, compilers, and applications to view the system as a number of processors sharing a large physical memory and a number of high bandwidth I/O ports.

The message passing programming model provides high performance for applications developed using explicit PVM or MPI messaging.

Compilers for the S-Class and X-Class servers use GSM to provide automatic, efficient parallelization while viewing memory as a single contiguous virtual address space.

GSM subsystem

GSM supports a two-level hierarchial memory subsystem; each level is optimized for a particular class of data sharing. The first level consists of the crossbar that connects memory, processors, and I/O in a single hypernode. The second level consists of the interconnection between hypernodes through the CTI rings.

The crossbar in GSM

The crossbar provides high bandwidth, low latency nonblocking access from processors and I/O channels to the hypernode-local memory. The crossbar implementation prevents the performance drop-off associated with systems that employ a system-wide bus for processor and I/O memory traffic.

Memory interleave

Sequential memory references (linearly ascending physical address) to GSM are interleaved across up to eight rings. The memory and CTI interconnects are interleaved on a 32-byte basis. Interleaving tends to balance traffic across the eight CTI rings. See Chapter 2, "Physical address space," for information regarding memory interleaving.

References from a processor to GSM on the same hypernode use the crossbar, and references from a processor to memory in another hypernode use the crossbar and CTI.

GSM and memory latency

Processor references to local memory have a minimum amount of latency. In X-Class servers, references to memory located in another hypernodes experience some additional latency because of the additional hardware required to transfer data from the remote node memory. Processors use caches for both instructions and data to reduce the number of memory accesses, thereby reducing traffic on the crossbar and CTI.

S-Class and X-Class servers use internal hardware that determines the access methods for each memory reference. The specific latency varies depending on the proximity of the referenced data.

Figure 3 illustrates the various stages that are required for memory reference instructions. It depicts processing of either a load or store instruction. The processor decodes the instruction and generates the appropriate address. For a load instruction, if the address produces a *hit* in the processor cache, the data is transferred from the cache to the processor register and the instruction is completed.

For a store instruction, the data is moved from the processor register to the cache. Otherwise, if the load or store address is not in cache (a cache miss) the processor must access either local or remote memory. In the case of X-Class server remote memory access, the CTI cache is used in addition to the processor caches. CTI caches are discussed in the "CTI cache layout" section on page 34.

The flow chart in Figure 3, illustrates how each type of memory reference experiences different latency, depending on the type.

GSM allows for different memory allocation, sharing, and latency, without requiring different physical memories in a hypernode. The system hardware automatically copies cache lines between hypernodes without software intervention. This translates to lower overhead for programs using GSM.

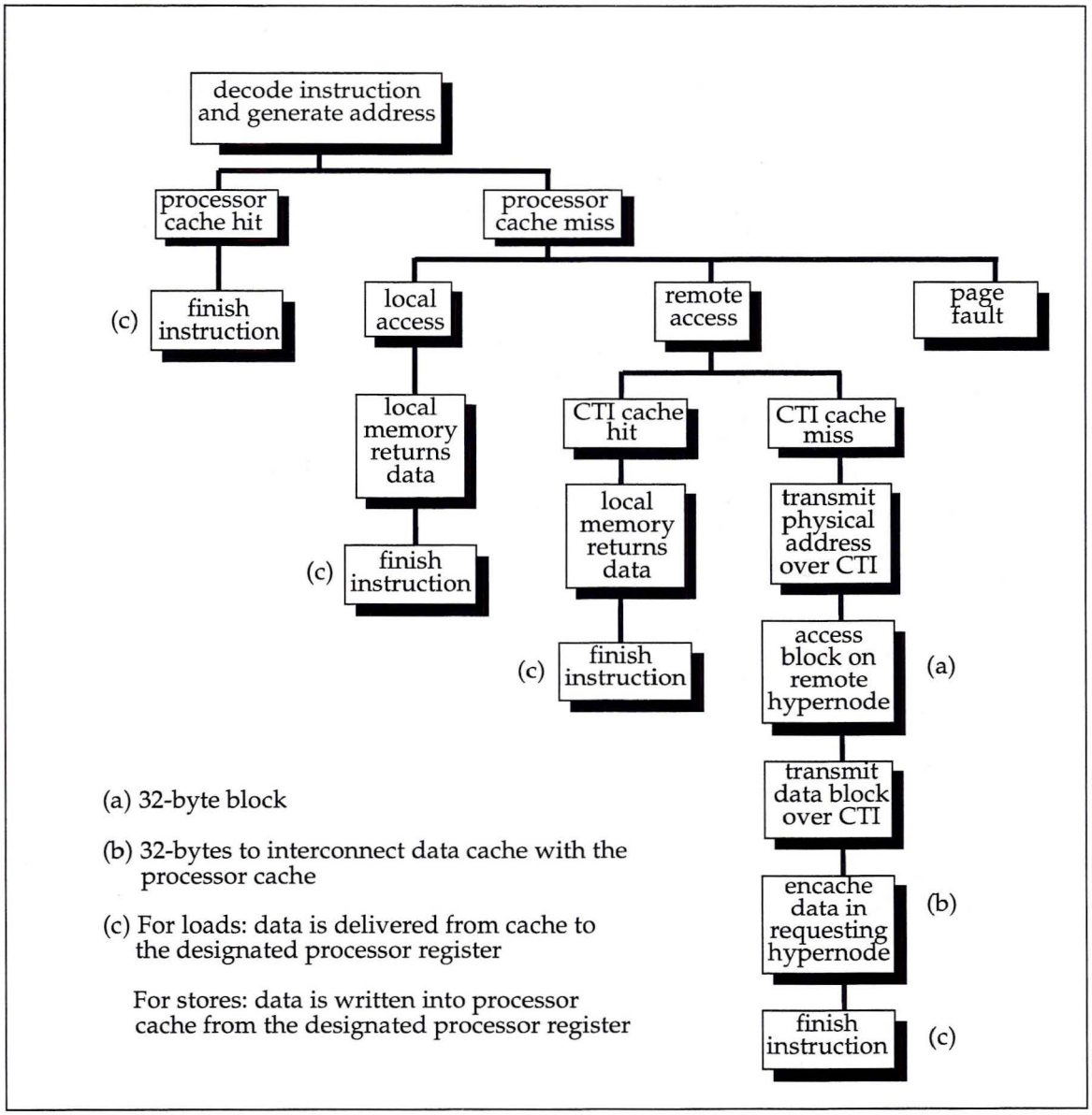


Figure 3 Hardware processing of a load or store instruction

GSM and cache coherence

With all processor references to memory (local and remote), copies of the accessed data are encached into either the instruction or data caches of each processor. If the data is from the remote memory of another hypernode in the case of the X-Class servers, it is also copied into a CTI cache.

If the processor making the memory reference modifies the data and if another processor references that same data while a copy is still in the first processor cache, a condition exists whereby the data has become stale. The hardware continually works to ensure that the second processor does not use an outdated copy of the data from memory. The state that is achieved when both processors' caches always have the latest value for the data is called *cache coherence.*

To maintain updated coherent copies, S-Class and X-Class servers operate under the following rules:

- Any number of read encachements of a cache line can be made at a single time. The cache line can be read-shared in multiple caches.
- To write data (store) into a cache line, the cache line must be "owned" exclusively by the processor. This implies that any other copies must be invalidated.
- Modified cache lines must be written back to memory from the cache before being overwritten.

Cache coherence protocols are different within a hypernode than across the CTI. Cache coherence can result in additional memory latency, because memory control logic may have to force write-backs of modified data before allowing a cache line to be copied into a processor or CTI cache. Providing cache coherence in hardware, however, also provides benefits:

- It avoids the requirement for the programmer to explicitly flush the caches.
- It avoids unnecessary synchronizations.

Physical address space

2

This chapter describes the S-Class and X-Class server physical address space including coherent memory, core logic, and CSR address regions.

Physical addresses

The PA-8000 processor is an implementation of the 64-bit PA-RISC 2.0 architecture. The processor translates all 32- and 64-bit, virtual and absolute addresses to 64-bit physical addresses. External to the PA-8000 chip, however, only 40 bits of the 64-bit physical address are implemented.

The I/O system uses controllers that have fewer than 40 address bits. The mapping of I/O addresses to the corresponding 40-bit physical address space occurs in the PIC I/O subsystem.

S-Class and X-Class server processors have four addressable physical address regions. These are:

- **Coherent memory space**—Memory (both local and remote) used for programs and data and available to every processor. This is the bulk of the memory space.
- **Core logic space**—The space occupied by a group of hardware registers that comprises the hypernode core logic function and is accessible to all processors within that hypernode.
- **Local I/O space**—The space occupied by the PCI buses, and prefetch and context RAM CSRs associated with input and output devices within a hypernode.
- **Non-I/O CSR space**—The space occupied by the group of all other CSRs within a hypernode.

Core logic space and local I/O CSR space are only accessible by processors in the local hypernode in S-Class and X-Class servers. The other two spaces, coherent memory space and non-I/O CSR

space, are accessible by all hypernodes in X-Class servers. Figure 4 shows how the address space is partitioned.

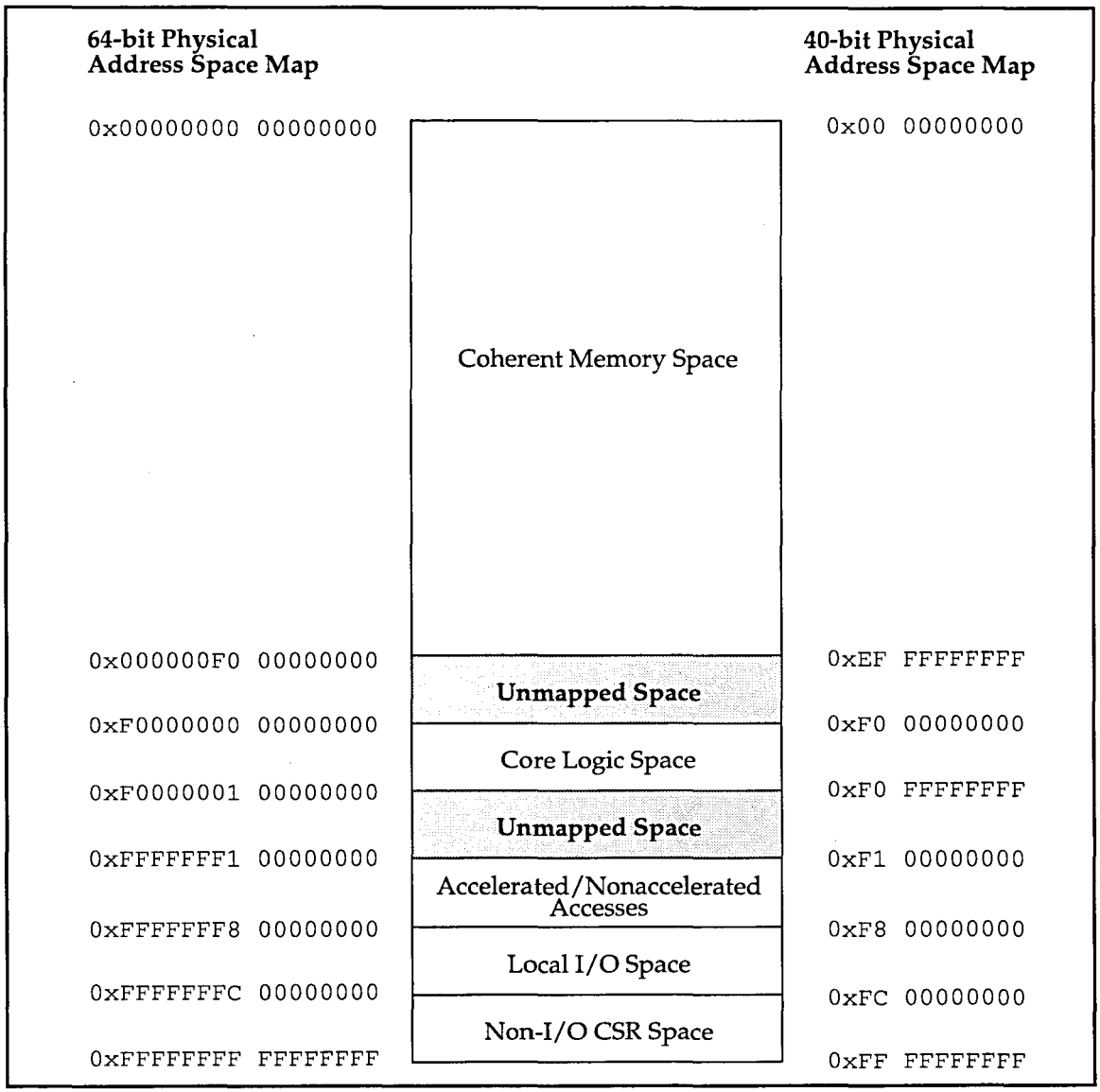


Figure 4 Physical address space partitioning

The left side shows the PA-8000 64-bit address map, and the right shows the 40-bit external address map used by the system. The two regions labeled "Unmapped Space" exist in the 64-bit physical address space but do not exist for the 40-bit physical address space.

Hypernode addressing

The physical memory within the hypernode is byte-addressable and is accessed by either 32-bit or 64-bit absolute pointers. An absolute pointer is an unsigned integer whose value is the address of the most significant byte of the operand it designates.

Addressable units (shown in Figure 5) are bytes, halfwords (two bytes), words (four bytes), and double-words (eight bytes). Bytes in memory and bits within larger units are always numbered from zero, starting with the most significant byte or bit, respectively.

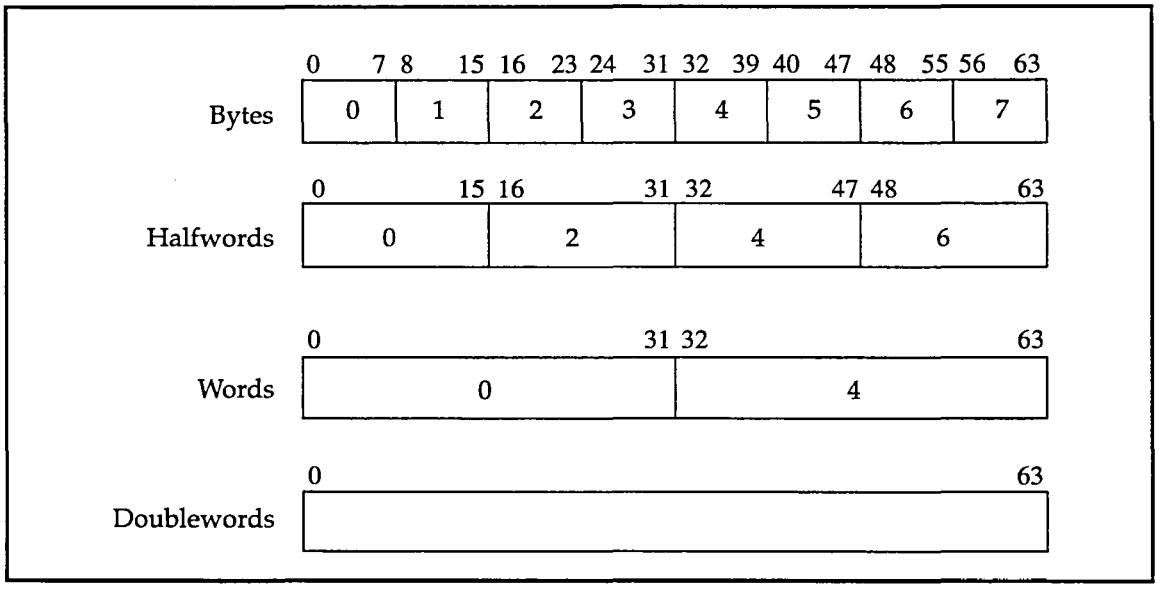


Figure 5 Physical memory addressing and storage units

All addressable units must be stored on their naturally aligned boundaries. A byte can be at any address, halfwords at any even address, words at any address that is a multiple of four, and double-words at any address that is a multiple of eight. If an unaligned virtual address accesses memory, a PA-8000 unaligned reference trap occurs.

Hypernode CSR space is referenced using bytes, halfwords, words, and double words with a strong preference for double-word accesses. Internode CSR accesses are double-word in size.

Hypernode identifier

Every hypernode in both S-Class and X-Class servers has a unique hypernode identifier (Node ID) of five bits.

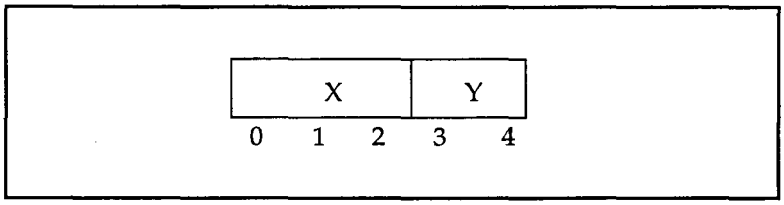


Figure 6 X-Class server Node ID field format

The X and Y subfields of the Node ID field correspond to two dimensions used for configuring multiple hypernode topologies.

Coherent memory space

From Figure 4 on page 18, coherent memory occupies the largest amount of physical address space. Figure 7 shows both the 64-bit and 40-bit physical address formats.

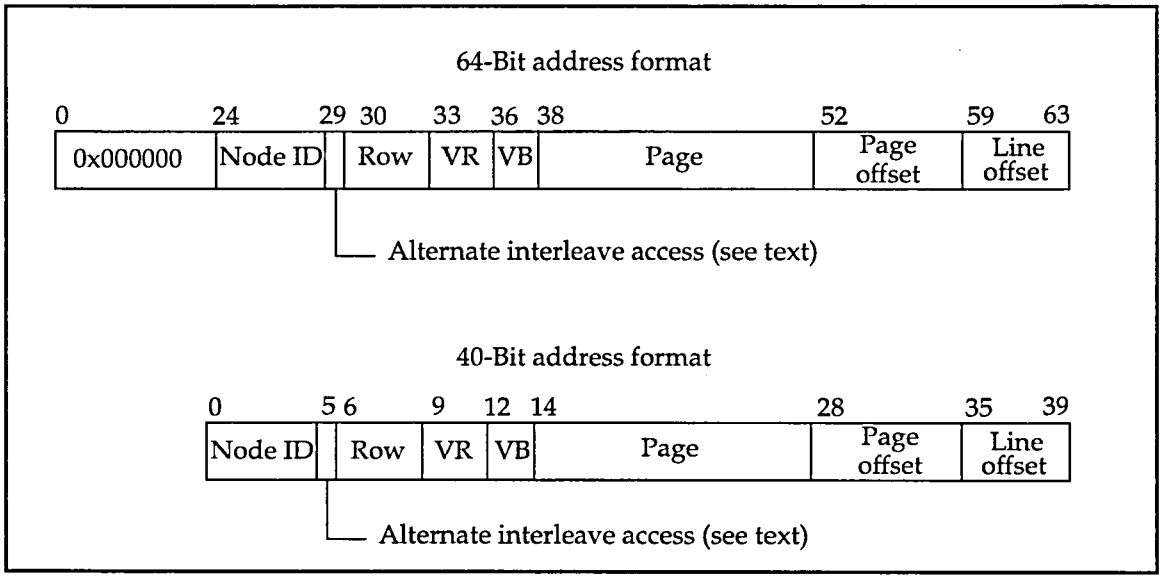


Figure 7 Coherent memory space address formats

The architecture supports a maximum of 16 GBytes per hypernode. Currently, the S-Class and X-Class servers are populated with up to four Gbytes of memory. These systems *do not* employ the alternate interleave bit.

Coherent memory layout

Memory physically resides in memory blocks, with each block controlled by a single MAC. Each block has four banks of memory. Each memory block is associated with a CTI ring for remote memory accesses. Coherent memory is further divided into memory lines 32-bytes in size.

Memory blocks are implemented with memory SIMMs, up to 16 SIMMs per block (one block per MAC). Each SIMM can have one or two rows of SDRAM chips, constructed with either 16-Mbit or 64-Mbit SDRAMs.

Addressing a byte of memory

Figure 8 represents a fully populated hypernode with 16 Gbytes of physical memory (currently, the maximum memory S-Class

and X-Class servers can have is four GBytes). It shows how a byte of memory is addressed.

Note

Figure 8 represents only the *concept* of how memory is configured in the hypernode. It does *not* depict the physical implementation.

The eight memory boards are at the top of the drawing. Each board has 16 SIMMS and each SIMM is loaded with memory chips on both sides. Each memory board has four banks comprised of four SIMMs in the vertical direction. Also, each board has eight rows along the horizontal direction. The top and bottom of each SIMM in the horizontal direction are part of two separate and adjacent rows. For example, Row 0 consists of the memory mounted on the bottom of each of the four SIMMs located on the bottom of the memory board in the horizontal direction. If the memory chips were 64-MBit SDRAM, each board would contain two GBytes of memory. A CTI ring is associated with each memory board for a total of eight rings per dimension.

A pair of rows per bank (that is, rows 0 and 1) is sufficient to maintain maximum memory bandwidth. Additional rows add memory capacity, not additional bandwidth.

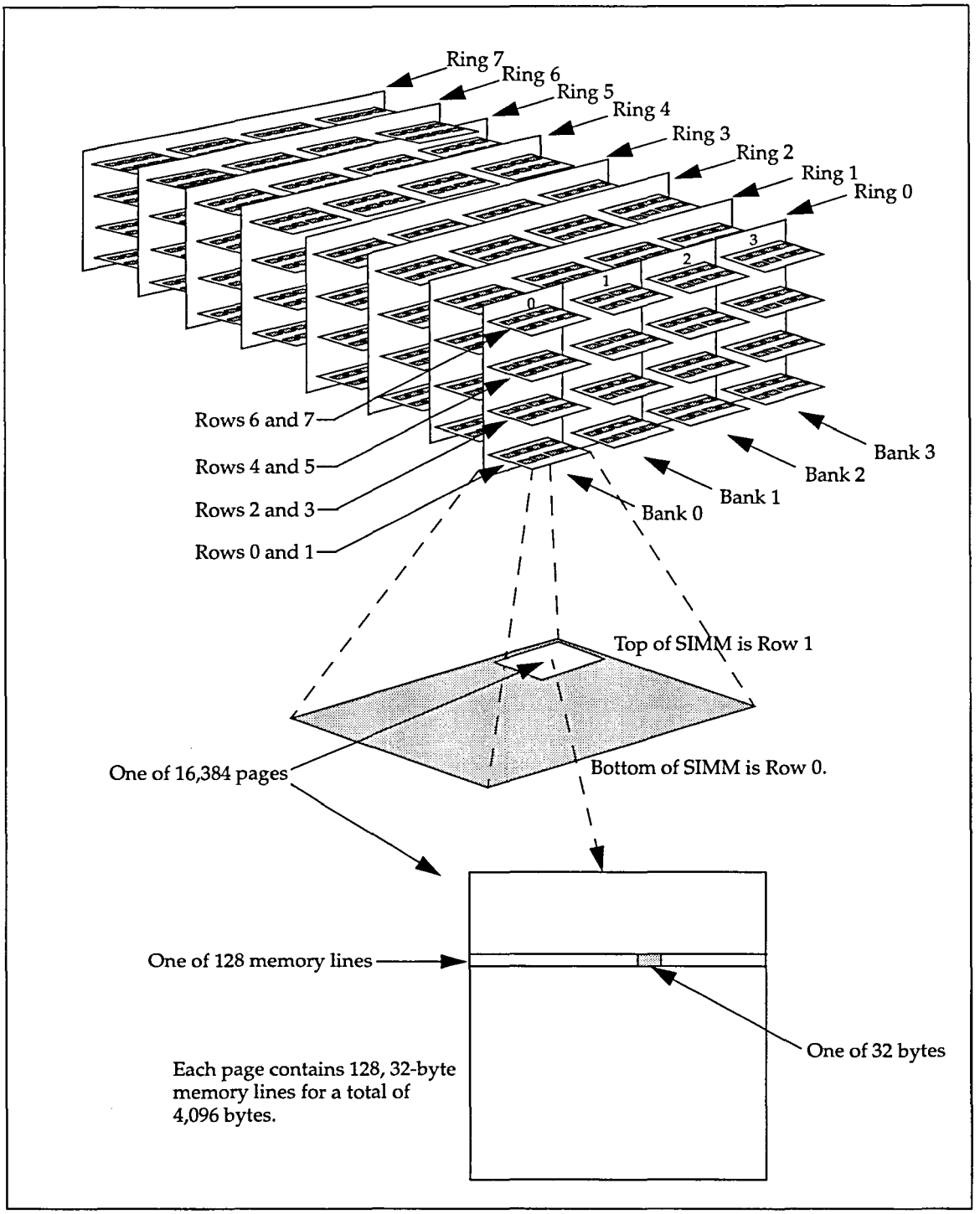


Figure 8 Conceptual layout of physical memory of a fully populated hypernode

As shown in the physical address in Figure 7 and the conceptual memory layout in Figure 8, a byte of memory is accessed as follows:

1. The Row field selects one of eight rows of SDRAMs.
2. The Virtual Ring (VR) field selects the memory board.
3. The Virtual Bank (VB) field selects one of four banks on the appropriate memory board.
4. These three components of the physical address point to one side of a SIMM which contains 16,384 pages of 4,096 bytes each.
5. The Page field selects one of 16K pages on one side of the SIMM.
6. The Page offset field selects one of 128 memory lines in the page.
7. The Line offset field selects the appropriate byte in the line.

Virtual rings and banks

The Virtual Ring field and Virtual Bank fields specify the ring or bank, respectively, in which the first line of a page of memory resides. Subsequent memory lines are interleaved across the configured memory banks on a ring first and then a bank, within ring rotation.

Rows

Each row contains 512 Mbytes of physical memory with 16-Mbit SDRAMs or 2 Gbyte with 64-Mbit SDRAMs. Within a row, 32 subpartitions exist, one for each ring-bank combination (eight rings with four banks per ring). Each subpartition is 64 Mbytes in size. If 16-Mbit SDRAMs are installed into a row of memory, then only the first 16 Mbytes of each subpartition of a row is accessible. Otherwise, with 64-Mbit SDRAMs, the entire 64 Mbyte subpartition is accessible.

Noninterleaved memory accesses are different from interleaved. They are divided into eight *rows*. Figure 9 illustrates the layout for the noninterleaved coherent memory space.

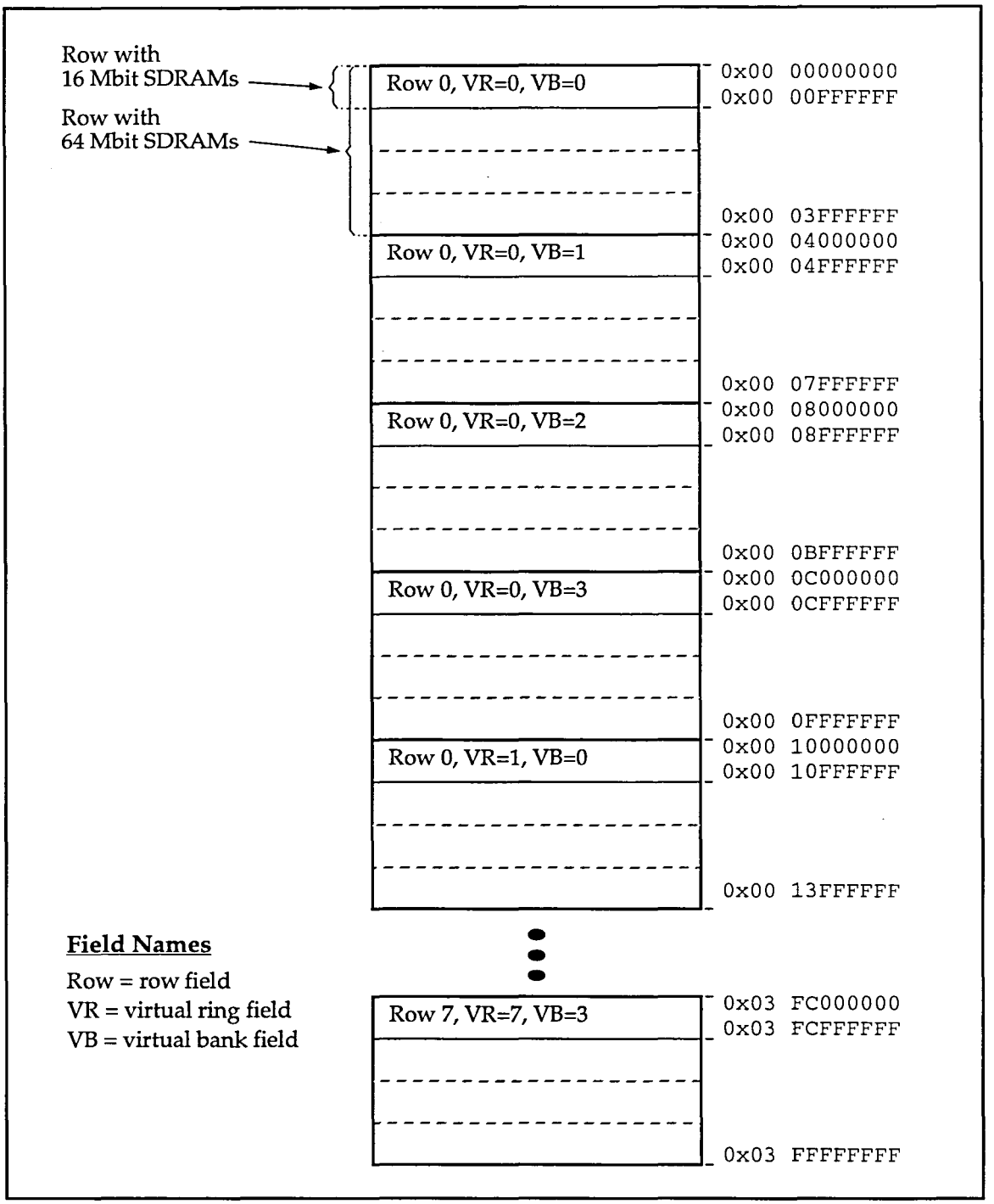


Figure 9 Coherent memory space layout

Coherent memory interleave

Memory interleaving distributes consecutive lines of memory across as many banks as possible. It is supported across four, eight, 16, and 32 banks of memory as follows:

- A single memory block (four banks)
- An even-odd pair of memory blocks (eight-way interleave)
- Two pairs of memory blocks (16-way)
- Four pairs of memory blocks (32-way)

For balanced bandwidth reasons, only pairs of memory boards are supported with normal memory interleave. Three pairs of memory blocks are interleaved with 16-way interleave.

Memory interleave generation

Coherent memory and CTI ring interleaving (for X-Class servers only) are performed on all memory references, except in the single memory block mode. A memory block is associated with each MAC and CTI ring within a hypernode. To optimize memory bandwidth, memory blocks must be installed in pairs, even and odd. There are a maximum of four even-odd pairs of memory blocks in a hypernode.

A memory block consists of either two or four memory banks. The two memory bank configuration is a reduced bandwidth configuration.

Figure 10 shows the mechanism for interleave. The 40-bit physical address provides the basis from which the memory block and memory bank are selected.

The Memory Board Configuration register supplies the map used to translate a memory block to the associated memory board where the memory line resides. See the "PAC Memory Board Configuration register" section on page 48 for more information.

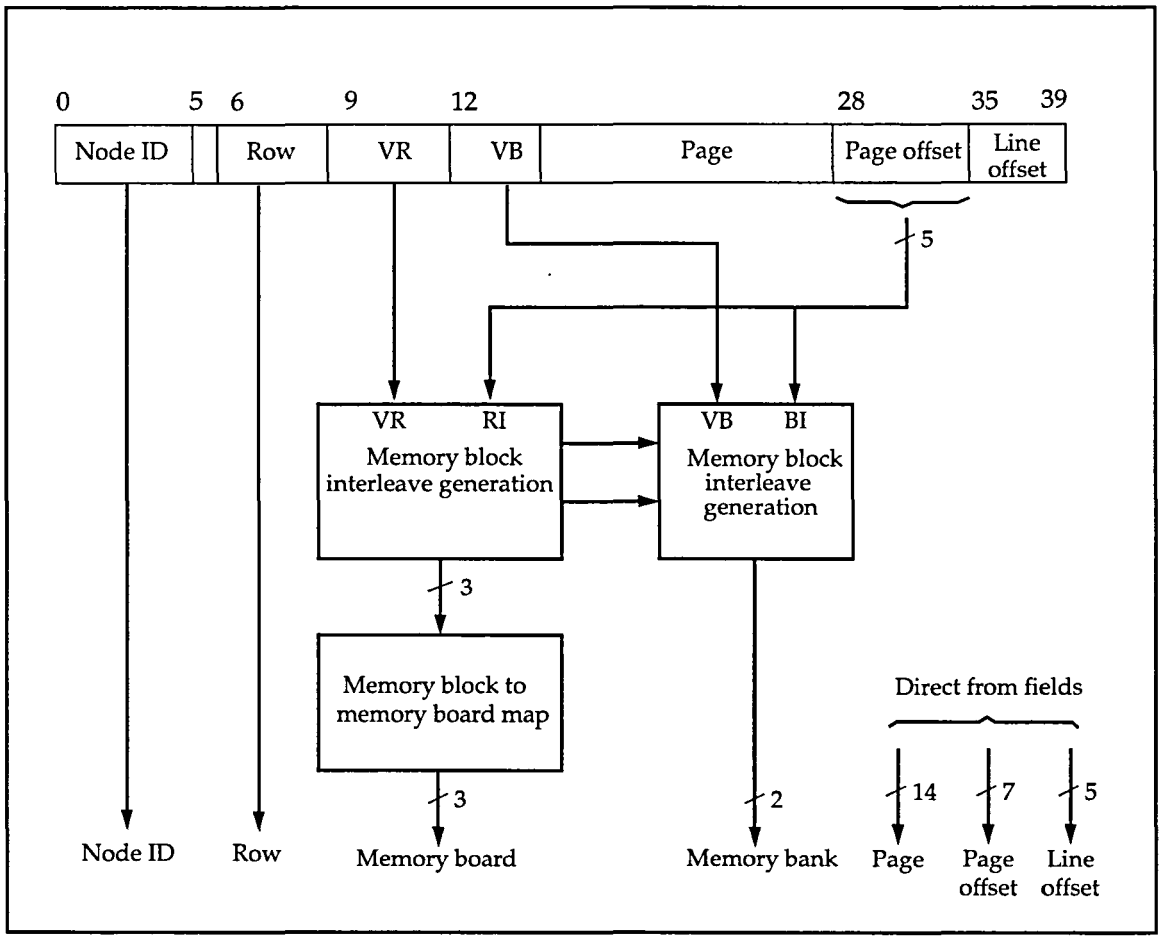


Figure 10 40-bit coherent memory address generation

The MAC online field of the System Configuration register checks that a valid virtual ring value is specified in the memory address. An invalid virtual ring value will result in a PA-8000 high-priority machine check trap.

Force hypernode ID function

In X-Class servers, the Force Node ID function allows all hypernodes to appear to have memory in the ranges of 0x0000000 to 0x0FFFFFF and 0x4000000 to 0x4FFFFFFF. It allows the operating system to be loaded into each hypernode without having to specify the Node ID.

The hypernode ID for the generated address is selected from either the Node ID field of the 40-bit physical address, or from the

Node ID field of the System Configuration register. The System Configuration register Node ID field is selected when the System Configuration register *Force Node ID Enable* bit is set and when the Row, VR, the most significant bits of the VB, and the two most significant bits of the Page (bits 14-15) are zero.

The original hypernode identification is checked to be the value zero prior to forcing it to the local hypernode's value. The check ensures cache coherency is maintained. If a processor accesses the first or fifth 16-Mbyte block and the original hypernode identification is not zero, a high-priority machine check trap occurs.

The Force Node ID function is also applied to the CSR Operation Address register *Address* field, but does not apply to the data mover or I/O subsystem addresses. Refer to Chapter 4, "Data mover," Chapter 5, "Synchronization," and Chapter 7, "I/O subsystem," for additional information.

Ring and bank index selection

The Page Offset bits indicate the bank index and ring index. If no ring interleaving is performed, the ring index is zero. Table 2 shows which offset bits are used. The numbers inside the brackets indicate the appropriate address bits.

Table 2 Bank/ring index selection

Board pairs	Bank index (BI)	Ring index (RI)
No Interleave	Offset 33:34)	RI=0
One Pair	Offset (32:33)	Offset (34)
Two or Three Pairs	Offset (31:32)	Offset (33:34)
Four Pairs	Offset (30:31)	Offset (32:34)

Memory block interleave pattern

The memory block generated for the noninterleaved case is simply the Virtual Ring field.

The memory blocks generated for interleave cases of one, two, three, and four board pairs are given in Table 3, Table 4, Table 5,

and Table 6, respectively. The tables show the memory board number with respect to the virtual ring and ring index.

Table 3 Memory block interleave pattern for one board pair

VR (9:11)	RI=0	RI=1
0	0	1
1	1	0
2	2	3
3	3	2
4	4	5
5	5	4
6	6	7
7	7	6

Table 4 Memory block interleave pattern for two board pairs

VR (9:11)	RI=0	RI=1	RI=2	RI=3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2
4	4	5	6	7
5	5	6	7	4
6	6	7	4	5
7	7	4	5	6

Table 5 Memory block interleave pattern for three board pairs

VR (9:11)	RI=0	RI=1	RI=2	RI=3
0	0	1	2	3
1	1	2	3	0
2	2	3	4	5
3	3	4	5	2
4	4	5	0	1
5	5	0	1	4
6	6	7	6	7
7	7	6	7	6

Table 6 Memory block interleave pattern for four board pairs

VR (9:11)	RI=0	RI=1	RI=2	RI=3	RI=4	RI=5	RI=6	RI=7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Memory bank interleave pattern

Memory bank interleaving occurs for all memory board interleave spans. Memory configurations allow either two or four memory banks per MAC. To support these two configuration options, two- and four-way memory bank interleaving are supported. Table 7 shows the memory bank interleave pattern.

Table 7 Memory bank interleave pattern for four banks

VB (12:13)	BI=0	BI=1	BI=2	BI=3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Bank interleaved memory pattern

Single memory board interleave mode forces contiguous memory lines to reside in the same memory block. Within the memory block, memory lines interleave across the four memory banks. Figure 11 shows the interleave pattern for this mode. The pattern is the same independent of the number of memory blocks in a hypernode The pattern shown is for a 4096-byte page of memory (128 memory lines per page).

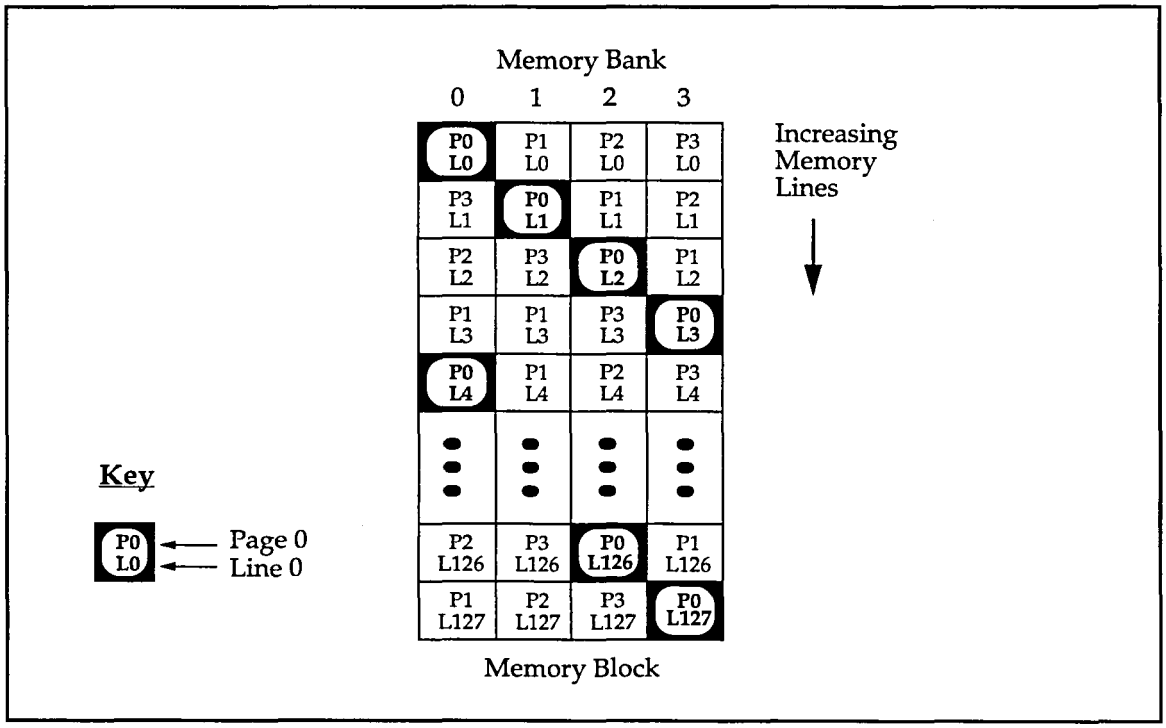


Figure 11 Single memory block interleave pattern

Block and bank interleave memory pattern

In multiple memory board interleave modes, memory lines are interleaved across the largest power-of-two memory banks that exist in the hypernode. There are up to eight memory blocks in a hypernode, resulting in 32-way memory line interleaving. The minimum system configuration has two memory blocks, resulting in eight-way memory line interleaving. With both four and six memory blocks in a hypernode, the interleave is 16-way. Figure 12 shows the interleave pattern for a system with four memory blocks (four MACs/TACs/CTI rings). The pattern shown is for 4096-byte pages of memory (128 memory lines per page).

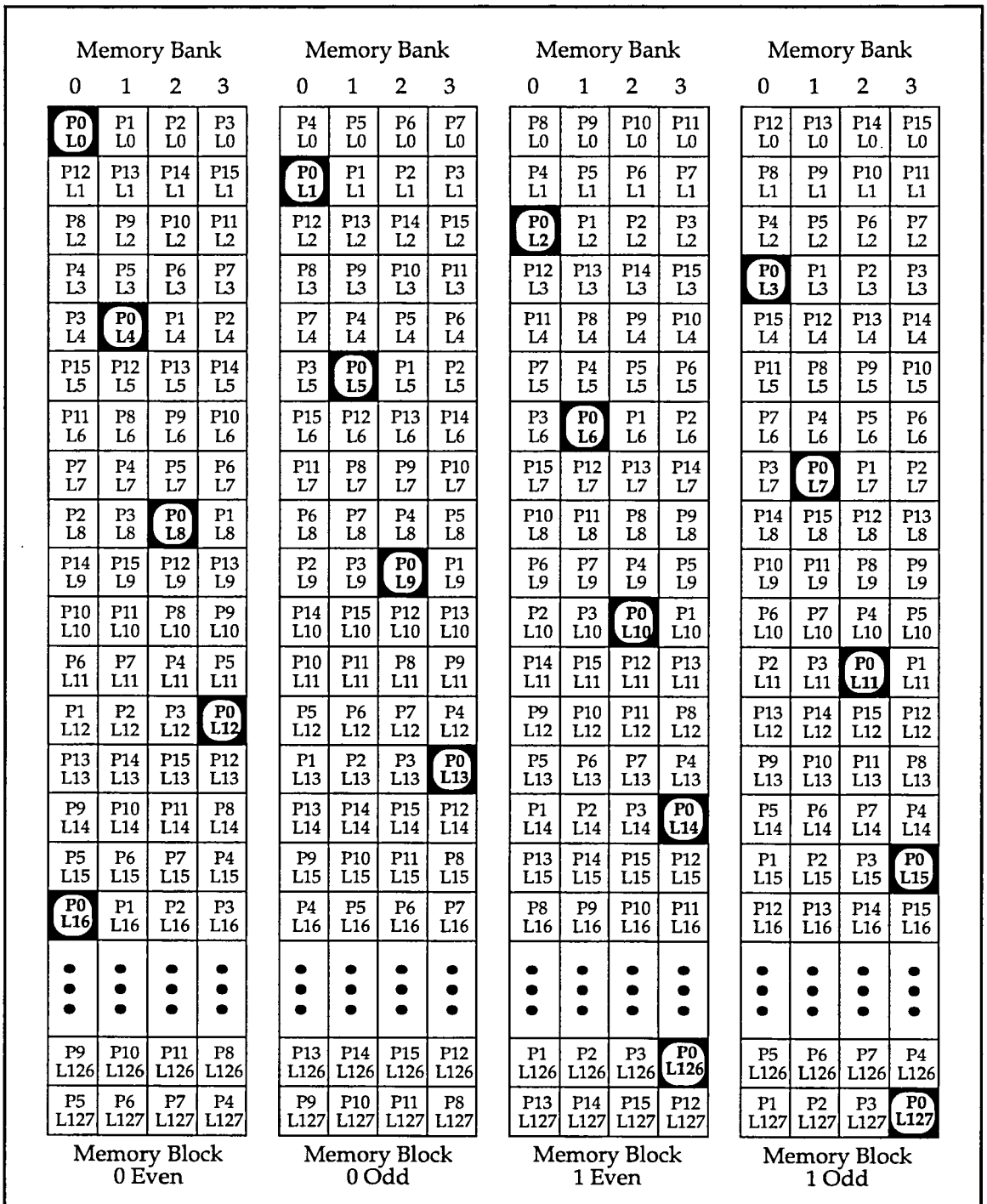


Figure 12 Memory line interleave pattern with four memory blocks

CTI cache layout

In X-Class servers, each hypernode has a CTI cache to decrease the access latency of frequently accessed remote-hypernode coherent memory lines. The size of the CTI cache is configurable from 16 Mbytes to over one Gbyte.

Figure 13 shows the bits of the 40-bit physical address which are forced to generate a CTI cache address.

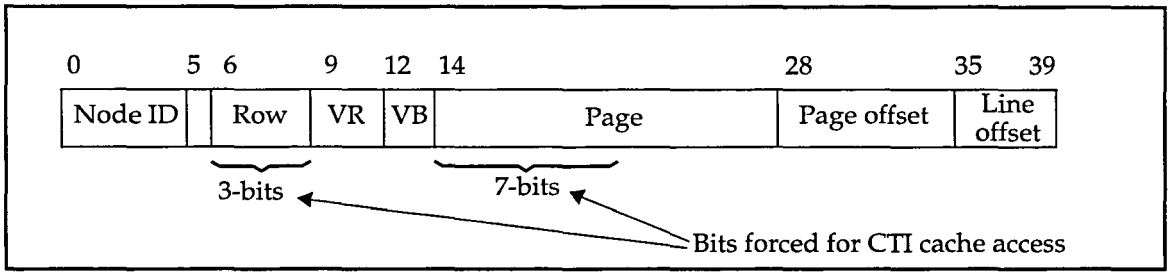


Figure 13 40-bit coherent memory address format

The ten bits shown in Figure 13 configure the size and location of the CTI cache. The number of bits set determines the size of the cache; the specific value determines its location. Table 8 shows the number of bits that must be set for each CTI cache size: 4, 8, 16, and 32 banks.

Table 8 CTI cache size options

Number of bits set	4 banks	8 banks	16 banks	32 banks
2	512	1024	2048	4096
3	256	512	1024	2048
4	128	256	512	1024
5	64	128	256	512
6	32	64	128	256
7	16	32	64	128
8	8	16	32	64
9	4	8	16	32
10	2	4	8	16

When the row(s) of memory in which the CTI cache resides are installed with 16-Mbit SDRAMS, the two most significant bits of the Page field must be set to zeros. Figure 14 illustrates the layout for a CTI cache with the Row field set to 0x4, and the most significant seven bits of the Page field set to 0x0F.

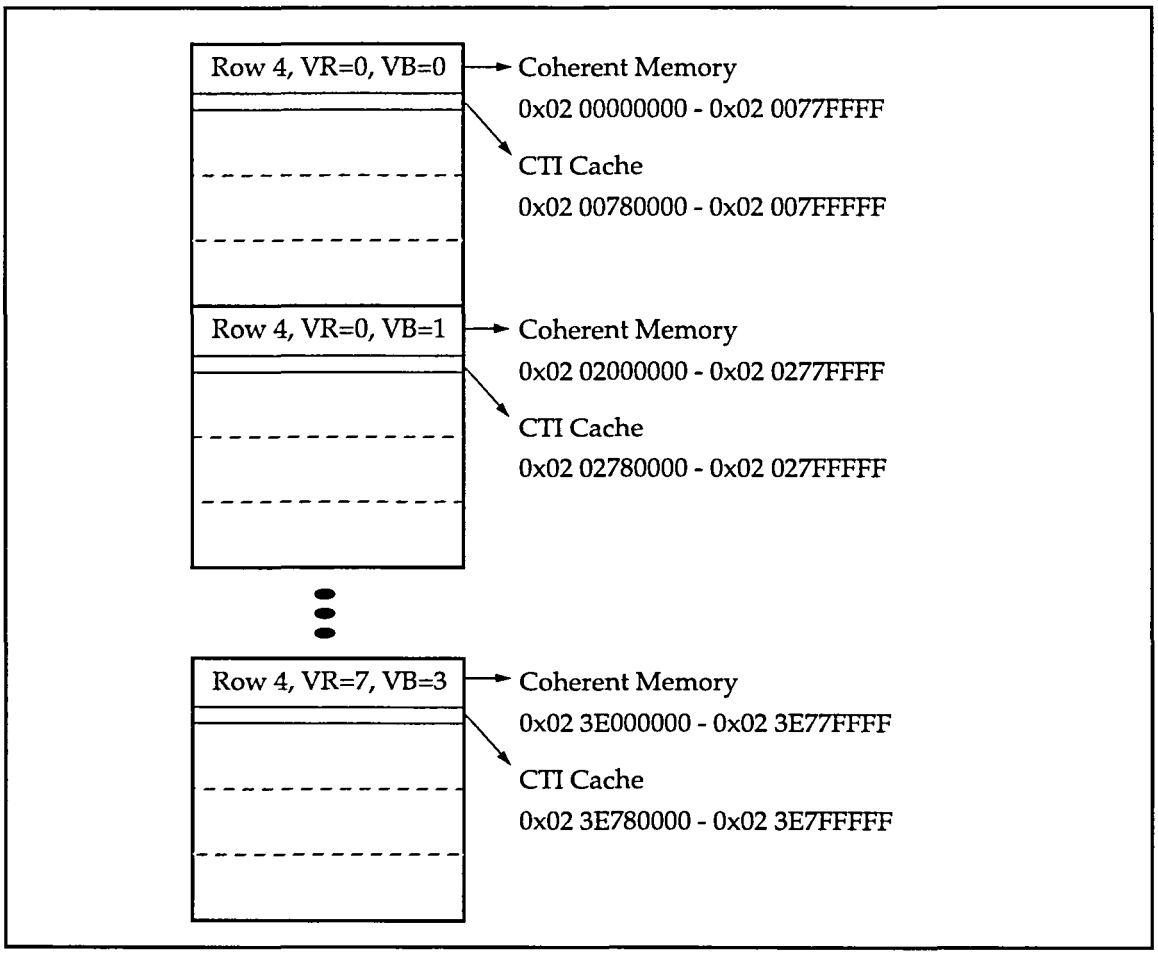


Figure 14 Coherent memory space layout with CTI cache

As shown in Figure 14, the CTI cache occupies the upper 512 Kbytes of memory of the subpartition of each row across which it is interleaved. In the example, the interleaving is 32-way so that the aggregate CTI cache size is 16 Mbyte (32-way, 512 Kbytes per subpartition). All other rows of memory would be available for coherent memory access.

Nonexistent memory

All coherent memory accesses are checked to verify that the address of the request points to existing physical memory. Accesses to nonexistent memory or to the address space where a CTI cache exists will result in an error response. The only exception is accesses to the CTI cache by a cache flush entry operation or a diagnostic memory operation.

The specific checks made are:

- The appropriate MAC online bit is set in the source PAC System Configuration register.
- The Exist bit is set for the appropriate SIMM row in the MAC Memory Row Configuration register.
- The request is to the appropriate region of memory.

Core logic space

Core logic space is used to access core logic hardware (EEPROM, SRAM, and core logic CSRs) and is only accessible within a hypernode. The processor has a fixed decode for this space. Bits 8 through 15 of the 40-bit physical address are ignored for address decoding.

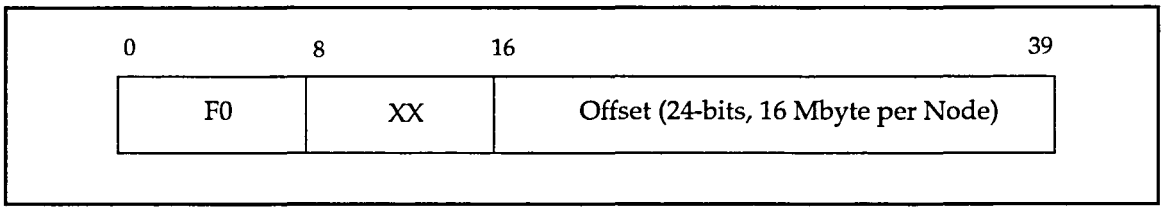


Figure 15 40-bit core logic space format

The addresses contain a 24-bit offset used as the core logic bus address. The PAC translates from the physical addresses to core logic bus addresses. It also splits 64-bit requests into two, 32-bit requests. The PUC translates the core bus address to utility address. See Figure 16.

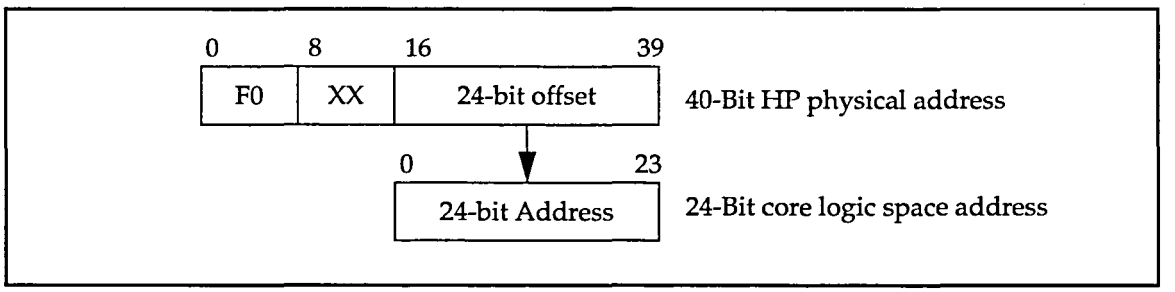


Figure 16 Core logic address translation

Core logic space is further partitioned for EEPROM, SRAM, and CSR Space. Table 9 shows the address ranges for each of these partitions.

Table 9 Core logic space partitions

Partition	Core logic space offset range
EEPROM	0x000000 - 0x7FFFFF
SRAM	0x800000 - 0xBFFFFF
CSR	0xC00000 - 0xFFFFFF

Processor-dependent code (PDC) space is accessed using the core logic bus attached to each processor. A PDC space access is not routed through the crossbar.

Local I/O space

A processor can directly access all I/O Space on the local hypernode (in both S-Class and X-Class servers) using the I/O controllers.

Refer to Figure 17. The PAC of the source processor checks the value of the DXbr field against the appropriate bit of the Processor Agent online field in the System Configuration register of the source processor to verify that the destination processor agent is online. The PAC of the destination processor checks to see if the PIC online bit in its Chip Configuration register is set. If either of the online bits are not set, the request will fail with a high-priority machine check trap.

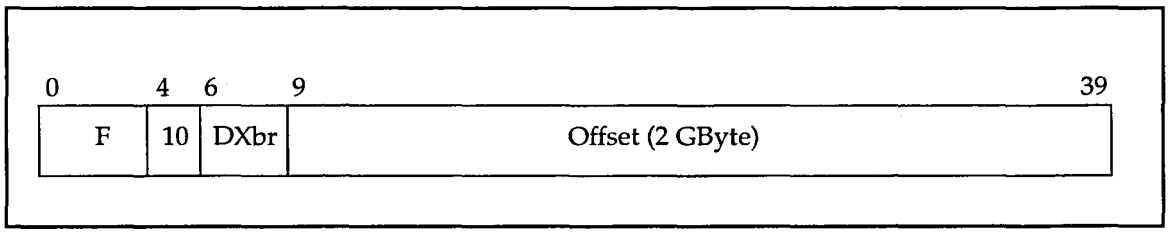


Figure 17 40-bit local I/O space format

The bits of the local I/O space address are as follows:

- ***DXbr*** field (bits 6:9)—Specifies which of the eight cross bar ports (which PAC chip) the request is to be routed to.
- ***Offset*** field (bits 33:39)—Specifies the offset into I/O Space. In this space, all PCI configuration, I/O CSRs and I/O memory space must be allocated. See the "Host-to-PCI address translation" section on page 117 for more information.

Non-I/O CSR space

Non-I/O CSRs reside within the PA-8000, PAC, MAC, TAC, and PIC.

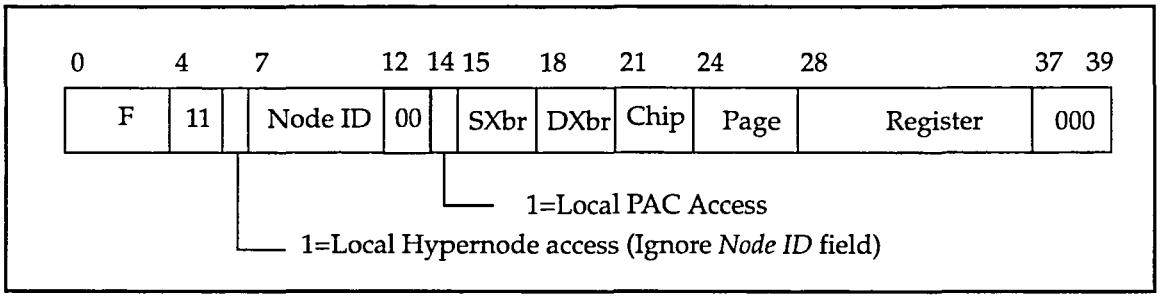


Figure 18 Non-I/O CSR space format

The bits and fields of the CSR space address are as follows:

- ***Local hypernode access*** bit (bit 6)—Indicates that the access is to the local hypernode. When the Local Hypernode Access bit is set, the Hypernode ID field is ignored for purposes of intrahypernode routing.
- ***Local PAC access*** bit (bit 14)—Indicates that the access is to the local PAC space within the associated PAC chip. When this bit is asserted, the Node ID, SXbr, and DXbr fields are ignored and specific bits of the Page field are forced to the particular processor issuing the request.
- ***SXbr*** field (bits 15:17)—Currently not used in S-Class and X-Class servers.
- ***DXbr*** field (bits 18:19)—Specifies which of the eight cross bar ports the request is to be routed on the destination hypernode.
- ***Chip*** field (bits 21:23)—Routes the packet to the appropriate chip at a crossbar port.
- ***Page*** field (bits 28:36)—Separates groups of CSRs into similar usage spaces.

CSR access

There are four different packet routing methods used for accessing CSRs. These four methods are:

- Processor-local
- PAC-local
- Hypernode-local
- Remote

The 40-bit physical address determines which access method will be used.

Processor-local access

Processor-local accesses reference CSRs that reside in the processor issuing the request. These accesses are sent out and brought back into the requesting processor on its Runway bus. The PAC, which is also connected to the runway bus, ignores the request. The processor online bits of the processor agent are not checked for processor-local accesses.

PAC-local access

PAC-local accesses are accesses to CSRs that reside in the PAC physically connected to the processor that is issuing the request. These accesses are identified as PAC-local and are not sent to the crossbar array (RAC). Table 10 shows which fields must be specified for PAC-local addressing.

Table 10 Field specifications for PAC-local access

Field	Specification
Bits 0:5	0x3F
Local Node	X
Node ID	X
Local PAC	1
SXbr	X
DXbr	X
Chip	X

This method accesses processor specific CSRs that reside in a PAC. All processor-specific PAC CSRs are identified as having bit 2 of the Chip field set (presently only pages two and three have processor specific CSRs defined). When the PAC detects a

processor specific page it forces bit three of the page field as appropriate for the processor issuing the request.

Note

The PAC online field of the System Configuration CSR is not checked for PAC-local accesses.

Hypernode-local access

Hypernode-local accesses are used to access CSRs that reside in the local hypernode. These accesses are sent to the crossbar even if the access is to the local PAC.

If the access is to a MAC or TAC, the DXbr field routes the request to the proper MAC. The MAC Online field of the source PAC System Configuration register is checked to ensure the destination MAC is online. A high-priority machine check trap will result if the destination MAC is not online.

If the access is to a PAC, PIC, or processor, the request is first routed to the MAC specified by the Intermediate MAC field of the PAC Configuration CSR. The PAC Online field of the source PAC System Configuration CSR is checked to ensure the destination PAC is online. A high-priority machine check trap results if the destination MAC is not online.

Table 11 Field specifications for hypernode-local access

Field	Specification
Bits 0:5	0x3F
Local Node	1
Node ID	X
Local PAC	0
SXbr	X
DXbr	Destination crossbar port
Chip	Destination chip

Remote access

Remote accesses are accesses to CSRs that reside in any hypernode of the X-Class server. All local hypernode CSRs and many remote hypernode CSRs can be accessed with this method. These accesses are sent to the crossbar, even if the access is to the local PAC. The request is sent to the local hypernode MAC specified by the SXbr

field. The local hypernode MAC checks if the Node ID is for the local hypernode.

If the request is for the local hypernode, the local hypernode MAC uses the DXbr and Chip fields to determine the local hypernode destination of the request. The original PAC checks that the destination PAC or MAC is online using the PAC Online and MAC Online fields of the System Configuration register. Only the MAC specified by the SXbr field is accessible using the remote access method.

If the Node ID of the access does not match the local hypernode ID, the request is sent to the MAC of the remote hypernode using a CTI ring. The remote hypernode MAC routes the request using the DXbr and Chip fields to the appropriate remote hypernode destination. The TAC of the remote hypernode checks that the destination chip is to (1) an online PAC, (2) a processor on an online PAC, or (3) the MAC on the same CTI ring as the TAC.

If the access is to a MAC or TAC (on a local or remote hypernode), the SXbr and DXbr fields must have the same value. Table 12 specifies the fields for remote addressing:

Table 12 Field specifications for remote access

Field	Specification
Bits 0:5	0x3F
Local Node	0
Node ID	Destination Node ID
Local PAC	0
SXbr	CTI ring to be used for routing
DXbr	Destination crossbar port
Chip	Destination chip

Access to nonexistent CSRs

It is possible that a request be sent to a CSR of a controller that is not online. Online bits are implemented for processors, PACs, MACs, PICs, and TACs. Memory uses existence bits.

Accesses to nonexistent CSRs terminate in one of the following ways:

- Requests with a response to a CSR covered by an online bit result in an error response being returned to the processor. The processor issues a high-priority machine check interrupt.

- Requests without a response to a CSR covered by an online bit result in a time-out when the next synchronization operation is performed. The synchronization time-out results in a high-priority machine check interrupt.
- Requests with a response to a CSR not covered by an online bit result in a time-out. The request time-out results in a high-priority machine check interrupt.
- Requests without a response to a CSR not covered by an online bit result in a time-out when the next synchronization operation is performed. The synchronization time-out results in a high-priority machine check interrupt.

System Configuration register

The System Configuration register specifies system configuration parameters. The register is replicated on the PAC, MAC, and TAC, but only fields used by each controller type are implemented for a particular controller. Therefore, not all fields exist on a PAC, MAC, or TAC.

Figure 19 shows the generic format of the register. All fields are written to by a write access and read from by a read access. All fields are unaffected by reset unless specified.

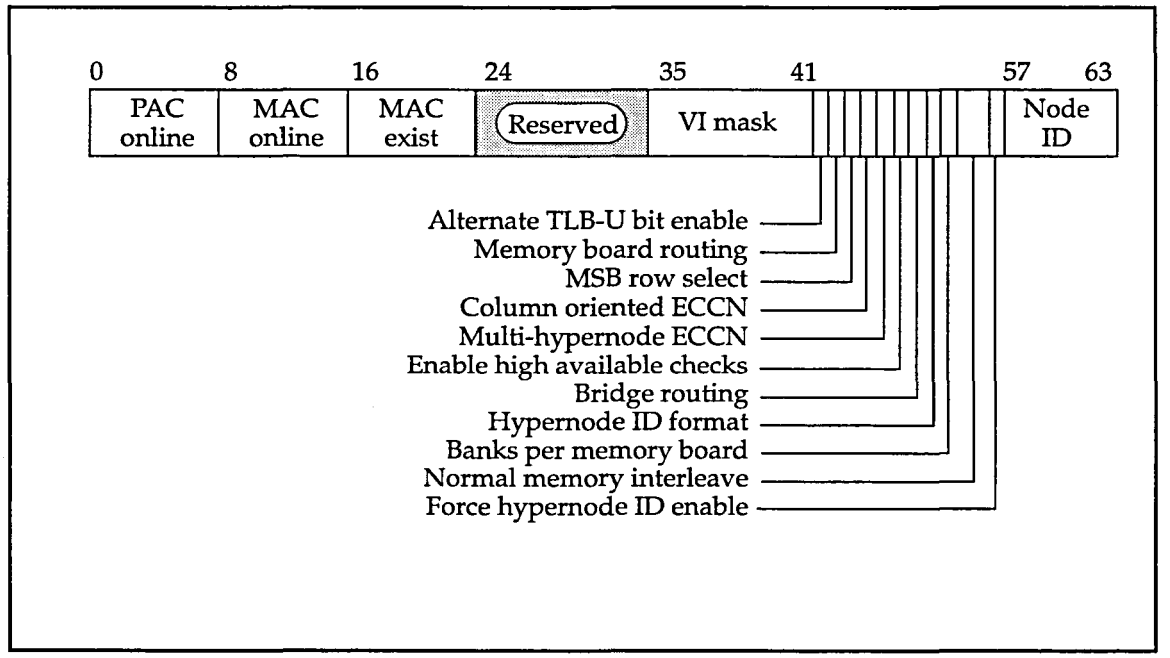


Figure 19 System Configuration register definition

The bits and fields of the System Configuration Register are defined as follows:

- *PAC online* field (bits 0:7)—Specifies which PACs are accessible. These bits are used to validate all I/O space and local CSR Space requests. The field is cleared by reset.
- *MAC online* field (bits 8:15)—Specifies which MAC ASICs are accessible. These bits are used to validate all Coherent Memory Space and CSR Space requests. The field is cleared by reset.
- *MAC exist* field (bits 16:23)—Indicates which MAC ASICs exist in the system. These bits are used by software to initialize the MAC online field. The field is initialized by reset. A CSR write is ignored.
- *VI mask* field (bits 35:41)—Specifies the Virtual Index bits generated by the PA-8000 processor that are masked (forced to zero).
- *Alternate memory interleave* field (bits 42:44)—Specifies the number of even-and-odd memory board pairs spanned by alternate memory interleaving. Alternate interleave is currently not implemented in S-Class and X-Class servers.
- *Alternate TLB-U bit enable* bit (bit 45)— Enables checking for coherent accesses to noncoherent semaphore memory.
- *Memory board routing* bit (bit 46)—Selects the coherent memory request to the memory board routing function. When the bit is zero, even coherent memory requests are routed to even memory boards and odd requests to odd boards. When the bit is one, even coherent memory requests are routed to odd memory boards and odd requests to even boards.
- *MSB row select* bit (bit 47)—Selects which bit of the 40-bit physical address is the most significant bit of the three-bit row selection information. When the MSB Row Select bit is zero,
- *Column oriented ECCN, Multihypernode ECCN, Enable high availability checks* bits (bits 48, 49, 50)—Control the high availability modes. These bits are currently not used in S-Class and X-Class servers.
- *Bridge routing* bit (bit 51)—Not currently used in S-Class and X-Class servers.
- *Node ID format* bit (bit 52)—Specifies one of two coherent memory format options. This bit is currently not used in S-Class and X-Class servers.
- *Banks per memory board* bit (bit 53)—Specifies whether two or four banks exist per memory board.

- *Normal memory interleave* field (bits 54:55)—Specifies the number of even/odd memory board pairs which normal interleaving should span.
- *Force Node ID enable* bit (bit 56)—Enables mapping the first 16 MBytes of coherent memory space to the first 16 Mbytes of the local hypernode.
- *Node identification* field (bits 57:63)—Specifies the hypernode identification number for the hypernode.

PAC Configuration register

Each PAC has one Processor Agent Configuration register which specifies information about the PAC. Each PAC can be configured differently.

Figure 20 shows the format of the PAC Chip Configuration register. All fields of the register are read by a read access.

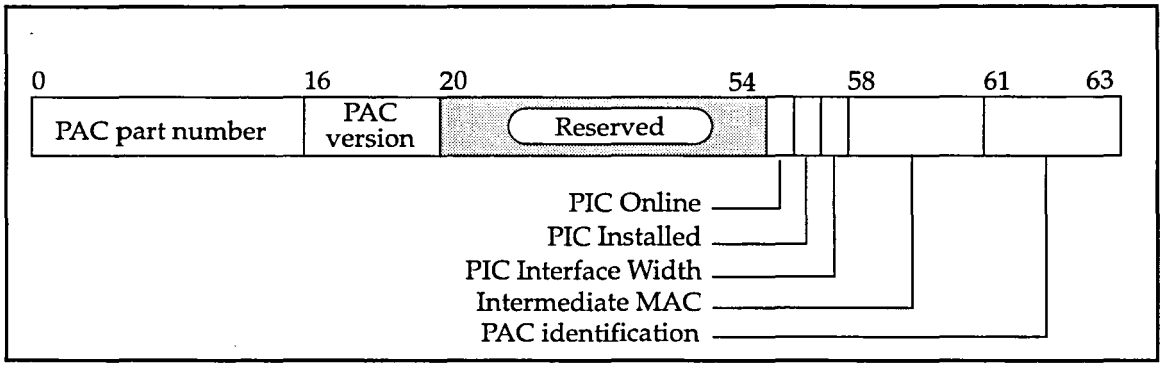


Figure 20 PAC Configuration register definition

The bits and fields in the PAC Chip Configuration register are defined as follows:

- *PAC part number* field (bits 0:15)—Specifies the part number for the PAC chip. A write is ignored and a read returns the hard-wired value.
- *PAC version* field (bits 16:19)—Specifies the version for the PAC chip. A write is ignored and a read returns the hard-wired value.
- *PIC online* bit (bit 55)—Set by software to allow CSR accesses to the PIC ASIC. The bit is cleared by reset.

- *PIC installed* bit (bit 56)—Specifies whether a PIC ASIC is connected to the PAC. A value of one indicates a PIC is installed. This bit is read only.
- *PIC interface width* bit (bit 57)—Specifies whether a 32-bit or 16-bit interface exists between the PIC and PACs. A value of one indicates a 32-bit interface, a value of zero indicates 16-bit. This bit is read only.
- *Intermediate MAC* field (bits 58:60)—Specifies the physical MAC used by the PAC when routing a packet to another PAC. Any MAC installed in the hypernode can be specified and packet routing will function properly.
- *PAC identification* field (bits 61:63)—Specifies the identification number for the physical PAC. The value is obtained from pins on the PAC. A write to this field is ignored and a read access will return the value of the pins.

PAC Processor Configuration register

Each PAC has a Processor Configuration register that contains specific information about the PAC processor. Each processor attached to the PAC can be configured differently.

Figure 21 shows the format of the PAC Processor Configuration register. All fields of the register are read by a read access.

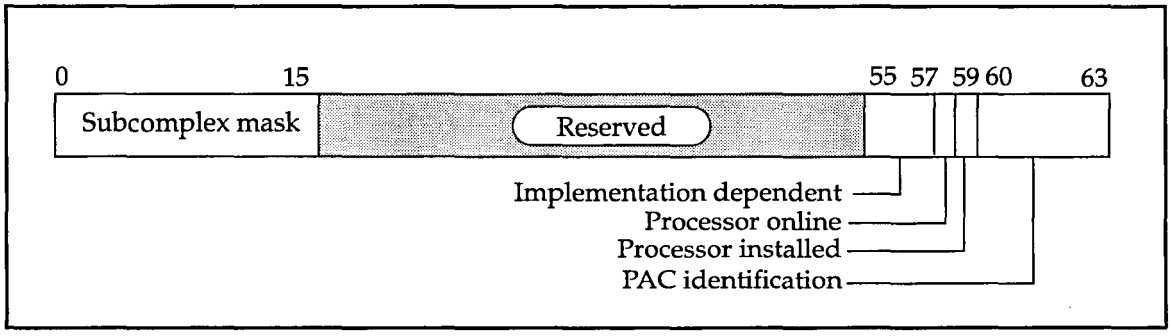


Figure 21 PAC Processor Configuration register definition

The bits and fields in the PAC Processor Configuration register are defined as follows:

- ***Subcomplex mask*** field (bits 0:15)—Specifies which processors within the hypernode belong to the same subcomplex as the processor associated with this configuration register. The mask determines which processors of the local hypernode should receive broadcasted transactions.

Note

If two processors are online on a PAC and either processor is in the subcomplex (as the processor associated with this processor configuration register), the bits for both processors must be set.

- ***Implementation dependent*** field (bits 55:57)—Used by low level implementation dependent software. The value in this field should not be modified during normal operation.
- ***Processor online*** bit (bit 58)—Indicates that the processor is assessable. The value is initialized to the value of the Processor Installed bit.
- ***Processor installed*** bit (bit 59)—Indicates that the processor is installed. The value of this bit comes directly from a pin on the PAC. Writes to this bit are ignored; a read access will return the value of the input pin.
- ***Processor identification*** field (bits 60:63)—Specifies the identification number for the physical processor. The value read is obtained by concatenating the three PAC ID pins and bit 27 of the 40-bit address used to read the register. A write to this field is ignored and a read access will return the value of the pins/address bit.

PAC Memory Board Configuration register

Each PAC has a Memory Board Configuration that specifies the memory block to memory board mapping. Figure 22 shows the format of the register. All fields are written by a write access and read by a read access. Reset has no effect. Writes to reserved bits are ignored and reads to reserved bits return the value zero.

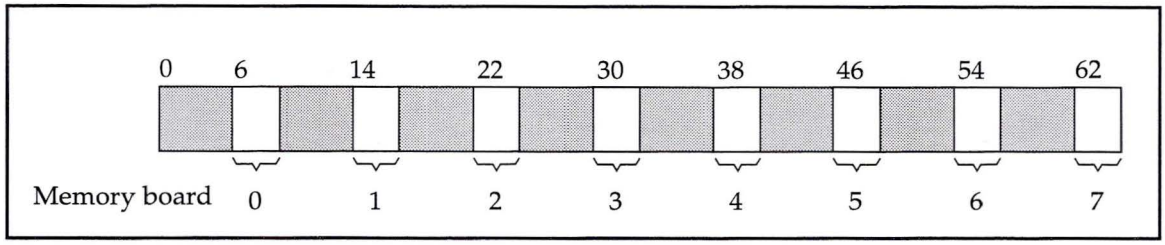


Figure 22 PAC Memory Board Configuration register definition

The three-bit memory block generated by the memory block interleave generation logic indexes into one of the eight memory board fields of the Memory Board Configuration register.

The bits and fields in the Memory Board Configuration register are defined as follows:

- ***Memory board*** fields—Specify the most significant two bits of the physical memory board. The least significant bit of the memory block index is the least significant bit of the physical memory board. This forces even memory blocks to be mapped to even memory boards and odd memory blocks to odd memory boards.

MAC Configuration register

Each MAC has a Configuration register that contains specific information. Each MAC can be configured differently.

Figure 23 shows the format of the register. All fields of the register are read by a read access.

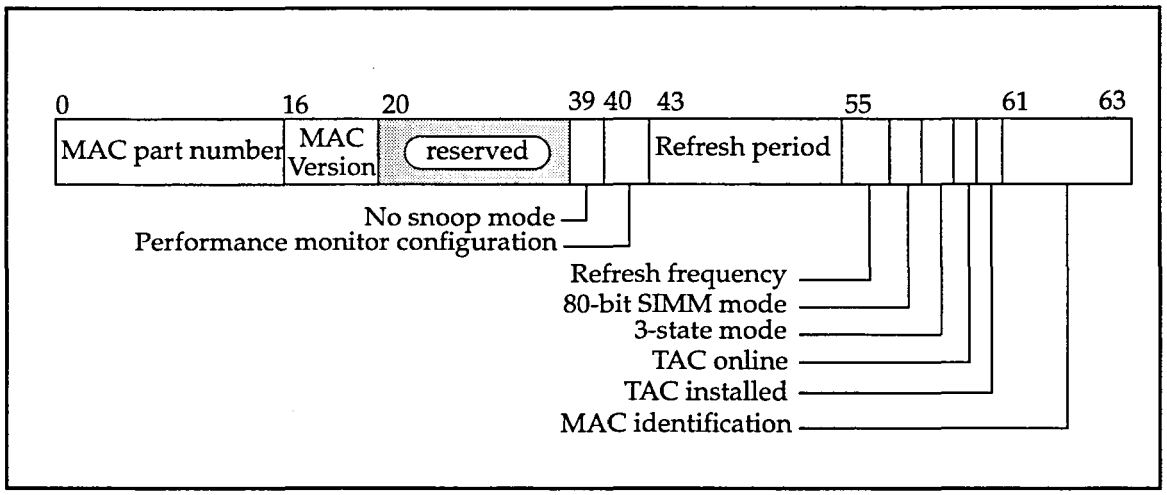


Figure 23 MAC Configuration register definition

The bits and fields in the MAC Configuration register are defined as follows:

- ***MAC part number*** field (bits 0:15)—Specifies the part number for the MAC chip. A write is ignored and a read returns the hard wired value.
- ***MAC version*** field (bits 16:19)—Specifies the version for the MAC chip. A write is ignored and a read returns the hard wired value.
- ***No snoop mode*** bit (bit 39)—Indicates that the snoop mode is set. The no snoop mode improves performance for cases where a processor owns a line, does a local castout, and re-requests the line.
- ***Performance monitor configuration*** field (bits 40:42)—Controls the output of performance information on the 2-bit PM field of certain crossbar packets.
- ***Refresh period*** field (bits 43:54)—Indicates how often refresh occurs. For SDRAMs that need to be refreshed every 15.6us, this value should be set to 0x3a8, which is 936 half clocks.

- *Refresh frequency* field (bits 55:56)—Controls the operation of refresh. The value of this field after reset is 3.
- *80-bit SIMM mode* bit (bit 57)—Forces the EMAC to operate with SDRAM SIMMs that are 80 bits wide for S-Class servers.
- *3-state mode* bit (bit 58)—Forces the MAC to return data with shared permission when a load request arrives at memory and the data is not encached. Returning it with private permission is the default case.
- *TAC online* bit (bit 59)—Set by software to allow accesses to the TAC ASIC. The bit is cleared by reset.
- *TAC installed* bit (bit 60)—Specifies whether a TAC is connected to the MAC. A value of one implies a TAC is installed.
- *MAC identification* field (bits 61:63)—Specifies the identification number for the physical MAC. The value is written by software.

MAC Memory Row Configuration register

Associated with each MAC are four banks of memory, each bank having up to eight rows of SDRAMs. A table maps the row specified in the physical address to a physically installed row of SDRAMs. The four banks of memory controlled by a MAC have the same memory row mapping.

Each MAC has a register that specifies the memory row mapping. The format of the register is shown in Figure 24. All fields are written by a write access and read by a read access. Reset has no effect. Writes to reserved bits are ignored and reads to reserved bits return the value zero.

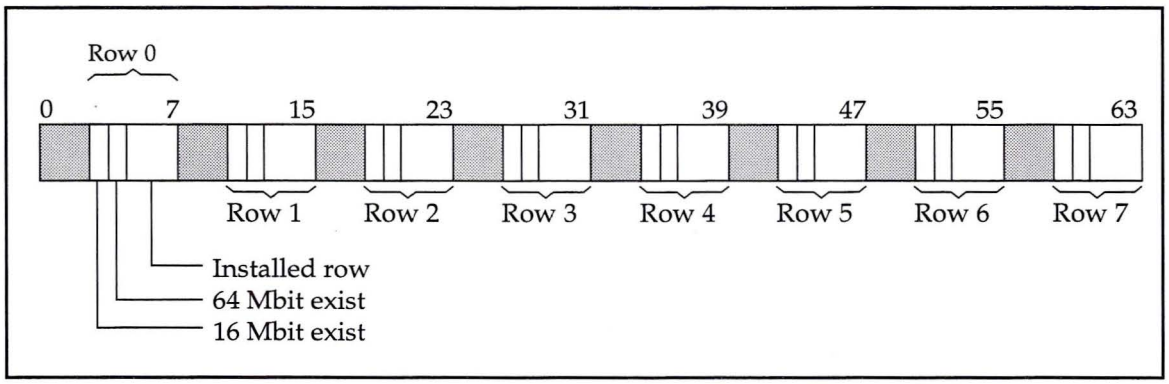


Figure 24 Memory Row Configuration register definition

The 3-bit *row* field of a physical address indexes one of the eight sets of *row* fields of a Memory Row configuration register.

The bits and fields in the MAC Memory Row Configuration Register are defined as follows:

- ***16 Mbit exist*** bits—Indicate a SIMM with 16-Mbit SDRAMs exists for the row. The field checks memory existence access.
- ***64 Mbit exist*** bits—Indicate a SIMM with 64-Mbit SDRAMs exists for the row. The field checks memory existence access.
- ***Installed row*** fields—Map the row specified by the physical address to the physical memory row.

MAC Memory Region registers

There are three CSRs on each MAC that define a region of memory. The three regions of memory are:

- Unprotected memory
- Normal CTI cache
- Unprotected CTI cache

All fields of the register are read by a read access and written by a write access.

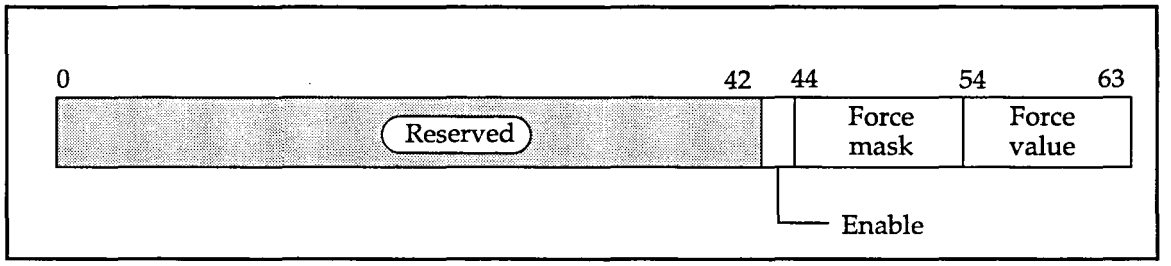


Figure 25 Memory Region register definition

The bits and fields in the MAC Memory Region register are defined as follows:

- ***Enable*** bit (bit 43)—Enables access to the memory region which the register specifies. Reset clears the bit.
- ***Force mask*** (bits 44:53) and ***Force value*** fields (bits 54:63)—Specify a region of memory. The mask field specifies which of the ten bits are set, and the value field specifies the value to set. The 10 bits of the physical address used for these fields are the three-bit row field and seven most-significant bits of the page field. The most-significant three bits of the

10-bit force fields apply to the row field, and the least significant seven bits apply to the page field.

Unprotected Memory register

The fields of the Unprotected Memory Region register check that requests from other hypernodes have access permission to the address of the request.

Normal CTI Cache Memory Region register

This memory region register has two purposes:

- It generates an address to access the normal CTI cache.
- It prohibits accessing the normal CTI cache memory region as local hypernode memory.

When the physical address accesses a remote hypernode, the normal CTI cache is accessed by setting bits of the physical address as specified in the force fields.

When the physical address references local memory, the address must be checked to ensure that it is not accessing the normal CTI cache memory region.

Unprotected CTI Cache Memory Region register

This memory region register has two purposes:

- It generates an address to access the unprotected CTI cache.
- It prohibits accessing the unprotected CTI cache memory region as local hypernode memory.

When the physical address references local memory, the address must be checked to ensure that it is not attempting to access the unprotected CTI cache memory region.

Memory region access checking summary

Table 13 summarizes the memory region access violation checks performed by the MAC. The memory types are defined as follows:

- Normal Memory—All existing memory not covered by any of the three memory region registers.
- Unprotected Memory—Memory covered by the Unprotected Memory Region register.
- Normal CTI Cache—Memory covered by the Normal CTI Cache Memory Region register.
- Unprotected CTI Cache—Memory covered by the Unprotected CTI Cache Memory Region register.

- Nonexistent Memory—All coherent memory address space not covered by the installed bits of the Memory Row Configuration register.

Table 13 Memory region access checking summary

Access type	Memory type				
	Main memory		CTI cache		Nonexistent memory
	Normal	Unprotected	Normal	Unprotected	
Normal memory	OK	OK	Error	Error	Error
Remote memory	Error	Error	OK	OK	Error
Diagnostic	OK	OK	OK	OK	Error
CTI Cache Flush Entry	Error	Error	OK	OK	Error

TAC Configuration register

Each TAC has one TAC Configuration register that specifies information that can be different for each TAC. The format of the TAC Chip Configuration register is shown in Figure 26. All fields of the register are read by a read access.

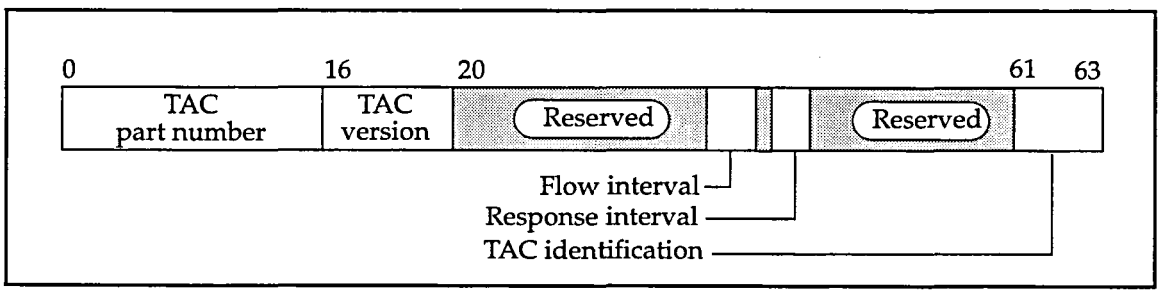


Figure 26 TAC Configuration register definition

The bits and fields in the TAC Configuration register are defined as follows:

- *TAC part number* field (bits 0:15)—Specifies the part number for the TAC chip. A write is ignored and a read returns the hard wired value.
- *TAC version* field (bits 16:19)—Specifies the version for the TAC chip. A write is ignored and a read returns the hard wired value.

- *Flow interval* field (bits 41:43)—Specifies the time-out interval for ERI flows.
- *Response interval* field (bits 45:47)—Specifies the timeout interval for CTI responses.
- *TAC identification* field (bits 61:63)—Specifies the identification number for the physical TAC. The value is written by software.

Cache management 3

Caches compensate for local and remote (in the case of most X-Class servers) memory latency. Caches contain frequently accessed memory, either data or instructions.

Processor cache management

Each processor in the S-Class and X-Class server has a data cache and an instruction cache. Copying information from memory into a cache is referred to as *move in*. Whenever a memory item (data or instruction) is moved into the cache, the processor can modify it there. If another processor references the same item while the copy is still in the first processor cache, the original item and the copy must be identical to maintain coherency.

The instruction cache and the data cache are one MByte in size each.

A move-in can cause more than a single cache line of data to be imported into the caches. All the cache lines of the referenced page can be imported for data references. For instruction references, all the cache lines in the referencing page and the following page (virtual or physical) can be imported. A flush cache, purge cache, or purge TLB instruction stops any subsequent move-in operations to that page until another reference is made.

Each processor supports speculative execution of code. Speculative execution is enabled for virtual address translation. It allows data cache move-in of any coherent memory line, provided that the virtual-to-physical address translation of the memory line is in the processor translation lookaside buffer (TLB). Also, the virtual-to-physical translation bit in the processor status word must be set and the TLB entry must have the U-bit (uncacheable bit) cleared.

The size of both the instruction and data cache lines is 32 bytes.

PA-8000 caches

The PA-8000 has both instruction and data caches. These caches are flushed by one of two methods:

- Specifying a memory line address
- Specifying the cache entry to flush

All processor cache operations are issued with a single PA-8000 processor instruction. These include:

- Flush Data Cache (FDC)
- Purge Data Cache (PDC)
- Flush Data Cache Entry (FDCE)
- Flush Instruction Cache (FIC)
- Flush Instruction Cache Entry (FICE)

The FDC and PDC instructions have identical functionality. Both operations flush cache lines from all processors on the local hypernode and from the node CTI cache. FDC and PDC instructions write data from dirty cache lines back to memory.

The FDCE and FICE instructions flush a cache entry independently of the tag value. They flush an entry from the executing processor caches only. If the cache line in the data cache is dirty, it will be written back to local memory or the CTI cache. The CTI cache, however, is not flushed.

Note

Follow all cache flush instructions by a `sync` instruction to ensure that all flushes have been written to memory.

Cache coherence between processors

Cache coherence causes the S-Class and X-Class servers to behave as if they had a single data cache and a single instruction cache (logically). Since there are many processors and, therefore, multiple data caches, each processor must cross-interrogate for current data and broadcast purges and flushes (except for FDCE and FICE). Purges and flushes do not cause TLB faults on other processors.

All coherent data references are satisfied using cache coherence checks. These checks ensure that the data has remained coherent since it was moved in. Cache coherence checks are performed on write buffers in order to ensure proper ordering of storage accesses.

Address aliasing

Through the correct management of TLBs and page directories (PDIR), the operating system ensures that a virtual address does not translate to two different physical addresses (see the *PA-RISC 2.0 Architecture* manual and the *Exemplar Programming Guide*).

Several virtual addresses often map to the same physical memory. The processor caches permit a physical memory location to be accessed by both a physical address and a virtual address where the two are identical. Such a virtual address is said to be equivalently mapped. Equivalently mapped virtual and physical references present a consistent view of memory.

Two or more distinct virtual addresses mapped to the same physical page are said to be *virtual aliases*. Virtual aliases do not always provide a consistent view of memory, depending on the difference in their virtual addresses (both the offset and space portions). In order for two mappings to present a consistent view, they must index the same cache line.

To provide a consistent mapping of virtual aliases, only address bits 0, 1, 2, 3, 8, 9, 10, 11, 16, 17, 18, and 19 can be used for space handling.

The operating system flushes the cache lines when:

- The address mapping in the TLBs changes.
- A physical access is made to a location that might reside in the caches as a result of a virtual access that was not equivalently mapped.
- A virtual access is made to a location that might reside in the caches as a result of a physical access that was not equivalently mapped.
- A virtual access is made to a location that might reside in the caches as a result of another virtual access that differs in bits from those listed above.

Store ordering

It is possible for stores and cache flushes to complete out of order. This is referred to as *weak-store ordering*. With weak ordering, the synchronization instruction ensures that pending stores and cache flushes have been completed. All flushes must be followed by the `cache sync` instruction to wait for completion. X-Class servers have two CSRs that set and clear weak ordering enables. See the "TAC weak ordering CSRs" section on page 64.

CTI cache maintenance

In X-Class servers, each hypernode maintains a cache of memory references sent to the CTI from other hypernodes. Also, any data that moves from one processor cache to another processor cache on the same hypernode (staying encached in the original processor cache) is encached in the CTI cache. Consequently, the CTI cache directory information locates any global data currently encached by the hypernode.

The CTI cache line size is 32 bytes. The CTI cache is physically indexed and tagged with the global physical address. Because the cache is physically indexed, there are no aliasing requirements.

The X-Class server ensures cache coherence between multiple hypernodes, that is, two or more hypernodes that map the same global address will get a consistent view. The system maintains a linked sharing list that contains a list of all the hypernodes sharing each cache line or the hypernode that exclusively owns the cache line. Within every hypernode a record is kept of which processors have encached each line in the CTI cache so that CTI coherency requests can be forwarded to the appropriate processors.

The system flushes the entire CTI cache by executing an implementation-dependent loop containing FDC instructions These instructions reference the physical addresses implementing the cache. A specially designated address must be used so that the CTI cache control logic flushes the line, regardless of whether the physical page number and virtual index tags match the reference made by the processor. This implies that the physical addresses devoted to the CTI cache must be addressable by the processor, even though in normal circumstances load or store instructions are not issued on these addresses.

CTI cache operators

There are four operators for CTI cache management unique to the Exemplar product line. These operators are:

- CTI Cache Global Flush
- CTI Cache Flush Entry
- CTI Cache Prefetch for Read
- CTI Cache Prefetch for Write

These four operators manipulate the contents of the CTI cache without affecting the contents of the processor caches.

CTI Cache Global Flush

This operator flushes the memory line for a specific address out of all CTI caches in the entire system. Once completed, the memory line is in memory and is not encached in any data cache or CTI cache.

CTI Cache Flush Entry

This operator performs a CTI cache flush entry operation when issued with a physical address that is mapped to a CTI cache memory region. The flush entry operation is performed only on the referenced CTI cache.

CTI Cache Prefetch for Read

This operator accelerates a memory line into the CTI cache of the local hypernode. The CTI cache contains the memory line for read access (not write accessible) after the operation is complete.

CTI Cache Prefetch for Write

This operator accelerates a memory line to the CTI cache of the local hypernode. The CTI cache contains the memory line for read and write access after the operation is complete.

CTI cache interfaces

The PA-RISC architecture does not support the concept of a CTI cache. Therefore, there are no PA-RISC instructions for issuing the CTI cache operations. The X-Class server provides two interfaces to these operations using non PA-RISC architected mechanisms.

User applications can not directly use these mechanisms. This ensures compatibility with previous and future generation platforms that may implement the interface mechanisms differently. Architectural Interface Library (AIL) routines that are consistent on other Exemplar systems, however, provide access to these mechanisms.

Methods for issuing CTI cache operations

There are two methods for issuing CTI cache operations in X-Class servers:

- Using a single PA-8000 instruction.
- Using CSR read and write operations.

Instruction method

There are three CTI cache instructions:

- CTI Cache Global Flush
- CTI Cache Prefetch Read
- CTI Cache Prefetch Write

These instructions are detailed in Appendix B.

These instruction pass a CTI cache operation directly to the PAC. Fields in the instructions specify the particular operation and specify whether read and/or write protection checking should be performed.

User applications can also use these instructions. However, the result will be nonportable code.

CSR method for CTI cache operations

Another method of issuing the CTI cache operations exists that uses PAC CSRs. This method is a lower bandwidth approach but does not require any non-architected support from the PA-8000 processor.

PAC CSRs include the following:

- Processor *n* CTI Cache Flush Global address
- Processor *n* CTI Cache Prefetch Read address
- Processor *n* CTI Cache Prefetch Write address
- Operation Context register
- Operation Address register

The first three registers exist for the three CTI caches operations, where *n* is either 0 or 1.

CSR-based cache management operations use only loads and stores to CSR space to issue the operations. The following sequence shows how the operations are implemented:

1. Arm the operation by writing to the PAC Operation Context register Armed bit.
2. Perform virtual-to-physical address translation.

3. Issue the Cache operation with the physical address provided as the data of the write.
4. Check the PAC Operation Context register Armed bit to make sure the operation was issued. If the Triggered bit is not set then the operation must be restarted. The Armed bit will be cleared when the sequence is interrupted by either an external interrupt, or a TLB miss.

As an example, the following sequence of instructions issues a CTI Cache Global Flush:

```
loop
LDI     1,%t1
STD     %t1,(CSR_OP_CNTX) ;Arm CSR Operations
LPA     ncfg_addr,%t2
STD     %t2,(CSR_NCG)     ;Issue NC Global Flush
LDD     (CSR_OP_CNTX),%t4
BB,*>=  %t4,62,loop       ;check if triggered
```

The following sections describe the CSRs used to issue CTI cache management operations.

PAC cache management operation addresses

Writing to the three CTI cache management addresses triggers the respective cache management operation.

If the Armed bit in the PAC Operation Context register is set, writing to one of the addresses results in sending the cache management operation to memory.

The address of the target memory line becomes the data of the double word store instruction that issues the operation. The target memory line address is written to the Operation Address register, the Armed bit is cleared, and the Triggered bit is set. If the Armed bit is not set, the PAC returns an undefined value to the processor, and the Armed and Triggered bits are not modified.

PAC Operation Context register

The PAC has two Operation Context registers, one for each processor. These registers are used for synchronization (see Chapter 5), as well as for CTI cache operations.

The operation context is applied to other CSRs in two ways. One is by *arming* a CSR and the other is by indicating that the armed CSR was *triggered*, that is, it performed a specific function.

The format of the PAC Operation Context register is shown in Figure 27.

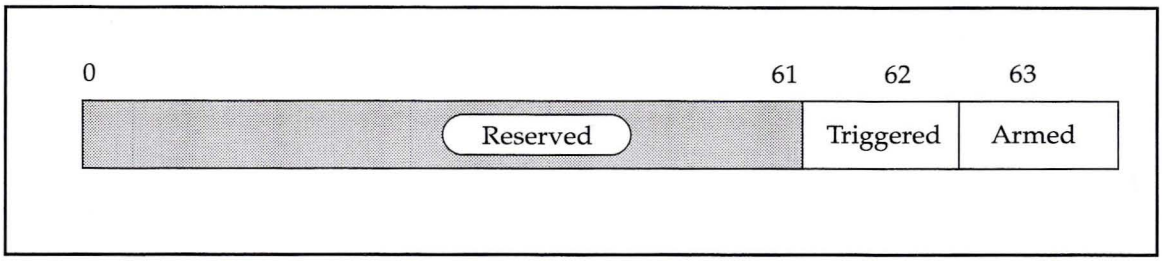


Figure 27 PAC Operation Context register definition

The fields of the Operation Context registers are defined as follows:

- ***Armed*** bit (bit 63)— Set by software to arm specific PAC processor CSRs. The following PAC CSRs are armed by this field:
  - Data Mover Input Command register
  - Fetch and Increment address
  - Fetch and Decrement address
  - Fetch and Clear address
  - Noncoherent Read address
  - Noncoherent Write address
  - Coherent Increment address
  - CTI Cache Global Flush address
  - CTI Cache Prefetch for Read address
  - CTI Cache Prefetch for Write address
- ***Triggered*** bit (bit 62)—Indicates that a CSR operation was issued when the Armed bit was set.

Table 14 shows how the Armed and Triggered bits work together.

Table 14 CSR Operation Context register state transition when operation is issued

Present value		Next value	
Triggered	**Armed**	**Triggered**	**Armed**
0	0	0	0
0	1	1	0
1	0	0	0
1	1	1	1

The operating system manipulates the Armed and Triggered bits when the process state is saved or restored. When the Armed bit is set and the Triggered bit is cleared, then both bits are restored as cleared. Otherwise, the bits are restored unmodified.

Context State Save/Restore

The PAC Operation Context register Armed and Triggered mechanism protects an operation from being corrupted by a processor interrupt (external, DTLB miss, etc.) that disrupts the sequence. The interrupt handlers examine the PAC Operation Context register and take appropriate action. The following situations can occur:

- No operation being setup (Armed bit cleared)—An operation setup sequence was not interrupted. No modification of the bits is necessary.
- Operation is being setup (Armed bit set) — An operation setup sequence was interrupted. The Armed and Triggered bits must be cleared to abort the sequence and force a retry. The Armed bit is cleared to prohibit an operation from starting. The Triggered bit is cleared to notify software that the operation did not complete.

If the interrupted thread is to be context switched, then the modified Armed and Triggered bits must be saved and later restored when the thread resumes. No other CSR context must be saved or restored for the CSR method of issuing operations to function properly.

PAC Operation Address register

Each PAC has two Operation Address registers, one for each processor. The register stores the address used for CSR operations.

The format of the PAC CSR Operation Address register is shown in Figure 28. The field of the register is read by a read access.

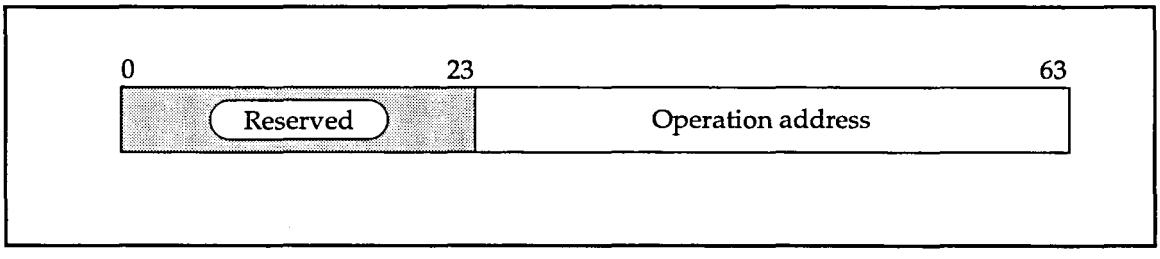


Figure 28 PAC CSR Operation Address register definition

The *Operation address* field (bits 24:63) designates the address for CSR operations. It also designates one of the following addresses:

- Coherent Increment address
- CTI Cache Global Flush address
- CTI Cache Prefetch for Read address
- CTI Cache Prefetch for Write address

Addresses that reference the first 16 MBytes of physical memory on hypernode zero are forced to reference the local hypernode. To perform this mapping, the hardware compares the Row, Virtual Ring, Virtual Bank, and most significant two bits of the Page fields for the value zero. When these fields are all zero, then the Node ID of the address is forced to that of the local hypernode.

TAC weak ordering CSRs

Two TAC CSRs enable inter-hypernode weak ordering:

- Set Weak Order Enable register
- Clear Weak Order Enable register

These CSRs are described in the following sections.

TAC Set Weak Order Enable register

Each TAC has one Set Weak Order Enable register. The register enables weak order mode for inter-hypernode coherency requests. Figure 29 shows the format for the register.

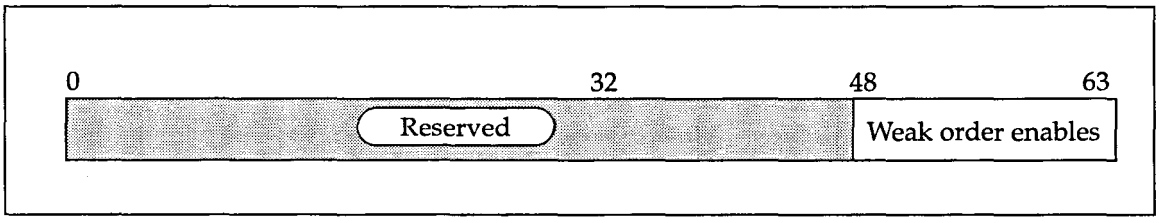


Figure 29 TAC Set Weak Order Enable

The *Weak order enables* field (bits 48:63) specifies which processors are enabled for weak ordered inter-hypernode requests. The data written to the register is OR'ed into the existing register contents. The 16 bits of the field correspond to the 16 processors within the hypernode with bit 48 corresponding to processor 0 and bit 63 corresponding to processor 15.

TAC Clear Weak Order Enable register

Each TAC has one Clear Weak Order Enable CSR. The register disables weak order mode for inter-hypernode coherency requests. Figure 30 shows the format for the register.

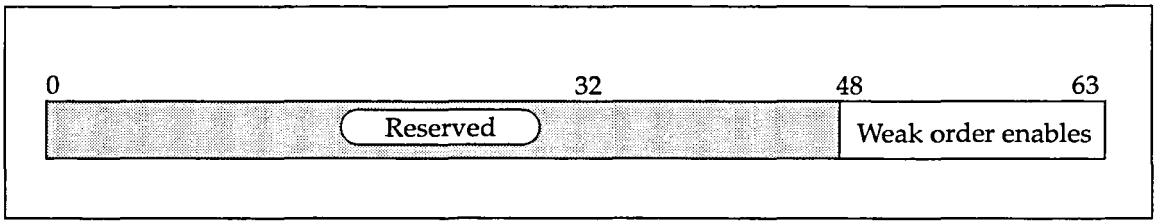


Figure 30 TAC Clear Weak Order Enable

The ***Weak order enables*** field (bits 48:63) specifies which processors are enabled for weak ordered inter-hypernode requests. The data written to the register is a clear mask. A bit set to one value clears the corresponding bit in the register. A bit set to zero does not modify the existing register bit. The 16 bits of the field correspond to the 16 processors within the hypernode with bit 48 corresponding to processor 0 and bit 63 corresponding to processor 15.

Data mover

4

Overview

In order for S-Class and X-Class servers to process separate threads, they must send process context messages from one thread in a process to another within the same or different process. The messages and data shared between threads are transferred between processors by way of common memory (globally shared memory).

The PAC contains a hardware section called the data mover that routes these messages and copies data between memory.

Message transfers

For message transfers, the message resides in memory on the source hypernode before the transfer and then in the memory of the destination hypernode after the transfer. The processor knows the address of the message in the source hypernode, but not in the destination hypernode. A MAC on the destination hypernode determines the destination address for the transfer. Source and destination addresses can be specified with either a virtual address or a physical address.

Data copy

For copy data, the data resides in the memory of both the source and destination hypernode after the transfer. The processor knows both source and destination addresses of the copied data.

The size of the data copy operation can vary, with the source and destination address being specified within a single page or with a list of pages.

Data mover features

The following list highlights the data mover hardware functions:

- Messaging and data copy can be to and from both interleaved and noninterleaved memory.
- Each processor has an independent interface for copy transfers.
- The initiating processor notifies the destination hypernode of impending messaging operations with queued completion status.
- The initiating processor receives a confirmation interrupt at the completion of the transfer.
- Copy operations can be from virtual-to-virtual, virtual-to-physical, physical-to-virtual, or physical-to-physical space.
- Messages can be performed from virtual or physical space.
- The source or destination space can be specified as a single page (where the page is up to four Mbytes in size), or with a Block Translation Table (BTT) with each entry mapping a four-Kbyte page.

Data mover implementation

S-Class and X-Class servers use CSRs in the PACs and MACs and memory addresses along with the data mover to transfer messages and data. These CSRs and addresses include the following:

- PAC Operation Context register
- PAC Input Command register
- PAC Source and Destination Physical Page Frame registers
- PAC Source and Destination Offset registers
- PAC Operation Status register
- MAC Message Reception Area Configuration registers
- MAC Message Reception Area Offset registers
- MAC Message Completion Queue Configuration registers
- MAC Message Completion Queue Offset registers
- MAC Memory Allocation address
- MAC Message Completion Enqueue address
- MAC Message Completion Dequeue address

The implementation uses memory structures for message reception, message completion queues, and the BTT for I/O data copy transfers. Memory structures are preallocated regions of memory. The actual data resides in memory before and after transfer.

Functional overview

Figure 31 shows a functional diagram of the messaging and data copy transfers.

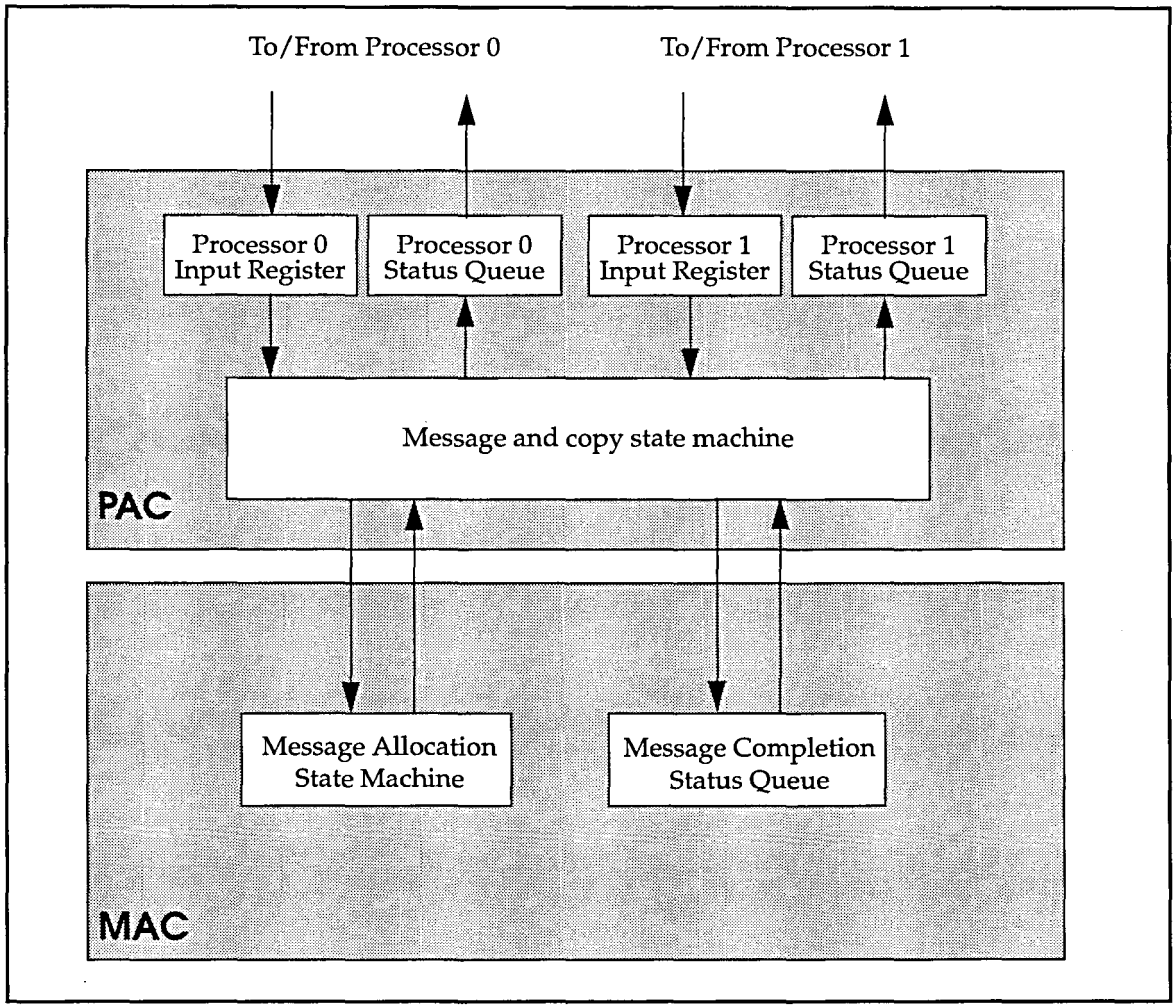


Figure 31 Messaging and data copy transfers implementation

Input registers

Each PAC has two input registers for its two processors. To initiate a messaging or data copying operation, a processor first determines if the input registers are available. All transfer operations remain in the input register until completed. Once the

input registers are available, a processor initiates an operation by programming the input registers.

Message and copy state machine

The message and copy state machine starts executing an input operation when a set of input registers has been set up and the message and copy state machine is idle. If both sets of input registers have operations ready to execute, the hardware arbitrates between the two sets of input registers to guarantee forward progress.

The state machine achieves the transfer operation by executing the following three phases:

- It determines the destination address for message operations. If the current operation is a copy operation, this phase of execution is skipped. The destination address is determined by sending a transaction to a MAC on the destination hypernode. The MAC performs a memory allocation operation and responds with a destination hypernode memory address.
- It copies data from the source memory to the destination memory. The copy operation executes until complete or until either a TLB purge or error occurs. For more information on TLBs, see the *PA-RISC 2.0 Architecture* manual.
- It sends a message completion transaction to the MAC on the destination hypernode. This phase is not performed if the operation is a data copy. The MAC enqueues the completion status in a memory-based queue and informs a processor on the destination hypernode via an interrupt.

Operation status queues

The two processors connected to the PAC each have an operation status queue. The state machine places the message and copy operation completion status in one of the appropriate status queues. Each status queue is three entries deep to provide status space for the input register and message and copy state machine stages.

Once status is enqueued, an interrupt is sent to the processor that initiated the operation.

Messaging and data copy CSRs

CSRs in both the PACs and MACs control messaging and data copy. This section specifies the addresses used to access each CSR of the messaging and data copy hardware.

PAC Operation Context register

Each PAC has two Operation Context registers, one for each processor. The operation context is applied to other CSRs in two ways. One is by arming a register and the other is by indicating that the armed register was triggered, that is, it performed a specific function. Figure 32 shows the format of the PAC CSR Operation Context register.

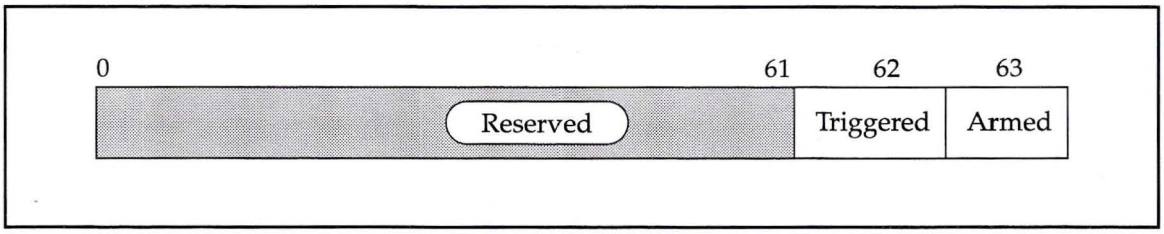


Figure 32 PAC CSR Operation Context register definition

The bits of the CSR Operation Context register are defined as follows:

- ***Armed*** bit (bit 63)—Set by software to arm the functionality of specific PAC processor CSRs. The PAC CSRs armed by this bit are:
  - Data Mover Input Command register
  - Fetch and Increment address
  - Fetch and Decrement address
  - Fetch and Clear address
  - Noncoherent Read address
  - Noncoherent Write address
  - Coherent Increment address
  - Network Cache Global Flush address
  - Network Cache Prefetch for Read address
  - Network Cache Prefetch for Write address

  Each of these CSRs and the effects on them by the Armed bit are discussed later in this chapter. The Armed bit is set by software and is cleared by either hardware or software.

- *Triggered* bit (bit 62)—Indicates that a CSR operation executed when the *Armed* bit was set. The Triggered bit is cleared by software and is set by hardware.

Table 15 shows the Armed and Triggered bit transitions that the hardware controls when software writes to one of the operation addresses.

Table 15 CSR Operation Context register transitions when the operation is issued

Present value		Next value	
Triggered	Armed	Triggered	Armed
0	0	0	0
0	1	1	0
1	0	0	0
1	1	1	1

Table 16 shows the *Armed* and *Triggered* bit transitions that hardware controls when a TLB invalidate transaction is detected.

Table 16 CSR Operation Context register transitions with TLB invalidate

Present value		Next value	
Triggered	Armed	Triggered	Armed
0	0	0	0
0	1	0	0
1	0	1	0
1	1	1	1

PAC Input Command register

Each PAC has two input command registers that set the modes and lengths of messaging and data copy operations.

The Input Command register can be written when the Ready bit of the CSR is zero and the CSR Operation Context register Armed bit is a one. There are no restrictions for reading this register.

The format of the Input Command register is shown in Figure 33.

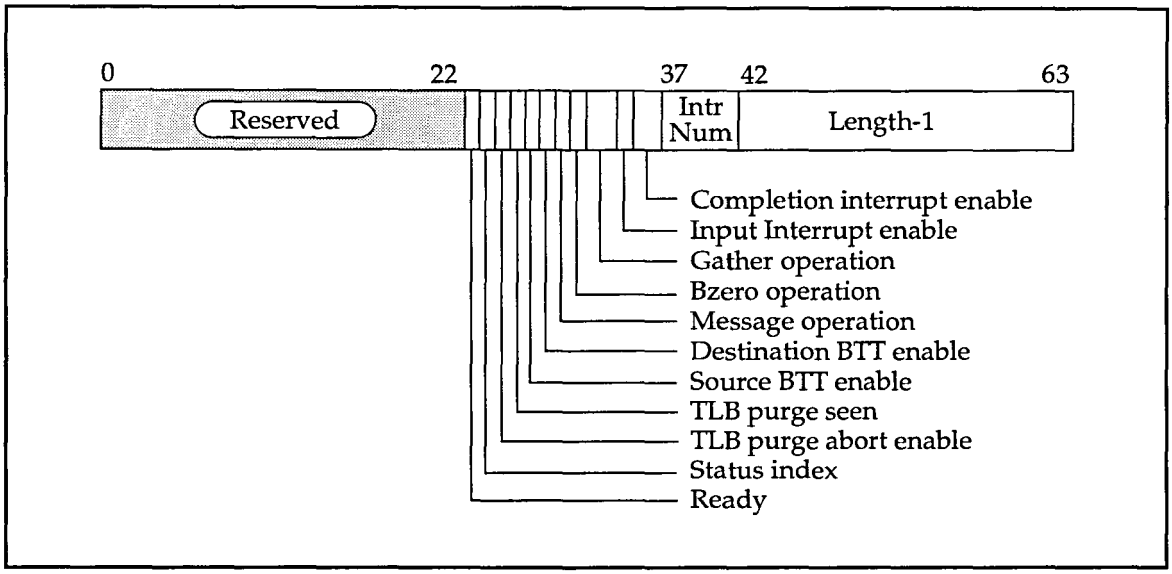


Figure 33 PAC Input Command register format

The bits and fields of the Input Command register are defined as follows:

- ***Ready*** bit (bit 23)—Indicates that the input registers are ready to perform an operation. Normally, this bit is set by software and cleared by hardware. It should be set by software when the input registers are completely set up for an operation. Hardware clears it when the messaging and copy state machine has accessed all required information from the input registers for the operation. The Ready bit is written by a CSR write access. A CSR read will read the current value. Reset clears the bit.
- ***Status index*** field (bits 24:25)—Indicates part of the status in the Operation Completion status queue. Reset clears the field.
- ***TLB purge abort enable*** bit (bit 26)—Enables an operation to be aborted if a TLB purge transaction is detected prior to or during the operation. In system operation, software sets and clears the bit. The operation aborts prior to starting if the TLB Purge Seen and TLB Purge Abort Enable bits are set at the time the messaging and copy state machine starts the operation. Completion status for an aborted operation is written to the appropriate status queue. The TLB Purge Abort Enable bit is written by a CSR write access and read by a CSR read. Reset clears the bit.

- *TLB Purge seen* bit (bit 27)—Indicates that a TLB purge transaction was detected by a PAC. The bit is cleared by software and set by hardware. It is written by a CSR write. A CSR read will read the current value. Reset clears the bit.
- *Source BTT enable* bit (bit 28)—Indicates the Source Physical Page Frame register contains the address of the BTT used for accessing the source memory region of the operation. The bit is written by a CSR write and read by a CSR read.
- *Destination BTT enable* bit (bit 29)—Indicates the Destination Physical Page Frame register contains the address of the BTT used for accessing the destination memory region of the operation. The bit is written by a CSR write and read by a CSR read.
- *Messaging operation* bit (bit 30)—Forces the messaging and copy state machine to use the messaging mechanism to determine the destination address rather than the destination address of the input register. The bit is written by a CSR write and read by a CSR read.
- *Bzero operation* bit (bit 31)—Forces the messaging and copy state machine to clear the destination memory region rather than copy the source to destination memory region. The bit is written by a CSR write and read by a CSR read.
- *Gather operation* field (bits 32:33)—Specifies the stride used for a gather operation. Currently, this field is disabled and set to zero.
- *Input interrupt enable* bit (bit 34)—Enables an interrupt to the associated processor when the Input Command Register is available for reprogramming by software. The most significant five bits of the interrupt number that is sent is specified by this field. The least significant bit of the interrupt number sent is zero. The bit is written by a CSR write and read by a CSR read.
- *Completion interrupt enable* field (bits 35:36)—Enables an interrupt to the associated processor when the messaging and copy state machine completes the operation. The field also determines whether an interrupt is sent when the operation completes with an error if it is sent independently from the status of the operation. The most significant five bits of the interrupt number are specified by this field. The least significant bit of the interrupt number is one. The field is written by a CSR write and read by a CSR read.
- *Interrupt number* field (bits 37:41)—Specifies the most significant five bits of the interrupt numbers to be sent to the processor that initiated the request. An interrupt is sent when either of two events occur:

- When the messaging and copy state machine has completed accessing the input registers.
- When the messaging and copy state machine completes the operation.

The least significant bit of the interrupt number is a zero for the first event and a one for the second. The bit is written by a CSR write and read by a CSR read.

- ***Length-1*** field (bits 42:63)—Specifies the length of the messaging and copy operation. Messaging operations ignore the least significant 5 bits, forcing the length to be an integer number of memory lines (32-byte increments). Copies, however, can be any byte length. A value of zero in the field copies one byte (one memory line for messaging), and a value of all ones in the field will clear four megabytes of memory. The bit is written by a CSR write and read by a CSR read.

PAC Source and Destination Physical Page Frame registers

There are two Source/Destination Physical Page Frame registers on each PAC to specify the source and destination of messaging and data copy operations.

The registers can be written to only when the Input Command CSR Ready bit is zero and the CSR Operation Context register Armed bit is a one. The register can be read at any time.

The format of the Physical Page Frame register is shown in Figure 34.

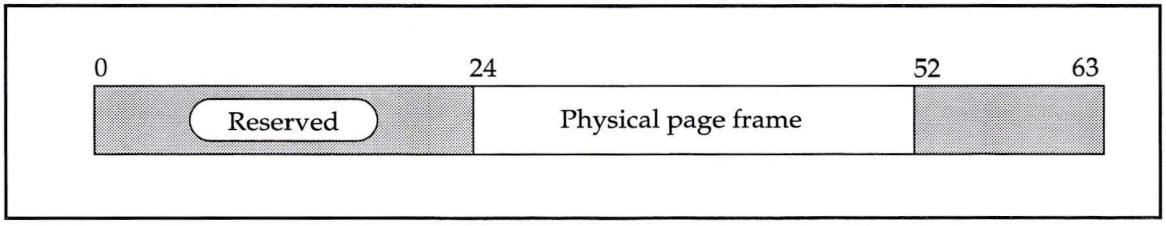


Figure 34 PAC Physical Page Frame register definition

The ***Physical page frame*** field (bits 24:51)—Indicates the physical page frame of a 40-bit PA-8000 address. If a BTT is being used, the field specifies the address of the BTT. Otherwise, the field specifies the source or destination page for the copy operation.

For messaging operations, the Destination Physical Page Frame register must be programmed with the Node ID and Virtual Ring of the destination MAC receiving the message. The Node ID and

VR information are written in the normal physical address field positions.

PAC Source and Destination Offset registers

There are two Source/Destination Offset registers on each PAC to specify the offset for the source and destination of a message or copy operation.

The registers can be written to only when the Input Command CSR Ready bit is zero and the CSR Operation Context register Armed bit is a one. The register can be read at any time.

The format of the Offset register is shown in Figure 35.

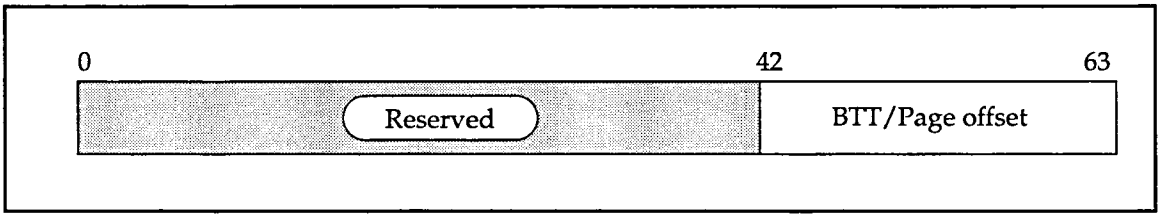


Figure 35 PAC Source/Destination offset register definition

The ***BTT/Page offset*** field (bits 42:63)—Used in one of two ways. When a BTT is being used, the most significant 10 bits specify the index into the BTT and the least significant 12 bits specify the offset into the selected BTE memory page. When a BTT is not being used, the field is used as the offset into a page of memory. With 22-bits, the offset within a page can be up to 4 Megabytes in size for support of larger page sizes.

For messaging operations, the Destination Offset register need not be programmed.

PAC Operation Status Queue register

There are two Operation Status registers on each PAC, one for each of the two processors attached to the PAC. Status can not be inserted in a status queue in the same order the processor sets up the input registers.

The format of the processor 0/1 Status registers is shown in Figure 36.

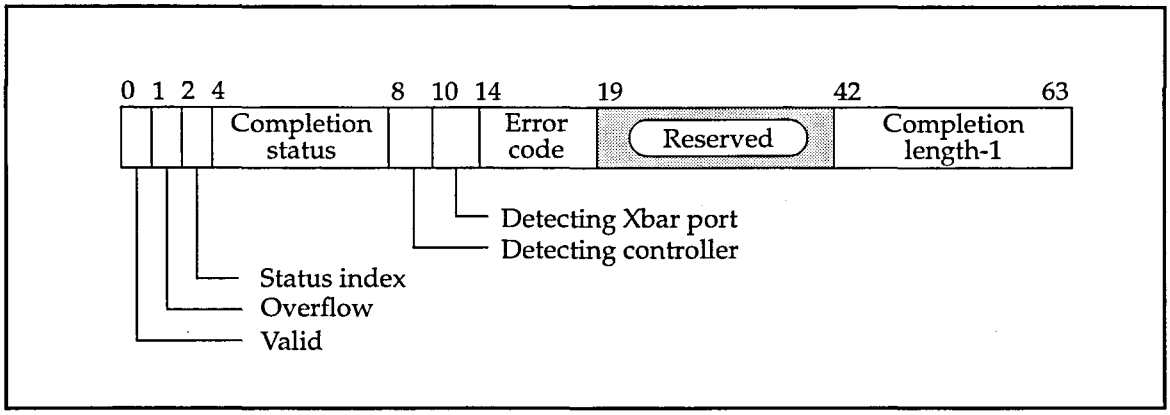


Figure 36 Processor Status Queue register definition

The bits and fields of the Operation Status register are defined as follows:

- ***Valid*** bit (bit 0)—Indicates that the Status Queue has valid messaging and copy state machine completion status. The bit is set when the state machine has completed and writes status into the queue. The bit is cleared when the status is read, and no other valid status remains in the status queue. A CSR read access will read the value, and a CSR write has no effect. Reset clears the bit.
- ***Overflow*** bit (bit 1)—Indicates that a status queue overflow occurred resulting in the loss of status information. The bit is set when a status queue is full and the messaging and copy state machine has completed an operation and its status is destined for that queue. The bit is cleared when the status register is read. A CSR write does not effect the value of the bit. Reset clears the bit.
- ***Status index*** field (bits 2:3)—Indicates the status index. The two bits are a direct copy of Input Command register Status Index field just before the operation was started.
- ***Completion status*** (bits 4:7)—Indicates the messaging and copy state machine completion status.
- ***Detecting chip*** and ***Detecting Xbar port*** fields—Obtained directly from a transaction error response. The fields specify which chip or crossbar port detected the error.
- ***Error code*** field—Specifies the type of error that caused the operation to fail.

- *Completion length-1* field (bits 42:63)—Indicates the amount remaining to copy when the operation finished.The field is only valid if the operation was aborted with the detection of a TLB Purge. The field contains the value of minus one when the operation completed successfully and zero or greater if the operation was aborted. The value restarts an operation when it aborted due to a TLB Purge being detected. A CSR read access reads the value, and a CSR write has no effect.

MAC Message Reception Area Configuration register

There is one Message Reception Area Configuration register on each MAC to specify the base address for the region of memory used to receive messages.

The format of the Configuration registers is shown in Figure 37.

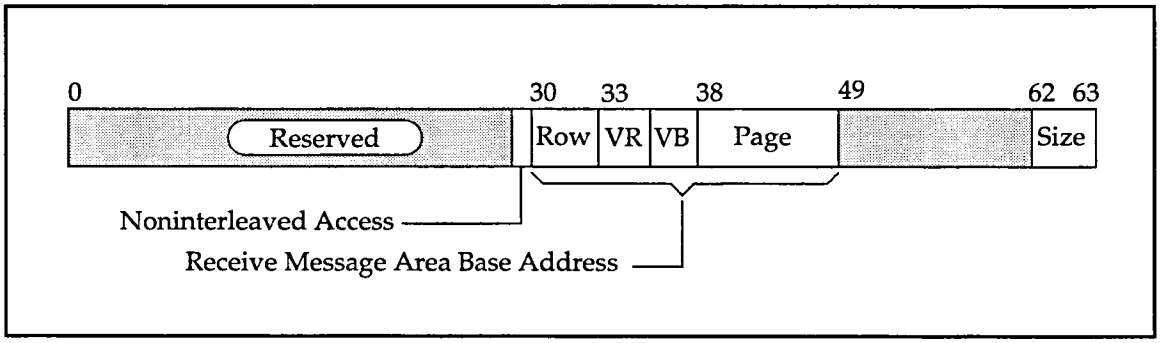


Figure 37 MAC Message Reception Area Configuration register definition

The bits and fields of the Message Reception Area Configuration register are defined as follows:

- ***Noninterleaved Access*** bit (29), ***Row*** (30:32), ***Virtual Ring*** (33:35), ***Virtual Bank*** (36:37), and ***Page*** (38:48) fields specify the Message Reception Area Base Address. The noninterleaved access bit is currently not used in S-Class and X-Class servers.
- *Size* field (bits 62:63)—Specifies the size of the Message Reception Area. Table 17 shows the possible sizes for the Message Reception Area.

Table 17 Message Reception Area size options

Field value	Size
0	32 Kbyte
1	256 Kbyte
2	2 Mbyte
3	16 Mbyte

MAC Message Reception Area Offset registers

There are two Message Reception Area Offset registers on each MAC:

- Message Reception Area Available Offset register—Specifies the region of the message reception area available for new messages.
- Message Reception Area Occupied Offset register—Specifies the region presently occupied by messages.

One register specifies the offset into the message reception area where the received messages start and the other specifies where occupied memory starts.

The format of the Message Reception Area Offset registers is shown in Figure 38.

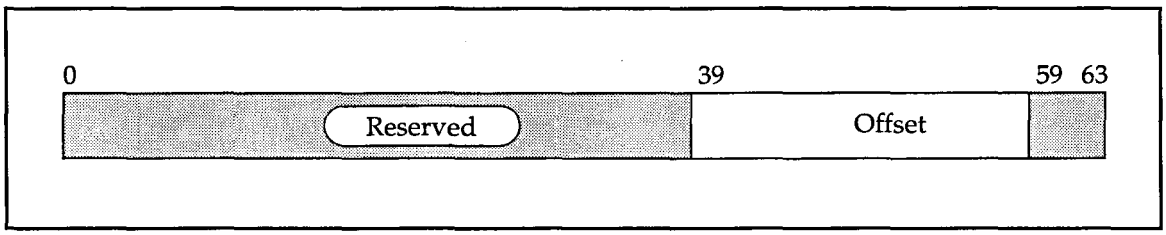


Figure 38 MAC Message Reception Area Offset register definition

The field of the register is defined as follows:

- *Offset* field (bits 39:58)—Specifies an offset into the message reception area. The register is normally read and written by hardware (to allocate space for new messages). It is read by hardware to check if sufficient available area exists for a new message and written by software to free memory consumed by previously received messages.

Depending on the Size field of the register, some of the most significant bits of the offset field are not used and must be set to

zero when written by software. Table 18 shows the bits for each possible size of the message reception area.

Table 18 Offset bits used for each size option

Size option	Bits used as offset
32 Kbyte	10-bits (49:58)
256 Kbyte	13-bits (46:58)
2 Mbyte	16-bits (43:58)
16 Mbyte	19-bits (40:58)

The message reception area is full when the Message Reception Area Available Offset is equal to the Message Reception Area Occupied Offset in the bits specified in Table 18 and the single bit more significant to that specified in the table is different. Bit 39 of the Offset field is never used as an offset to the Message Reception Area, but rather is only used to determine the full status of the Message Reception Area when the size is 16 Mbytes.

MAC Message Completion Queue Configuration register

Each MAC has one Message Completion Queue Configuration register that specifies the base address for a region of memory used to write message completion status.

Figure 39 shows the format of the register. All fields of the register are read by a read access and written by a write access.

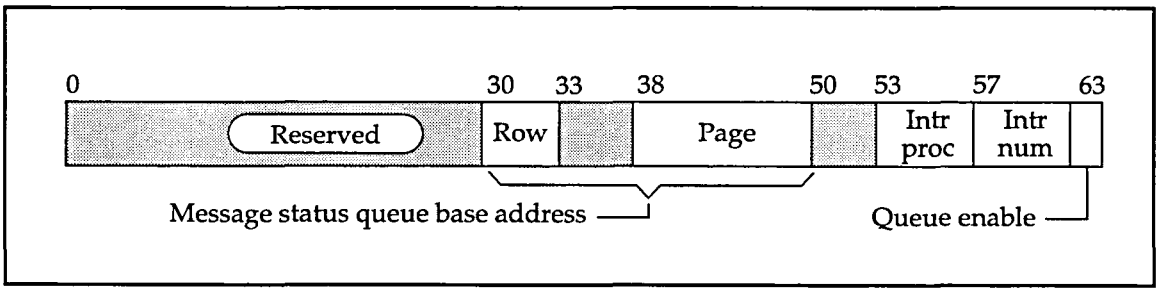


Figure 39 MAC Message Completion Queue Configuration register definition

The bits and fields of the Message Completion Queue Configuration register are defined as follows:

- ***Row*** (bits 30:32) and ***Page*** (bits 38:49) fields—Specify the Message Completion Queue base address. The Virtual Bank is not part of the base address, because the hardware uses all

banks on the MAC with specified Row and Page values as message completion queue area memory.

- ***Interrupt processor*** field (bits 53:56)—Specifies which of the 16 processors within the destination hypernode to interrupt when message completion status is placed in the message completion queue.
- ***Interrupt number*** field (bits 57:62)—Specifies the interrupt number used to interrupt a processor on the destination hypernode when message completion status is placed in the message completion queue.
- ***Queue enable*** bit (bit 63)—Enables receiving messages to the associated message reception area. The bit is cleared by reset.

MAC Message Completion Queue Offset registers

Each MAC has three Message Completion Queue Offset registers:

- Message Completion Queue Reserve Offset—Specifies the offset into the message completion queue memory area where space has been reserved for message completion status.
- Message Completion Queue Write Offset—Specifies the offset where received message status is written.
- Message Completion Queue Read Offset—Specifies the offset where message completion status is read.

Software must initialize these registers by writing a zero value, but, thereafter, only hardware needs to read or write the registers.

Figure 40 shows the format of the three Message Completion Queue Offset registers.

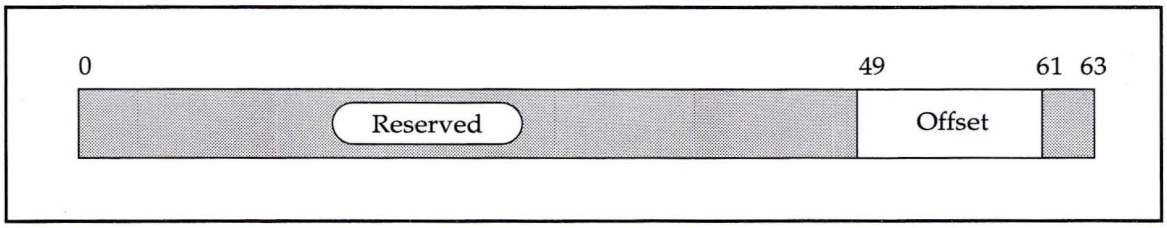


Figure 40 MAC Message Completion Queue Offset register definition

The ***Offset*** field (bits 49:60) specifies an offset into the message completion queue memory area. The most significant bit of the field (bit 49) is not part of the offset, but determines the full or empty status of the queue.

The Message Completion Queue is full when bits 50:60 of the Message Completion Queue Read Offset are equal to bits 50:60 of the Message Completion Queue Write Offset register, and bit 49 of each register is different. The queue is empty when bits 49:60 of each offset register have the same value.

MAC Message Allocation address

Each MAC has a message allocation address. This address is special in that it does not have registers associated with it but rather manipulates other CSRs when accessed. The operation performed is to check that space exists in the message reception area and message completion queue, and if it does exist, to allocate space in the reception area and reserve an entry in the message completion queue.

The following functionality is performed by an access to this address:

- Checking that the Message Reception Area has been enabled to receive a message.

  This is performed by checking the Queue Enable bit of the Message Completion Queue Configuration register.
- Checking that an entry exists in the Message Completion Queue.

  The information required for the check is the Message Completion Queue Reserved Offset and Message Completion Queue Read Offset registers. The check which is made is that the comparison of the two offsets does not result in queue full.
- Checking that space exists in the message reception area.

  The information needed for this check is the length of the message, the Message Reception Area Available Memory Offset register, and the Message Reception Area Occupied Memory Offset register. The check which is made is that the occupied offset less the available offset is greater than the length of the message.
- Returning status of the unsuccessful allocation attempt if any of the above checks fail.

  Otherwise, information from the Message Reception Area Configuration register and the Message Reception Area Available Offset register are returned to the source hypernode specifying the memory address.
- Incrementing the Message Reception Area Available Offset register by the length of the message.
- Incrementing the Message Completion Queue Reserved Offset register by one indicating one less entry available.

MAC Message Completion Enqueue address

Each MAC has a Message Completion Enqueue address that is special in that it does not have registers associated with it but rather other CSRs are manipulated when the address is written. The operation performed is writing the completion status to a memory-based message completion queue.

The message completion queue should not be full, because any previous access to the Message Allocation register address will have reserved space in the queue for the completion status.

The functionality performed by a write to this address is listed below:

- Writing of the completion status to the memory-based message completion queue.

  The memory address to be written is formed by the Row and Page fields of the Message Completion Queue Configuration register and the Offset field of the Message Completion Queue Write Offset register. The data to be written is contained in the write request packet.
- Incrementing by one the Offset field of the Message Completion Queue Write Offset register.

If the Message Completion Queue was empty prior to the accessing the Message Completion Enqueue address, the processor specified by the Message Completion Queue Configuration register is interrupted.

Figure 41 shows the format for the request data sent with a write to a Message Completion Enqueue address.

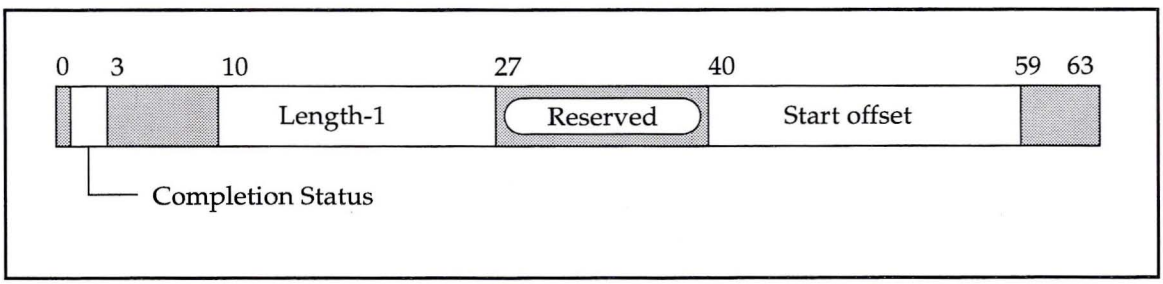


Figure 41 MAC Message Completion Enqueue definition

The bits and fields of the response data returned from a read to the address are defined as follows:

- ***Completion status*** field (bits 1:2)—Specifies the completion status of a received message. Table 19 shows the possible completion status field values.

Table 19 Message Completion Status field values

Field value	Completion status
0	Message received successfully
1	Message aborted
2-3	Reserved

  For completion status values 0 and 1, the space for the message was allocated in the Message Reception Area and the memory must be freed.

- ***Length-1*** field (bits 10:26)—Specifies the length of the allocated memory in memory lines (32-byte increments) for the message. A zero value specifies one memory line (32 bytes) and a value of all ones specifies 131,072 memory lines (4 Mbytes).
- ***Start Offset*** field bits 40:58)—Specifies the offset into the memory reception area to the start of the message.

MAC Message Completion Dequeue address

Each MAC has a Message Completion Dequeue address that is special in that it does not have registers associated but rather manipulates other CSRs when the address is read. The operation performed is reading the completion status from a memory based message completion queue.

The functionality performed by a read to this address is listed below:

- Returning a response with the valid bit as zero if the Message Completion Queue is empty.
- Reading the completion status from the memory-based Message Completion Queue.

  The memory address read is formed by using the Row, and Page fields of the Message Completion Queue Configuration register and the Offset field of the Message Completion Queue Read Offset register. The data that read is returned in the response packet.
- Incrementing by one the Offset field of the Message Completion Queue Read Offset register.

Figure 42 shows the format for the response data returned from a read to a Message Completion Dequeue address.

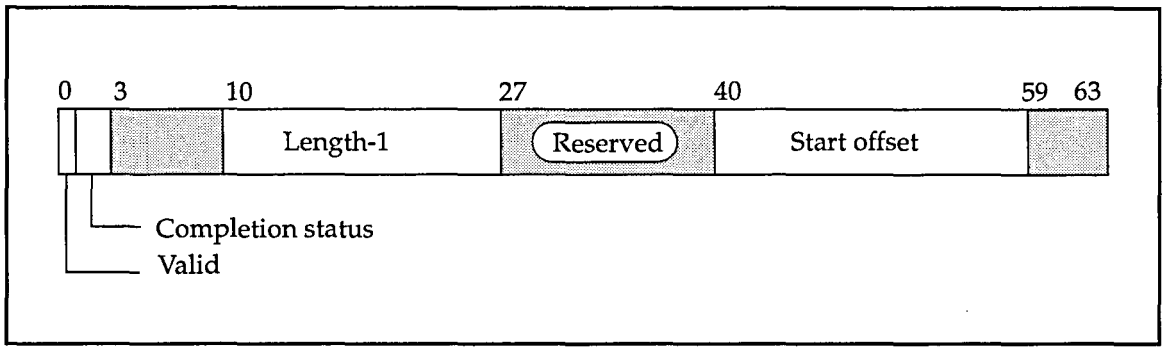


Figure 42 MAC Message Completion Dequeue definition

The bits and fields of the response data returned from a read to the address are defined as follows:

- ***Valid*** bit (bit 0)—Indicates the empty status of the Message Completion Queue at the time of the read access.
- ***Completion status*** field (bits 1:2)—Specifies the completion status of a received message. Table 20 shows the possible completion status field values.

Table 20 Message Completion Status field values

Field value	Completion status
0	Message received successfully
1	Message aborted
2-3	Reserved

For completion status values 0 and 1, the space for the message allocated in the Message Reception Area and memory must be freed.

- ***Length-1*** field (bits 10:26)—Specifies the length of the allocated memory in memory lines (32-byte increments) for the message. A zero value specifies one memory line (32 bytes) and a value of all ones specifies 131,072 memory lines (4 Mbytes).
- ***Start offset*** field (bits 40:58)—Specifies the offset into the memory reception area to the start of the message.

Memory structures

This section describes the memory structures used by the messaging and data copy hardware. The three data structures are:

- Message Reception Area
- Message Completion Queue
- Block translation table

Message Reception area

The Message Reception area is a preallocated region of memory to which messages can be written. The memory is controlled by hardware that enqueues messages as they are received.

All accesses to message reception areas are through coherent memory accesses. A processor can copy a message out from the Message Reception area directly or by using the data copy hardware.

Message Completion Queue area

The Message Completion Queue area holds message completion status until software is ready to process a received message.

The size of the message completion queue area is fixed at 16 Kbytes. Each entry is 8 bytes in size, resulting in 2048 entries per queue. The Message Completion Queue area resides in memory that is physically connected to the MAC.

Figure 43 shows the format for a Message Completion Queue and one of its entries.

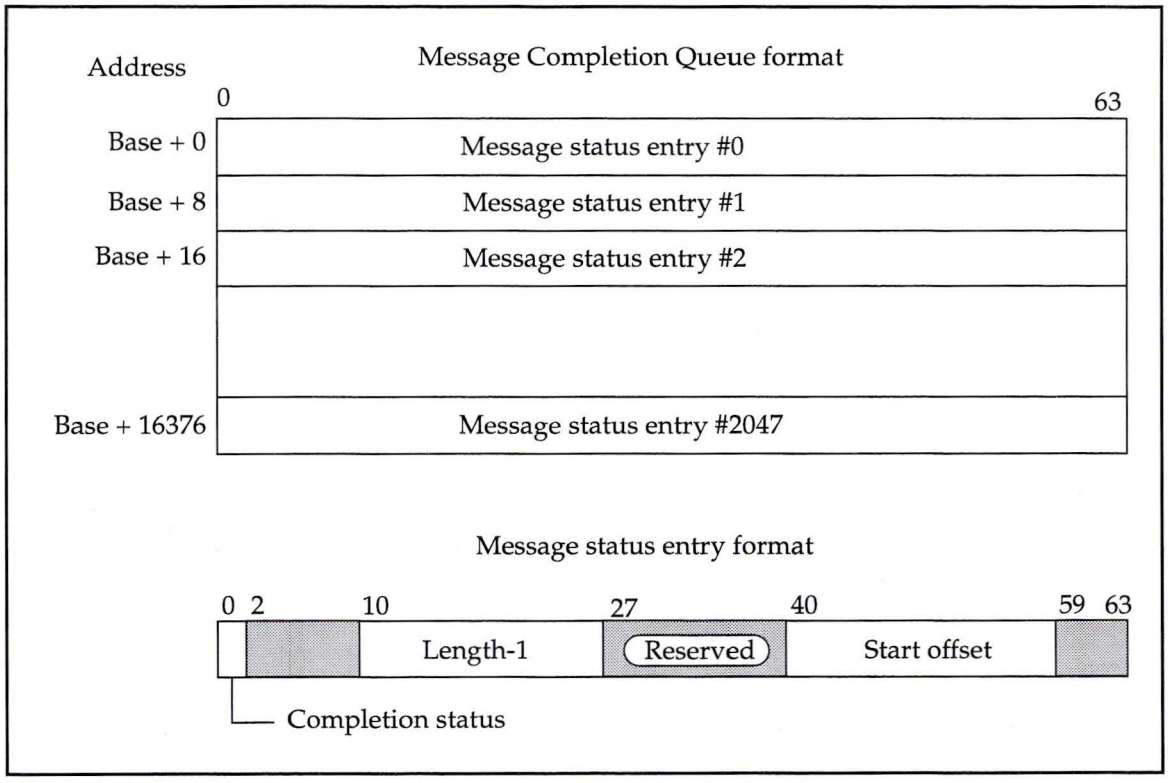


Figure 43 Message Completion Queue and Entry definition

The fields of the message status entry are as follows:

- ***Completion status*** field (bits 0:1)—Specifies the completion status of a received message. Table 21 shows the possible completion status field values.

Table 21 Message Completion Status field values

Field value	Completion Status
0	Message received successfully
1	Message aborted
2-3	Reserved

For completion status values 0 and 1, the space for the message allocated in the Message Reception Area and the memory must be freed.

- *Length-1* field (bits 10:26)—Specifies allocated memory in number of memory lines (32-byte increments) for the message. A value of zero specifies one memory line (32 bytes), and a value of all ones specifies 131,072 memory lines (4 Mbytes).
- *Start offset* field (bits 40:58)—Specifies the offset into the Memory Reception Area to the start of the message.

Block translation table definition

The BTT provides the I/O system a means to translate from a peripheral's address space to physical memory. It specifies a mapping of contiguous addresses to pages of physical memory. The table is limited to a single page of memory, with each entry being a word (four bytes) in size.

Each entry in the table is called a Block Translation Entry (BTE), and it specifies the page frame for a page of physical memory. A page is 4096 bytes. The BTT specifies a maximum address space of four Mbytes.

Figure 44 shows the format for a BTT and one of its entries.

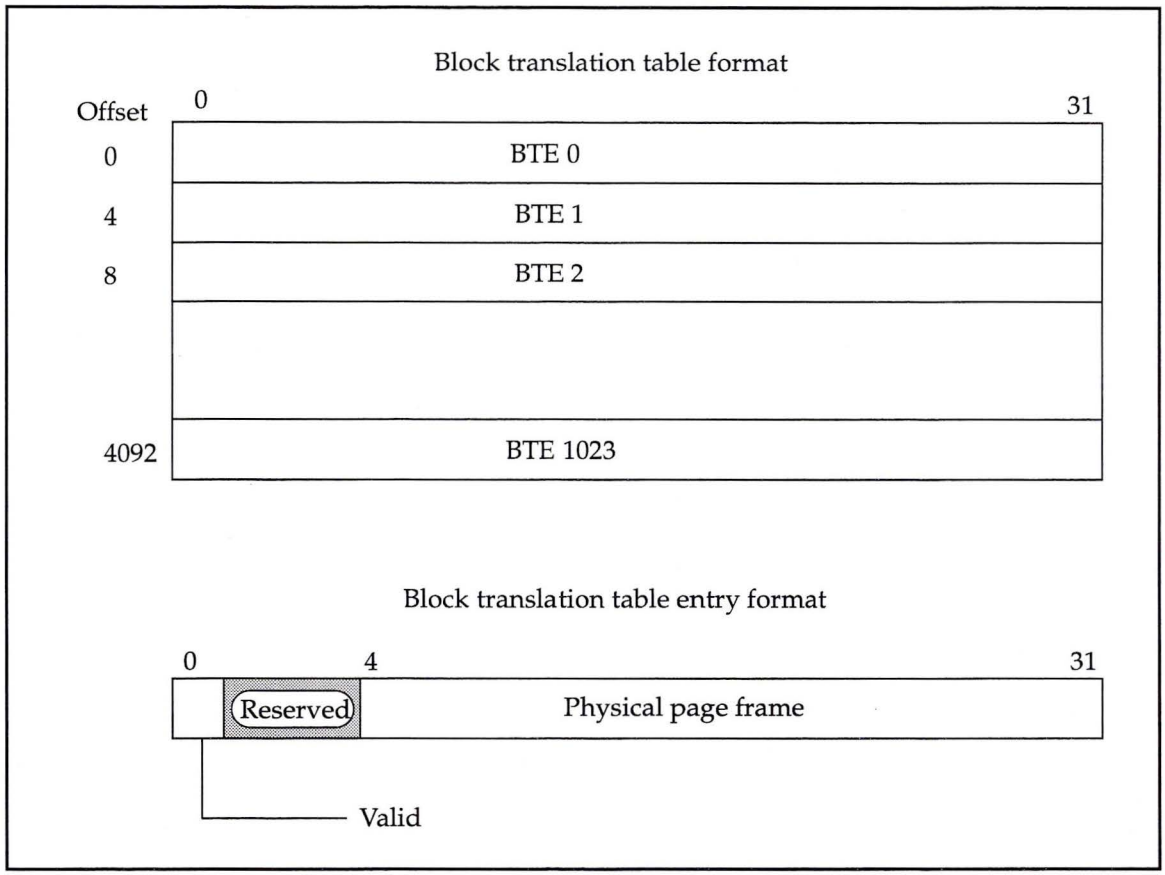


Figure 44 Block translation table and entry definition

The bits and fields of the BTE are as follows:

- ***Valid*** bit (bit 0)—Indicates a valid entry. If the messaging and copy state machine needs to use an entry without the *Valid* bit set, the operation is aborted with completion status indicating the problem.
- ***Read/Write*** bit (bit 1)—Ignored by the messaging and copy state machine.
- ***Physical page frame*** field (bits 4:31)—Indicates the page frame for either the source or destination of the operation.

Software interface

The software interface for the data mover consists of the following functions:

- Resetting and initialization
- Programming the input registers
- Obtaining operation completion status
- Obtaining message completion status
- Reading a message
- Freeing message allocation area memory

Reset and initialization

After reset, all hardware is in a state ready to accept and perform a copy operation. Reset disables all message reception queues. In order to send a message, the destination hypernode Message Reception Area and Message Completion Queue CSRs must be initialized.

There are multiple MACs in a fully configured hypernode, each providing independent control of a message reception area. Any message queue can be disabled to reduce the amount of messaging memory. This does not reduce the bandwidth used for messaging, but rather decreases the number of independently managed message reception areas.

All memory used for message reception areas and the message completion queues must be *wired-down* (the term wired-down means that the virtual-to-physical address translation must remain the same).

Initializing with virtual addresses

The Source or Destination Physical Page Frame registers can be initialized from a virtual address. To initialize with a virtual address, the LPA (Load Physical Address) instruction provides the virtual-to-physical address translation.

When either the source or destination address is obtained from nonwired virtual memory, the TLB Purge Abort Enable bit of the Input Command register must be set. This bit enables the messaging and copy state machine to abort the operation if a TLB purge transaction is detected prior to or during the copy operation. An aborted operation can be restarted to complete the operation.

Copy operations can be restarted with the length remaining to copy when the operation is aborted, guaranteeing forward progress. An aborted message operation must be restarted with

the original length. Forward progress is not guaranteed for messages that use virtual source addressing.

Determining when the input registers are available

There are two ways to determine when the input registers are available

- An Input Command Register can be read at any time to determine the state of the Ready bit. When the bit is set, the messaging and copy state machine has control of the input registers, and software should not write to them. Once the bit is cleared by hardware, software can program the input registers to set up an operation. Software completes programming the input registers by setting the Ready bit.
- An interrupt is sent to the processor when hardware clears the Ready bit. The interrupt is masked off within the processor, allowing the processor to poll the interrupt pending register to determine the availability of the input registers.

Obtaining operation completion status

To obtain the status of messaging and copy operations, the processor reads the Operation Status register. There are two status queues maintained by a PAC, one for each processor attached to the chip. Each queue is three elements deep. The messaging and copy state machine puts status into a queue when an operation finishes, and an entry is removed from a queue when the Operation Status register is read. The ordering is maintained between operations programmed in the input registers and status read out of the Operation Status register.

An interrupt is sent to a processor at the completion of a messaging or copy operation. The interrupt number is specified by the Interrupt Number field of the Input Command register. The interrupt is sent to the processor that initialized the Input Command register with the operation.

Obtaining message completion status

When message completion status is enqueued, a processor is interrupted. The processor reads each Message Completion Queue entry one at a time, determining the successful arrival of the message, the starting offset of the message, and the length of the message. A message completion queue entry with the Valid bit cleared indicates that the queue is empty.

Reading a message

The start and length of a message is obtained by reading a Message Completion Queue entry. A message can be accessed from memory (interleaved or noninterleaved) by either the data copy hardware or the processor.

Freeing message allocation area memory

Once a message has been copied out of the message reception area, the memory it was occupying must be freed for reuse. The process of freeing memory is complicated by the different order of memory allocation. A software structure is used to collate and sort the freed memory by the use of a simple bit map. To free a message, the appropriate bits of the bit map are set. Once all bits of messages to be freed are set, the bits are scanned from the current Message Reception Area Occupied Offset register value looking for the first occurrence of a cleared bit. The Message Allocation Area Occupied Offset register is then updated to the offset associated with the end of the string of bits that were set. Finally, the string of set bits is cleared.

Synchronization

5

Synchronization allows processors executing multiple threads of the same process to share data by locking data structures until all threads requiring the structure have completed. It is accomplished by using semaphore variables associated with each data structure.

The semaphore variable provides a flag to all processors involved in the multiple thread process. Each processor has access to the semaphore variable, but it can only manipulate the variable atomically. That is, once a processor starts a semaphore operation, it must complete it before any other processor may perform an operation.

Semaphore operations take the form of instructions for both coherent and noncoherent memory accesses.

Coherent semaphore instructions

There are two instructions for semaphore operations in coherent memory:

- Load and clear word (LDCW)
- Load and clear double (LDCD)

The load implies that the semaphore variable memory line is loaded into the processor data cache. The clear operation is performed by the processor cache. A cache hint bit must be set for these instructions to work properly.

SPP1000-series systems did not require the accelerated cache hint bit to be set for proper operation of the LDCW instruction. S-Class and X-Class servers, however, require it be set, because the LDCW instruction without the accelerated cache hint bit set is a fetch and clear operation and does not use the data cache.

Noncoherent semaphore operators

S-Class and X-Class servers support six semaphore operators not defined in the PA-RISC architecture. These special operators may enhance semaphore operations in some applications, because they operate directly on semaphore variables located in unencacheable memory pages (pages with the U-bit set in the TLB entry; see the "PA-8000 TLB Entry U-bit" section on page 101). They do not accelerate the semaphore into the data cache.

These noncoherent semaphore operators include the single and double versions of the following instructions:

- Fetch and clear
- Fetch and increment
- Fetch and decrement

The fetch implies that the semaphore variable goes directly into a processor register. When either the fetch and increment or fetch and decrement instructions read the variable, the MAC automatically increments or decrements it.

In addition to these fetch instructions, noncoherent read and write operations are also available to access semaphore variables. If a noncoherent semaphore operator accesses a memory line that is encached by a processor, the semaphore operation will fail, resulting in an error being returned to the processor. All semaphore variables are 16-byte aligned. Semaphore operations to nonaligned variables produce undefined results. Non-coherent semaphore operations can be issued to any memory word for word sized operations, and any double word for double word sized operations.

Implementing noncoherent semaphore operations requires a sequence of instructions using PAC CSRs.

The steps of the sequence are:

1. Check write access privilege for the semaphore address.
2. Arm the operation by writing to the PAC Operation Context register Armed bit. See the "PAC Operation Context register" section on page 61 section for information on this register.
3. Write the physical address to the PAC Operation Address register.
4. Read the Fetch Operation address. The value read is the return value for the semaphore operation.
5. Check the PAC Operation Context register Triggered bit to make sure the operation was issued. If the Triggered bit is not set, the operation must be restarted. The Armed bit is cleared when the sequence is interrupted by either an external interrupt or a TLB miss.

6. The other noncoherent semaphore operations and the noncoherent read operation can also use this sequence by using a different Fetch Operation CSR address. The noncoherent write operation is similar to the above sequence, except that the load instruction is replaced with a store instruction with the value to be stored.

As an example, the sequence of instructions for a `fetch_and_inc32` is as follows:

```
      PROBEW   fetch_addr              ;Check protection
loop
      LDI      1,%t1
      STD      %t1,(CSR_OP_ARMED)      ;Arm CSR Operations
      LPA      fetch_addr,%t2
      STD      %t2,(CSR_OP_ADDR)       ;Fetch operation addr
      LDW      CSR_FETCH_INC),%t3      ;Issue fetch semaphore
      LDD      (CSR_OP_ARMED),%t4
      BB,*>=   %t4,62,loop             ;check if triggered
```

Barrier synchronization

Not all threads in a multithread process complete at the same time. All threads, however, must typically wait until the last thread finishes. The threads *hit* a *barrier* and must be synchronized before continuing.

The barrier synchronization semaphore is a running count of the number of threads that have reached the barrier. The last processor to finish writes a nonzero value to the semaphore address, signalling the other processor that the threads are synchronized.

The instruction, Coherent Increment Double, for barrier synchronization is described in the *PA-RISC 2.0 Architecture* manual.

An alternate method exists for barrier synchronization semaphore operations that requires a sequence of instructions using PAC CSRs similar to the sequence discussed in the "Noncoherent semaphore operators" section on page 96.

This method is not as efficient as the instruction.

The following sequence of instructions provides an alternate method for the `coherent_inc64()` function:

```
    PROBEW  cincd_addr      ;Check protection
loop
    LDI     1,%t1
    STD     %t1,(CSR_OP_CNTX);Arm CSR Operations
    LPA     cincd_addr,%t2
    STD     %t2,(CSR_CINCD)   ;Issue Coh. Inc.
    LDD     (CSR_OP_CNTX),%t4
    BB,*>= %t4,62,loop       ;check if triggered
```

The steps of the sequence are:

1. Check write protection for the operation address.
2. Arm the operation by writing to the CSR Operation Armed register Armed bit.
3. Perform virtual-to-physical address translation.
4. Issue the Coherent Increment operation with the physical address provided as the data of the write.
5. Check the PAC Operation Context register Armed bit to make sure the operation was issued. If the Triggered bit is not set then the operation must be restarted. The Armed bit is cleared when the sequence is interrupted by either an external interrupt or a TLB miss.

PAC semaphore registers

The PAC has two registers and several addresses to implement CSR-based semaphore operations. The Operations Context registers and the Operation Address registers are detailed in Chapter 3, "Cache management." The address are discussed in the following sections.

PAC Fetch Operation address

Each PAC has six Fetch Operation addresses, three for each processor pair. A read of these addresses triggers one of the following noncoherent fetch semaphore operations:

- Fetch and Increment
- Fetch and Decrement
- Fetch and Clear

If the Armed bit in the Operation Context register is set, an access to one of the fetch operation addresses results in a fetch operation to memory. When the Armed bit is set, the address contained in the Fetch Operation Address register becomes the address for the fetch operation. If the Armed bit is not set, the PAC returns the value zero to the processor rather than the data intended for the fetch operation.

The size field determines whether the operation is word or double word. Any word-aligned address can be used for word operations, and any double-word-aligned address can be used for double-word addresses.

PAC Noncoherent Read and Write Operation addresses

Each PAC has two Noncoherent Read Operation addresses, one for each processor pair. Each PAC also has two Noncoherent Write Operation addresses, one for each processor pair. A read of the noncoherent read address triggers the noncoherent read operation. A write to a noncoherent write address triggers a noncoherent write operation.

If the Armed bit in the Operation Context register is set, the address contained in the Operation Address register becomes the address for the operation. If the Armed bit is not set, the PAC returns the value zero to the processor rather than the data intended for the noncoherent read operation. For a Noncoherent Write Operation, if the Armed bit is not set, the PAC drops the noncoherent write.

The size field determines whether the operation is word or double word. Any word-aligned address can be used for word operations, and any double-word-aligned address can be used for double-word addresses.

PAC Coherent Increment address

Each PAC has two Coherent Increment addresses, one for each processor pair. Writes to these addresses trigger coherent increment operations. If the armed bit in the Operation Context register is set, the address contained in the Operation Address register becomes the address for the fetch operation. If the Armed bit is not set, then the PAC ignores the write access.

CTI Cache Global Flush address

Each PAC has two CTI Cache Global Flush addresses, one for each processor pair. Writes to these addresses triggers CTI cache global flush operations. If the armed bit in the Operation Context register is set, the address contained in the Operation Address register becomes the address for the flush operation. If the Armed bit is not set, then the PAC ignores the write access.

CTI Cache Prefetch Read and Write addresses

The PAC has two CTI Cache Prefetch Read addresses, one for each processor pair. The PAC also has two CTI Cache Prefetch Write addresses, one for each processor pair.

A write to the CTI Cache Prefetch Read addresses triggers a CTI cache prefetch for read operation. A write to the CTI Cache Prefetch Write addresses triggers a CTI cache prefetch for write operation.

If the Armed bit in the Operation Context register is set, the address contained in the CSR Operation Address register becomes the address for the prefetch operation. If the Armed bit is not set, then the PAC ignores the write access.

PA-8000 TLB Entry U-bit

Each PA-8000 TLB entry contains a bit that controls whether an access to coherent memory space should accelerate the memory line into its data cache. S-Class and X-Class servers use this bit to inhibit noncoherent operations from being moved into data cache.

Table 22 lists the supported semaphore operators and the associated PA-8000 instructions used to issue the operations for the accessed memory page.

Table 22 Semaphore operation instructions

TLB entry U-bit	PA-8000 instruction	Semaphore operation
0 (Coherent)	LDCW	Load and Clear 32-bit
	LDCD	Load and Clear 64-bit
	CINCD	Coherent Increment 64-bit
1 (Noncoherent)	FINCW	Fetch and Increment 32-bit
	FINCD	Fetch and Increment 64-bit
	FDECW	Fetch and Decrement 32-bit
	FDECD	Fetch and Decrement 64-bit
	FCLRW	Fetch and Clear 32-bit
	FCLRD	Fetch and Clear 64-bit
	LDW	Non-Coherent Load 32-bit
	LDD	Non-Coherent Load 64-bit
	STW	Non-Coherent Store 32-bit
	STD	Non-Coherent Store 64-bit

Mixing coherent and noncoherent accesses to a memory line generates an error to the issuing processor.

Interrupts

6

This chapter discusses the interrupt mechanism of the S-Class and X-Class servers.

The *PA-RISC 2.0 Architecture* manual presents a detailed discussion of the interrupt mechanism implemented for the PA-8000 processor and that material is not presented in this book. Instead, this chapter discusses interrupts unique to the S-Class and X-Class servers.

Overview

Interrupts cause process control to be passed to an interrupt handling routine. Upon completion of interrupt processing, a return from interrupt (RFI) instruction restores the saved processor state, and the execution proceeds with the interrupted instruct.

When responding to an interrupt, the processor behaves as if it were not pipelined. That is, it behaves as if a single instruction is fetched and executed. Any interrupt conditions raised by that instruction are handled at that time. If there are none, the next instruction is fetched, and so on.

Faults, traps, interrupts, and checks are different classes of interrupts.

A fault occurs when an instruction requests a legitimate action that cannot be carried out due to a system problem. After the problem has been corrected, the instruction causing the fault executes normally. Faults are synchronous with respect to the instruction stream.

A trap occurs when a function requested by the current instruction can not or should not be carried out. For example, attempting to access a page for which a user does not have privilege causes a trap. Another example is when the user requires system intervention before or after the instruction is executed,

such as page reference traps used for debugging. Traps are synchronous with respect to the instruction stream.

An interrupt occurs when an external device requires processor attention. This is done by setting a bit in the external interrupt request register. Interrupts are asynchronous with respect to the instruction stream.

A check occurs when the processor detects a malfunction. The malfunction may or may not be correctable. Checks can be either synchronous or asynchronous with respect to the instruction stream.

Processor interrupts

System interrupts are applied to a processor by writing to its External Interrupt Request Register (EIRR). The S-Class and X-Class server interrupts occur from several sources which include:

- Other processors
- I/O subsystem
- Memory subsystem
- Messaging and data copying mechanism
- Time of Century counter loss of synchronization
- Utilities board

The EIRR is written to using either a word or double word store. All fields of the register are undefined when read.

The format of the EIRR is shown in Figure 45.

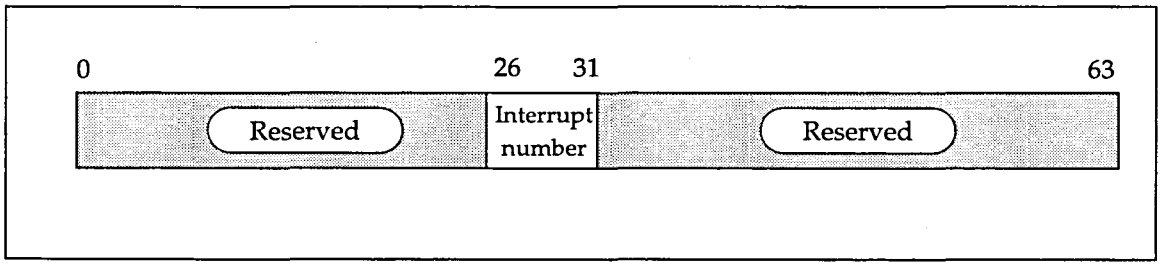


Figure 45 PA-8000 External Interrupt Request register definition

The ***Interrupt Number*** field (bits 26:31)—Specifies the external interrupt to be set in the EIRR. The value of the interrupt, 0-63, is encoded in the six bit field.

Utilities board interrupts

Almost all interrupts are sent directly to the processor EIRR with the exception of those associated with the core logic bus. The utilities board collects hypernode environmental interrupts and applies them to the EIRR. The utilities board handles the following types of interrupts:

- Environmental conditions
- Transfer of external control
- External communications
- System warnings and failure

Figure 46 shows how these interrupts are presented to the processor.

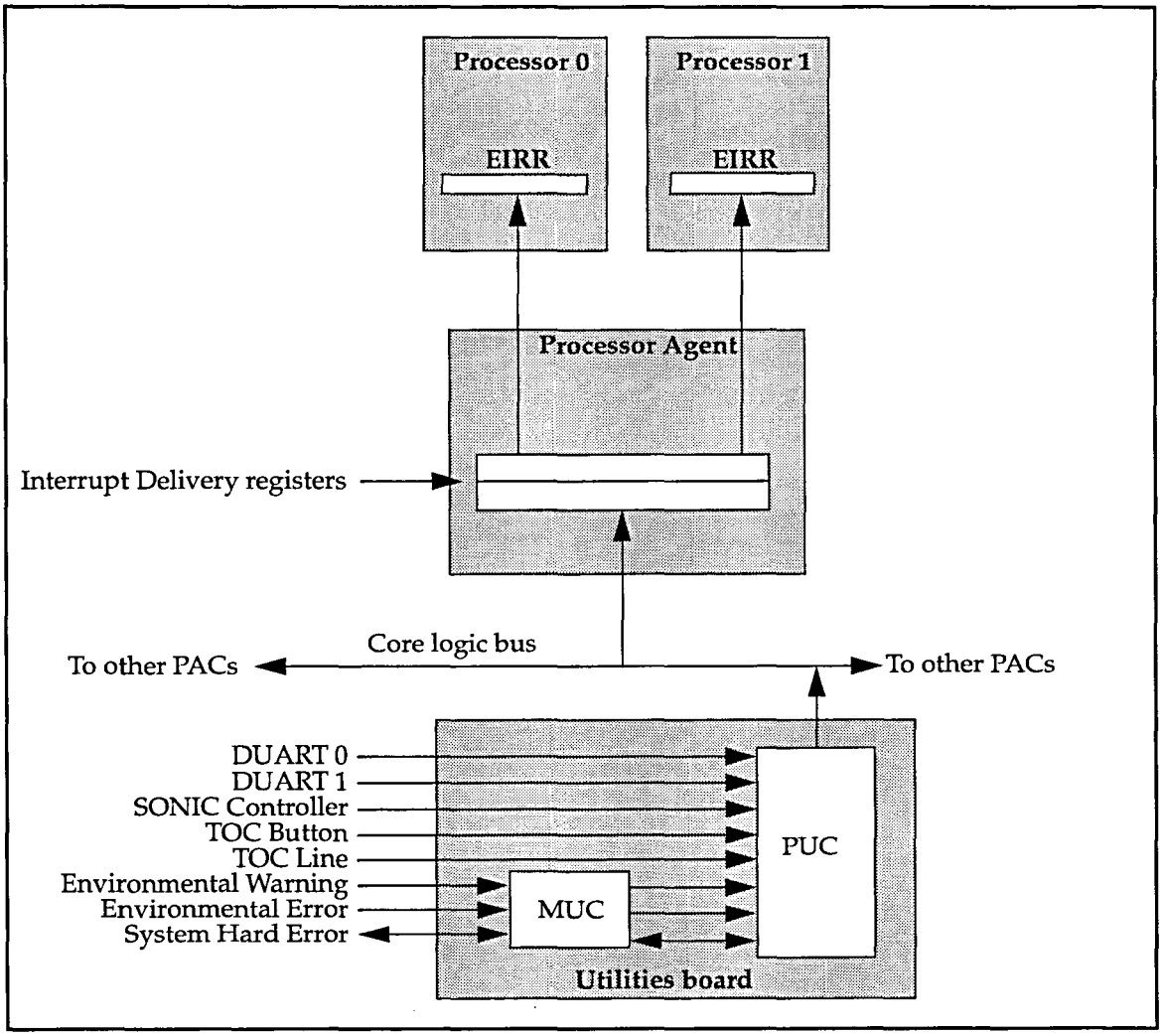


Figure 46 Core logic interrupt system

The utilities board provides interrupt information to all PACs in the hypernode. Each PAC must determine if one of its two processors is enabled to handle the pending interrupt. All processors enabled, process the interrupt.

The utilities board accepts eight separate interrupt sources listed in Table 23.

Table 23 Core logic interrupt sources

Core logic interrupt source	Interrupt bit
Duart channel 0	0
Duart channel 1	1
SONIC controller	2
Transfer of control button	3
Transfer of control line (from test station)	4
Environmental warning	5
Environmental error	6
System hard error	7

The utilities board processes interrupts as follows:

- Interrupts are latched into the Interrupt Status register in the MUC. Interrupts can also be forced into the Force Interrupts register for testing purposes (these are not masked).
- The interrupts are compared to data in the Interrupt Mask register, and, if they are not masked out, are sent across the core logic bus to the Interrupt Delivery register in the PAC.
- If the PAC determines that one of its processors has the interrupt enabled, it delivers the interrupt information to the processor by writing to the processor EIRR with the level of the interrupt in bits 26:31 of the 64-bit register.

The EIRR also receives other system exceptions and interrupts. See the *PA-RISC 2.0 Architecture* manual for more information.

PAC interrupt logic

Each PAC receives an eight-bit mask from the PUC (via the core logic bus) that specifies the interrupt sources sent to the processors. Each PAC has interrupt delivery information for each

of the eight possible interrupt sources. Figure 47 shows the interrupt delivery data.

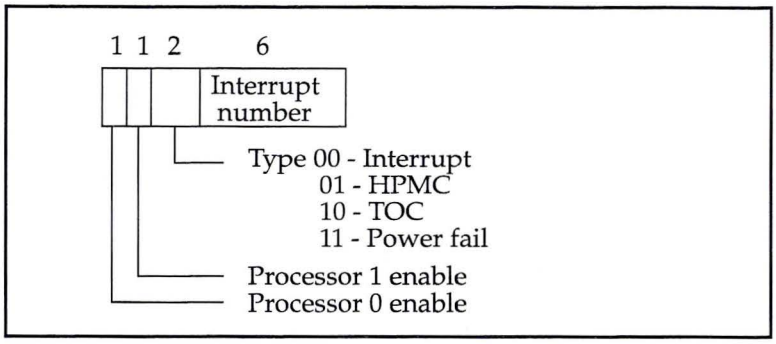


Figure 47 PAC Interrupt Delivery Information

The information contains individual enables for each of the two processors connected to a PAC, the type of exception, and the interrupt number or an interrupt exception type.

Figure 48 shows where these bits are located in the PAC Interrupt Delivery register.

PAC Interrupt Delivery registers

There are two 64-bit Interrupt Delivery registers on each PAC. Each register specifies the delivery information for four of the eight interrupt sources.

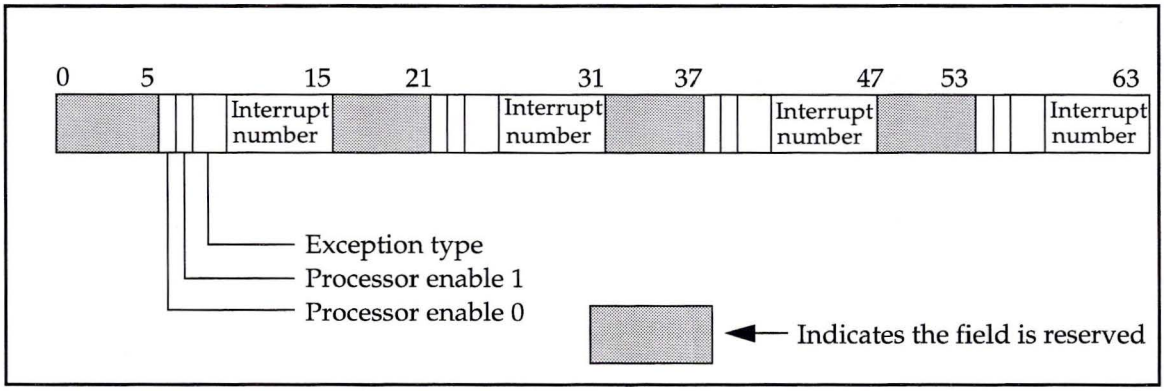


Figure 48 PAC utilities board interrupt delivery register definition

The fields and bits of the PAC interrupt delivery registers are defined as follows:

Processor enable bits—Indicate that the processor is enabled to handle the exception.

Exception type fields—Indicate the type of interrupt:

- Interrupt
- HPMC
- TOC loss of synchronization
- Power failure

Interrupt number fields—Indicate the interrupt source to the deliver registers.

The utilities board interrupts map to the core logic interrupt delivery registers as shown in Table 24. All fields are written by a CSR write and read using a CSR read. Reset has no effect on the register.

Table 24 Core logic interrupt delivery registers

Utilities board interrupt source	Register and bits
DUART channel 0	Register 0, bits 6:15
DUART channel 1	Register 0, bits 22:31
SONIC controller	Register 0, bits 38:47
Transfer of control button	Register 0, bits 54:63
Transfer of control line (from test station)	Register 1, bits 6:15
Environmental warning	Register 1, bits 22:31
Environmental error	Register 1, bits 38:47
System hard error	Register 1, bits 54:63

PUC interrupt logic

PUC interrupt logic is comprised of the following PUC interrupt registers:

- Interrupt status register
- Interrupt mask register
- Interrupt force register

PUC Interrupt Status register

The PUC contains one Interrupt Status register. The register maintains the status of the pending utilities board interrupts given in Table 24. Figure 49 shows the definition of the register.

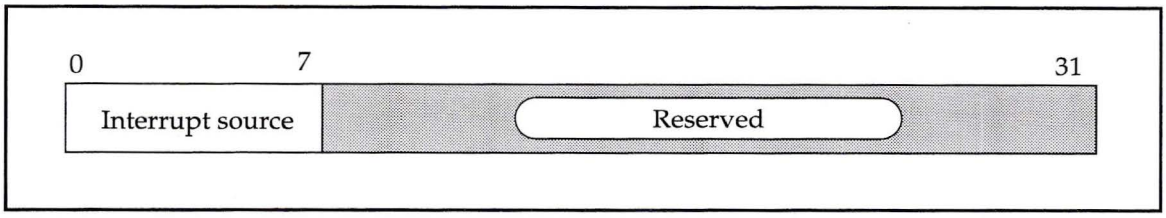


Figure 49 PUC Interrupt Status register definition

The ***Interrupt source*** field (bits 0:7)—Indicates the source of the PUC interrupt. The bits are set when the PUC detects an active input interrupt signal. The contents of the register are read by a CSR read operation. Each bit set in the data of a CSR write operation clears the associated bit of the status register, and a reset clears the register.

Table 25 shows the bit field assigned to each core logic interrupt source.

Table 25 PUC Interrupt register field definitions

Register bit	Interrupt source
0	DUART channel 0
1	DUART channel 1
2	SONIC controller
3	Transfer of control button
4	Transfer of control line (from test station)
5	Environmental warning
6	Environmental error
7	System hard error

Each individual bit of the Interrupt Status register can be cleared without affecting the other bits, even when the CSR is receiving an interrupt. For all bits except the transfer of control button, clearing a bit has precedence over setting it. This means that if an input interrupt is still asserted when the bit is cleared, the status bit is set on the following cycle, and a new interrupt is sent to each PAC.

The transfer of control button interrupt is edge-level sensitive, and the other interrupts are level sensitive.

PUC Interrupt Mask register

The PUC contains one Interrupt Mask register. The register enables sending pending interrupts to the PAC. It provides the ability to mask out any of the eight interrupt sources. Figure 50 shows the definition of the register.

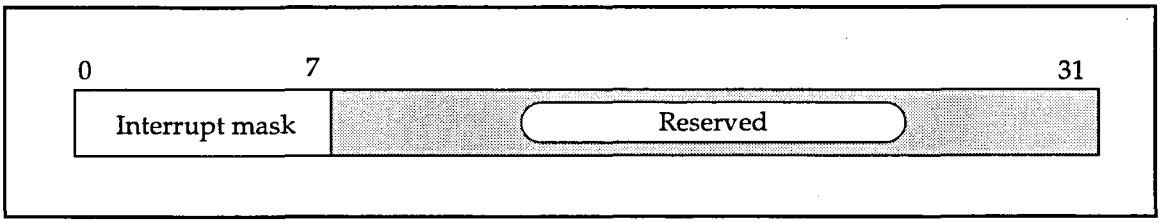


Figure 50 PUC Interrupt Enable register definition

The ***Interrupt mask*** field (bits 0:7)—Indicates interrupts are masked. The contents of the register are read by a CSR read operation and written by a CSR write operation. A reset clears the register.

PUC Interrupt Force register

The PUC contains one Interrupt Force register. The register allows software to force an interrupt on any of the eight interrupts. Figure 51 shows the definition of the register.

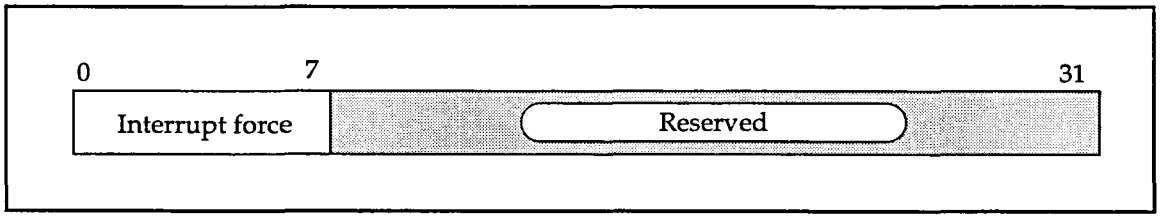


Figure 51 PUC Interrupt Force register definition

The ***Interrupt force*** field (bits 0:7)—Indicates the interrupt(s) being forced. The contents of the register are read by a CSR read operation and written by a CSR write operation. Setting a bit forces an interrupt regardless if it is enabled of not. A reset clears the register. Table 25 shows the interrupts assigned to each core logic interrupt force bit.

I/O subsystem

7

Overview

The S-Class and X-Class I/O subsystem serves as a bridge between the system and peripheral devices. It connects the system to the industry standard 32- or 64-bit peripheral component interface (PCI) bus. A fully configured hypernode provides up to eight PCI buses, one for each PAC. Each bus supports three controllers for a maximum of 24 PCI controllers per hypernode.

Unlike the I/O subsystems of the SPP1000-series, the S-Class and X-Class I/O subsystem transfers data coherently to and from the system main memory, eliminating the need for flushing the processor caches. Figure 52 shows a block diagram of the I/O subsystem, based on the PCI-bus Interface Controller (PIC).

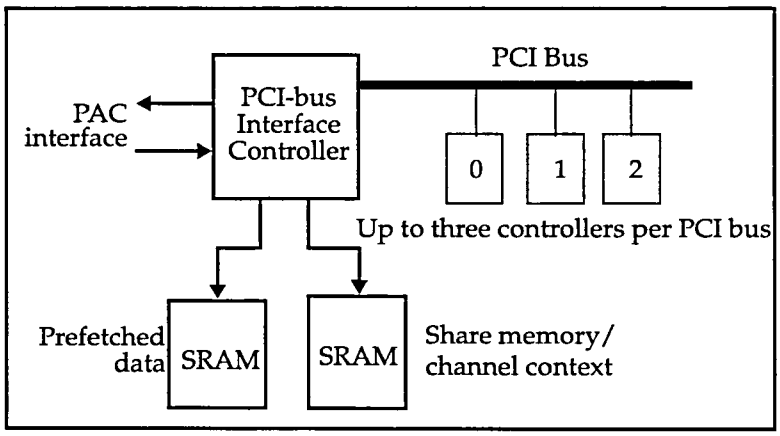


Figure 52 I/O system block diagram

The PIC provides memory-mapped access from the processor to the I/O controllers and allows external devices to transfer data into and out of system memory. There is one PIC per PAC. Each PIC has a pair of unidirectional links to the associated PAC. Each PIC has two physical SRAM banks, one for PIC data prefetch and one for PCI controller-shared memory.

Logical I/O channel

The PIC uses the concept of a logical I/O channel to translate PCI addresses and prefetch system coherent memory. A logical channel defines a pipe between four MBytes of PCI memory space to four MBytes of system coherent memory.

Each channel has a distinct address mapping between the PCI bus address space and the system main memory. It also has a buffer for storing prefetched data during read data transfers. The buffer hides PCI start-up latencies associated with read data transfers.

The logical I/O channel also has a posted write buffer for collecting 32-byte data cache lines before flushing them to system coherent memory. Figure 53 depicts the logical I/O channel concept, and Figure 54 shows the PCI bus command and address format.

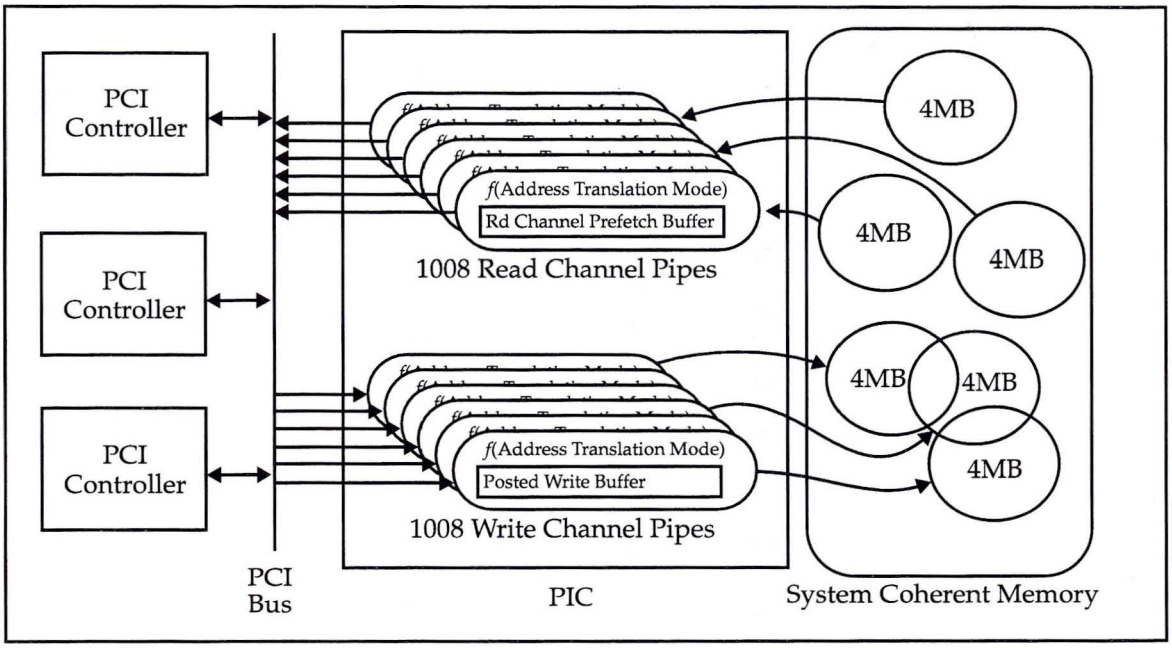


Figure 53 Logical I/O channel model

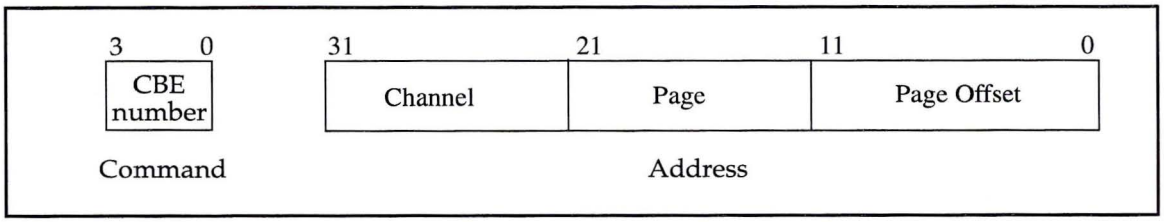


Figure 54 PCI bus command and address

The 10 most significant bits of the PCI address define the logical channel number, providing a total of 1,024 logical channels. The PIC provides a separate read and write pipe per controller to memory. Therefore, it uses the PCI command (CBE number) to distinguish between a read and write channel, effectively doubling the number of available channels to 1,024 read and 1,024 write. PIC implementation reserves channels 1008-1023, leaving a maximum of 1,008 read channels and 1,008 write channels. The 22-bit channel offset gives each channel a four-MByte data space. Consecutive channels may be chained to allow transfers larger than four MBytes.

Note

Each channel can be used for one or more DMA transfers on a controller. However, best performance is usually realized with a single I/O transfer per channel. Due to PIC caching algorithms for prefetched data, a channel can *not* be used by multiple controllers at the same time.

Channel initialization

Before a processor initializes an I/O operation, it must set up a channel for the appropriate controller by writing to the PIC Channel Builder register. There are two types of channel builds: accelerated and nonaccelerated.

The accelerated build consists of a single write to the Channel Builder register. See the "PIC Channel Builder Register" section on page 133. In this flow, the PIC initializes all the external SRAM channel context state and prefetches any needed data and TLB entries.

The nonaccelerated build consists of multiple writes to the Channel Builder register. The first one initializes the channel. The second and optional third ones initialize TLB entries in the channel TLB cache. The optional fourth write schedules any needed data prefetch.

Channel context and shared memory SRAM

The PIC maintains both channel context and shared memory in its external Channel Context SRAM (CCSRAM). The channel context space reserves 64 Kbytes from the base of the SRAM and shared

memory is available for the remainder. The PIC supports up from 256 Kbytes to 2 Mbytes of external CCSRAM. See figure Figure 55.

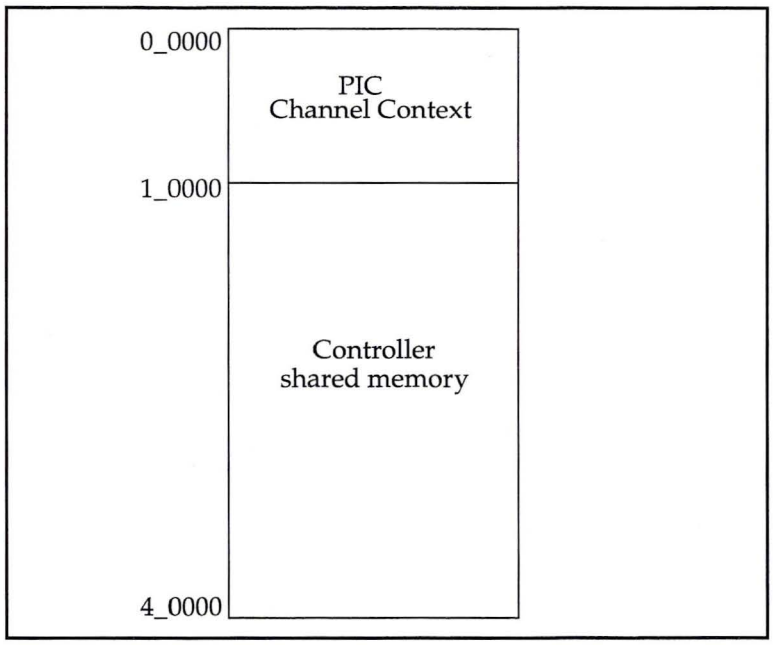


Figure 55 CCSRAM Layout

Channel context

The channel context portion of the SRAM contains information to determine how to perform the DMA transfer between PCI and system memory. Channel context is mapped into both PCI memory space and processor I/O space. The channel context region, however, is only directly accessed for diagnostic use. The processor programs channel context state through the PIC Channel Builder register.

Shared memory

The PIC provides a local, byte-addressable, shared memory region in the CCSRAM for all status and control structures that support the PCI controllers. This SRAM is not coherent with main memory. It is *visible* from both PCI memory space and processor I/O space.

Host-to-PCI address translation

The 40-bit hypernode address map, shown in Figure 56, reserves 16 GBytes from F8 0000 0000 to FB FFFF FFFF for host access to PCI devices.

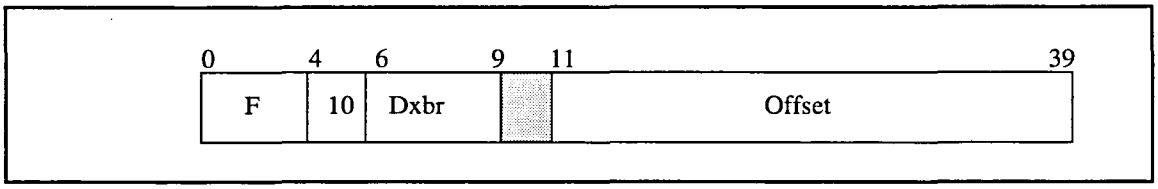


Figure 56 Hypernode-local I/O address space format

The fields for the hypernode-local I/O address space are defined as follows:

- **Dxbr** field bits (6:9)—Specifies one of eight PACs
- **Offset** field bits (11:39)—Specifies 29-bit PIC mapping.

PCI configuration space

The PCI specification establishes three PCI address spaces: configuration, IO, and memory. Dedicated read and write commands select a particular space for a PCI bus operation.

The PCI configuration space contains a set of configuration registers that must be implemented by all bus targets except host bridges. The configuration registers allow the PIC to setup PCI I/O and memory space requirements in the system address map. PCI configuration space is 16 MBytes (24 bits). Figure 57 shows the PCI configuration address format from the hypernode perspective.

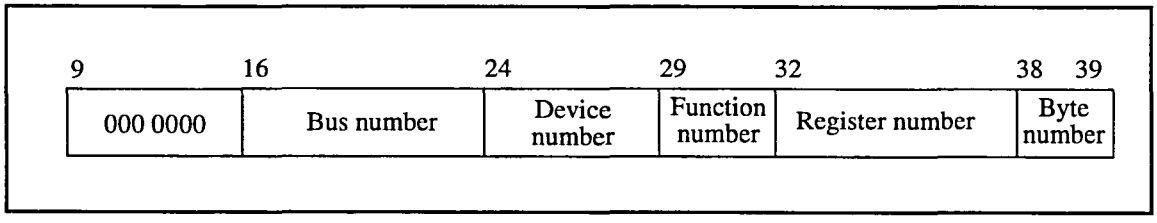


Figure 57 Hypernode-local I/O configuration space format

The fields for the hypernode-local I/O configuration space format are defined as follows:

- ***Bus number*** field (bits 16:23)—Indicates PCI bus number. Bus 0 is the bus directly attached to the PIC. Any other PCI buses must be assigned Bus numbers 1 to 255 during the software probe.
- ***Device number*** field (bits 24:28)—Specifies the device on one PCI bus segment. Bus 0 only supports Device 0 through Device 3.
- ***Function number*** field (bits 29:31)—Specifies the function on a PCI device.
- ***Register number*** field (bits 32:37)—Specifies the register within a PCI function.
- ***Byte number*** field (bits 38:39)—Provides the byte address. This field and the packet size code establish the PCI byte enables during the access. Accesses must be aligned to their natural size. The PIC does not support 64-bit double-word accesses to PCI.

PCI I/O and memory space

PCI I/O and PCI memory space allow host access to device-specific CSRs. Target implementation of either space is optional. However, if a device implements either space, it must also implement a corresponding base address register in PCI configuration space to allow consistent address mapping.

PCI I/O and PCI memory space may each be as large as four GBytes. PCI I/O space uses a full byte address, so the PIC combines the least significant bits of the hypernode address with the packet size code to create the PCI byte address and the PCI byte enables. PCI memory space uses four byte-aligned addresses; smaller entities are addressed by bus byte enables.

Hypernode I/O space-to-PCI map

As shown in Figure 58, the PIC maps its partition of hypernode-local I/O space into the three PCI spaces. It also reserves an area for diagnostic windows into the external PIC context/shared memory and external PIC prefetch memory.

The PCI defines eight GBytes of I/O and memory space, but the PIC only has 0.5 GByte of hypernode space in which to operate. Therefore, the address map is necessarily sparse. Only the PCI configuration space maps on a one-to-one basis.

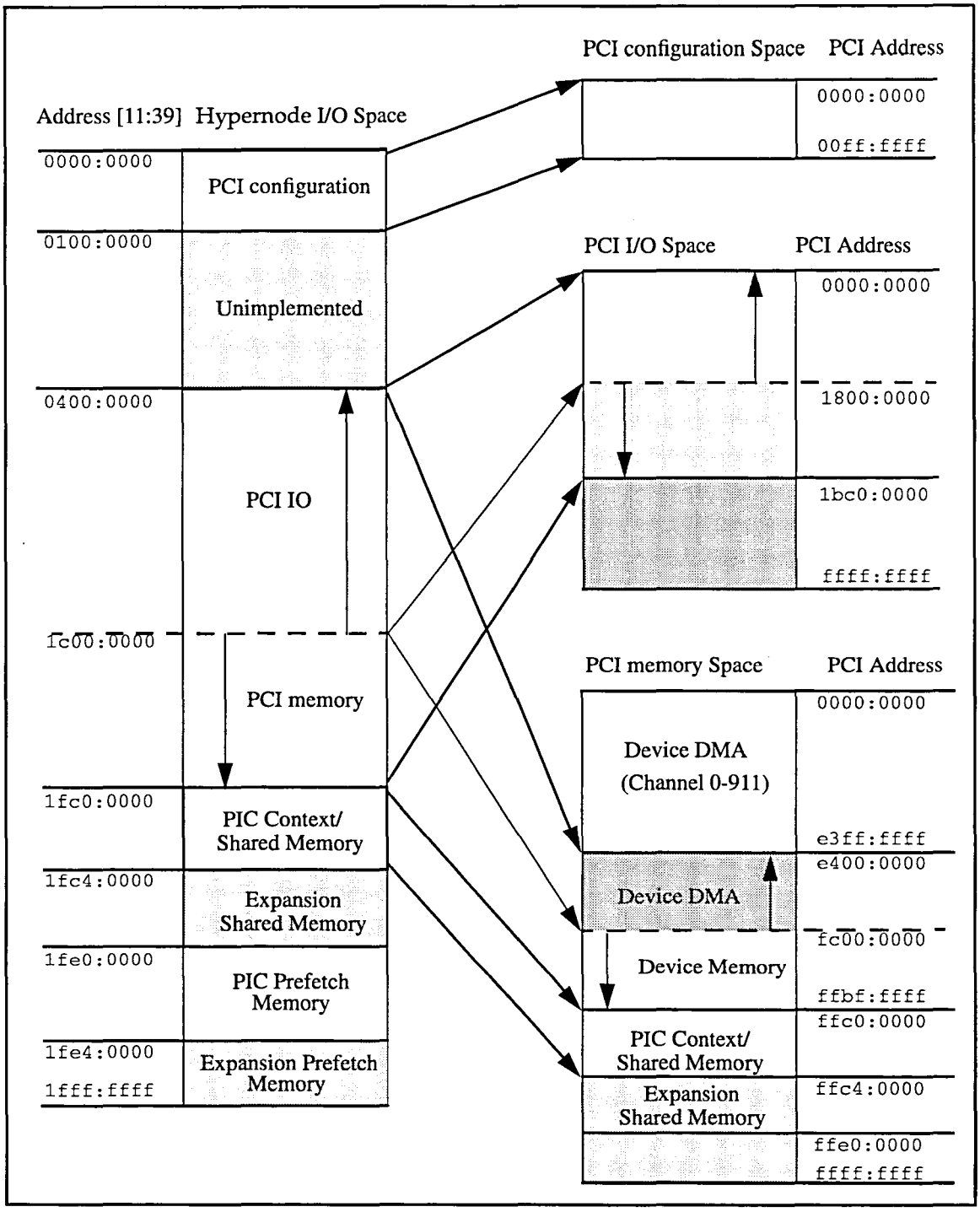


Figure 58 Hypernode I/O space to PCI space mapping

To increase installation flexibility, the PIC can generate PCI addresses increasing from 0000 0000 in PCI I/O space and decreasing downward from FFBF FFFF in PCI memory space. The allocation boundary between I/O and memory space is programmable in 64-MByte increments and can range from no I/O space and all memory space to no memory space and all I/O space.

Maximizing the PCI I/O space also maximizes the number of available PCI DMA channels, while increasing the PCI memory space comes at the cost of 16 PCI DMA I/O channels per 64-MByte increment.

PCI-to-host memory address translation

Since most PCI controllers generate a 32-bit address, they are capable of addressing up to four GBytes. S-Class and X-Class servers can have more than this amount. Therefore, they provide for translating the 32-bit addresses to system addresses. There are two types of address translation: physical and logical. An address translation enable bit (ATE) for each channel determines the translation type used to perform the address translation between PCI and system coherent memory

In the physical translation mode, data is fetched directly from a four-Mbyte buffer in system main memory.

Logical address translation implies that the translation process uses an intermediate step to derive the system address. The process uses translation tables in system memory for data transfers. A data transfer of two pages or less is a special case of the logical translation process.

Most modern I/O controllers use part of the host memory for storing control and status blocks. Typically, these are accessed using word accesses over the PCI bus. Since the main memory access latency is relatively large, accesses to status and control structures in main memory use part of the channel context SRAM for storing the control and status structures.

By addressing logical channel 1023, a controller accesses the entire SRAM.

Physical address translation

The simplest translation mode is the physical translation mode. In this mode, the four-MByte PCI channel directly maps into a four-MByte, physically contiguous block of system memory. The 22-bit PCI channel offset is combined directly with the 18-bit channel physical base pointer to generate the 40-bit hypernode address.

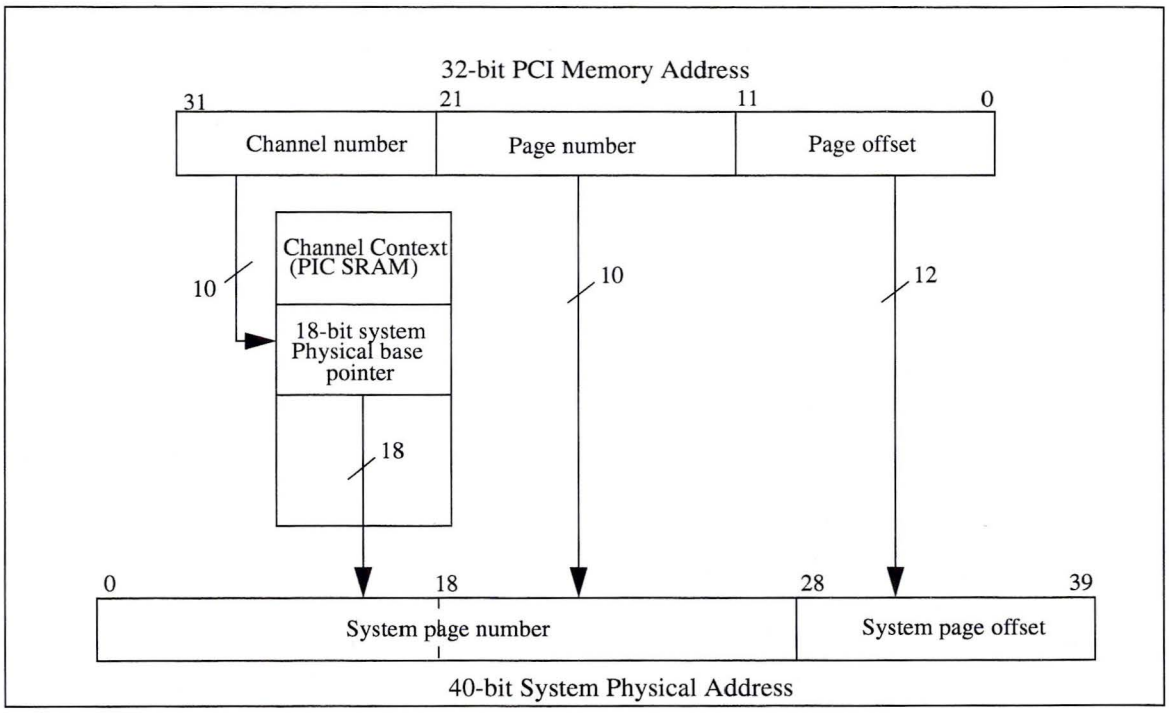


Figure 59 Physical mode address translation

Some I/O transfers, specifically remote receive transfers with many small I/O streams, need to be handled with a *nondeterministic* usage order. If each transfer were located in its own channel, software could run out of channels. If the transfers are packed into a single logical channel, the TLB miss overhead when switching streams would then limit the controllers throughput.

Software can pack remote receive buffers into a single physical channel and reduce the number of channels used, reduce the number of channel swaps, and eliminate TLB miss latencies.

Logical Address Translation

The more common way to map the 32-bit PCI address into the 40-bit hypernode address is using a logical translation mode channel. For logical translations, a level of indirection is used to

generate the 40-bit hypernode address from the 32-bit PCI address.

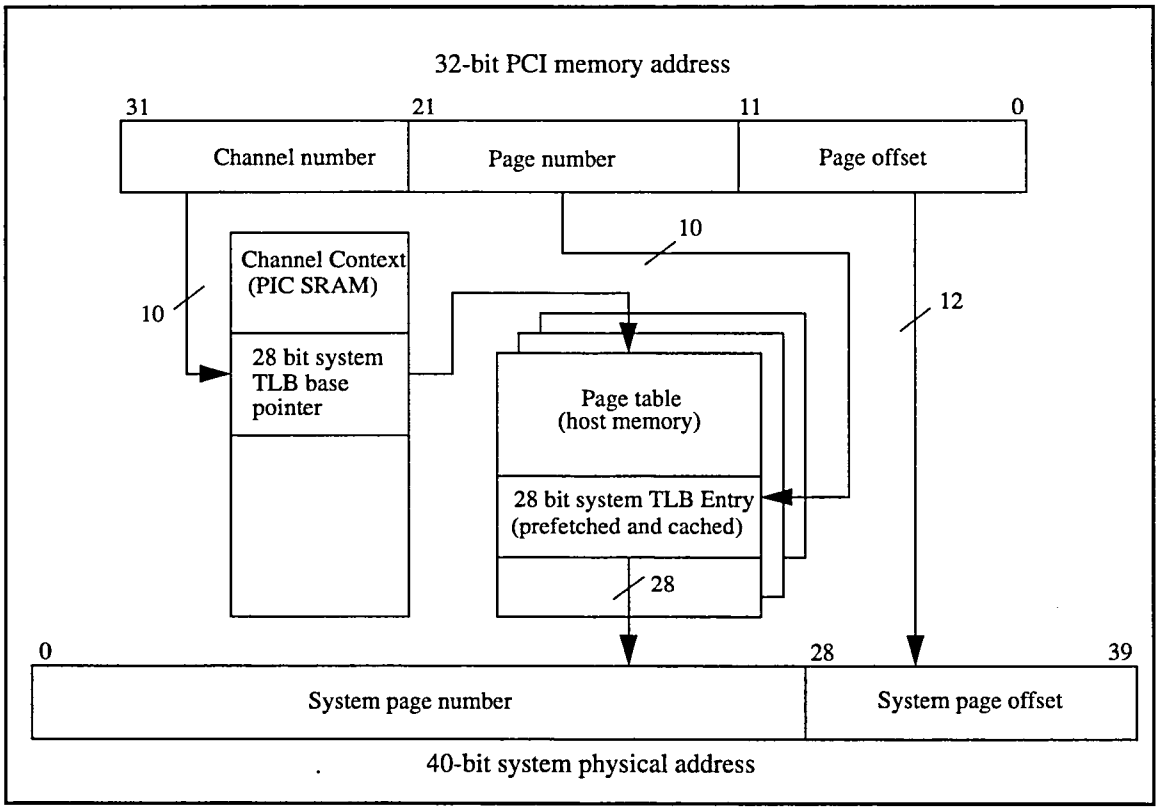


Figure 60 Logical mode address translation

The logical address translation mode uses translation tables much like the ones that hypernode processors use in that they are based on a translation lookaside buffer (TLB). The translation table converts the PCI bus addresses into system addresses on a page (four-KByte) granularity, and therefore, are aligned on page boundaries. A TLB base pointer points to the page of TLB entries in system memory. The PCI page number indexes into this table, pointing to a system page number. This system page number and the PCI page offset are combined to generate the 40-bit system address.

When using the logical translation mode, storing translation tables can cause demands on main memory to become quite high. To alleviate this burden, the logical translation mode is further divided into two submodes:

- Page tables are written directly into PIC TLB channel state (TLB_FE cleared)
- Page tables reside in main memory (TLB_FE set)

TLB fetch disabled

When TLB Fetch Enable (TLB_FE) is disabled, the PIC does not fetch TLB entries from main memory. Instead, software can initialize up to two TLB entries in the PIC channel TLB cache. The logical mode, (ATE is set to 1), without TLB prefetch, (TLB_FE is set to 0), is used for transfers spanning less than two pages. In these instances, a full four-KByte page table in memory would be a waste of valuable memory space.

TLB fetch enabled

When TLB_FE is enabled, the PIC uses the TLB base pointer to fetch the needed TLB entries from main memory as needed, storing them into the channel TLB cache. This cache is two entries deep and tagged with the page number. It normally contains the TLB entries for the current PCI page and current PCI page plus one more. As data transfers walk across page boundaries, that is, into the next page (page + 1), the PIC will fetch the next needed TLB entry (page + 2).

I/O TLB Entry Format

The I/O TLB entries in system coherent memory are the same as that used by the data mover. They are not the same as the processor TLB entries.

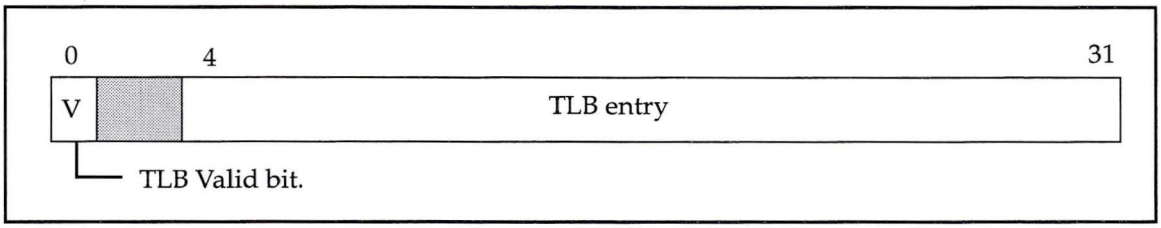


Figure 61 I/O TLB entry format

An I/O page table consists of 1,024 TLB entries. Each 28-bit TLB entry points to a four-KByte page of system coherent memory. Therefore, the table consumes four KBytes of system coherent memory. The channel TLB base pointer points to its channel page table. The PCI page number indexes the page table to address the needed TLB entry. See Figure 60 on page 123.

PCI memory read transfers

To handle long and variant system memory latencies, the PIC uses several different prefetch techniques in combination to ensure that the data needed by a controller is available at the time it is needed. These techniques include:

- Channel Prefetch/Refetch
- Device consumption-based prefetch
- Device stall prefetch

These techniques allow the PIC to:

- Cover the start-up latencies from memory.
- Maintain a minimal prefetch depth that matches the memory latency needed for prefetch data to the controller's consumption rate of that data.
- Increase prefetch depth dynamically, as needed, to account for larger latencies than the current depth of the prefetch buffer can hide.

Read data is coherent at the point that the request is satisfied in system memory. However, I/O transfers are not included in system memory sharing lists. Therefore, if the data is modified later, the PIC prefetched data will be stale. This fact dictates that direct memory access (DMA) from system memory be used only for buffered data that is defined prior to the PIC issuing any prefetches for that data. To purge prefetched data from the PIC prefetch buffers, the channel must be either reinitialized or rebuilt through the Channel Builder register.

To accommodate these prefetch techniques, the PIC provides two types of data prefetch storage:

- Channel Prefetch space
- Device Prefetch space

The channel prefetch space hides start-up latencies, and the device prefetch space maintains the streaming data. When a PCI transfer starts, data is supplied from the smaller channel prefetch buffer. Once that data from this buffer is exhausted, the data is pulled from the larger device prefetch buffer.

Channel prefetch space

Channel prefetch space stores channel prefetch data. This space hides the typical start-up latency for a local hypernode access when a controller switches from one channel to another. Only channel Prefetch/Refetch is stored in the channel prefetch buffer space.

There is one channel prefetch buffer per read channel for a total of 1,008 channel prefetch buffers. The amount of storage space needed to cover the start-up latency for a local hypernode access determines the depth of each channel prefetch buffer. The depth of the buffer is sized with the following formula:

Channel prefetch depth = PCI bandwidth * Local memory latency

Device prefetch space

Device prefetch space stores the prefetch data of a streaming device. It buffers a single controller stream of data from memory. The depth of the buffer is sized to hide latencies for local or remote hypernode block transfers (for X-Class servers) with minimal stalls on the PCI bus. A device consumption based prefetch or stall prefetch are stored in device prefetch space.

There is one device prefetch buffer per controller. The depth of the buffer is sized with the following formula:

Device prefetch depth = PCI bandwidth * Remote memory latency

Channel prefetch/refetch modes

For small transfer, high bandwidth controllers, the start-up latency dictates the effectiveness of the controller to move data. The start up latency is the time from which a controller provides the PIC a new address stream to the time the PIC provides the first data word.

The PIC provides a Channel Prefetch Enable (P) bit to hide controller start-up memory latencies. When enabled (P is set to 1), the PIC channel builder register prefetches data at channel initialization time. The prefetched data is stored locally in the channel space of the PIC external SRAM. Therefore, when a controller presents the PIC with an address mapping into this channel, the data is already local to the PIC, reducing the latency to first data word.

A controller that uses time-multiplexing on its read streams (for example, an ATM controller), can also be programmed with a Channel Refetch bit. When this bit is enabled (R is set to 1) and one channel is swapped for another, the PIC first refetches data into the channel prefetch buffer, starting where the controller left off. This guarantees that when the controller comes back to this address stream, the next needed data is available in channel prefetch space.

Device consumption-based prefetch

Each PIC can be connected to controllers with different bandwidth requirements. The PIC must insure that each controller has fair access to the system memory.

The PIC consumption-based prefetch algorithm keeps the prefetch request rate matched to the amount of data that a controller is consuming from the PIC. Each time a line of data is transferred across the PCI interface to a controller, a prefetch is scheduled for that PIC device prefetch buffer. This also ensures that the depth of the prefetch buffer is maintained at the minimal level that satisfies the consumption rate of the controllers, keeping the PIC from over prefetching for a particular controller.

Stall prefetch

If a controller needs data that is not available in the PIC device prefetch buffer (for example, when the controller address stream jumps outside the depth of the device prefetch buffer), the stall prefetch mechanism causes the PIC to issue a constant stream of data prefetches at a programmable interval until the critical line returns to the PIC. Once prefetch data is available, the prefetch algorithm reverts to the consumption-based prefetch algorithm.

PCI memory write transfers

The PIC has four independent write buffers, one per PCI controller. In order to minimize the write traffic to memory, a write buffer accumulates sequential bytes into a cache line of data prior to sending it to system memory. Any of the following events can cause the PIC to flush this write buffer to memory:

- The controller writes the last byte of a cache line.
- The controller writes a noncontiguous byte stream (a jump).
- A synchronization event forces a write pipe flush.

When the controller write buffer accumulates a cache line of data, a `WritePurge` operation flushes the line of data to memory. When the memory subsystem receives the data, it purges this line from all processor and CTI caches.

When a partial line needs to be flushed to memory, however, a `WritePurge` can not be used, since the current cache line in memory must be merged with the partial cache line of the PIC. The PIC provides a Write Purge Partial (W) mode bit per channel that defines how the PIC should perform this partial cache line merging.

Write_Purge_Partial disabled

If the Write_Purge_Partial bit is cleared, the nonwritten portion of the cache line in memory must be maintained coherently. Therefore, the PIC must perform a `Dflush_Alloc/Write_Mask` flow. In this flow, the PIC first issues a `Dflush_Alloc` to flush the line back to memory, locking down the line. When `Dflush_Alloc` is complete, the PCI issues the `Write_Mask` operation. This operation writes the data to memory with a mask to allow the memory subsystem to merge the two lines together and release the line in memory.

Write_Purge_Partial enabled

If the Write_Purge_Partial bit is set, the nonwritten portion of the cache line in memory is not guaranteed to be coherently maintained. This mode provides accelerated partial line transfers to system coherent memory. It is suitable for transfers like those to kernel buffers but is not suitable for I/O transfers directly to user space.

This write provides a mask that defines the data to be written to memory. The remaining bytes of the cache line come from what is currently in memory. When this data is received, the memory subsystem purges any users of the cache line.

I/O subsystem CSRS

The PIC is controlled by CSRs. All CSRs are aligned on a 64-bit basis and may only be accessed using noncoherent Read Short and Write Short packets. The PIC registers include:

- PIC Chip Configuration
- PIC PCI Master Configuration
- PIC PCI Master Status
- PIC Channel Builder
- PIC Interrupt Configuration
- PIC Interrupt Source
- PIC Interrupt Enable
- PCI Slot Configuration
- PCI Slot Status
- PCI Slot Interrupt
- PCI Slot Synchronization

PIC CSR address decoding

The PIC CSR address decoder looks at hypernode address bits [4:5] to determine the target address space (I/O or PIC CSRs) and bits [24:39] to index the CSR space. Accesses to unimplemented PIC CSR space return error responses to the requestor. Reserved addresses and bits ignore writes and return zeros on reads.

The PIC CSRs may be accessed by hypernode local addressing, the mapping of which is shown in Figure 62.

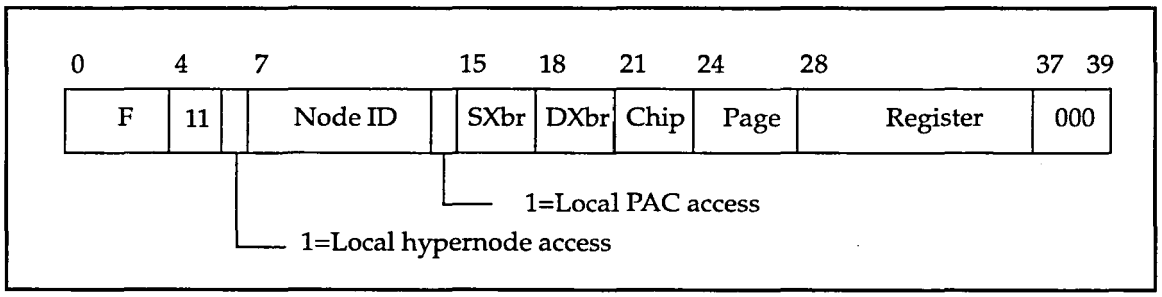


Figure 62 PIC CSR 40-bit address format

The bits and fields of the PIC CSR space address are as follows:

- ***Local hypernode access*** bit (bit 6)—Indicates that the access is to the local hypernode. When the local hypernode access bit is set, the hypernode ID field is ignored for purposes of intrahypernode routing.

- ***Local PAC access*** bit (bit 14)—Indicates that the access is to the local PAC space within the associated PAC chip. When this bit is asserted, the Node ID, SXbr, and DXbr fields are ignored and specific bits of the Page field are forced to the particular processor issuing the request.
- ***SXbr*** field (bits 15:17)—Routes a packet to the appropriate port on the crossbar on the source hypernode. This field is ignored for local hypernode accesses, but is needed to allow highly available systems that do not have full connectivity of all rings to send requests to remote hypernodes.
- ***DXbr*** field (bits 18:19)—Specifies which of the eight cross bar ports the request is to be routed to the destination hypernode.
- ***Chip*** field (bits 21:23)—Routes the packet to the appropriate chip at a crossbar port.
- ***Page*** field (bits 28:36)—Separates groups of CSRs into similar usage spaces.

PIC CSR definition

This section describes the PIC CSRs

PIC Chip Configuration register

There is one PIC Chip Configuration register on each PIC. It specifies configuration information.

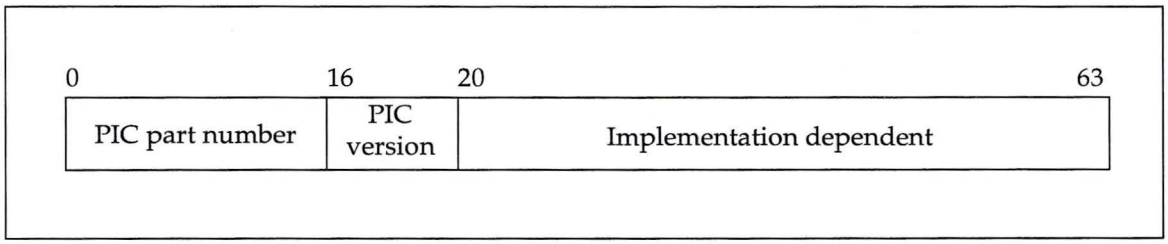


Figure 63 PIC Chip Configuration register definition

The fields of the PIC Chip Configuration register are defined as follows:

- ***PIC part number*** field (bits 0:15)—Specifies the part number for the PIC chip. A write is ignored and a read returns the hardwired value.
- ***PIC version code*** field (bits 16:19)—Specifies the version of the PIC chip. A write is ignored and a read returns the hardware value.
- ***Implementation dependent*** field (bits 20:63)—Specifies implementation dependent information. The value in this field should not be modified during normal use.

PCI Master Configuration register

There is one PCI Master Configuration register on each PIC that provides configuration information about the PCI Master interface. The format of the register is shown in Figure 64. All reserved fields are read as zero, and writes are ignored. All implemented fields are read with the last value written.

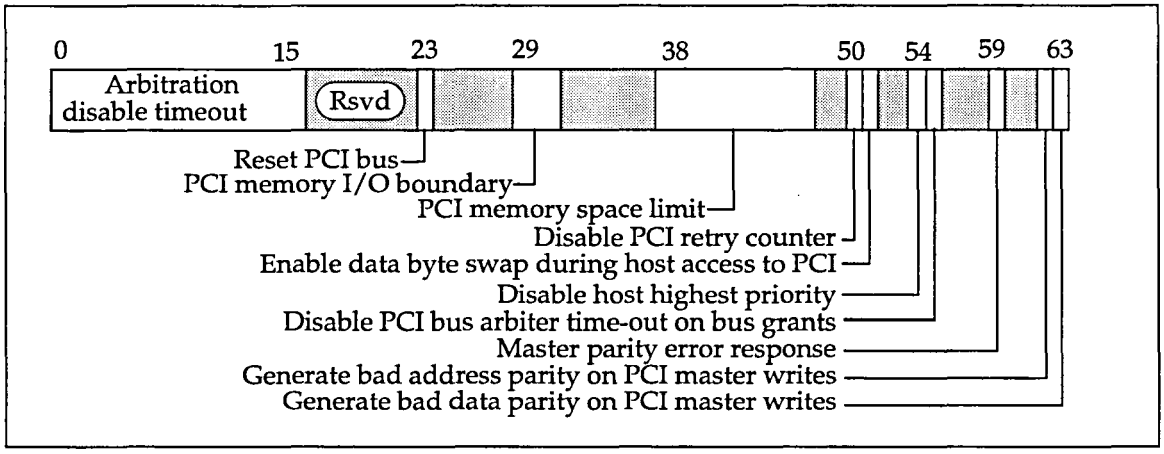


Figure 64 PCI Master Configuration register definition

The fields and bits in the PIC PCI Master Configuration register are defined as follows:

- ***Arbitration disable timeout*** field (bits 0:15)—Defines the PCI arbitration disable timeout threshold.
- ***Reset PCI bus*** field (bit 23)—Resets the PCI bus.
- ***PCI memory I/O boundary*** field (bits 29:31)—Controls the amount of hypernode local-I/O space mapped to PCI MEM and PCI I/O address space in contiguous 64MB blocks.

- ***PCI memory space limit*** field (bits 38:47)—Resets to 0x3f0 or 1008 decimal. The PIC reserves the upper 60MB (channels 1008-1022 inclusive) for PCI controller shared memory address space and the highest four MBytes (channel 1023) for PIC context SRAM. The PIC does not respond to PCI Controller DMA from channels 1022 down to the value stored in this register. This register value updates on write to the PCI Memory_I/O Boundary field, and is read-only.
- ***Disable PCI retry counter*** bit (bit 50)—Specifies the PCI retry counter is enabled.
- ***Enable data byte swap during host access to PCI*** bit (bit 51)—Causes host accesses to the PCI to swap data bytes.
- ***Disable Host highest priority*** bit (bit 54)—Indicates the host is participating in rotating priority with other devices.
- ***Disable PCI bus arbiter timeout on bus grants*** bit (bit 55)—Indicates that the PCI bus arbiter does not timeout on bus grants.
- ***Master Parity Error Response*** bit (bit 59)—Indicates that the PIC performs its normal operation when it detects a parity error as bus master.
- ***Generate bad address parity on master writes to PCI*** bit (bit 62)—Forces bad address parity out to PCI (for diagnostic use only).
- ***Generate bad data parity on master writes to PCI*** bit (bit 63)—Forces bad data parity out to PCI (for diagnostic use only).

PCI Master Status register

Each PIC has one PCI Master Status register. The register provides status information for the PCI Master interface. All fields are

cleared by a reset except the PIC Device Select Timing field; it is hardwired to the value one.

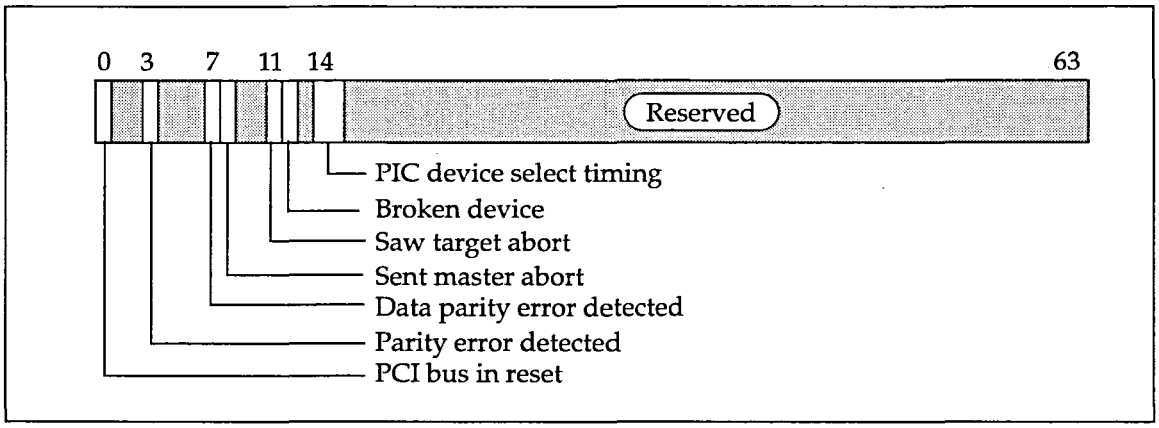


Figure 65 PCI Master Status register definition

The bit in the PIC PCI Master Status register are defined as follows:

- ***PCI bus in reset*** bit (bit 0)—Indicates that the PCI bus is reset.
- ***Parity error detected*** bit (bit 3)—Indicates that the PIC detected a parity error on incoming read data while the PIC was bus master.
- ***Data parity error detected*** bit (bit 7)—Indicates that a parity error was detected on the bus while PIC was PCI bus master.
- ***Sent master abort*** bit (bit 8)—Indicates that the PIC was master and sent a Master Abort (no target claimed the bus cycle). The Host receives an error response.
- ***Saw target abort*** bit (bit 11)—Indicates that the PIC was master and received a Target Abort. The Host will receive an error response.
- ***Broken device*** bit (bit 12)—Indicates that the PIC PCI interface received a grant and an *Idle* bus for 16 clocks but did not run a bus cycle. The PIC returns an error response to the requestor and sets the Master error in the Error Cause register.
- ***PIC device select timing*** field (bits 14:15)—Sets medium-speed address decode on PCI. This field is read-only.

PIC Channel Builder Register

There is one PIC Channel Builder Register on each PIC. The register sets up channel context in preparation for an I/O operation. The format of the register is shown in Figure 66. A write

will write the stored value to all fields, and a read will return the last written value. A reset clears all fields.

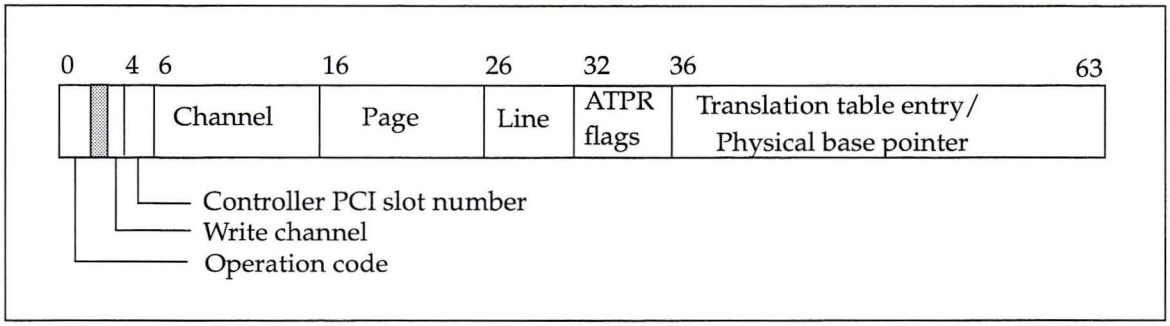


Figure 66 PIC Channel Builder register definition

The fields and bits in the PIC Channel Builder register are defined as follows:

- ***Operation code*** field (bits 0:1)—Determines the operation the channel builder will perform.
- ***Write channel*** field (bit 3)—Indicates a memory write channel. If set, a memory read channel is cleared.
- ***Controller PCI slot number*** field (bits 4:5)—Determines the PCI slot that this channel uses.
- ***Channel number*** field (bits 6:15)—Indicates the PCI channel to be built. The valid range is from 0 to the PCI memory space limit.
- ***Page number*** field (bits 16:25)—Indicates the 10-bit PCI page number.
- ***Line number*** field (bits 26:31)—Indicates the line number inside the page from which to start prefetch (only applies if a read channel, i.e. the Write Channel bit = 0).
- ***A - Address translation enable*** bit (bit 32)—Indicate the channel is in logical mode (translation on), if the A bit is set to a one value. If the A bit is set to a zero value, the channel is in physical mode (address translation off). This field also interprets the translation table base Pointer field.
- ***T -TLB fetch enable*** bit (bit 33)—If set, TLBs are fetched from memory; if T=0, only pre-cached TLBs are available.
- ***P -Prefetch/Write purge partial enable*** bit (bit 34)—Indicates that data prefetch starts at the same time as channel build for read channels. It indicates that write purge partials are enabled for a write channel.

- *R -Refetch* bit (bit 35)—Enables data refetch prior to a read channel being swapped out.
- *Translation table entry/Physical base pointer* field (bits 36:63)—Indicates PIC function as follows:
  - If the A bit is set to a one value and operation code is either a `Build` or `Init`, then the field is translation table base pointer. This 28-bit field points to the translation table base address where TLBs are fetched.
  - If the A bit is set to a one value and operation code is a `Prefetch`, this field is the 28-bit Translation Table Entry for the data prefetch.
  - If the A bit is set to a zero value the field is an 18-bit physical base pointer for this channel number, indexing into a four MByte physically contiguous block of memory.

A read from the Channel Builder register returns the current state. A write to the Channel Builder register causes channel state to be modified as defined by the written data.

PIC Interrupt Configuration register

The PIC has one PIC Interrupt Configuration register. The register specifies the interrupt number and processor when an interrupt occurs. The PIC forwards the interrupt by writing a value to a local processor EIRR register. Since the PIC can only send interrupts to one of the 16 processor EIRR registers on a local hypernode, only four bits of the address are programmable. All programmable fields are reset to zero.

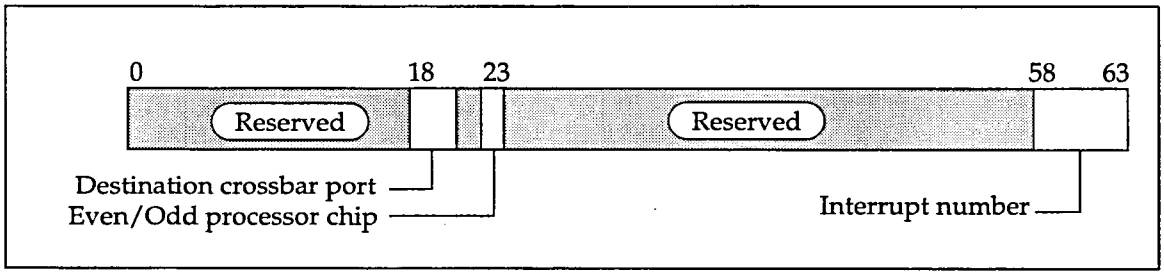


Figure 67 PIC Interrupt Configuration register definition

The fields and bits in the PIC Interrupt Configuration register are defined as follows:

- ***Destination crossbar port*** field (bits 18:20)—Indicates the crossbar port (and therefore which PAC) to which the interrupt will be sent.
- ***Even/Odd processor chip*** field (bits 23)—Specifies which of the two processors for the given PAC the interrupt is to be sent.
- ***Interrupt number*** (bits 58:63)—Indicates the processor External Interrupt register interrupt bit to be set.

PIC Interrupt Source Register

Each PIC has one Interrupt Source register that holds pending PIC interrupts. Source bits are set when the source of the interrupt occurs and remains set until cleared. Interrupts are accumulated regardless of the state of the enable. If the interrupt is enabled in the PIC Interrupt Enable register, then PIC sends an interrupt. If the interrupt enable is written to a one value while the interrupt is pending in the Interrupt Source register, the PIC generates an interrupt following the response to the register write.

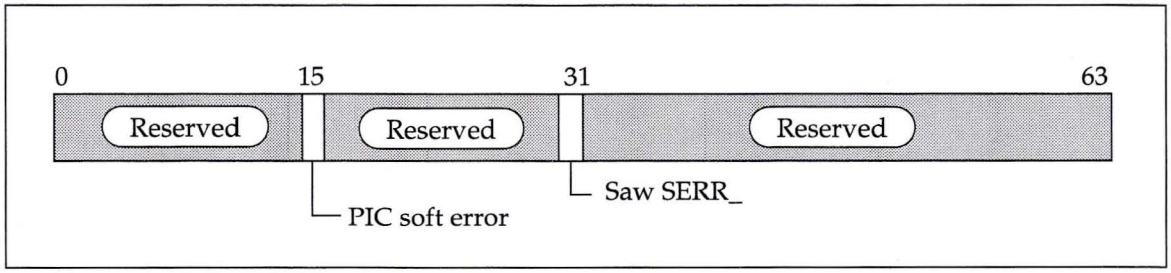


Figure 68 PIC Interrupt Source register definition

The bits in the PIC Interrupt Source register are defines as follows:

- ***PIC soft error*** bit (bit 15)—Indicates that a bit has been set in the Error Cause register that is configured as a soft error.
- ***Saw SERR_*** field (bit 31)—Indicates the PIC received an SERR_ on the PCI bus.

PIC Interrupt Enable register

PIC Interrupt Enable register has a bit for every source interrupt in PIC Interrupt Source. A one value in any PIC Interrupt Enable bit causes an interrupt when the corresponding source event in PIC Interrupt Source occurs. This register resets to zero (all

interrupts disabled). The format for this register is shown in Figure 69.

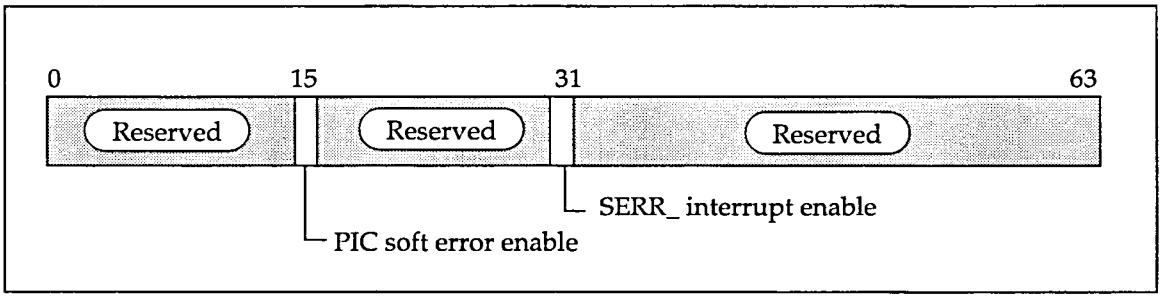


Figure 69 PIC Interrupt Enable register definition

The bits in the PIC Interrupt Enable register are defined as follows:

- ***PIC soft error enable*** field (bit 15)—Indicates that an interrupt can be sent when a soft error occurs. A value of one enables the interrupt.
- ***SERR_ Interrupt Enable*** field (bit 31)—Indicates that an interrupt can be sent when a PCI SERR_ occurs.

PCI Slot Configuration register

There are four PCI Slot Configuration registers, one for each supported PCI expansion slot. These registers provide control of the PCI interface.

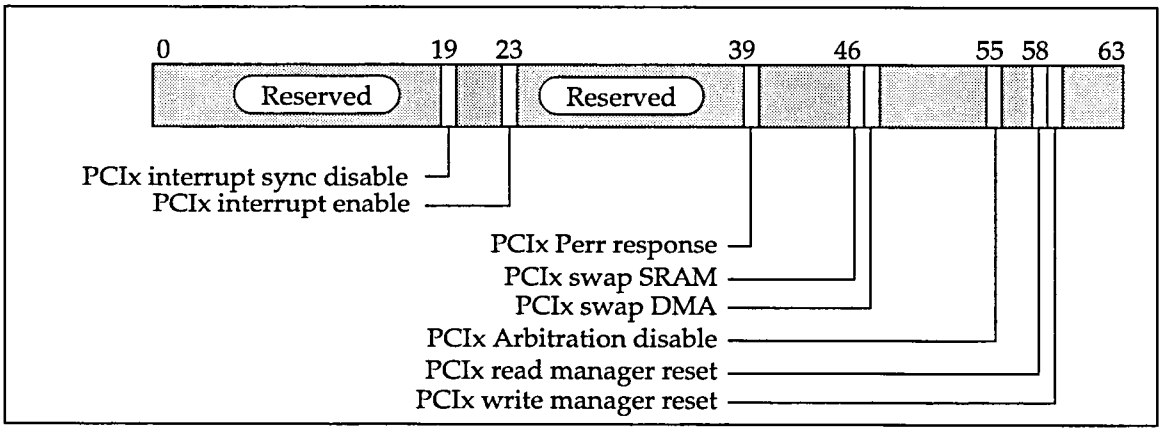


Figure 70 PCI Slot Configuration register definition

The bits in the PCI Slot Configuration register are defined as follows:

- ***PCIx interrupt synchronization disable*** bit (bit 19)—Disables the synchronization of a device on interrupt.
- ***PCIx interrupt enable*** bit (bit 23)—Enables the forwarding of a device interrupt.
- ***PCIx Perr response*** bit (bit 39)—Enables PCI_PERR_ data parity error signalling.
- ***PCIx_Swap SRAM*** bit (bit 46)—Enables byte swapping on PCI*x* shared memory transfers.
- ***PCIx_Swap DMA*** bit (bit 47)—Enables byte swapping on PCI*x* DMA transfers.
- ***PCIx Arb Disable*** bit (bit 55)—Disables bus arbitration for PCI*x*.
- ***PCIx Read Manager Reset*** bit (bit 58)—Resets PIC Read manager *x*.
- ***PCIx Write Manager Reset*** bit (bit 59)—Resets PIC Write manager *x*.

PCI Slot Status register

Each PIC has four PCI Slot Status registers that specify the status of slot specific events.

Bits in these registers are set when PIC is the target of one of the four controllers and a status event occurs. All writable fields are reset to the value zero.

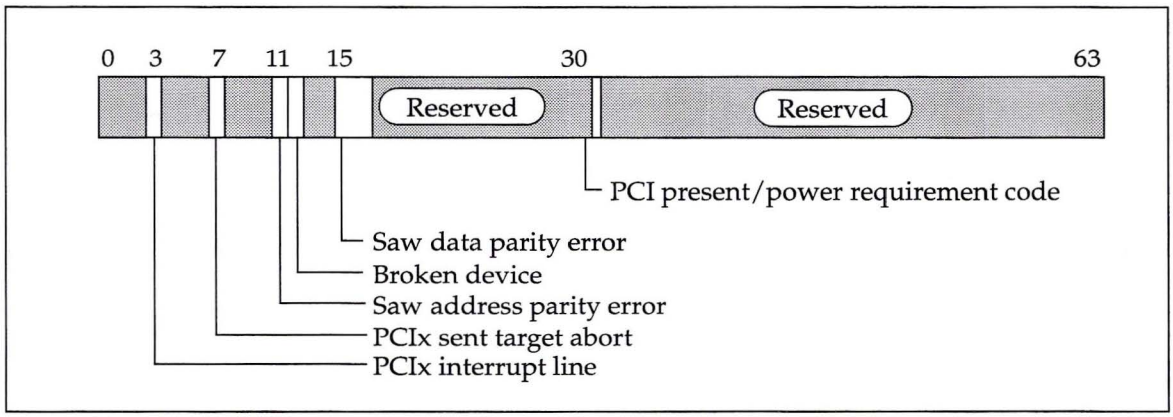


Figure 71 PCI Slot Status register definition

The fields and bits in the PCI Slot Status register are defined as follows:

- ***PCIx interrupt line*** bit (bit 3)—Indicates the current state of the PCI*x* INTA_ line (read only).
- ***PCIx sent target abort*** bit (bit 7)—Indicates that the PIC sent this slot a Target Abort bus cycle termination. This bit does *not* set the PCI Controller *x* bit in the Error Cause register.
- ***Saw address parity error*** bit (bit 11)—Indicates that the PIC detected an address phase parity error on a transfer from this slot. The PIC terminates the transfer with target abort and relies on the transaction master to report the error to software. This bit sets the PCI Controller *x* bit in the Error Cause register.
- ***Broken device*** bit (bit 12)—Indicates this slot received a grant during an idle bus for 16 clocks but did not run a bus cycle.
- ***Saw data parity error*** bit (bit 15)—Indicates the PIC (as a target) detected a PCI data phase parity error on incoming (write) data from this slot. This bit does not *not* set the PCI Controller *x* bit in the Error Cause register.
- ***PCI card present/Power requirements code*** field (bits 30:31)—Indicates a PCI controller is present and the power requirements of that controller. This field is read only.

PCI Slot Interrupt Configuration register

Each PIC has four PCI Slot Interrupt Configuration registers specify the interrupt number and processor when an interrupt occurs on the corresponding PCI slot. The PIC forwards the interrupt by writing a value to a local processor EIRR register. Because the PIC can only send interrupts to one of the 16 processor EIRR registers on a local hypernode, only four bits of the address are programmable. All programmable fields are reset to zero.

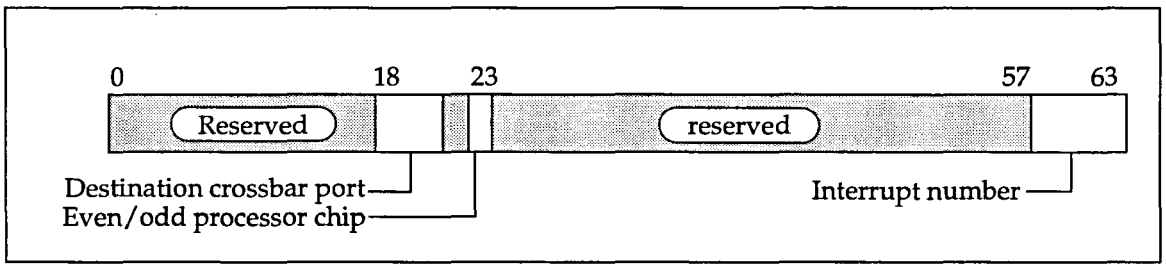


Figure 72 PCI Slot Interrupt Configuration register definition

The fields and bits in the PIC PCI Slot Interrupt Configuration register are defined as follows:

- ***Destination crossbar port*** field (bits 18:20)—Determines to which crossbar port (and therefore which PAC) the interrupt will be sent.
- ***Even/Odd processor chip*** field (bits 23)—Specifies which of the two processors for the given PAC the interrupt is to be sent.
- ***Interrupt number*** field (bits 58:63)—Specifies the processor External Interrupt register interrupt bit to be set.

PCI Slot Synchronization register

The PIC has four PCI Slot Synchronization registers, one for each of the four PCI bus slots. Software polls these registers to determine when the write pipe has been flushed.

A processor performs a read to these registers to synchronize the write pipe for the corresponding device. The registers are read-only and the CSR interface returns zero status after the requested device operation completes. The format of the PCI Slot Synchronization register is shown in Figure 73.

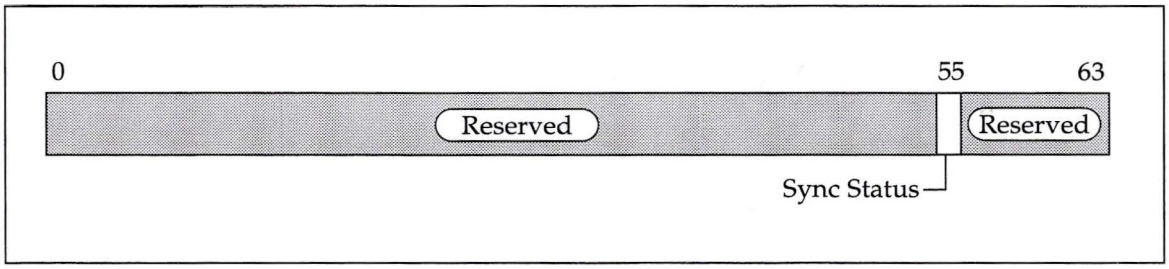


Figure 73 PCI Slot Synchronization register definition

The fields and bits in the PCI Slot Synchronization register are defined as follows:

- ***Synchronization status*** bit (bit 55)—Specifies the completion status of the device write manager for the corresponding slot.

Byte swapping

In order to address different byte ordering between the PCI bus and the rest of the system, the PIC provides CSR-configurable bits to define how to handle byte ordering of data crossing from one domain into the other. The CSRs configure byte swapping on the following data paths:

- PCI read and write of system coherent memory on a per controller basis via the PCI Slot Configuration register (see the "PCI Slot Configuration register" section on page 137).
- PCI read and write of shared memory on a per controller basis via the PCI Slot Configuration register.
- Host read and write of PCI IO, Memory, and Configuration space via the PCI Master Configuration register (see the "PCI Master Configuration register" section on page 131).

Performance monitors

8

Performance factors

Application performance in the S-Class and X-Class servers depends upon several factors. These systems provide ways to measure each of the principal factors. The measurements indicate the overall results and provide data that enables programmers to identify algorithmic changes that provide improvement to overall performance. Some of the important factors, and concepts for their measurement, are:

- Cache-hit rate—Algorithms must optimize cache use to perform well. Consequently the S-Class and X-Class servers provide data on overall hit ratios, hit ratios per processor, hit ratios over time, cache miss trace data, and data that tells which program statements are causing the most misses.
- Communication costs—Communication costs include communication within and across hypernodes (in X-Class servers). Concerns in communication include the following:
  - Ratio of local-to-remote memory usage
  - Memory access patterns of particular code sections
  - Topology of how the application is laid out across many hypernodes
  - Communication costs between threads
- Efficient parallel algorithms—Parallel algorithms pose both validity and performance problems for the programmer, and often prove more difficult to debug than single-threaded applications. Useful tools include:
  - Trace data correlated between threads
  - Deadlock detection
  - Synchronization statistics
  - Measurement of effective parallelism
  - Granularity measurements of parallel regions
  - Lock order enforcement

- IO performance—The largest IO factor is data transfer rates in the disk subsystem. Of particular interest are peak measurements, as most IO is done in a burst mode. Also, information about disk access patterns should be available.

The S-Class and X-Class servers provide a means of measuring the key parameters in each of the above areas. The process of measuring performance is an intrusive process and can impact the system performance.

Performance monitor hardware

S-Class and X-Class servers provide registers to record events and enable performance measurement. These registers include:

- The PCX-U interval timer
- The time-of-century clock (TIME_TOC)
- The performance-monitor set for each processor

Interval timer

Each processor has a 32-bit timer in control register 16. These timers count at a frequency between twice the peak instruction rate and half the peak instruction rate. They are not synchronized, nor can they be loaded by software. The timers may optionally generate an interrupt when the count reaches a certain value.

These timers are used for:

- Generating periodic clock interrupts to the processors for scheduling purposes.
- Measuring fine granularity time intervals within processors independent of other processors. An example of such a measurement would be the length of time necessary to execute a code segment.
- Implementing thread timer register (TTR) in software. The operating system allows user-read access to this timer in order to use the TTR.

Time-of-Century clock

S-Class and X-Class servers provide facilities for parallel program development that include very large memory and a Time-of-Century Clock (TIME_TOC) register. Together they provide low-overhead, time-stamped trace data stored within the hypernode. For X-Class servers, the time-stamped trace data from each hypernode can be merged in a post processing step to provide an accurate global picture, with event sequences in the 5-10 microsecond range.

The TIME_TOC also provides time-stamping of transmitted messages. The receiver can determine the transmission time by subtracting time-stamp from its current time.

S-Class and X-Class servers also allow virtual-time measurement. Maintenance of process virtual time is expensive in software, because the number of executing threads is adjusted frequently (as a result of each system call, for example). Accurate process virtual time can be compared with the total thread CPU time to get an approximation of parallel speedup.

TIME_TOC implementation

Each PAC has a 64-bit TIME_TOC register. It is accessed with a single 64-bit read, making latency to the TIME_TOC register significantly less than on earlier Exemplar SPP1000-series systems.

Hypernode core logic generates a 16-Mhz clock for the TIME_TOC logic on each PAC. The PAC synchronizes the 16-Mhz clock to its own clock and generates a TIME_TOC clock every seven or eight PAC clocks. The TIME_TOC logic generates a synchronization pulse every 256 TIME_TOC clocks.

A single PAC acts as the TIME_TOC synchronization master of the hypernode (all other PACs on that hypernode have their outputs in high impedance mode). The master PAC sends its synchronization pulse to the non-master PACs.

In X-Class servers, the pulse is distributed to the other hypernodes by setting a bit in the header of CTI packets. The amount of time required to distribute the TIME_TOC pulse is a function of system size. Each TAC has a register that specifies the source for the TIME_TOC pulse (TIME_TOC synchronization signal, CTI incoming X link, or CTI incoming Y link). The register also specifies whether the TIME_TOC pulse should be propagated to the CTI outgoing X link and/or the CTI outgoing Y link.

Logic ensures that the TIME_TOC registers maintain synchronization within their specified resolution. It checks to ensure the time between TIME_TOC synchronization pulses is in the range of TIME_TOC synchronization period plus or minus one-half the TIME_TOC synchronization resolution. If it detects a TIME_TOC synchronization pulse that occurs early or late, it sends an interrupt to one of the processors connected to the PAC.

When the resolution field of the PAC TIME_TOC configuration register has the value zero (TIME_TOC off), the TIME_TOC synchronization pulse checking is disabled.

PAC TIME_TOC configuration register

The format of the PAC TIME_TOC configuration register is shown in Figure 74.

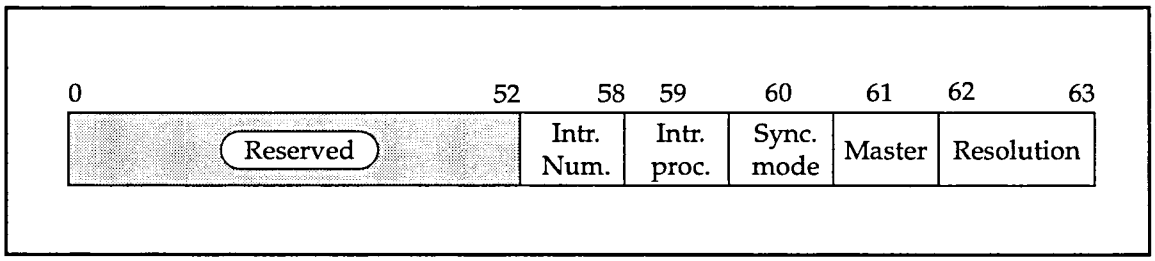


Figure 74 PAC TIME_TOC configuration register definition

The bits and fields of the PAC TIME_TOC configuration register are defined as follows:

- ***Interrupt number*** field (bits 53:58)—Specifies the interrupt number when sending an interrupt to one of the two processors when a synchronization pulse check problem is detected. The field is not initialized by reset.
- ***Interrupt processor*** bit (bit 59)—Specifies the processor to which an interrupt is sent when a synchronization pulse check problem is detected. The field is not initialized by reset.
- ***Synchronization mode*** bit (bit 60)—Indicates whether the synchronization pulse starts incrementing or synchronizing the TIME_TOC and prescale registers. At reset the field is cleared and indicates that the next synchronization pulse received will start incrementing the TIME_TOC and prescale registers. The reception of a synchronization pulse sets this bit. When the synchronization mode field is set, the reception of a synchronization pulse causes the prescale and least significant bits of the TIME_TOC register to be rounded.
- ***Master*** bit (61)—Specifies that the PAC is the TIME_TOC master. The PAC generates and delivers the synchronization pulse to the other PACs. A value of one enables this operation. The field is reset to the value zero.
- ***Resolution*** field (bits 62:63)—Specifies the TIME_TOC register resolution. The field is reset to the value zero (TIME_TOC off). Table 26 shows the supported resolutions.

Table 26 TOC register resolutions

Value	Resolution
0	TIME_TOC off
1	1 µSec
2	2 µSec
3	4 µSec

Avalue of zero (TIME_TOC off) disables TIME_TOC synchronization pulse checking.

PAC TIME_TOC Clock register

The TIME_TOC Clock register is a 64-bit, read-write register of which only the least significant 48 bits are implemented (48 bits supports an uptime of 8.9 years). Read access supports normal operation, and write access supports initialization as well as testing.

The TIME_TOC register increments its 48-bit value when the pre-scale register reaches the value 0xF. The format of the PAC TIME_TOC register is shown in Figure 75.

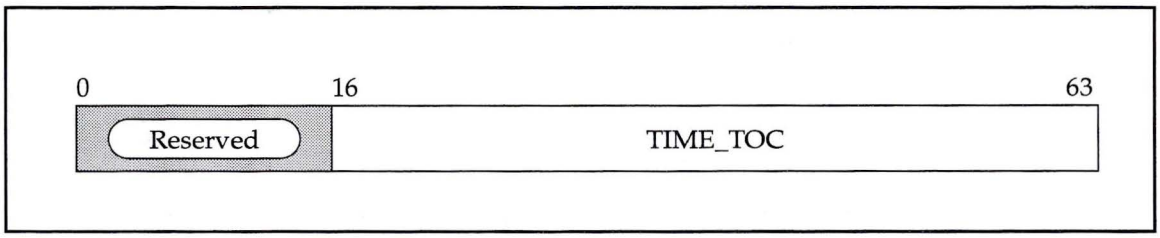


Figure 75 TIME_TOC Clock register definition

The ***TIME_TOC*** field (bits 16:63) increments each time the prescale logic has the value of 0xF. The register is accessible using a 64-bit CSR read or write. Reset does not effect this register. The least significant two bits may be rounded up or down when a synchronization pulse is received, depending on the resolution selected for the TIME_TOC logic.

TAC TIME_TOC configuration register

The physical addresses for the TIME_TOC Configuration register is 0xFF FC05 0300. The format of the TAC TIME_TOC configuration register is shown in Figure 76.

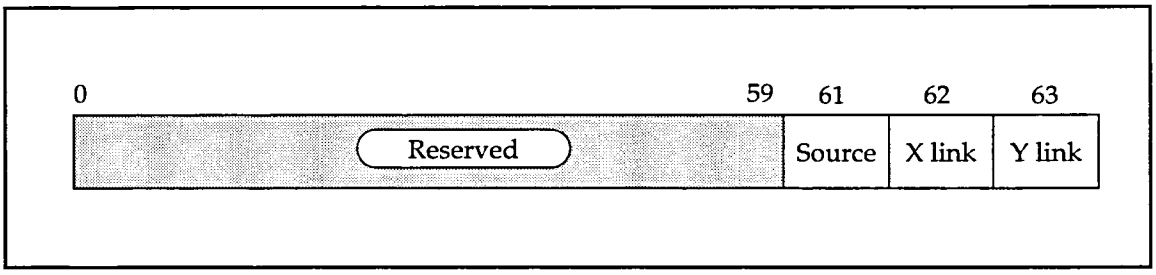


Figure 76 TAC TIME_TOC configuration register definition

The bits and fields of the TAC TIME_TOC configuration register are defined as follows:

- *Source* field (bits 60:61)—Specifies the TIME_TOC synchronization pulse inputs (TIME_TOC synchronization signal, X incoming link, or Y incoming link) should be propagated to the enabled TIME_TOC synchronization pulse outputs. Table 27 shows the source field selection values.

Table 27 TIME_TOC synchronization pulse source selection

Value	Selection
0	None
1	TOC synchronization signal
2	X CTI link
3	Y CTI link

The Source field is cleared by reset.

The TIME_TOC synchronization pulse propagates to the local hypernode (driven to the TIME_TOC synchronization signal) when the Source field is set to X CTI link, or Y CTI link.

- *X link* bit (bit 62)—Enables a synchronization pulse from the source selected by the Source field to be propagated to the X CTI outgoing link.
- *Y link* bit (bit 63)—Enables a synchronization pulse from the source selected by the Source field to be propagated to the Y CTI outgoing link.

TIME_TOC reset and initialization

Reset has the following effect on the TIME_TOC logic:

- The synchronization mode bit of the PAC TIME_TOC configuration CSR register is cleared, forcing the prescale register to the value zero.
- The TIME_TOC register is inhibited.
- The master field of the PAC TIME_TOC configuration register and the source field of the TAC TIME_TOC configuration register are cleared. With these two fields set to zero, the TIME_TOC synchronization pulse is disabled.
- The resolution field of the PAC TIME_TOC configuration CSR is cleared, disabling the TIME_TOC synchronization checking logic.

Performance monitoring counters

The PA-8000 processor has extensive performance monitoring capabilities on board. It does, however, require a minimal amount of additional hardware.

Each PAC has a set of five performance monitoring counters for both processors that includes one latency counter and four memory access type event counters. Coherent memory accesses fall into one of the following four classes:

- Local memory with no remote hypernode access required—The data resides within the memory of the local hypernode.
- Local memory with a remote access required—The data has migrated to another hypernode and requires CTI assistance.
- CTI cache with no remote hypernode access required—There is a CTI cache hit.
- CTI cache with remote hypernode access required—The data is not found in the CTI cache and requires CTI assistance.

The four event counters are grouped in pairs and concatenated into a two 64-bit register, thereby requiring only two accesses instead of four to read all four counters.

Latency counter

The latency counter is a 40-bit read-write register. It increments by the number of outstanding processor data reads (0 to 10) at the system clock frequency. The PAC has two latency registers, one for each processor. Figure 77 shows the bit definition of the

Performance Monitor Latency P*n* register counter (where *n* is the processor number).

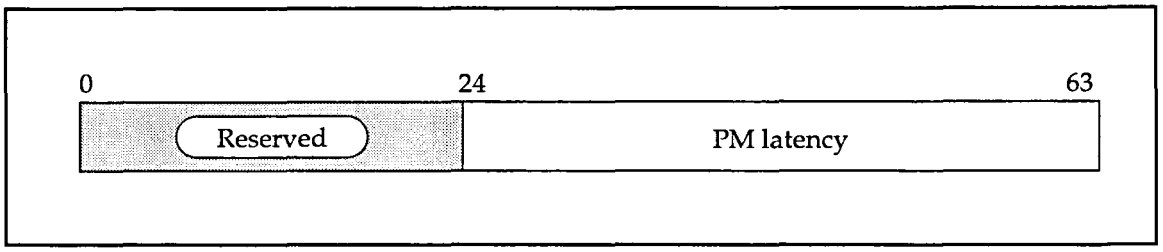


Figure 77 PAC Performance Monitor Latency register definition

The ***PM latency*** field (bits 24:63) accumulates the total latency of all coherent read requests.

Event counters

X-Class servers have four memory access event type counters corresponding to the four types of memory accesses. These counters are 32-bit read-write registers. The two local memory counters are concatenated for CSR access. The same is true for the two CTI cache counters, making two 64-bit registers for each of the processors on a PAC. S-Class servers do not use the CTI cache counters, only the memory counters

Two bits of performance information are returned with every ownership reflection. These two bits, along with a valid bit, indicate the type of memory access. Table 28 shows the meaning of the read access type bits.

Table 28 Read access type information

Field value	Read Access Type
00	Local memory, No remote hypernode access required
01	Local memory, Remote hypernode access required
10	CTI cache, No remote hypernode access required
11	CTI cache, Remote hypernode access required

The appropriate counter increments when these bits appear on the runway bus.

Figure 78 shows the two local memory counters in the Performance Monitor Memory Access Count register.

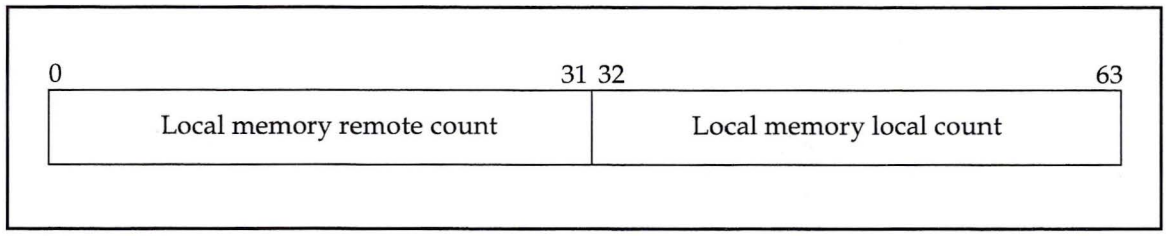


Figure 78 PAC Performance Monitor Memory Access Count P*n* register definition

Figure 79 shows the two CTI cache counters in the CTI Access register. These registers are not used in S-Class servers.

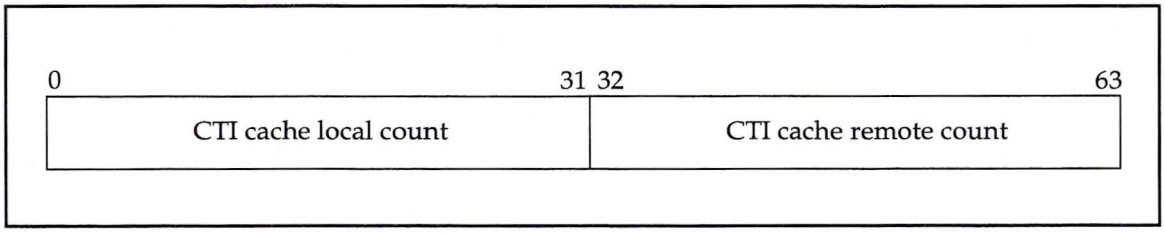


Figure 79 PAC CTI Access P*n* register definition

The ***CTI cache*** fields (bits 0:31, 32:63) accumulate the number of CTI cache accesses.

Hypernode utilities

9

Each hypernode has a section of hardware known as the hypernode utilities, centered around the Utilities board. On this board are two FPGAs: the PUC and the MUC. The PUC FPGA provides the utilities board a means to send interrupts and error messages to the processors and to receive control messages from the processors. The MUC FPGA performs all environmental monitoring. The utilities board connects to all hypernode PACS through the core logic bus.

Overview

The utilities board handles all of the hypernode housekeeping chores. It connects directly to the midplane board where it attaches to the core logic bus, the environmental sensors, and other test points. The board also interfaces to the hypernode liquid crystal display (LCD), an optional test terminal (via an ethernet connection), and other external devices.

Figure 80 on page 155 shows the utilities board functional layout.

The heart of the utilities board is the core logic. This section of hardware connects internally with the MUC for receiving environmental interrupts and to the PUC as an interface to the core logic bus. The core logic contains initialization and booting firmware. It also interfaces to the LCD and to serial RS232 links, as well as to ethernet links. An optional terminal can be connected via these links to run diagnostics and configure the hypernode.

The MUC latches hypernode interrupts, most of which are from environmental sensors located throughout the hypernode. The MUC and the power-on circuit together control system power-up. The MUC interfaces to a light-emitting diode (LED) diagnostic display through the power-on circuit.

The PUC provides the core logic an interface to the core logic bus. There are actually two buses, each one connects up to four PACs. The PUC communicates to the PACs using data packets.

The JTAG (Joint Test Action Group) interface supports a test station and a mechanism to fanout JTAG to all the boards in a hypernode. It is used only for testing.

S-Class and X-Class servers use a test method called scanning to test boards and other hardware units. With the test station connected to the ethernet between hypernodes, you can scan any hypernode in a multiple hypernode system without moving connectors.

The JTAG Interface contains a microprocessor to capture packets from the ethernet and apply them to the JTAG test bus controller or to take scan information from the JTAG test bus controller and send it out on the ethernet. The test station can also read and write every CSR in the hypernode.

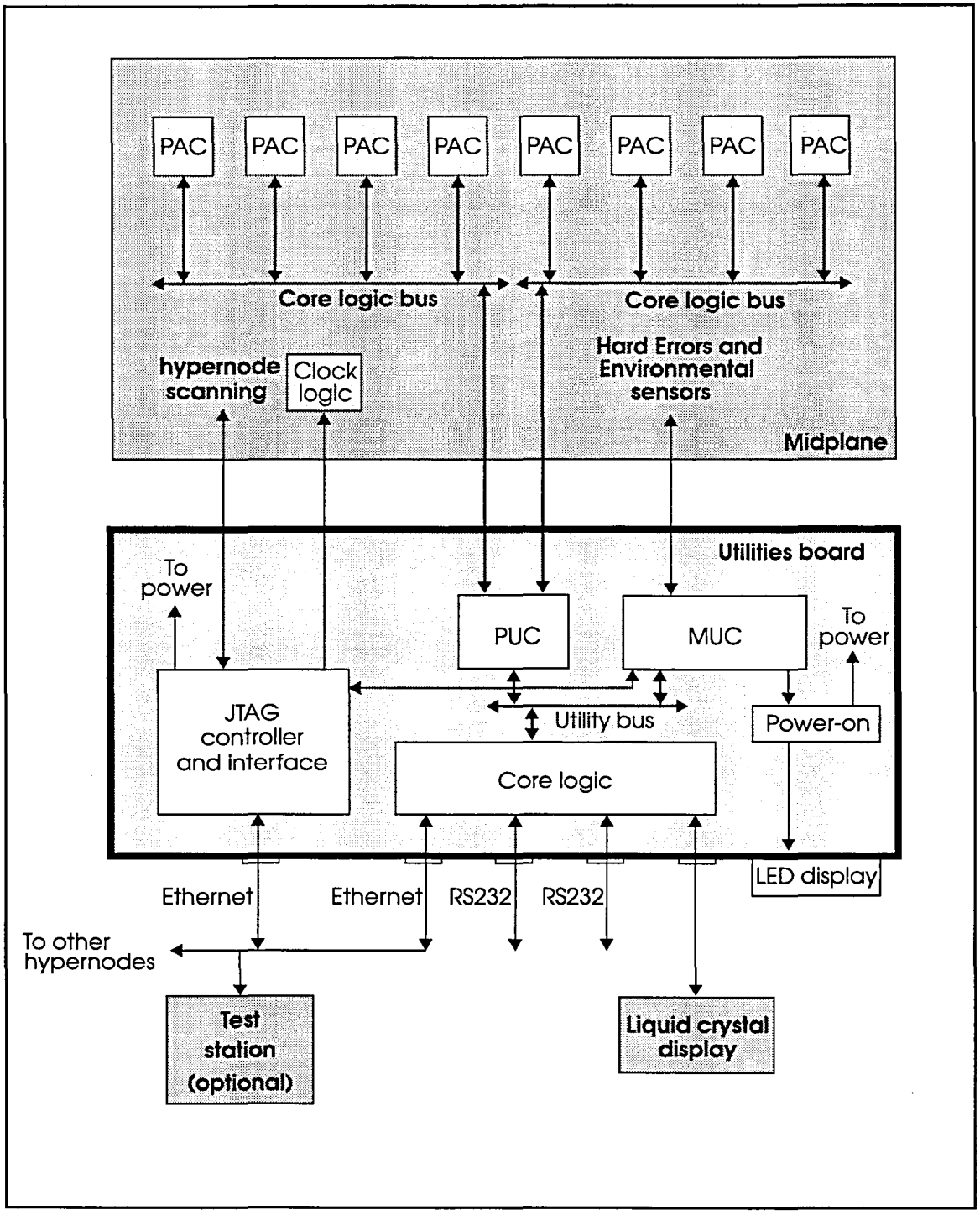


Figure 80 Hypernode utilities board

Core logic

This section describes the core logic bus and core logic hardware functions.

Flash memory

The core logic contains nonvolatile storage for processor-dependent code. This code consists of primary loader code, the Open Boot PROM (OBP) code, and power-on self test software (see Chapter 10, "Booting and testing"). This EEPROM memory is 2 MBytes, configured as 1 million addresses by 32 data bits with only 32-bit read and write accesses allowed. It is writable by the processors for field upgrades and can be written when the PUC is *scanned*.

Nonvolatile static RAM

The core logic section contains a nonvolatile battery-backed static RAM (NVSRAM). The NVSRAM is used to write system log information (failures) and store configuration information. This RAM is byte addressable and can be accessed even after power failures occur.

DUART

The utilities logic contains a Dual Universal Asynchronous Receiver-Transmitter (DUART). One port, configured as a basic RS232 port, provides an interface to the simplest core system functions. With this interface, you can connect a terminal as a local console to analyze problems, reconfigure the system, or provide other user access.The parallel port of the DUART drives the LCD. The second RS232 port can be used for a modem for field service.

RAM

RAM is needed to support the simple core system functions. When the hypernode powers up, the processors operate out of this RAM. They run self test software to test and configure the rest of the hypernode. Once the system is fully configured, the processors execute out of main memory. The RAM is byte addressable and is 128 KBytes, configured as 32K addresses by 32 data bits (with parity).

Console ethernet

The ethernet I/O port connects to another optional system console that has an ethernet port. You can use the console for initializing, testing, and troubleshooting the hypernode.

LEDs and LCD

Light Emitting Diodes display environmental information, such as the source of an environmental error that caused the utilities board to power down the hypernode.

The LCD is driven by one of the processors via the utilities board. A large amount of information can be displayed on the LCD. The core logic drives the LCD via the parallel port on the DUART.

COP interface

COP chips (serial EEPROMs) are located on the major boards with information such as serial number, error history, configuration information, and so on. The MUC connects to the COP bus selector (CBS) chip on the midplane and allows the system to read any COP in a hypernode.

PUC

The PUC provides the utilities board a means to apply interrupts and error messages to the processors and to receive control messages from the processors. It has two 18-bit, bidirectional buses. Each interface connects up to four PACs. The PUC provides core logic bus arbitration for the sixteen hypernode processors.

Through the PUC, the PAC has an interface to the utilities bus on the utilities board. This bus connects the PUC, the MUC, and the core logic section together.

PUC Processor Agent Exist register

The PAC Exist register indicates which PACs exist in a hypernode. During reset, all PACs assert their REQ lines. This sets corresponding bits in this register. The PUC ignores the REQ lines (with respect to core logic bus requests) approximately eight clocks after reset to allow the PACs to change from "exist" mode to "request" mode.

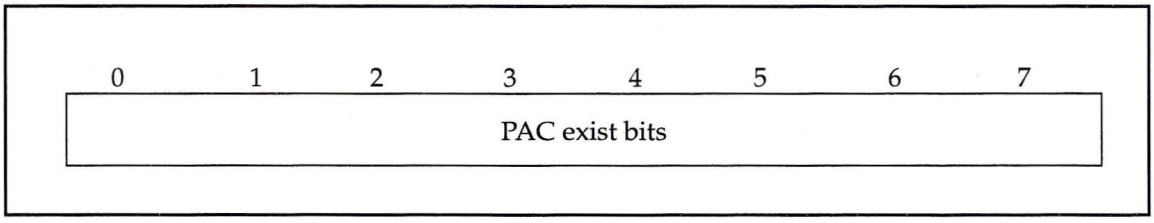


Figure 81 PUC processor agent exist register definition

A value of one on any bit indicates that the respective PAC *0-7* is active and exists.

PUC revision register

The Revision register indicates the revision level of the PUC FPGA.

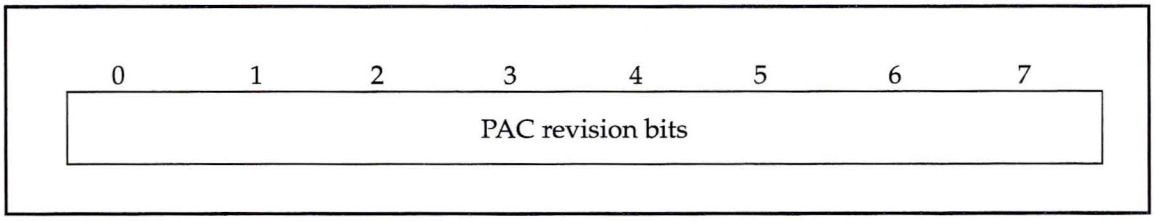


Figure 82 PUC revision register

Revision bits are read to determine the revision of the PAC.

MUC and Power-on

The MUC performs all environmental monitoring on the utilities board. It attaches to the utilities bus so that processors can monitor the hypernode by accessing these CSRs.

The MUC works in conjunction with a hardware section on the utilities board known as the power-on circuit. This circuit controls powering up the entire hypernode. It is able to operate when the rest of the hypernode is powered off or in some indeterminate state. It drives the environment LED display which is a basic (minimal hardware, no software) indication of what environmental error caused the utilities board to power down the hypernode.

The Test Station can also read the environmental LED display.

Environmental monitoring functions

The MUC and the power-on circuit monitor the following environmental conditions:

- ASIC installation error sensing
- FPGA configuration and status
- Thermal sensing
- Fan Sensing
- Power failure sensing
- 48-V failure
- 48-V maintenance
- Ambient air temperature sensing
- Power-on

Table 29 Environmental conditions monitored by the MUC and power-on circuit

Condition	Type	Action
ASIC Not Installed OK	Environmental error	Power not turned on, LED indication
FPGA not OK	Environmental error	Power not turned on, LED indication
48-V Fail	Environmental error	Power turned off, LED indication
Midplane power fail	Environmental error	Power turned off, LED indication
Board over temp	Environmental error	Power off in one second, LED indication, Interrupt
Fan not turning	Environmental error	Power off in one second, LED indication, Interrupt

Table 29 Environmental conditions monitored by the MUC and power-on circuit—**(continued)**

Condition	Type	Action
Ambient air hot	Environmental error	Power off in one second, LED indication, interrupt
Other power fail	Environmental error	Power off in one second, LED indication, interrupt
Ambient air warm	Environmental warning	LED indication, interrupt
48-Volt maintenance	Environmental warning	LED indication, interrupt
Hard error	Hard error	LED indication, interrupt

Environmental condition detected by power-on function

The power-on function detects environmental errors (such as ASIC Install or FPGA Not OK) immediately and does not turn on power to the hypernode until the conditions are corrected. It also detects environmental errors such as 48-V Fail while the system is powering up and Midplane Power Fail after the system has powered up. If a failure is detected in these two cases, the power-on circuit turns off power to the system.

Environmental warnings such as 48-Volt Maint are also detected by the power-on circuit. It applies these to the MUC which then sends an environmental warning interrupt to the hypernode processors.

In all cases, the power-on circuit lights an environmental LED display code. The environmental LED display code is prioritized so that it only displays the highest priority error or warning. See the "Environmental LED display" section on page 161 for a list of codes.

Environmental conditions detected by MUC

The MUC detects most of the environmental conditions. It samples error conditions during a time period derived from a local 10-Hz clock that drives the power-on circuit. It registers all the environmental error conditions twice and then ORs them together. If the conditions persist for 200 mS, the environmental error bit is set, and an environmental error interrupt is sent to the PUC, which sends it on to the processors. The MUC then waits 1.2 seconds and commands the power-on circuit to power down the system.

This same procedure exists for an environmental warning except that an environmental warning interrupt is sent and the power-on circuit does not power down the system.

The environmental error interrupt and the 1.2 second delay provide the system adequate time to read CSRs to determine the cause of the error, log the condition in NVRAM, and display the condition on the LCD.

After the system is powered down, the utilities board is still powered up, but all outputs are disconnected from the system.

Environmental LED display

Second-level registers in the MUC drive the 6-bit display. The MUC prioritizes the environmental errors and warnings and passes the information to the power-on circuit. This circuit prioritizes the 6-bit field with its environmental conditions and produces a 7-bit field plus an attention bit (ATTN) that drives the LED Display. ATTN is on if there is an environmental warning.

In general, the power-on-detected errors are a higher priority than MUC-detected errors, the lower the error code number, the higher its priority. Environmental warnings are lower priority than the environmental errors. Table 30 shows the LED display error codes.

Table 30 Environmental LED display

ATTN bit	LED Display	Description
1	00	ECUB 3.3-V error (highest priority)
1	01	ASIC Install 0 (ENRB)
1	02	ASIC Install 1 (MEM)
1	03	FPGA not OK
1	04-07	DC OK error (UL, UR, LL, LR)
1	08-11	48-V error, NPSUL fail, PWRUP=0-9
1	12-1B	48-V error, NPSUR failure, PWRUP=0-9
1	1C-25	48-V error, NPSLL failure, PWRUP=0-9
1	26-2F	48-V error, NPSLR failure, PWRUP=0-9
1	30-39	48-V error, no supply failure, PWRUP=0-9
1	3A	48-V yo-yo error
1	3B	ENRB power failure (ENRBPB)
1	3C	Clock failure
1	3D-3F	Not used (3)
1	40-47	MB0-MB7 power failure

Table 30 Environmental LED display—**(continued)**

ATTN bit	LED Display	Description
1	48-4F	PB0L, PB1R, PB2L, PB3R, PB4L, PB5R, PB6L, PB7R power failure
1	50-57	PB0R, PB1L, PB2R, PB3L, PB4R, PB5L, PB6R, PB7L power failure (possibly switch R and L)
1	58-5B	IOB (LR,LF,RF,RR) power failure
1	5C-61	Fan failure (UR,UM,UL,LR,LM,LL)
1	62	Ambient hot
1	63	Overtemp ENRB
1	64-67	Overtemp quadrant (RL, RU, LL, LU)
1	68	Hard error
1	69	Ambient warm
1	6A-6F	Not used (6)
1	70-73	DC supply maintenance (UL,UR,LL,LR)
1	74-7F	Not used (12)
0	00-09	PWRUP state (00=System all powered up), attention LED off

The top of the table is the highest priority, the bottom the lowest. If a higher condition occurs, that one is displayed.

Monitored environmental conditions

This section describes each environmental condition that is monitored by power-on circuit and the MUC.

ECBU 3.3-Volt error

This error indicates that the ECUB 3.3-volt power supply has failed, but the 5-volt supply has not.

ASIC installation error

Each ASIC in the hypernode has ASIC Install lines to prevent power-up if an ASIC is installed incorrectly (such as a PAC installed in a RACs position). If an ASIC is improperly installed, the utilities board does not power up the system. This condition is not monitored after power up.

DC OK error

When this error is displayed, the power-on circuit did not power up the system, because one or more 48-volt power supplies reported an error. In systems with redundant 48-volt power supplies, this error means that two or more 48-volt supplies reported an error.

48-volt error

If the 48-volt supply has dropped below 42 volts for any reason other than normally turning off the system or an AC failure, then this error is displayed by the power-on circuit. Also, the 48-volt supply that reported the error and the power-up state of the system at the time of the error is displayed.

48-volt yo-yo error

This error indicates that a 48-volt error occurred and the ECBU lost and then later regained power without the machine being turned off. The power-on circuit will display this error and not power on the system, because the 48-volt supply is likely at fault.

Clock failure

If the system clock fails, then the MUC will be unable to monitor environmental errors that could possibly damage the system. If the power-on circuit receives no response from the MUC, it powers down the system and displays this error.

FPGA configuration and status

The MUC is programmed by a serial data transfer from EEPROM upon utility board power-up. If the transfer does not complete properly, the MUC cannot configure itself and many environmental conditions cannot be monitored. The power-on circuit monitors both the MUC and PUC and does not power up the system, if they are not configured correctly.

Board over-temperature

There is one temperature sensor per board in the hypernode that detects board overheating. The sensors are bussed together into four hypernode quadrants plus the midplane and applied to the MUC.

Fan sensing

Sensors in the six fans determine if the fans are running properly. The MUC waits 12.8 seconds for the fans to spin up after power-up before monitoring them.

Power failure

Because a power failure on a board could cause damage to other boards, a mechanism is in place to detect 3.3-Volt failures on each board. Power failures are considered environmental errors, and the system is powered down after they are detected.

Midplane power failure

If the midplane power fails, the power-on circuit powers down the entire hypernode. The utilities board is still active, but the power-on circuit displays the power failure condition and disables all utilities board outputs that drive the hypernode. This condition persists until power is cycled on the utilities board.

48-volt maintenance

There are four 48-volt power supplies. Each sends a signal to the power-on circuit. If any supply fails at any time, the circuit asserts the 48-V maintenance line to the MUC, which reports the environmental warning to the processors. The power-on circuit displays the "highest priority" 48-volt supply that failed.

Ambient air sensors

The ambient air sensors detect a too warm or too hot condition in the input air stream to the utilities board (and therefore the entire hypernode). Ambient air too warm is an environmental warning; ambient air too hot is an environmental error that powers down the system.

The temperature set points are set by the test station. The digital temperature sensor has nonvolatile storage for the temperature set points. Power-on reset starts the digital temperature sensor without the core logic microprocessor intervening.

Environmental control

The utilities board performs the following functions to control the hypernode environment.

Power-on

When the power switch is turned on, the outputs of the 48-volt power supplies become active. Several hundred milliseconds after the utilities board 5-volt supply reaches an acceptable level, the power-on circuit starts powering up the other DC-to-DC converters of the hypernode. It does not turn on all converters at the same time. Instead, the converters are powered-on in succession.

The power-on circuit does not power up the hypernode if an ASIC is installed incorrectly (see the "ASIC installation error" section on page 162) or if an FPGA is not configured (see the "FPGA configuration and status" section on page 163). It keeps the system powered up unless an environmental condition occurs that warrants a power-down.

Voltage margining

Voltage margin is divided into four groups to minimize control, but allows all boards that communicate with each other to be margined separately for nominal, upper, and lower voltage.

MUC CSRs

This section provides a description of some of the MUC CSRs.

MUC Processor Report register

The Processor Report register indicates the processors that are working in the hypernode. Each processor reports by writing to this register and setting the bit corresponding to the processor number.

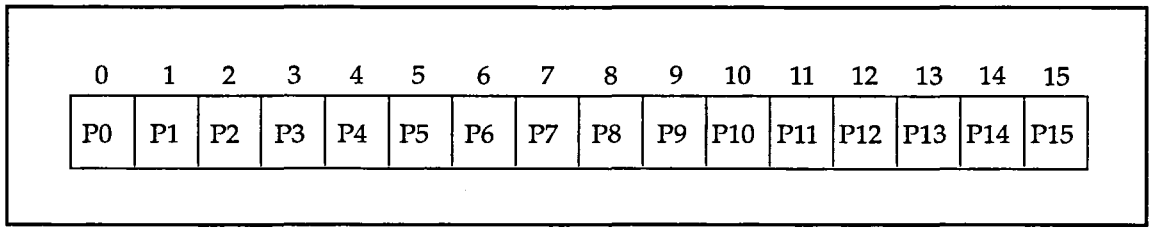


Figure 83 Processor Report register definition

P0-P15 comprise a fully readable and writable field.The bits are cleared on reset. Once a bit is written to a one value, it remains set until cleared by reset. Writes of zero value do nothing. The bit, *Px*, set to a one value indicates that processor *x* has reported in working.

MUC Processor Semaphore register

The Processor Semaphore register provides a signaling function for processor synchronization. This is an atomic read-and-increment register.

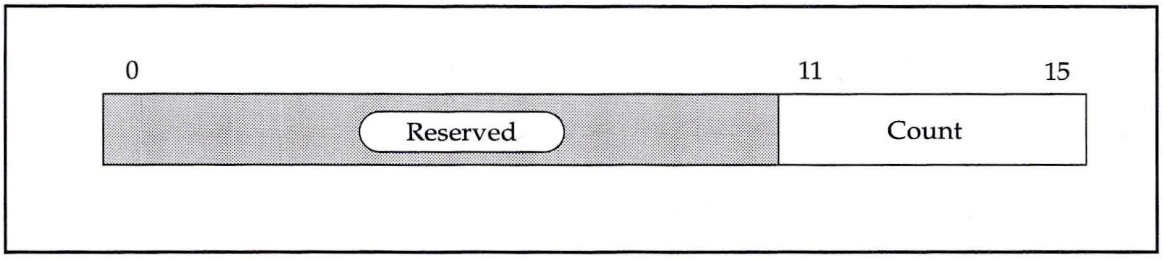


Figure 84 Processor Semaphore register definition

Count is cleared on reset. Writes load any value. Reads return the value of Count and then increment Count atomically.

MUC RAC Data register

The RAC data register holds the data to be written to the destination RAC CSR or the data that has been read from the RAC CSR.

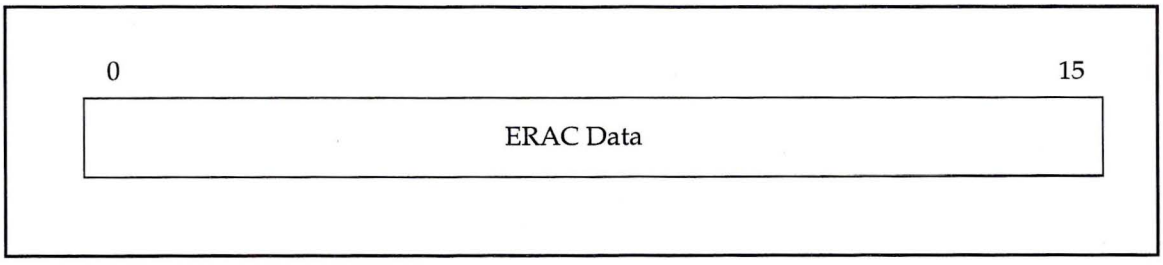


Figure 85 RAC Data register definition

RAC Data bits comprise a fully readable and writable field. After a RAC read operation, the RAC Data register holds the data. After the RAC write operation, the data is stored in the RAC register, and RAC Data is undefined.

MUC RAC Configuration Control register

The RAC Configuration Control register selects the target RAC, the address of the CSR within that RAC, and the type of CSR access (read or write). It controls the RAC CSR operation and then returns status of the operation.

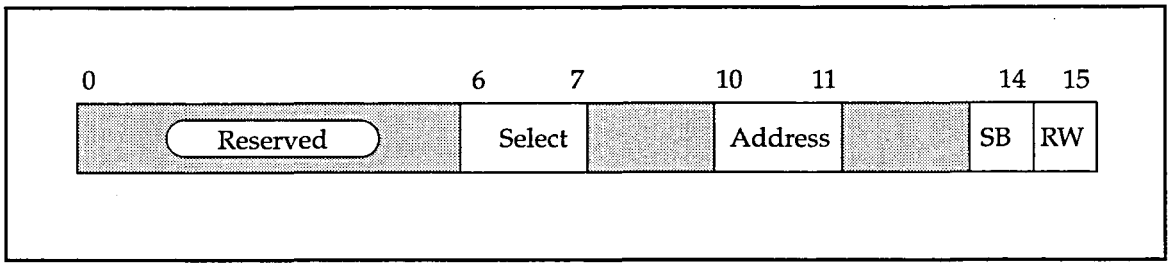


Figure 86 RAC Configuration Control register definition

The fields and bits of the RAC configuration Control registers are defined as follows:

- ***Select*** field (bits 6:7)—Selects the target RAC. This field is write only.
- ***Address*** field (bits 10:11)—Selects the address of the CSR within that RAC.
- ***SB Start/Busy*** bit (bit 14)—Starts the RAC operation by writing a one value. Reading the bit returns the status of the operation (0=idle, 1=busy). SB and SEL must be written together.
- ***RW*** bit (bit 15)—Selects the type of operation: Read (**RW**=1), or Write (**RW**=0).

MUC Reset register

The MUC Reset register initiates a reset or displays the type of the last reset. This CSR also contains the revision status.

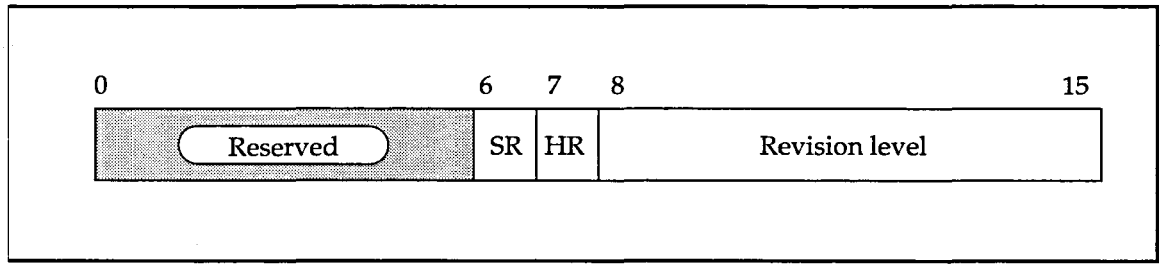


Figure 87 MUC Reset register definition

The bits and field of the Reset register are defined as follows:

- *SR* (Soft Reset) bit (bit 6)—Initiates a soft reset.
- *HR* (Hard Reset) bit (bit 7)—Initiates a hard reset.

The combination of SR and HR bits in the read mode indicate the resets given in Table 31. The combination of SR and HR bits in the write mode indicate the resets given in Table 32.

Table 31 Reset register read codes

SR HR - Read	Last reset was
0 0	Power-on reset
0 1	Hard reset
1 0	Soft reset

Resets are initiated by writing to this register. Reset is asserted according to the codes in Table 31. The only difference between a hard and soft reset is the action taken by the software upon reading the codes.

Table 32 Reset register write codes

SR HR - Write	Action taken
X 1	Hard reset
1 0	Soft reset

- ***Revision*** field (bits 8:15)—Indicates the revision of the MUC FPGA. This field is read only.

JTAG interface

The JTAG interface supports a test station and a mechanism to fanout JTAG to all the boards in a hypernode. It is used only for testing.

The JTAG functions are described in the following sections.

Test station interface

The test station can be a PA-RISC based workstation or laptop computer. The interface to the test station is an ethernet AUI port for flexibility in connecting to many workstations and laptops. It is also easily expandable to all the hypernodes in a single wall.

AC test of a hypernode

An AC test is performed by a Test Bus Controller (TBC) scanning in data to all boards in a hypernode and loading an `AC Test` instruction into all ASICs on one board.

Once all boards have been almost loaded with the AC test instruction and paused, the TBC takes all boards out of pause mode simultaneously causing them all to exit update together and execute the AC test.

The AC test enables clocks inside the ASICs so that they test internal and external paths at the system clock rate. They all execute on the same system clock.

Clock margining

Parallel ports on the core logic microprocessor select the nominal, upper, or external clock that drives the hypernode.

Multiple hypernode JTAG fanout

The test station interface is thin ethernet. This port is also used for console ethernet. There is one cable that connects to all the hypernodes and to the test station (if it exists) and to whatever device or network that will display the console.

Booting and testing 10

S-Class and X-Class servers implement booting and testing functions in specialized hardware in each hypernode.This hardware initializes the system and allows you to configure it. It has an ethernet port for connecting a terminal to perform system initialization and booting remotely. It allows you to perform hypernode diagnostic testing.

Booting

Booting a system refers to a sequence of events that loads and executes the operating system code. This sequence, or *boot procedure,* begins at power-on with the system in an unknown state and ends when the system begins executing the operating system.

Hardware reset

When power is applied to each hypernode, all controllers (ASICs) in the hypernode receive a power-up reset signal. Hardware initialization occurs within the first few clocks after the reset pulse is negated.

The reset signal has the following effects:

- PAC initialization—Hard error reporting is disabled, and all error registers hold their previous values if a hard error was logged before reset was applied. The registers can only be cleared by software. The identification number of each processor is loaded into a CSR.
- PUC initialization—Hard error reporting is disabled, and all error registers are cleared. All other PUC CSRs hold their previous values and can only be cleared by software.
- PIC initialization—Hard error reporting is disabled, and all error registers hold their previous values. The registers can only be cleared by software.

- RAC initialization—Hard error reporting is disabled, and all error registers hold their previous values. The registers can only be cleared by software. All ports are enabled.
- MAC initialization—Hard error reporting is disabled, and all error registers hold their previous values if a hard error was logged before reset was applied. The registers can only be cleared by software.
- TAC initialization—Hard error reporting is disabled, and all error registers hold their previous values. The registers can only be cleared by software.

Power on selftest (POST) routine

When the system first powers up, all processors and supporting hardware must be initialized before the hypernode proceeds with booting. The PUC contains firmware in EEPROM known as Power On Self Test (POST) routine.

Upon power up, POST begins executing and brings up the hypernode from an indeterminate state and then executes another routine called a *secondary loader*, also known as Open Boot Prom (OBP).

POST determines the hypernode hardware configuration before running OBP. If POST encounters an error during initialization, it passes the appropriate error code to an LCD.

Figure 88 shows of how POST initializes a hypernode.

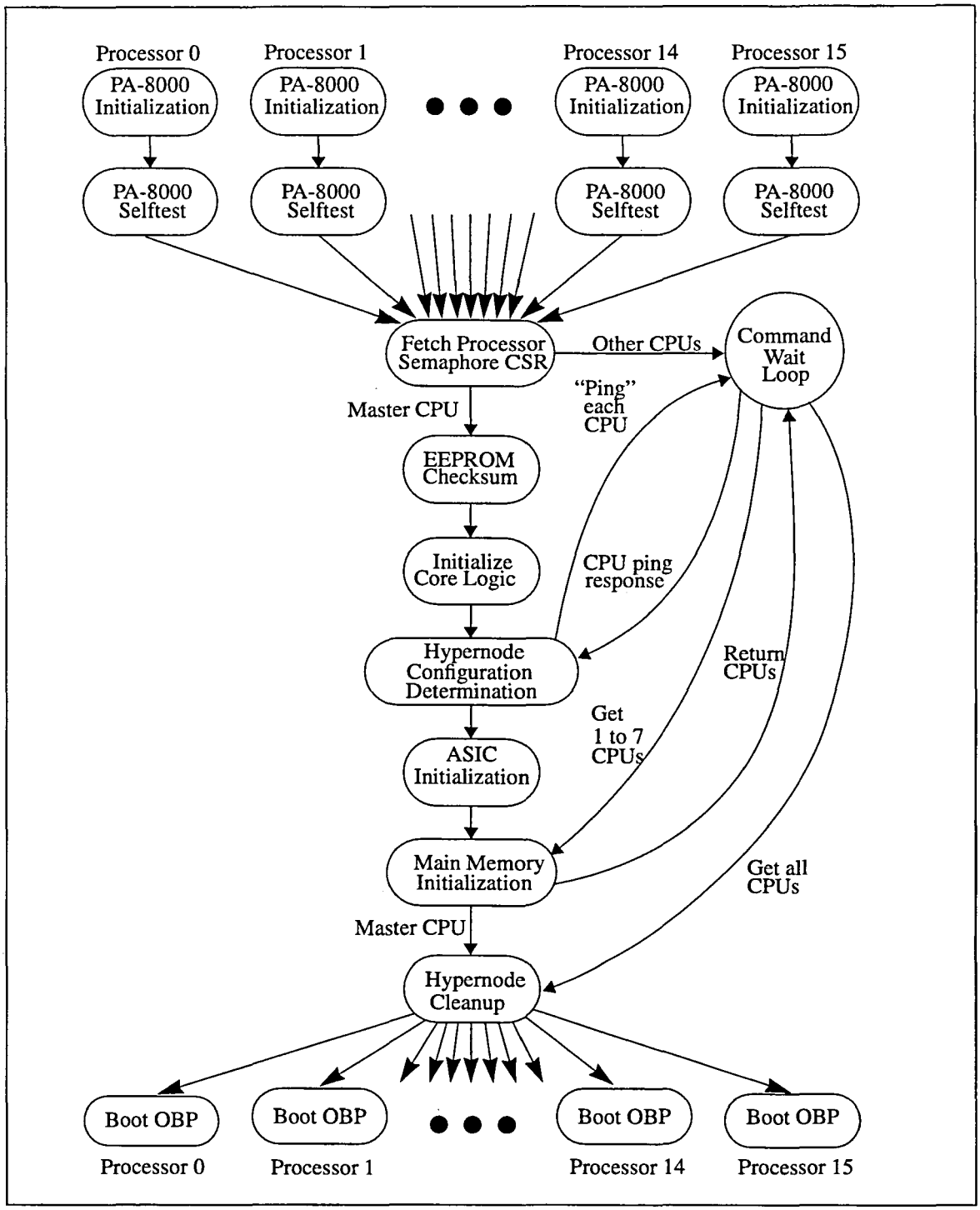


Figure 88 POST program flow

POST provides a controlled hypernode initialization and prepares it for booting by the OBP.

Basic processor initialization and selftest

The processor initialization and selftest routines are modified subsets of the PA-8000 test suite. Upon reset, all processors are initialized and placed in selftest. The extent of the selftest is determined by a mode bit in NVSRAM. The default is to perform, as a minimum, instruction set tests (primarily load and store) used by OBP.

Upon successful completion of processor selftest, the existence of each possible slice is determined. This is accomplished by reading the Revision register of each PAC, PIC, MAC, and TAC. One processor is responsible for each slice. Each processor will determine its processor ID from the PAC. Also, each processor fetches the Processor Semaphore register on the PUC. Because register requests are queued, one processor will fetch this CSR before the others and becomes the booting processor. All others go into a command wait idle loop. The booting processor begins executing POST code from the PUC EEPROM.

Checksum verification of the core logic EEPROM

The PUC EEPROM contains the POST, OBP, and hypernode diagnostics code. In addition, these three routines use shared data structures. The routines and shared data structures all reside in sections of the EEPROM know as code spaces. Each code space has an embedded checksum word. POST checks the validity of each code space by reading and comparing its checksum.

Core logic initialization

The core logic contains SRAM and DUARTs that support external terminal connection for self test and the LCD panel. POST initializes the SRAM, DUARTs, and all ASIC CSRs in the hypernode.

Hypernode configuration determination

POST determines which and how many ASICs reside in the system (not every system contains a full complement of support hardware). It also determines the number of memory modules and their sizes. Any ASIC that does not respond to any CSR access is considered to be not installed.

Hypernode ASIC initialization

POST sets every hypernode ASIC to a known state. It verifies each CSR write and checks all bits that should be valid in a read. The

initial value of each CSR is stored in a shared data structure in the PUC EEPROM.

Hypernode main memory initialization

The processor reads the hypernode ID from the COP and uses its information to load the Local Hypernode ID register. Then the Address Format and CTI Cache Enable are loaded with their initial value from EEPROM.

Next, a processor from each slice determines the memory configuration of the matching MAC. It determines the size, population, and installation of each SIMM on a memory board and returns this information to the POST. If there is not a one-to-one mapping of memory boards to processor pairs, the installed processors set up the memory boards. The results are compared with the results of each other memory mapping and the least common denominator is determined and mapped in. Once the memory population is determined, the memory and tags are initialized.

Hypernode clean up and OBP boot process

The POST resets the PUC Processor Semaphore CSR and cleans up any residual state information from the initialization process. All processors now begin to execute the OBP routine at approximately the same time.

Testing

The Utilities board contains an ethernet port for testing. This external connection allows running diagnostics on the system, as well as reconfiguring the system manually.

From the terminal, memory, CTI cache tags and data, and ECC state can be accessed.

All diagnostic accesses to memory occur through CSR space. A 64-bit register holds eight bytes for writing and reading memory. The memory transfer size is 64 bits, because memory tags are 64-bits wide. The diagnostic operations are:

- Memory line tag read
- Memory line tag write
- Memory line data read
- Memory line data write
- Memory line initialization
- Memory read ECC
- Memory write ECC
- Memory line scrub

Diagnostic Memory Read Operations

A diagnostic memory read is performed by writing to the memory line address of the Diagnostic Address CSR on the MAC. The operation that triggers the memory read is a CSR write to the appropriate diagnostic CSR address. The MAC interprets the write to the CSR address as a request to read the data for the addressed memory line. The 64-bit data is read from the SDRAM memory and written to the MAC diagnostic data register. The processor requesting the memory read can then access the data with a 64-bit CSR access. Figure 89 illustrates the flow for a CSR memory read operation.

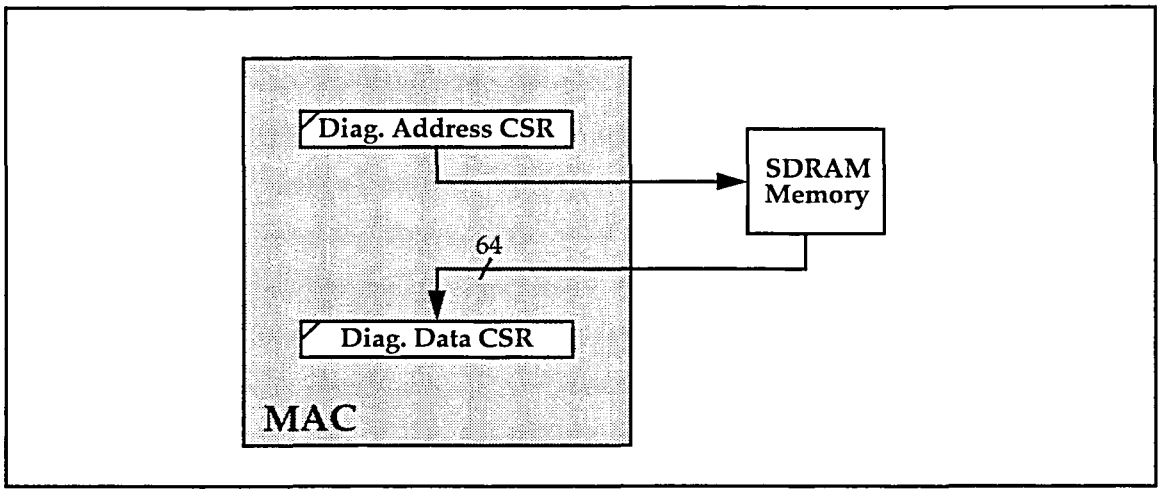


Figure 89 CSR memory read operation

Diagnostic memory write operations

A diagnostic memory write operation uses the same CSRs as read. The data is written to the CSR data register, and the address at which the data is to be stored is written to the CSR diagnostic address register. The operation that triggers the memory write is a CSR write to the appropriate diagnostic CSR address.

MAC diagnostic CSRs and addresses

This section defines some of the MAC CSRs that perform the diagnostic memory operations.

MAC Diagnostic Address register

There is one Diagnostic Address register on each MAC that supplies the address for diagnostic memory accesses.

The format of the Diagnostic Address register is shown in Figure 90.

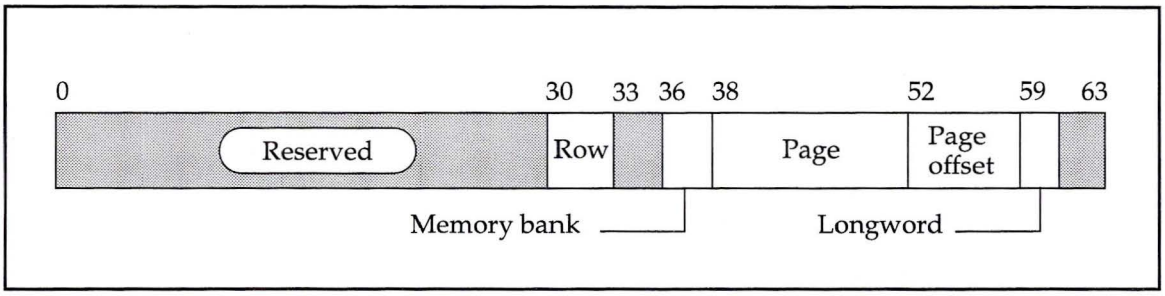


Figure 90 Diagnostic Address register definition

The ***Row*** field (bits 30:32), ***Memory bank*** field (bits 36:37), ***Page*** field (bits 38:51), ***Page offset*** field (bits 52:58), and ***Long word*** field (bits 59:60) together specify the memory address for diagnostic accesses.

The Long Word field is used for diagnostic memory data reads and writes where a specific eight-byte longword must be accessed.

Bank interleaving is not performed by the MAC; the processor must perform the mapping from the virtual bank to the memory bank.

All fields are written by a CSR write and read by a CSR read. Note that a Diagnostic Memory Initialization operation increments the concatenated Page and Page Offset fields as part of the operation.

MAC Diagnostic Data register

The Diagnostic Data register holds eight bytes of data used for diagnostic memory reads and writes.

The format of the Diagnostic Data register is shown in Figure 91.

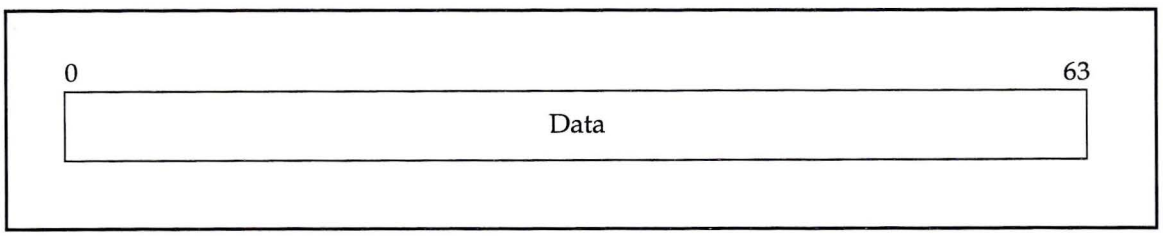


Figure 91 Diagnostic Data register definition

MAC Diagnostic Read Memory Tag address

A write to this address triggers a memory read tag operation to the address in the Diagnostic Address register. The 64-bit value read from the memory tag is written in the Diagnostic Data register. The data in this register can then be read by the processor with a CSR read access. The Long Word field of the Diagnostic Address register is ignored, because memory tags are associated with 32-byte memory lines. The Diagnostic Address register is not modified by this operation.

MAC Diagnostic Write Memory Tag address

A write to this address performs a memory write tag operation to the address in the Diagnostic Address register. The 64-bit value written to the memory tag is contained in the Diagnostic Data register. The data in this register should be initialized prior to writing to the Diagnostic Write Memory Tag address. Neither the Diagnostic Address nor Diagnostic Data registers are modified by the operation. The Long Word field of the Diagnostic Address register is ignored, because memory tags are associated with 32-byte memory lines. ECC is regenerated for the entire memory line by this operation.

MAC Diagnostic Read Memory Data address

A write to this address triggers a memory read operation to obtain eight bytes of data from the address specified by the Diagnostic Address register. The operation is similar to the memory tag read operation, except that the Diagnostic Address register also specifies which of the four eight-byte longwords of a 32-byte line is to be written to the Diagnostic Data register. The Diagnostic Address register is not modified by this operation.

MAC Diagnostic Write Memory Data address

A write to this address triggers a memory write operation to eight bytes of data at the address specified by the Diagnostic Address register. The operation is similar to the memory data write operation, except that the Diagnostic Address register also specifies which of the four eight-byte longwords of a 32-byte line is to be written with the data in the Diagnostic Data register. Neither the Diagnostic Address nor Diagnostic Data registers are modified by the operation. ECC is regenerated for the entire memory line by this operation.

MAC Diagnostic Memory Read ECC address

The tag and data of a memory line is stored in four consecutive SDRAM locations. Each SDRAM location is protected with an

eight-bit ECC. A write to this address triggers a memory read operation of the ECC associated with the address specified in the Diagnostic Address register. The Long Word field of the Diagnostic Address register specifies to which of the four SDRAM locations the ECC is to be accessed. The accessed ECC is written to the eight least significant bits of the Diagnostic Data register. The Diagnostic Address register is not modified by the operation.

MAC Diagnostic Memory Write ECC address

A write to this address triggers a memory write operation of the ECC associated with the address specified in the Diagnostic Address register. The *Long Word* field of the Diagnostic Address register specifies which of the four SDRAM locations the ECC is to be written. The SDRAM ECC is written with the least significant eight bits of the Diagnostic Data register.

MAC Diagnostic Memory Initialization address

A write to this address triggers a memory write operation to the tag and data of a memory line. The address of the memory line is specified by the Diagnostic Address register. The memory tag is written with the contents of the Diagnostic Data register, and the 32-bytes of memory data associated with the memory line are written with the value zero. The Diagnostic Data register is not modified. The concatenated Page and Page Offset fields of the Diagnostic Address register are incremented to address the next sequential memory line. The Long Word field of the Diagnostic Address register is ignored.

MAC Diagnostic Scrub Memory address

A write to this address triggers a read and write to the memory line at the address specified by the Diagnostic Address register. If a single bit ECC error occurs when the data is read, the data is corrected before it is written back into memory. If no ECC error occurs, the data is written back unmodified.

Error handling 11

This chapter discusses the mechanisms that handle errors. An error (or fault) is an abnormal condition with hardware or firmware (processor-dependent code). The cause of the abnormality can be either transient or permanent. It can also be classified as a recoverable (soft or advisory) error or an unrecoverable (hard) error, depending on whether continued operation of the system is possible.

Most hardware faults are transient; that is, they are not usually caused by a hardware fault. S-Class and X-Class server error-handling mechanisms attempt to contain the effect of the error while continuing to operate. There are, of course, occasions when hardware fails, continued operation is not possible, and the system must be taken down for diagnostic evaluation.

Soft errors

A recoverable error that results in the loss of one or more processes but allows continued operation of the system, is a *soft* error. An example of a soft error is a parity error on data read by a process. The process cannot continue, but the system continues to operate.

Soft errors can occur only during transactions that require a response. The error is reported to the requesting processor in one of two ways:

- The requesting processor detects the error itself (for example, a parity error).
- The detecting hardware sends an error response instead of its normal response. The error response contains some useful information about the error.

Whenever a soft error is reported to a processor, it invokes an HPMC. Any process running on a processor when an HPMC occurs is aborted by the OS. If the process is a kernel or server, an OS *panic* occurs. In this case, the system must be rebooted.

Advisory errors

A special type of recoverable error is an advisory error. Advisory errors are usually corrected by hardware or firmware. They are logged in the appropriate CSRs of the detecting controller and do not affect any processes running on the system. An example is a single-bit ECC memory error.

Advisory errors are not reported to software. Software must poll the CSRs periodically to determine their occurrence. Reading the CSRs when a soft error is detected can determine if it propagated from an earlier-detected advisory error.

If a detected error causes corrupted data and another error (soft error) is detected before corrupted data is consumed, it is also considered an advisory error. An example is a data parity error detected during a "responseless" request, such as a write-back, where corrupted data is written to memory This error is logged as an advisory error. Any further reference to the line will cause a soft error, telling the consumer of the data that it is corrupt.

There are some uncorrectable errors that can be classified as advisory errors. This would be the case when corrupted data resulting from the uncorrectable error is consumed and the consumer is notified; yet, the error can be detected again. Normally, for soft errors the consumer is only notified once.

Some advisory errors are reported by an interrupt to the processor specified for servicing error interrupts.

Hard errors

Hardware can fail in such a manner that a process can receive corrupt data without detecting it. If an error is detected that prevents returning an error response or corrupts data so that future references cannot detect the corruption, it is considered a hard error. An example is a parity error detected on the address of a memory transaction. The appropriate memory line cannot be updated, and future *consumers* of the line are not notified of the corruption.

A hard error is sent to the MUC and then the PUC which generates an HPMC to one or all processors in the hypernode. See the "PUC interrupt logic" section on page 109 for more information on PUC interrupts.

All error CSRs are locked (or *frozen*) when a hard error is detected so that additional errors caused by the propagation of the hard error are not logged. Usually, only one controller fails in the error condition. If more than one controller detects a hard error, however, the state in the MUC and clock phase information in each chip indicate the chip that first detected the hard error.

When a hard error occurs, the system must be rebooted. Failing resources can be deconfigured, as part of reboot, to allow the system to quickly resume operation (possibly with degraded performance) in the presence of broken hardware.

Figure 92 illustrates the characteristics of the three error types: advisory, soft and hard.

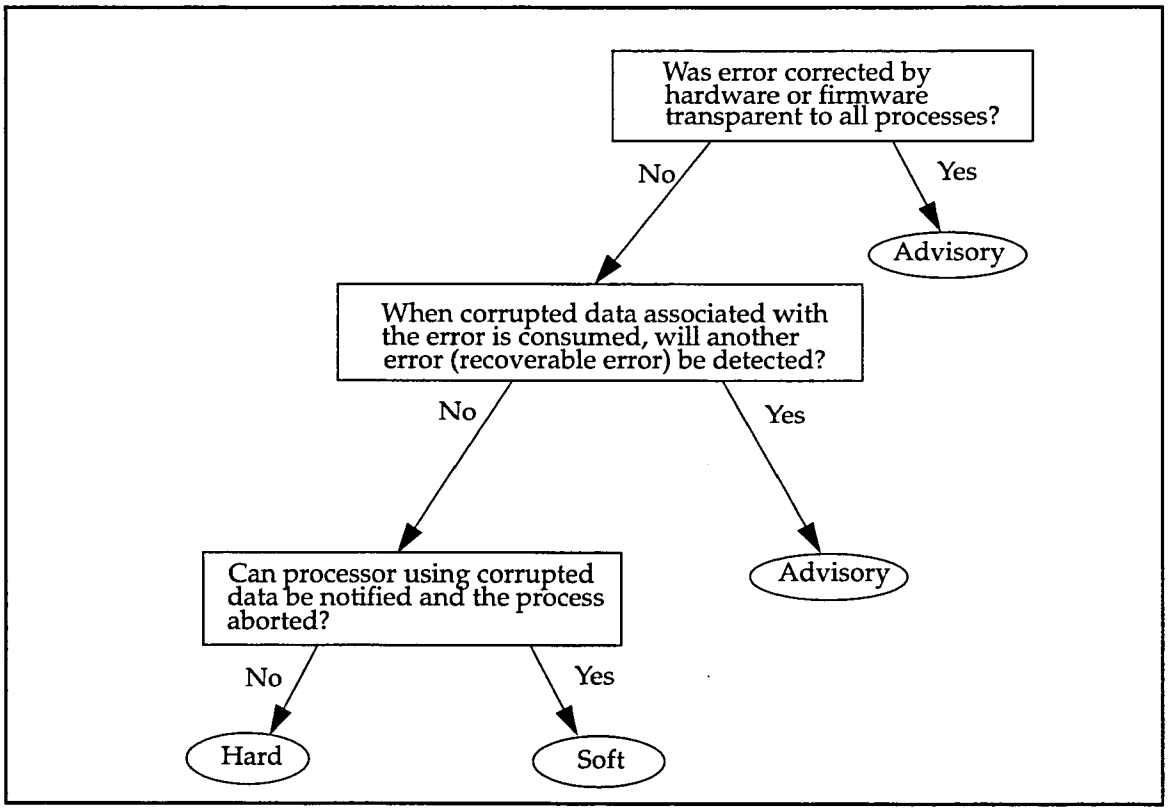


Figure 92 Error Types

Error responses

When an error occurs during a *response-expected* data request, the requested data is not returned. Instead, an error response is returned to the requestor. This type of error is called a soft error and is logged as such in the appropriate CSR of the requesting controller.

The response contains information that specifies the detecting hypernode, the detecting controller, and the detected error condition or *error code*. It does not contain data. After receiving an error response, the PAC logs the error and returns a directed error response packet to the requesting processor or PIC (in the case of an I/O request). The processor logs the error information in its SADD_LOG register and the PIC logs the error information in its internal CSRs. Error recovery software can then read these CSRs and take appropriate action.

Figure 93 shows the format for the processor SADD_LOG after an error response.

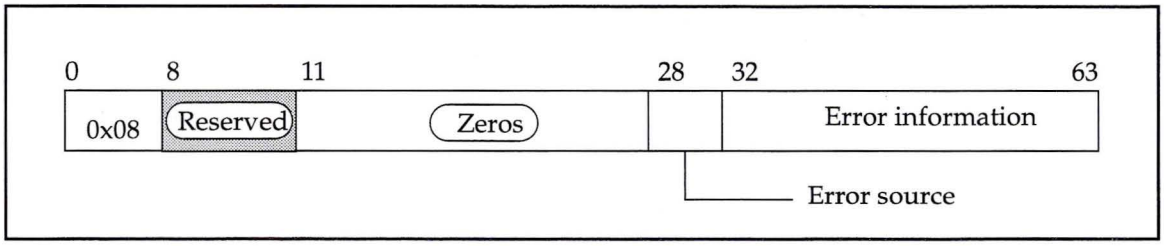


Figure 93 SADD_LOG after error response

The ***Error Source*** field (bits 28:31) indicates one of the sources as shown in Table 33:

Table 33 SADD_LOG error source field definition

Error source field	Source
0x0	Crossbar 0 input
0x1	Crossbar 1 input
0x2	Crossbar 0 output
0x3	Crossbar 1 output
0x4	Runway input
0x5	PAC CSRs
0x6-0xf	Reserved

The value in the ***Error Information*** field (bits 32:63) depends on the source of the error response.

If the error source field indicates the response was from either crossbar inputs (that is, external to the PAC), the contents of the ***Error information*** field (bits 32:63) contains the information shown in Figure 94.

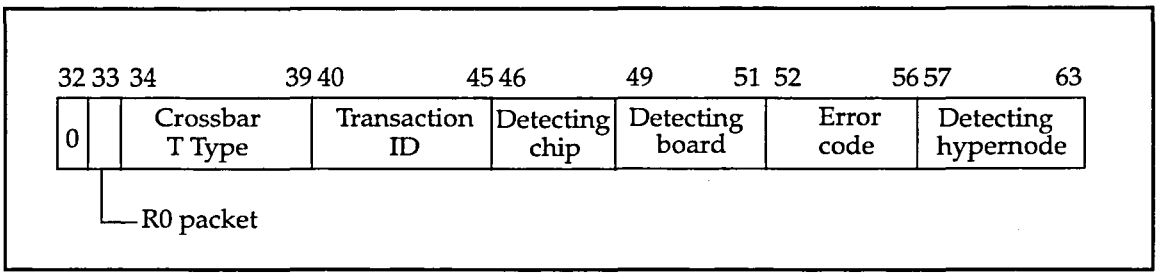


Figure 94 PAC error response information when received from either crossbar input

The subfields of the Error information field are defined as follows:

- ***R0*** bit (bit 33)—Indicates that the intended response in the crossbar was an R0 packet.
- ***Crossbar T type*** field (bits 34:39)—Specifies the transaction type of the intended crossbar response.
- ***Transaction type*** field (bits 40:45)—Specifies the transaction ID of the intended response.
- ***Detecting chip*** field (bits 46:48)—Specifies the controller that detected the error.
- ***Detecting board*** field (bits 49:51)—Specifies the instance of the controller that detected the error (there are eight instances of each controller that can return an error response).
- ***Error code*** field (bits 52:56)—Specifies the error condition detected by the chip that sent the error response.
- ***Detecting hypernode*** field (bits 57:63)—Specifies the hypernode that sent the error response (in X-Class servers only).

If the error source field indicates the response was from the crossbar output logic (internal to the PAC), the contents of the ***Error information*** field (bits 32:63) contains the transaction ID (TID) of the outgoing transaction. If the error source field indicates an error detected by PAC runway bus input logic, the contents of the ***Error information*** field (bits 32:63) contains the type of runway error. If the error source field indicates an error in one of the PAC CSRs, the contents of the ***Error information*** field (bits 32:63) contains the CSR error code.

Hard error logging

Hard errors result when data has been corrupted as a result of a hardware failure and the requesting processor can not be notified.

The hard error is logged in the MUC System Hard Error register. This register logs the first hard error detected, allowing isolation to the controllers or group of controllers that detected the error first.

When a hard error occurs, the PUC sends a directed error to one or more processors over the core logic bus, forcing the destination processor to take an HPMC. The SADD_LOG register in the target processor or processors contains information indicating that a directed error due to a hard error was received. The contents of the SADD_LOG, after a hard error has been received, are shown in Figure 95.

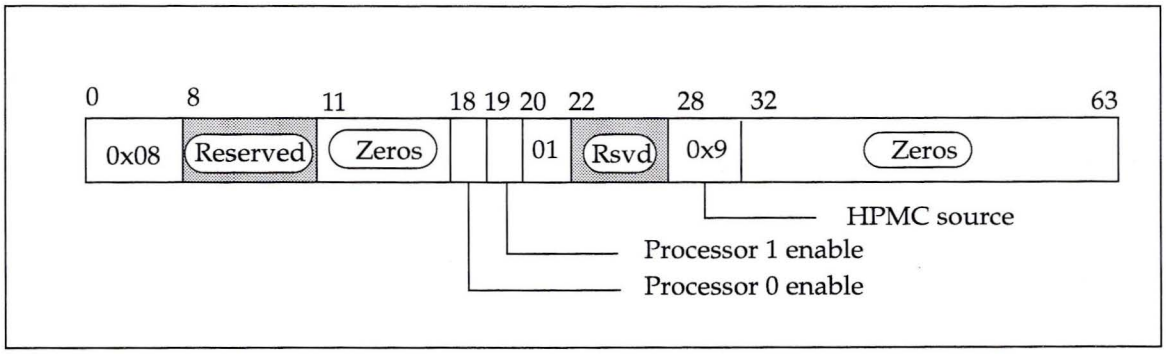


Figure 95 Processor SADD_LOG register definition after directed error due to hard error

The bits and fields of the SADD_LOG register after a directed error due to a hard error are as follows:

Processor 0 bit (bit 18) and ***1 enable*** bit (bit 19)—Specify which processors are enabled to receive hard error directed errors.

HPMC source field (bits 28:31)—Indicates the source of the high-priority machine check (value equals 0x9).

Error handling CSRs

Most controllers contain at least one the following registers:

- Error Cause
- Error Address
- Error Information
- Error Configuration

These registers are accessible through load and store instructions and diagnostic scan.

Error Cause register

The Error Cause register logs multiple errors. It has a *sticky* bit for every possible error condition that can be detected for the controller. Because an error condition can occur under multiple circumstances, a different bit exists in the Error Cause register for each unique circumstance.

A Hard Error Group (G) bit indicates that a hard error was detected by another controller. When this occurs, all other error sources are masked out and remain that way until the G-bit is reset. They are also masked out when the controller detects its own hard error.

If a hard error is logged, or the G bit is set, the contents of this register are not changed during reset.

Error Address register

Whenever an error occurs that can be isolated to a specific address, that address is loaded into the Error Address register. This register is loaded or held under the same error conditions as the Error Information register.

If a hard error is logged, or the G bit is set, the contents are not changed during reset.

Error Information register

The Error Information register contains error recovery or diagnostic information. This register contains the type of error (listed in increasing severity):

- None (00)
- Advisory (01)
- Soft (10)
- Hard (11)

It also contains the error number for advisory or soft errors (the error number is undefined for hard errors). If an error is detected of the same or lower severity as that already stored in the register, the Error Information register is not overwritten, but the Multiple Error (M) bit is set. If an error type of greater severity is detected, the Error Information register is overwritten with new error information. When error information is overwritten, the Overwritten (O) bit in the register is set.

A Simultaneous Group Errors (S) bit indicates that another chip was asserting the hard error signal input to this chip when it logged an error.

If a hard error is logged, or the G bit is set, the contents of the Error Information register are not changed during reset.

Error Configuration register

The Error Configuration register contains two bits for every bit in the Error Cause register. They are encoded as follows:

- 00—Disable the error
- 01—Treat as an advisory error
- 10—Treat as a soft error
- 11—Treat as a hard error

These bits control updating of the Error Information and Error Address registers. They have no effect on the controller behavior except for the following conditions:

- If an error is disabled (bit-pair value is 00) and the controller can still do something meaningful toward completing the operation, the error is ignored. For example, a chip might ignore an address alignment error and assume a certain address when the error is disabled. If no behavior makes sense when an error occurs (that is, the error cannot be ignored), then the disable has no effect on chip operation.
- If the controller scan ring option stop_on_hard bit is set and an error is configured as a hard error (bit-pair value is 11), registers containing information associated with the error must be held. This allows access to the information through scan.

Processor error detection

The processor detects errors that occur during transfers to and from its runway bus and cache interfaces. It logs these errors and invokes either an LPMC or an HPMC. An LPMC is similar to a trap and allows the process to be restarted. An HPMC is usually fatal to the process, but it may not require a system reboot. The PA-8000 has no Error Configuration register. The error handling code determines if a processor detected error is treated as advisory, soft or hard.

When errors occur outside the processor that result in an error response to the processor, error information is stored in the processor SADD_LOG register. For timeout errors occurring during noncoherent load or fetch operations, the address associated with the error is stored in the Read Short Logging register. Miscellaneous diagnostic registers contain information about cache parity errors.

PAC error detection

When the PAC receives an error response from the crossbar destined for a processor, it sends a directed error followed by a dummy response (in most cases) to the requesting processor. The error response informs the processor that a soft error was detected during its request and forces the processor to invoke an HPMC. When the PAC receives an error response destined for the PIC, it forwards it as an error response packet on the PIC interface.

When detecting an error, the PAC stores addresses and other error information in two databases. The information in these databases is accessible with loads and stores. When errors are detected, the PAC copies information from the database into certain error CSRs.

If an error response to a processor is received or a parity error is detected on the crossbar during a response to a processor, the PAC copies the error information in the database and into the error CSRs.

If a timeout transaction is detected on the runway bus, the information corresponding to the timed out response is copied into error CSRs.

If an error is encountered during a message or copy operation, a processor is interrupted, and the detected error condition is logged in the operation status queue.

The PAC has an interface to utilities board functions via the core logic bus. Processors receive interrupts, fetch instructions, and log error information over the bus. Access to the bus is unaffected by most errors, including a large percentage of hard errors, allowing the processors to perform error logging and recovery.

RAC error detection

The RAC routes crossbar packets between the PAC and the MAC. It checks parity on these packets to and from internal queues. It does not regenerate parity, but passes received parity through queues to its output ports. When the RAC detects an error, it is logged in the Error Cause and Error Information registers.

There is no Error Address register in the RAC. The RAC does not detect when an address is being sent in a packet; therefore, it cannot log the address in an Error Address register.

MAC error detection

The MAC detects a parity error on data received from the crossbar and from the TAC. It writes the data as usual, but sets the tag state of that line to "error." An error response received from the crossbar or TAC also sets the tag state to "error." When memory is read and the tag state is set to "error," the MAC returns an error response to the requesting processor indicating the data is corrupted.

The MAC contains memory error correction and detection hardware that corrects and logs single-bit errors. When it detects a multibit error on a memory tag, the MAC generates a hard error. If it detects a multibit error on requested memory data, it sends a parity error with the data to the requestor. If the MAC detects a multibit error on a read-modify-write operation, it writes bad ECC (that results in a multibit error) to the line to notify any potential users the data line is corrupted.

TAC error detection

In X-Class servers, hypernodes connect through the X- and Y-ring CTIs. The TAC provides an interface between the hypernode and the ring interconnect. It provides error detection and containment at both the hypernode side and ring side.

A cyclic redundancy code (CRC) covers every packet transferred between the TAC and the CTI (except for idle packets that have redundant information). If a CRC error is detected on a packet, the CRC is *stomped* (logically XORd with a pattern) allowing packets to be quickly passed from hypernode to hypernode and providing error handling software a means of determining which hypernode first detected the CRC error.

When the TAC sends a request or response on a CTI, an echo must return within a certain time limit. If it does not, it logs an echo timeout error. The same is true for packets sent to the TAC.

An echo timeout is an advisory error, because a response can still follow a request. A response timeout is advisory when the timeout mechanism of the requester inside the hypernode (processor, PIC

or data mover) detects the problem. If a response timeout cannot be reported to a requester, it is a hard error.

When the TAC detects a parity error on data received from the MAC, it logs the error but preserves the bad parity in its internal queues and data structures. When the bad data is transmitted on the CTI, the CRC is modified to indicate that a packet containing corrupted data is being sent. When the packet is received at the destination, bad parity is forced on the data. This scheme contains the error so that only consumers of the corrupt data are affected by the error.

CSR map

A

This appendix lists all architecturally defined CSRs in the Exemplar X-Class server. The S-Class server contains all these CSRs, but some are not used. It has the capability to be upgraded to an X-Class server.

Hypernode-local CSRs

The following CSRs are accessible to processors local to the hypernode only. This includes both S-Class and X-Class servers.

Table 34 Hypernode-local CSR map

40-Bit physical address	CSR space	CSR register name
0xF0 0000 0000 - 0xF0 FFFF FFFF	**Hypernode Local, Core Logic**	
0xF0 xx00 0000 - 0xF0 xx7F FFFF	PDC EEPROM	X-Class implements 4 MBytes (0x0-0x1FFFFF)
0xF0 xx80 0000 - 0xF0 xxBF FFFF	SRAM	X-Class implements 128 KBytes (0x800000-0x81FFFF)
0xF0 xxC0 0000 - 0xF0 xxC0 FFFF	PUC byte access	PUC CSR space
0xF0 xxC0 0000		Interrupt Status register
0xF0 xxC0 0004		Interrupt Enable register
0xF0 xxC0 0008		Interrupt Force register
0xF0 xxC0 000C		PAC Exist register
0xF0 xxC0 0010		PUC Revision register
0xF0 xxC0 0014		PUC Error register

Table 34 Hypernode-local CSR map **(continued)**

40-Bit physical address	CSR space	CSR register name
0xF0 xxC1 0000 - 0xF0 xxCF FFFF	MUC	MUC CSR space
0xF0 xxC1 0000	Half Word Access	Processor Report register
0xF0 xxC1 0004		Processor Semaphore register
0xF0 xxC1 0008		RAC Scan Data register
0xF0 xxC1 000C		RAC Scan Control register
0xF0 xxC1 0010		System Hard Error register
0xF0 xxC1 0014		System Hard Error Enable register
0xF0 xxC1 0018		System Hard Error Control register
0xF0 xxC1 001C		Error Cause register
0xF0 xxC1 0020		Environment Error A register
0xF0 xxC1 0024		Environment Error B register
0xF0 xxC1 0028		Environment Error C register
0xF0 xxC1 002C		Environment Control register
0xF0 xxC1 0030		Reset register
0xF0 xxD0 0000 - 0xF0 xxD2 FFFF	M48T35Y	Nonvolatile Ram and Real Time Clock
0xF0 xxD0 0000 - 0xF0 xxD0 7FF7	Byte, Half, Word or Double Word Access	Nonvolatile SRAM
0xF0 xxD0 7FF8		RTC Control register
0xF0 xxD0 7FF9		RTC Seconds register
0xF0 xxD0 7FFA		RTC Minutes register
0xF0 xxD0 7FFB		RTC Hour register
0xF0 xxD0 7FFC		RTC Day register
0xF0 xxD0 7FFD		RTC Date register
0xF0 xxD0 7FFE		RTC Month register
0xF0 xxD0 7FFF		RTC Year register

Table 34 Hypernode-local CSR map **(continued)**

40-Bit physical address	CSR space	CSR register name
0xF0 xxD3 0000 - 0xF0 xxD4 5FFF	83932B Sonic Ethernet	Ethernet Interface Chip
0xF0 xxD3 0000	Half Word Access	Command register
0xF0 xxD3 0004		Data Configuration register
0xF0 xxD3 0008		Receive Control register
0xF0 xxD3 000C		Transmit Control register
0xF0 xxD3 0010		Interrupt Mask register
0xF0 xxD3 0014		Interrupt Status register
0xF0 xxD3 0018		Upper Transmit Descriptor address
0xF0 xxD3 001C		Current Transmit Descriptor address
0xF0 xxD3 0020		Transmit Packet Size register
0xF0 xxD3 0024		Transmit Fragment Count register
0xF0 xxD3 0028		Transmit Start address 0 register
0xF0 xxD3 002C		Transmit Start address 1 register
0xF0 xxD3 0030		Transmit Fragment Size register
0xF0 xxD3 0034		Upper Receive Descriptor address
0xF0 xxD3 0038		Current Receive Descriptor address
0xF0 xxD3 003C		Current Receive Buffer address 0 register
0xF0 xxD3 0040		Current Receive Buffer address 1 register
0xF0 xxD3 0044		Remaining Buffer Word Count 0 register
0xF0 xxD3 0048		Remaining Buffer Word Count 1 register
0xF0 xxD3 004C		End of Buffer Word Count register
0xF0 xxD3 0050		Upper Receive Resource address register
0xF0 xxD3 0054		Resource Start address
0xF0 xxD3 0058		Resource End address
0xF0 xxD3 005C		Resource Read Pointer
0xF0 xxD3 0060		Resource Write Pointer
0xF0 xxD3 0064		Temporary Receive Buffer address 0 register
0xF0 xxD3 0068		Temporary Receive Buffer address 1 register

Table 34 Hypernode-local CSR map **(continued)**

40-Bit physical address	CSR space	CSR register name
0xF0 xxD3 006C	83932B Sonic Ethernet (Cont.)	Temporary Buffer Word Count 0 register
0xF0 xxD3 0070		Temporary Buffer Word Count 1 register
0xF0 xxD3 007C	Half Word Access	Last Link Field address
0xF0 xxD3 0080		Temporary Transmit Descriptor address
0xF0 xxD3 0084		CAM Entry Pointer
0xF0 xxD3 0088		CAM address Port 2 register
0xF0 xxD3 008C		CAM address Port 1 register
0xF0 xxD3 0090		CAM address Port 0 register
0xF0 xxD3 0094		CAM Enable register
0xF0 xxD3 0098		CAM Descriptor Pointer register
0xF0 xxD3 009C		CAM Descriptor Count register
0xF0 xxD3 00A0		Silicon Revision register
0xF0 xxD3 00A4		Watchdog Timer 0 register
0xF0 xxD3 00A8		Watchdog Timer 1 register
0xF0 xxD3 00AC		Receive Sequence Count
0xF0 xxD3 00B0		CRC Error Tally register
0xF0 xxD3 00B4		FAE Tally register
0xF0 xxD3 00B8		Missed Packet Tally register
0xF0 xxD3 00BC		Maximum Deferral Timer register
0xF0 xxD3 00FC		Data Configuration register 2
0xF0 xxD4 6000 - 0xF0 xxD4 9FFF	16552 DUART Serial 0 Port	Serial Port 0, Used for Console communication
0xF0 xxD4 6000	Byte Access	Receiver Buffer register/ Transmitter Holding register/ LSB Divisor Latch
0xF0 xxD4 6004		Interrupt Enable register/ MSB Divisor Latch
0xF0 xxD4 6008		Interrupt Identification register/ FIFO Control register
0xF0 xxD4 600C		Line Control register

Table 34 Hypernode-local CSR map **(continued)**

40-Bit physical address	CSR space	CSR register name
0xF0 xxD4 6010	16552 DUART Serial Port 0 (Continued)	Modem Control register
0xF0 xxD4 6014		Line Status register
0xF0 xxD4 6018		Modem Status register
0xF0 xxD4 601C		Scratch Pad register (SCR)
0xF0 xxD4 A000 - 0xF0 xxD4 BFFF	16552 DUART Serial 1 Port	Serial Port 1, Not used
0xF0 xxD4 A000	Byte Access	Receiver Buffer register/ Transmitter Holding register/ LSB Divisor Latch
0xF0 xxD4 A004		Interrupt Enable register/ MSB Divisor Latch
0xF0 xxD4 A008		Interrupt Identification register/ FIFO Control register
0xF0 xxD4 A00C		Line Control register
0xF0 xxD4 A010		Modem Control register
0xF0 xxD4 A014		Line Status register
0xF0 xxD4 A018		Modem Status register
0xF0 xxD4 A01C		Scratch Pad register
0xF0 xxD4 C000 - 0xF0 xxD4 FFFF	16552 DUART Parallel Port	Parallel Port, Used for LCD communication
0xF0 xxD4 C000	Byte Access	Read Data/ Write Data
0xF0 xxD4 C004		Read Status
0xF0 xxD4 C008		Read Control/ Write Control
0xF4 0000 0000 - 0xF7 FFFF FFFF	**Hypernode Local, Nonaccelerated IO**	
0xF8 0000 0000 - 0xFB FFFF FFFF	**Hypernode Local, Accelerated IO**	

Global CSRs

The following CSRs are globally accessible to all processors in an X-Class servers.

Table 35 Global CSR map

40-Bit physical address	CSR space	CSR register name
0xFC 0000 0000 - 0xFC 003F FFFF	**Hypernode 0**	**Globally accessible CSRs physically residing on Hypernode 0**
0xFC 0000 0000 - 0xFC 0000 FFFF	**Hypernode 0, PAC 0**	
0xFC 0000 0000	PAC 0, Page 0	System Configuration register
0xFC 0000 0008		PAC Chip Configuration register
0xFC 0000 0010		PAC Core Logic Interrupt Delivery register 0
0xFC 0000 0018		PAC Core Logic Interrupt Delivery register 1
0xFC 0000 0020		Memory Board Configuration register
0xFC 0000 0080		PAC Error Cause register 0
0xFC 0000 0088		PAC Error Info register
0xFC 0000 0098		PAC Error Configuration register 0
0xFC 0000 00A0		PAC Error Configuration register 1
0xFC 0000 00A8		PAC Error Cause register 1
0xFC 0000 0300		Time-of-Century Configuration register
0xFC 0000 1308	PAC 0, Page 1	Time-of-Century Count register
0xFC 0000 2000	PAC 0, Page 2 Processor 0 Specific	Processor 0 Processor Configuration register
0xFC 0000 2010		Processor 0 CSR Operation Context register
0xFC 0000 2018		Processor 0 CSR Operation address register
0xFC 0000 2020		Processor 0 Fetch and Increment address
0xFC 0000 2028		Processor 0 Fetch and Decrement address
0xFC 0000 2030		Processor 0 Fetch and Clear address
0xFC 0000 2038		Processor 0 Noncoherent Read address
0xFC 0000 2040		Processor 0 Noncoherent Write address
0xFC 0000 2060		Processor 0 Coherent Increment address
0xFC 0000 2068		Processor 0 CTI Cache Flush Global address

Table 35 Global CSR map **(continued)**

40-Bit physical address	CSR space	CSR register name
0xFC 0000 2070		Processor 0 CTI Cache Prefetch for Read register
0xFC 0000 2078		Processor 0 CTI Cache Prefetch for Write register
0xFC 0000 2100		Processor 0 DM Input Command register
0xFC 0000 2110		Processor 0 DM Source Physical Page Frame register
0xFC 0000 2118		Processor 0 DM Source Offset register
0xFC 0000 2120		Processor 0 DM Destination Physical Page Frame register
0xFC 0000 2128		Processor 0 DM Destination Offset register
0xFC 0000 2130	PAC 0, Page 2 (Continued)	Processor 0 DM Operation Status Queue register
0xFC 0000 2200		Processor 0 Performance Monitor Memory Access Count 0 register
0xFC 0000 2208		Processor 0 Performance Monitor Memory Access Count 1 register
0xFC 0000 2210		Processor 0 Performance Monitor Memory Access Latency register
0xFC 0000 3000	PAC 0, Page 3 Processor 1 Specific	Processor 1 Processor Configuration register
0xFC 0000 3010		Processor 1 CSR Operation Context register
0xFC 0000 3018		Processor 1 CSR Operation address register
0xFC 0000 3020		Processor 1 Fetch and Increment address
0xFC 0000 3028		Processor 1 Fetch and Decrement address
0xFC 0000 3030		Processor 1 Fetch and Clear address
0xFC 0000 3038		Processor 1 Noncoherent Read address
0xFC 0000 3040		Processor 1 Noncoherent Write address
0xFC 0000 3060		Processor 1 Coherent Increment address
0xFC 0000 3068		Processor 1 CTI Cache Flush Global address
0xFC 0000 3070		Processor 1 CTI Cache Prefetch for Read register
0xFC 0000 3078		Processor 1 CTI Cache Prefetch for Write register
0xFC 0000 3100		Processor 1 DM Input Command register

Table 35 Global CSR map **(continued)**

40-Bit physical address	CSR space	CSR register name
0xFC 0000 3110		Processor 1 DM Source Physical Page Frame register
0xFC 0000 3118		Processor 1 DM Source Offset register
0xFC 0000 3120		Processor 1 DM Destination Physical Page Frame register
0xFC 0000 3128		Processor 1 DM Destination Offset register
0xFC 0000 3130		Processor 1 DM Operation Status Queue register
0xFC 0000 3200		Processor 1 Performance Monitor Memory Access Count 0 register
0xFC 0000 3208		Processor 1 Performance Monitor Memory Access Count 1 register
0xFC 0000 3210		Processor 1 Performance Monitor Memory Access Latency register
0xFC 0001 0000 - 0xFC 0001 FFFF	PIC 0	Hypernode 0, PIC 0 CSR space
0xFC 0001 0000	PIC 0, Page 0	System Configuration register (Reserved on PIC)
0xFC 0001 0008		PIC Chip Configuration register
0xFC 0001 0010		PCI Master Configuration register
0xFC 0001 0018		PCI Master Status register
0xFC 0001 0020		PIC Channel Builder register
0xFC 0001 0080		PIC Error Cause register
0xFC 0001 0088		PIC Error Info register
0xFC 0001 0090		PIC Error Address register
0xFC 0001 0098	PIC 0, Page 0	PIC Error Configuration register
0xFC 0001 00A0		PIC Interrupt Configuration register
0xFC 0001 00A8		PIC Interrupt Source register
0xFC 0001 00B0		PIC Interrupt Enable register
0xFC 0001 0100		PCI Slot 0 Configuration register
0xFC 0001 0108		PCI Slot 0 Status register
0xFC 0001 0110		PCI Slot 0 Interrupt Configuration register
0xFC 0001 0118		PCI Slot 0 Synchronization register

Table 35 Global CSR map **(continued)**

40-Bit physical address	CSR space	CSR register name
0xFC 0001 0120		PCI Slot 1 Configuration register
0xFC 0001 0128		PCI Slot 1 Status register
0xFC 0001 0130		PCI Slot 1 Interrupt Configuration register
0xFC 0001 0138		PCI Slot 1 Synchronization register
0xFC 0001 0140		PCI Slot 2 Configuration register
0xFC 0001 0148		PCI Slot 2 Status register
0xFC 0001 0150		PCI Slot 2 Interrupt Configuration register
0xFC 0001 0158		PCI Slot 2 Synchronization register
0xFC 0001 0160		PCI Slot 3 Configuration register
0xFC 0001 0168	PIC 0, Page 0	PCI Slot 3 Status register
0xFC 0001 0170		PCI Slot 3 Interrupt Configuration register
0xFC 0001 0178		PCI Slot 3 Synchronization register
0xFC 0002 0000 - 0xFC 0002 FFFF	Processor 0	Hypernode 0, Processor 0 CSR space
0xFC 0002 0000		External Interrupt Request register
0xFC 0003 0000 - 0xFC 0003 FFFF	Processor 1	Hypernode 0, Processor 1 CSR space
0xFC 0003 0000		External Interrupt Request register
0xFC 0004 0000 - 0xFC 0004 FFFF	MAC 0	Hypernode 0, MAC 0 CSR space
0xFC 0004 0000	MAC 0, Page 0	System Configuration register
0xFC 0004 0008		MAC Chip Configuration register
0xFC 0004 0020		Memory Row Configuration register
0xFC 0004 0028		Unprotected Memory Region register
0xFC 0004 0030		Normal Network Cache Memory Region register
0xFC 0004 0038		Unprotected Network Cache Memory Region register
0xFC 0004 0080		MAC Error Cause register
0xFC 0004 0088		MAC Error Info register
0xFC 0004 0090		MAC Error address register
0xFC 0004 0098	MAC 0, Page 0	MAC Error Configuration register 0

Table 35 Global CSR map **(continued)**

40-Bit physical address	CSR space	CSR register name
0xFC 0004 00A0		MAC Error Configuration register 1
0xFC 0004 00B0		MAC Error Interrupt register
0xFC 0004 0100		Message Reception Area Configuration register
0xFC 0004 0110		Message Reception Area Available Offset register
0xFC 0004 0118		Message Reception Area Occupied Offset register
0xFC 0004 0120		Message Completion Queue Configuration register
0xFC 0004 0128		Message Completion Queue Reserve Offset register
0xFC 0004 0130		Message Completion Queue Write Offset register
0xFC 0004 0138		Message Completion Queue Read Offset register
0xFC 0004 0140		Message Completion Dequeue address
0xFC 0004 0200		Diagnostic Address register
0xFC 0004 0208		Diagnostic Data register
0xFC 0004 0210		Diagnostic Read Memory Tag address
0xFC 0004 0218		Diagnostic Write Memory Tag address
0xFC 0004 0220		Diagnostic Read Memory Data address
0xFC 0004 0228		Diagnostic Write Memory Data address
0xFC 0004 0230	MAC 0, Page 0	Diagnostic Read Memory ECC address
0xFC 0004 0238		Diagnostic Write Memory ECC address
0xFC 0004 0240		Diagnostic Initialize Memory address
0xFC 0004 0248		Diagnostic Scrub Memory address
0xFC 0004 F148	MAC 0, Page F	Message Completion Enqueue address
0xFC 0004 F150		Message Allocation address
0xFC 0005 0000 - 0xFC 0005 FFFF	Hypernode 0, TAC 0	Hypernode 0, TAC 0 CSR space
0xFC 0005 0000	TAC 0, Page 0	System Configuration register
0xFC 0005 0008		TAC Chip Configuration register
0xFC 0005 0010		TAC Ring Configuration register
0xFC 0005 0018		TAC Chip Status register
0xFC 0005 0020		TAC Set Weak Order Enable register

Table 35 Global CSR map **(continued)**

40-Bit physical address	CSR space	CSR register name
0xFC 0005 0028		TAC Clear Weak Order Enable register
0xFC 0005 0080		TAC Error Cause register
0xFC 0005 0088		TAC Error Info register
0xFC 0005 0090		TAC Error Address register
0xFC 0005 0098		TAC Error Configuration register
0xFC 0005 0300	TAC 0, Page 0	Time-of-Century Configuration register
0xFC 0006 0000 - 0xFC 0006 FFFF	Reserved	
0xFC 0007 0000 - 0xFC 0007 FFFF	Reserved	
0xFC 0008 0000 - 0xFC 000F FFFF	**Hypernode 0, XBar Port 1**	
0xFC 0010 0000 - 0xFC 0017 FFFF	**Hypernode 0, XBar Port 2**	
0xFC 0018 0000 - 0xFC 001F FFFF	**Hypernode 0, XBar Port 3**	
0xFC 0020 0000 - 0xFC 0027 FFFF	**Hypernode 0, XBar Port 4**	
0xFC 0028 0000 - 0xFC 002F FFFF	**Hypernode 0, XBar Port 5**	
0xFC 0030 0000 - 0xFC 0037 FFFF	**Hypernode 0, XBar Port 6**	
0xFC 0038 0000 - 0xFC 003F FFFF	**Hypernode 0, XBar Port 7**	
0xFC 0400 0000 - 0xFC 043F FFFF	**Hypernode 1**	
0xFC 0800 0000 - 0xFC 083F FFFF	**Hypernode 2**	
	• •	
0xFD FC00 0000- 0xFD FC3F FFFF	**Hypernode 0x7F**	

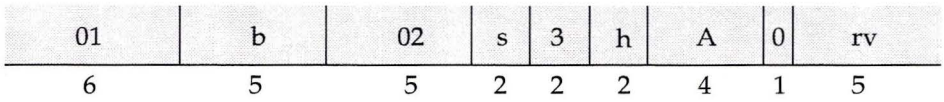
CTI cache instructions

B

There are three CTI cache instructions. The formats for these instructions are shown on the following pages.

CTI Cache Flush Global — NCFG

Format: **NCFG (s,b)**



01	b	02	s	3	h	A	0	rv
6	5	5	2	2	2	4	1	5

Description: The memory line specifed by the effective address generated by the instruction is written back to memory if it is dirty in any cache on any node of the system, all shared copies of the memory line are marked invalid. After execution of the instruction, the memory line will not be cached anywhere in the system. The offset is formed from the base register, *b*.

The hint field, *h*, determines whether read or write access protection checks are performed. The h-field of the instruction also specifies whether speculative execution of the instruction is allowed. *Speculative execution* occurs when the h-field value is 2. (Table 36 defines the assembly language h-field values.)

Table 36 NCFG access protection checking

Description	h
Read access protection check	2
Read and write access protection check	3

The PSW D-bit (Data address translation enable) determines whether a virtual or absolute address is used.

Operation:

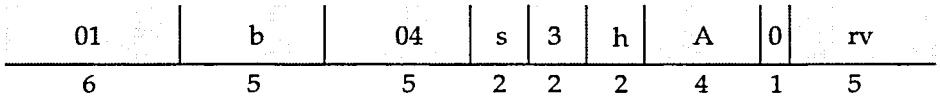
```
If (RDR enabled) {
    space ← space_select(s,GR[b],SHORT);
    offset ← GR[b];
    Global_flush(space, offset);
} else
    Illegal_instruction_trap
```

Exceptions: Nonaccess data TLB miss fault Illegal_instruction_trap
Data memory access rights trap

Restrictions: The base register must be 32-byte aligned.

CTI Cache Prefetch Read — NCPR

Format: **NCPR (s,b)**



Description: The memory line specifed by the effective address generated by the instruction is prefetched for read access into the local node's CTI cache. If the memory line was being held for private use (cached for write) prior to the instruction, then the cached line is written back to memory. The line may be cached in other node's CTI cache for read access after the instruction is performed. The offset is formed from the base register, *b*.

The hint field, *h*, determines whether read or write access protection checks are performed. The h-field of the instruction also specifies whether speculative execution of the instruction is allowed. *Speculative execution* occurs when the h-field value is 2. (Table 37 defines the assembly language h-field values.)

Table 37 NCPR access protection checking

Description	h
Read access protection check	2
Read and write access protection check	3

The PSW D-bit (Data address translation enable) determines whether a virtual or absolute address is used.

Operation:

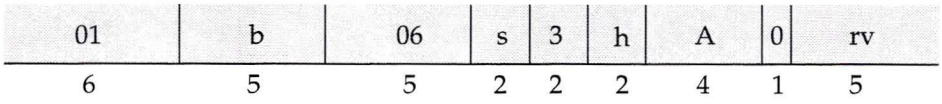
```
If (RDR enabled) {
    space ← space_select(s,GR[b],SHORT);
    offset ← GR[b];
    CTIcache_prefetch_read(space, offset);
} else
    Illegal_instruction_trap
```

Exceptions: Nonaccess data TLB miss fault Illegal_instruction_trap
Data memory access rights trap

Restrictions: The base register must be 32-byte aligned.

CTI Cache Prefetch Write — NCPW

Format: **NCPW (s,b)**



01	b	06	s	3	h	A	0	rv
6	5	5	2	2	2	4	1	5

Description: The memory line specifed by the effective address generated by the instruction is prefetched for write access into the local node's CTI cache. If the memory line was being held for private use (cached for write) prior to the instruction, then the cached line is written back to memory. The line will not be cached in any other node's CTI cache after the instruction is performed. The offset is formed from the base register, *b*.

The hint field, *h*, determines whether read or write access protection checks are performed. The h-field of the instruction also specifies whether speculative execution of the instruction is allowed. *Speculative execution* occurs when the h-field value is 2. (Table 36 defines the assembly language h-field values.)

Table 38 NCPW access protection checking

Description	h
Read access protection check	2
Read and write access protection check	3

The PSW D-bit (Data address translation enable) determines whether a virtual or absolute address is used.

Operation:

```
If (RDR enabled) {
    space ← space_select(s,GR[b],SHORT);
    offset ← GR[b];
    CTIcache_prefetch_read(space, offset);
} else
    Illegal_instruction_trap
```

Exceptions: Nonaccess data TLB miss fault Illegal_instruction_trap
Data memory access rights trap

Restrictions: The base register must be 32-byte aligned.

Glossary

A

absolute address
An address that does not undergo virtual-to-physical address translation when used to reference memory or the I/O register area.

address
A number used by the operating system to identify a storage location. Also a unique number or character string that identifies a particular network hypernode.

address space
Memory space, either physical or virtual, available to a process.

architecture
The physical structure of a computer's internal operations, including its registers, memory, instruction set, and I/O structure.

B

barrier synchronization
A control mechanism used in parallel programming that ensures all CPUs have completed the prior operation before continuing with the next operation.

block
A group of data containing a fixed number of bytes.

block TLB
A type of TLB entry that translates many contiguous virtual pages to an equal number of contiguous physical pages.

boot
The procedure by which a program is initiated the first time. Typically, a bootstrap is performed when power is first applied to the processor.

buffer
A temporary storage area. Several types of buffers are used in computer systems, in both hardware and software. The most common types of buffers are those maintained by a computer operating system to mediate between processes and I/O devices.

bus
A data path shared by several components within a computer system.

C

cache
See *cache memory*.

cache hit
The term used when a processor locates the address of a memory object in a cache.

cache memory
A small, high-speed buffer memory used in modern computer systems to hold temporarily those portions of the contents of the main memory that are, or believed to be, currently in use.

cache miss
The term used when a processor fails to locate the address of a memory object in a cache.

cache purge
The act of removing entries in a cache memory.

cacheable
Memory references that are eligible for cache move ins.

check
A type of *interruption* caused by the detection of an internal hardware detected malfunction.

coherency
A term frequently applied to caches. If a data item is referenced by a particular processor on a multi-processor system, the data is copied into that processor cache and is updated there if the processor modifies the data. If another processor references the data while a copy is still in the first processor cache, a mechanism is needed to ensure that the second processor does not use an outdated copy of the data from memory. The state that is achieved when both processors' caches always have the latest value for the data is called cache coherency.

communication, interprocessor
The process of moving or sharing data, and synchronizing operations between processors on a multiprocessor system.

crossbar
A switching device used in multiprocessor, shared-memory computer systems that connects CPUs to the various banks of memory in the system.

CSR
Control and status register. A CSR is a software-addressable hardware register used to hold control or state information.

CTI cache
In an X-Class server, it is a portion of coherent memory that contains data fetched from other hypernodes.

CTI ring
The ring interconnect that connects all the hypernodes of an X-Class server together.

D

default processor set
All processors in a hypernode are initially assigned to a default set when the hypernode is booted. A processor remains in the default set until it is explicitly assigned to a sub-complex. When a processor is removed from a sub-complex, it is returned to the default set.

diagnostic insertion
A method used on S-Class and X-Class servers using diagnostic instructions for explicitly specifying the TLB entry to be used for the next TLB insertion.

direct memory access (DMA)
A procedure or method defined for gaining direct access to main storage and achieving data transfers without involving the processor.

DMA
See direct memory access.

E

EEPROM
See electrically erasable programmable read-only memory.

electrically erasable programmable read-only memory (EEPROM)
A read-only memory module that can be programmed repeatedly by first erasing the previous contents of the memory module. Unlike the EPROM, the EEPROM can be reprogrammed without removal form the circuit board by applying an erase signal to the device.

error code
The status returned by a function call.

error correction code (ECC)
Code used to decide which bit of a memory read operation is in error.

exception
A hardware-detected event that disrupts the running of a program, process, or system. See also *fault*.

F

far-shared memory
Memory that is globally accessible from all hypernodes in an X-Class server and has equal access latency from all hypernodes.

fault
A type of *interrupt* caused by an instruction that requests a legitimate action that cannot be carried out immediately due to a system problem.

G

Globally shared memory (GSM)
A memory architecture in memory can be accessed by all processors in the system. This architecture can also support virtual memory. This type of memory is sometimes referred to as shared virtual memory or global virtual memory.

H

hard error
An uncorrectable data error.

hardware address
The device-dependent physical address of a hypernode attached to a communication line.

High-Performance Parallel Interface (HIPPI)
Currently the fastest industry standard for connecting high-performance computers. HIPPI hardware consists of HIPPI channel control unit (CCU) that supports dual simplex connections (one input, one output) that currently provides a data rate of up to 800 megabits per second over distances of up to 25 meters.

high-priority machine check (HPMC)
Similar to a trap. It indicates that a process error occurred and that the process cannot continue. In most cases, the system does not require rebooting after an HPMC.

HPMC
See *high priority machine check.*

hypernode
In S-Class and X-Class servers, a set of up to 16 processors and up to 16 Mbytes of physical memory organized as a symmetric multiprocessor (SMP) running a single image of the operating system microkernel. An X-Class server consists of one or more hypernodes, with a high speed CTI connecting the hypernodes.

hypernode-private memory
Memory that is accessible by any CPU on a single hypernode. Compare with *CPU-private memory, near-shared memory,* and *far-shared memory.*

I

instruction cache (Icache)
Memory used to hold frequently accessed instructions.

interface
A physical path between any two modules or systems.

interleaved memory
Memory that is divided into multiple banks to permit concurrent memory accesses. The number of separate memory banks is referred to as the memory interleave. Programs can optimize memory accesses by using stride intervals so that each of the banks can be refreshed between memory accesses.

interrupt
An occurrence that changes the normal flow of instruction execution. An interrupt originates from hardware, such as an I/O device. See also *maskable interrupt.*

interval timer
An interval timer is used to generate an interrupt based on the passage of time.

I/O hypernode
A hypernode that contains a chassis filled with peripheral devices (e.g., disk drives) in place of a CPU chassis.

J

JTAG
The Joint Test Action Group was formerly a group of European and later American companies that developed a boundary scan technique to facilitate in-circuit testing and functionality testing of circuit boards. It was handed off to the IEEE and refined as IEEE Standard 1149.1.

L

latency
The time delay between the issuing of an instruction and the completion of the operation. A common metric used for comparing parallel processor systems is the latency of coherent

logical memory
The memory space as seen by the program. Also called virtual memory. See also *page*.

M

MAC
Memory Array Controller. The gate array that controls hypernode memory. It interfaces to the routing array controller (RAC) and the CTI ring or toroidal interface controller (TAC). Each MAC controls four banks of memory, allowing up to 32 banks in an eight-MAC hypernode.

main memory
See physical memory.

maskable interrupt
An interrupt to which the operating system may choose not to respond.

message passing
A type of programming in which program modules (often running on different processors or different hosts) communicate with each other by means of system library calls that package, transmit, and receive data.

most significant bit (MSB)
The bit contributing to the largest quantity to the value of a binary numeral. Bit numbering in S-Class and X-Class servers is left to right, N-1 through 0. The MSB is N-1

move-in
The operation of bringing information from memory into a cache.

multiprocessing
The creation and scheduling of processes on any subset of CPUs in a system configuration.

N

near-shared memory
Memory that is globally accessible by all hypernodes of an X-Class server, but has affinity for its home hypernode. Latency is lowest from the home hypernode and higher from other hypernodes. Compare with *CPU-private memory*, *hypernode-private memory*, and *far-shared memory*.

noncoherent memory reference
A memory reference that 1) does not cause a cache move-in, or 2) causes a cache move-in, but fails to obey cache coherency rules.

O

ownership reflection
Data sent to a processor from another indicating that a particular message or packet of data "belongs" to it. For the processor, any data received without an ownership reflection is considered invalid.

P

PA-RISC
The Hewlett-Packard precision architecture reduced instruction set computer. A RISC instruction set is easy to decode in hardware and for which a compiler can generate highly optimized code.

PAC
Processor agent controller. The gate array that interfaces to pairs of PA-RISC processors.

packet
A group of related items. A packet may refer to the arguments of a subroutine or to a group of bytes that is transmitted over a network.

page
A page is the unit of logical memory controlled by the memory management algorithms. A page in S-Class and X-Class servers is 4 Kbytes (4,096) contiguous bytes.

page fault
A page fault occurs when a process requests data that is not currently in main memory. The machine first saves the state of all controllers onto a context stack in main memory. The operating system creates a free page of physical memory to bring the data in from the disk. The appropriate Page Table Entries (PTEs) are set up so that the proper logical-to-physical translation occurs. The machine reads back from memory the state of the machine from the context stack, and restores the processor to the same state that it was in when it determined that the data it needed was nonresident. The CPU can then continue with normal operation of the process.

page frame
A page frame is the unit of physical (main) memory in which pages are placed. Referenced and modified bits associated with each page frame aid in memory management.

PIC
PCI Interface Controller. The heart of the S-Class and X-Class I/O subsystem. The PIC connects to the PCI bus and provides and interface between I/O devices and the PAC.

PDIR
Physical page directory. PA-RISC CPUs that implement hardware TLB miss handlers, may fetch TLB entries from a PDIR entry in the event of a TLB miss. The PDIR serves as a cache of virtual-to-physical page translations, and is maintained by the operating system.

physical address
A unique identifier that selects a particular device from the set of all devices connected to a particular bus.

physical address space
The set of possible addresses for a particular bus.

physical memory
Memory devices, usually RAM, connected as a subsystem that provide fast-access storage for the operating system, applications, and data.

process
The fundamental unit of a program that is managed by the job scheduler. A collection of one or more execution streams within a single logical address space; an executable program. A process is made up of one or more threads.

process memory
The portion of system memory that is used by an executing process.

R

RAC
Routing Array Controller. The ASIC (four required) used to in the hypernode crossbar. The crossbar provides an interface between the processors and I/O devices and hypernode memory.

read
A memory operation in which the contents of a memory location are copied and passed to another part of the system.

register
A hardware entity that contains an address, operand, or instruction status information.

reset
The process of establishing a known state in a machine register.

RISC
See reduced instruction set computer.

runway bus
The data interface of the Hewlett-Packard PA-8000 processor. It allows multiprocessor systems to interface to memory and I/O without additional components.

S

Scalable Parallel Processing (SPP)
A computer architecture that permits an application to run with a relatively small number of processors or in a processor array containing many processors. A key design goal for SPP systems is to enable performance to increase linearly with respect to its number of processors.

SDRAM
Synchronous Dynamic Random Access Memory.

semaphore
A group of bits associated with data structures that act as a flag to all processors to synchronize the threads of a multiple-thread process. See also *synchronization*.

server
A process that fulfills a request issued by a client process, and transmits a response back to the client.

snooping
Externally flushing or invalidating a cache line from a processor cache. The flushing or invalidating transaction is issued by the processor agent and is referred to as a "snoopy transaction". This action occurs when one processor in the system loads or stores to a *dirty* line in the cache of another processor, or when a processor stores to a line that is shared by one or more processors in the system.

soft error
Correctable single-bit memory error. May further be defined as transient (non-reproducible) or stuck (reproducible).

space
A contiguous range of virtual addresses within the system wide virtual address space.

SPP
See *scalable parallel processing (SPP).*

subcomplex
In S-Class and X-Class servers, a logical entity that provides control over the allocation of processors and physical memory to different applications and users.

subcomplex server
In S-Class and X-Class servers, a subsystem within the process manager that provides the system call interface for creating and manipulating subcomplexes and server sets on the system.

synchronization
A way to keep two threads from accessing the same critical region simultaneously. You can synchronize programs using compiler directives, thread library calls, or assembly-language instructions. You do so, however, at the cost of additional overhead; synchronization may cause at least one CPU to wait for another.

system console
The terminal or workstation that serves as a communication device between the system manager and the computer system. On S-Class and X-Class servers, the *test station* serves as the system console.

system subcomplex
In a S-Class and X-Class server, a subcomplex that is automatically created at boot time by the operating system to run system processes, including `init` and processes spawned by `init`. The Complex Manager will not allow users to destroy this subcomplex, nor to remove the last processor from this subcomplex.

T

TAC
Toroidal Access Controller. The ASIC that provides hypernode an interface to other hypernodes across the CTI.

TLB
Translation Lookaside Buffer. A hardware structure in each processor of S-Class and X-Class servers that contains the information necessary to translate a virtual memory reference to a physical page and to validate memory accesses.

TOC
Time of Century. In S-Class and X-Class servers, the TOC register is used to time stamp trace data and interprocess messages.

test station
The workstation that is used to diagnose problems and install system software.

thread
An independent execution stream that is fetched and executed by a CPU. One or more threads, each of which can execute on a different CPU, make up each process. Memory, files, signals, and other process attributes are generally shared among threads in a given process, enabling the threads to cooperate in solving the common problem. Threads are created and terminated by instructions that can be automatically generated by compilers, inserted by adding compiler directives to source code, or coded explicitly in Fortran, C, or C++ programs.

thread-private or thread-specific
Data that is accessible by a single thread only (not shared among the threads constituting a process). Thread-specific data allows the same virtual address to refer to different physical memory locations.

trap
A type of *interrupt* caused when either the function requested by the current instruction cannot or should not be carried out, or system intervention is desired by the user before or after the current instruction is executed. Typically, this condition is a result of unexpected arithmetic results.

V

virtual address
An address used by a program to access data or instructions. The S-Class and X-Class servers map each virtual address to physical memory location.

virtual alias
Two different virtual addresses that map to the same physical memory address.

W

wall-clock time
The time an application requires to complete its processing. If an application starts running at 1:00 p.m. and finishes at 5:00 a.m. the following morning, its wall-clock time is sixteen hours.

weak-store ordering
A memory reference that is not guaranteed to be completed after all previous memory references are completed, or before any subsequent memory references are completed.

wired-down
The term that applies to virtual-to-physical address translation indicating that the two addresses remain the same after translation.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

A4716-90001