Architecture Reference Guide

V2500 Server

First Edition

A5074-96004

V2500 Server

Customer Order Number: A5074-90004

June, 1999

Printed in: USA

Revision History

Edition: First

Document Number: A3725-96004
Remarks: Initial release June, 1999.

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.

Contents

Figures

Tables

Preface

The document describes the architecture of the Hewlett-Packard V2500 on the PA-8500, the latest in a line of high performance Precision Architecture—Reduced Instruction Set Computer (PA-RISC) processors from Hewlett-Packard Company.

Notational conventions

This section describes notational conventions used in this book.

`bold monospace`	In command examples, **`bold monospace`** identifies input that must be typed exactly as shown.
`monospace`	In paragraph text, `monospace` identifies command names, system calls, and data structures and types. In command examples, `monospace` identifies command output, including error messages.
italic	In paragraph text, *italic* identifies titles of documents and provides emphasis on key words. In command syntax diagrams, *italic* identifies variables that you must provide. The following command example uses brackets to indicate that the variable *output_file* is optional: `command` *input_file* [*output_file*]
Brackets ([ ])	In command examples, square brackets designate optional entries.
Curly brackets ({}), Pipe (|)	In command syntax diagrams, text surrounded by curly brackets indicates a choice. The choices available are shown inside the curly brackets and separated by the pipe sign (|). The following command example indicates that you can enter either `a` or `b`: `command {a | b}`

Horizontal ellipses (...)	In command examples, horizontal ellipses show repetition of the preceding items.
Vertical ellipses	Vertical ellipses show that lines of code have been left out of an example.
Keycap	**Keycap** indicates the keyboard keys you must press to execute the command example.

NOTE A note highlights important supplemental information.

1 Introduction

The V2500 server provides multipurpose, scalable computing resources through shared memory and I/O designed to provide high throughput and quick time to solution.

V2500 server uses the PA-8500, the latest in a line of high performance Precision Architecture-Reduced Instruction Set Computer (PA-RISC) processors from Hewlett-Packard Company.

V2500 Systems can be configured to consist of one to four cabinets. A cabinet is often referred to as a node. Each node may have eight to 32, 440-Mhz PA-8500 processors. The processors and memory of each node are tightly coupled as a cache-coherent Non-Uniform Memory Architecture (ccNUMA) system. Each node in a V2500 system is tightly intergrated with memory and processors in all other nodes.

The PA-8500 processor

The V2500 server uses the Hewlett-Packard PA-8500 processor 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-8500. This document does not attempt to duplicate information in that manual. Instead, it presents only V2500 server-specific information.

The processors of the system are supported by several Application-Specific Integrated Circuit (ASIC) hardware controllers, an enhanced memory system, and a high-bandwidth I/O subsystem. Special hardware and software allow these processors to perform both as conventional single processors or together in parallel to solve more complex problems.

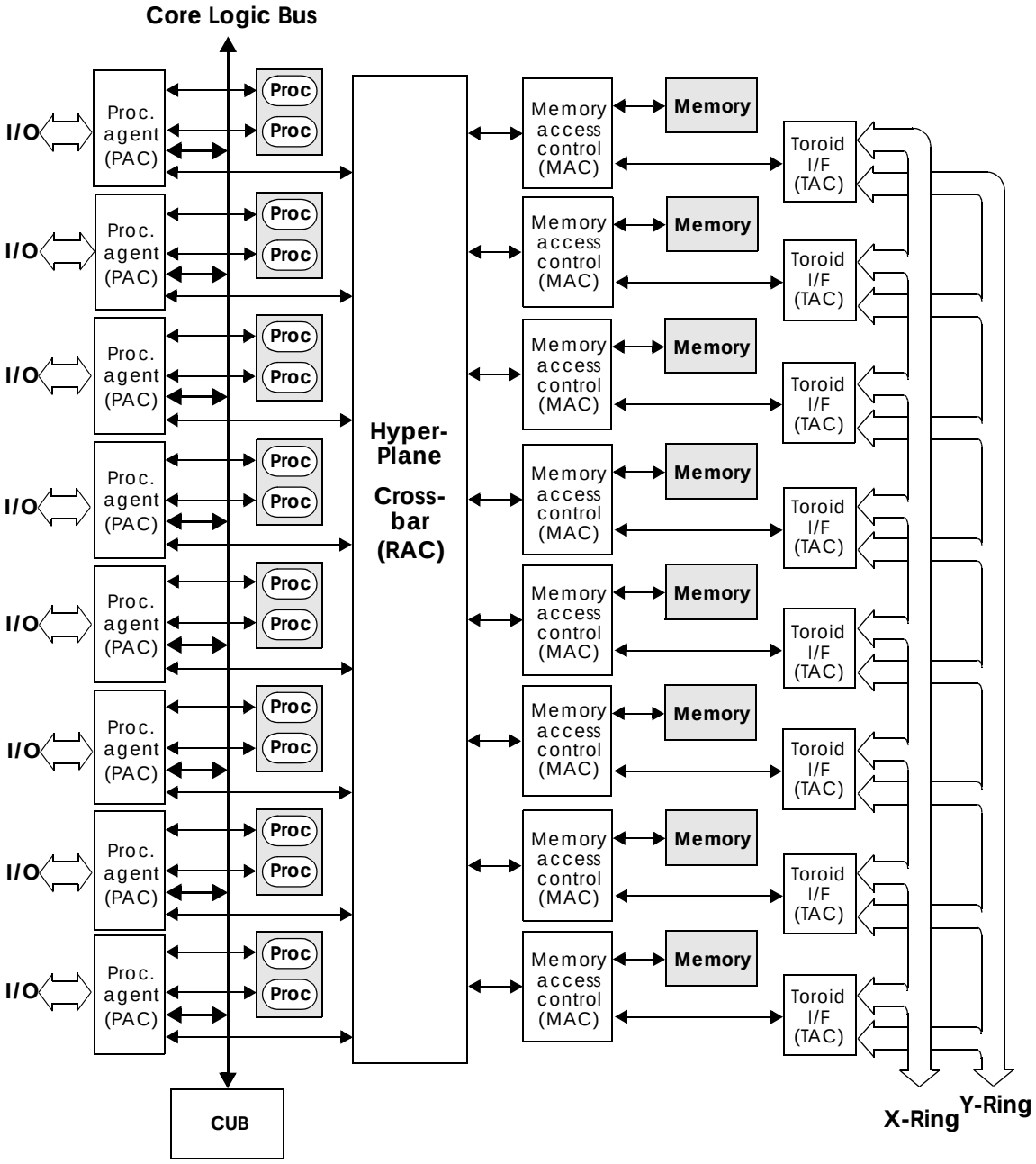
The node

The V2500 Server can contain eight to 32 processors. The processors and the associated hardware comprise what is commonly called a node. The terms node and system are used interchangeably in this book. The node uses a symmetric multiprocessor (SMP) design that can exploit fine-grain parallelism.

A conceptual block diagram of the system is shown in Figure 1. Centrally located in the diagram is the Hyperplane crossbar which is comprised of four Routing Attachment controllers (RAC). The Hyperplane crossbar allows all of the processors to access all available memory. Processors are installed on Processor Agent controllers (PACs). A PAC allows the processor and the I/O subsystem (the PCI-bus Interface controller or SAGA) access to the Hyperplane crossbar. Also connected to the Hyperplane crossbar are the Memory Access controllers (MAC). Up to four processors are located on each PAC. Memory is controlled by the MAC. Input and output devices connect to the system through SAGA which is connected to the processor agents.

The Core Utilities board (CUB—commonly called the Utilities board) in the node contains a section of hardware called the core logic. It provides interrupts to all of the processors in the system through the core logic bus which connects to each PAC. The CUB attaches to the Midplane Interconnect Board (MIB) centrally located in the node.

Figure 1 Functional block diagram of a V2500 system



Control and status registers (CSRs)

V2500 systems use CSRs located in the processors and controllers to provide control, status, or both to the processors and other hardware in the system. Each CSR is memory mapped and is available to all processors in the system. Many of the registers are described in detail by functional groups, such as system configuration, messaging and data copy, I/O, and so on. These descriptions appear throughout this book.

Description of functional blocks

Each block in Figure 1 is described in the following sections.

Processor agent controller

The PAC can connect to zero or one to four PA-8500 processors. It can also connect to zero or one I/O controller (SAGA). With no processors, the PAC serves as an I/O-only 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 per second.
- Hyperplane crossbar port bus (0, 1)—Four 32-bit, unidirectional buses connected to two Hyperplane crossbar RACs, two in each direction. These buses have a total raw bandwidth of 1.9 GBytes per second.
- 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 per second or 240 Mbytes per second, depending on the width of the interface.
- Core Logic Bus interface—A single bidirectional bus that supports boot and support services.

The PAC sends and receives transactions from the RACs using four unidirectional data paths. There are four RACs in the Hyperplane crossbar. Each processor agent, however, communicates with only two of the four RACs.

Routing attachment controller—Hyperplane crossbar

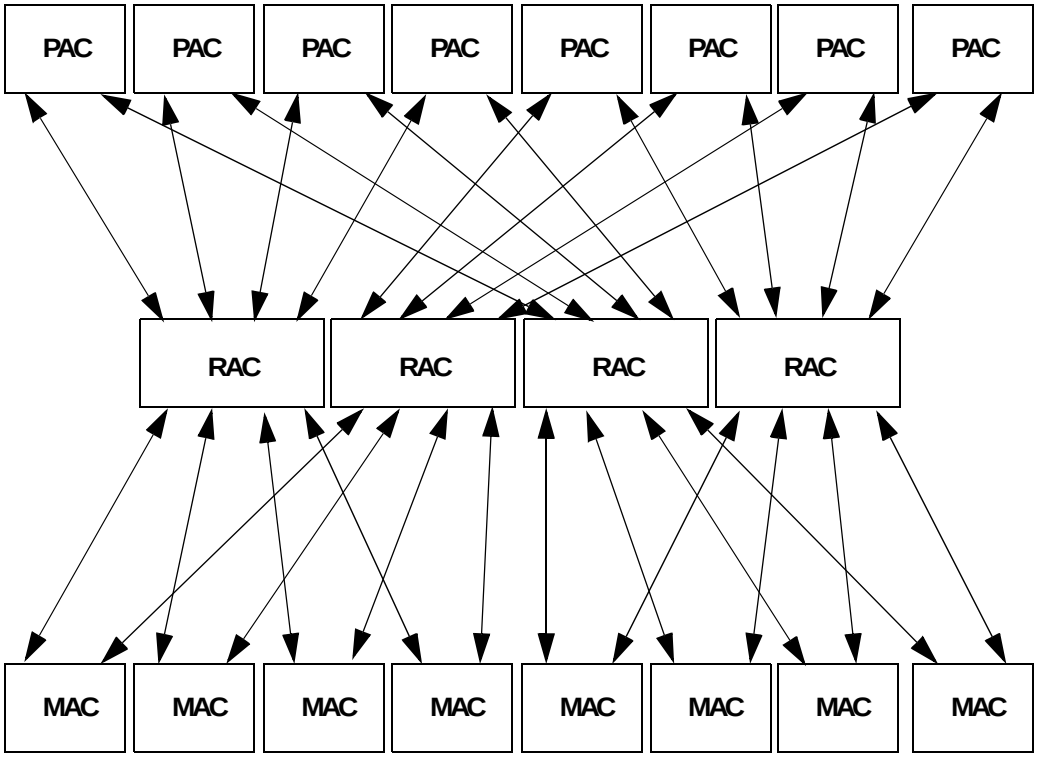
The Hyperplane crossbar is comprised of four RACs that provide an interconnect for each processor and I/O device to memory.

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 in each direction. Each port has simultaneous (input and output) bandwidth of 960 MBytes per second.
- MAC Port (A, B, C, D)—Eight 32-bit, unidirectional interfaces to four MAC ports, four in each direction. Each port has simultaneous (input and output) bandwidth of 960 MBytes per second.

Figure 2 shows how the RACs connect to each PAC and MAC.

Figure 2 RAC interconnection



Memory access controller

The MAC controls all accesses to memory. Each MAC controls 32 banks of memory, allowing up to 256 banks in an eight-MAC system. Memory banks consist of Dual In-line Memory Modules (DIMMs) of Synchronous Dynamic Random Access Memory (SDRAM).

The MAC has the following buses:

- RAC Port (A, B)—Four 32-bit, unidirectional interfaces, two in each direction. This interface supports a total simultaneous read-write bandwidth of 1.9 GBytes per second.
- Even Memory—A single 88-bit, bidirectional interface to the even memory banks associated with the MAC.
- Odd Memory—A single 88-bit, bidirectional interface to the odd memory banks associated with the MAC.

A processor accesses memory by sending a request, in the form of packets, to a RAC. The request is then forwarded to one of the MACs. The MAC routes requests into even and odd pending queues. Some packets not destined for memory are routed from processor to processor through the MAC. These packets are routed directly to the output ports.

The MAC accesses one of 32 available memory banks, checking the Error Correction Code (ECC). The data accessed from memory is returned to the processor by sending a response back to the RAC, which forwards the response to the PAC.

CUB and core logic bus

The CUB, or Utilities board, connects to the core logic bus and contains two field-programmable gate arrays (FPGAs): the Processor Utilities controller (PUC) and the Monitoring Utilities controller (MUC). The PUC allows processors access to the system core logic and booting firmware, and the MUC processes the environmental state of the system and interrupts the processors when appropriate. V2500 Servers use the core logic bus primarily to boot the system and to issue 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). A processor can write to control and status registers (CSRs) accessed using the core logic bus to initialize and configure the RAC chips and Utilities board logic.

Shared memory

V2500 servers use a shared-memory architecture to provide high-performance. This allows the developer, compilers, and applications to view the system as processors sharing a large physical memory and high-bandwidth I/O ports.

Compilers use shared memory to provide automatic, efficient parallelization, while viewing memory as a single contiguous virtual address space.

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

Sequential memory references (linearly ascending physical address) to shared memory are interleaved across up to eight memory boards on a 32-byte basis. See the chapter "Physical address space," for more information.

With all processor references to memory, copies of the accessed data are encached into either the instruction or data caches of each processor. 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 V2500 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*.

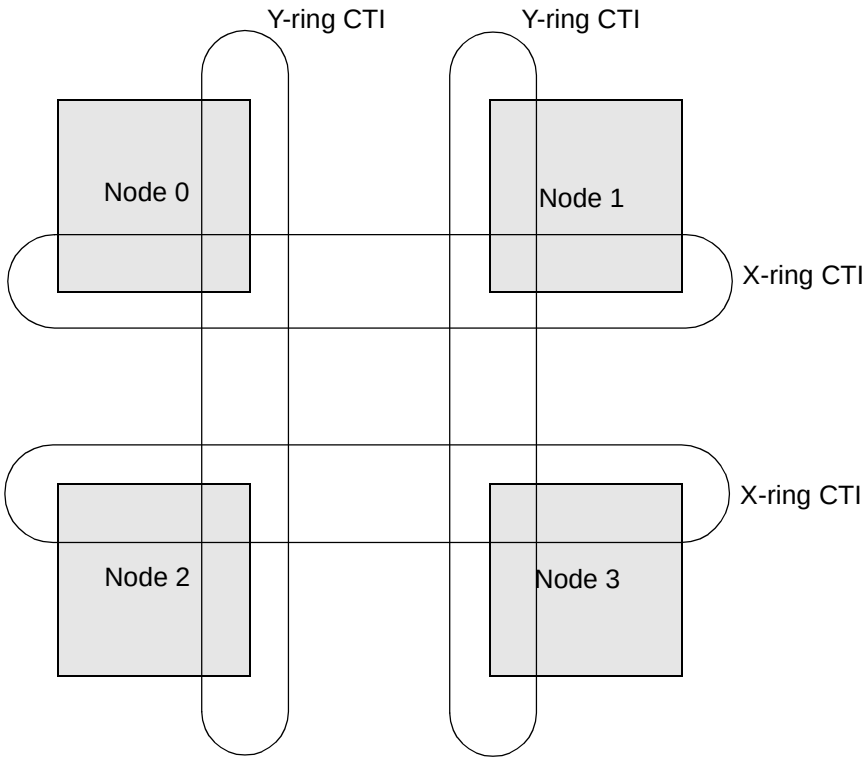
To maintain updated coherent copies, V2500 servers operate under the following rules:

- 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 (store) data 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.

Multiple nodes

Up to 128 processors in four nodes can be connected together in the V2500 multinode system. Nodes are tightly coupled by globally shared memory and CTI rings to provide minimum latency in global memory accesses. Figure 3 depicts how four nodes might be connected. The two Y rings and two X rings provide a low-latency interconnect that allows processors to access memory anywhere in the system and communicate with processors in other nodes. Each ring in the figure represents multiple interconnects. One inherent advantage to this interconnect scheme is built-in redundancy afforded by multiple CTI rings.

Figure 3 Four-node interconnection



Coherent toroidal interconnect

Multiple nodes are interconnected with one or more low-latency interconnections. The CTI refers to a collection of rings used by V2500 server nodes 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.

V2500 servers have eight physical CTI rings per dimension between nodes. 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 (CTI cache line size is also 32 bytes). Ring interleaving tends to balance the traffic across all eight rings (global memory references from a processor to the memory on the same node do not use the node 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.

V2500 servers use a coherency protocol that 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 (GSM)

All of the processors in the V2500 servers share memory both within a node (local memory) and across the entire array of nodes (remote memory), in the case of multiple node V2500 servers. Shared memory makes the system easy to program.

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.

Compilers for the V2500 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 Hyperplane crossbar that connects memory, processors, and I/O in a single node. The second level consists of the interconnection between nodes through the CTI rings.

The Hyperplane crossbar provides high-bandwidth, low-latency nonblocking access from processors and I/O channels to the node-local memory. The Hyperplane crossbar 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 node use the Hyperplane crossbar, and references from a processor to memory in another node use the Hyperplane crossbar and CTI.

GSM and memory latency

Processor references to local memory have a minimum amount of latency. In V2500 servers, references to memory located in other nodes experience some additional latency because of the additional hardware required to transfer data from the remote-node memory. Processors use instruction and data caches to reduce the number of memory accesses, thereby reducing traffic on the Hyperplane crossbar and CTI.

V2500 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.

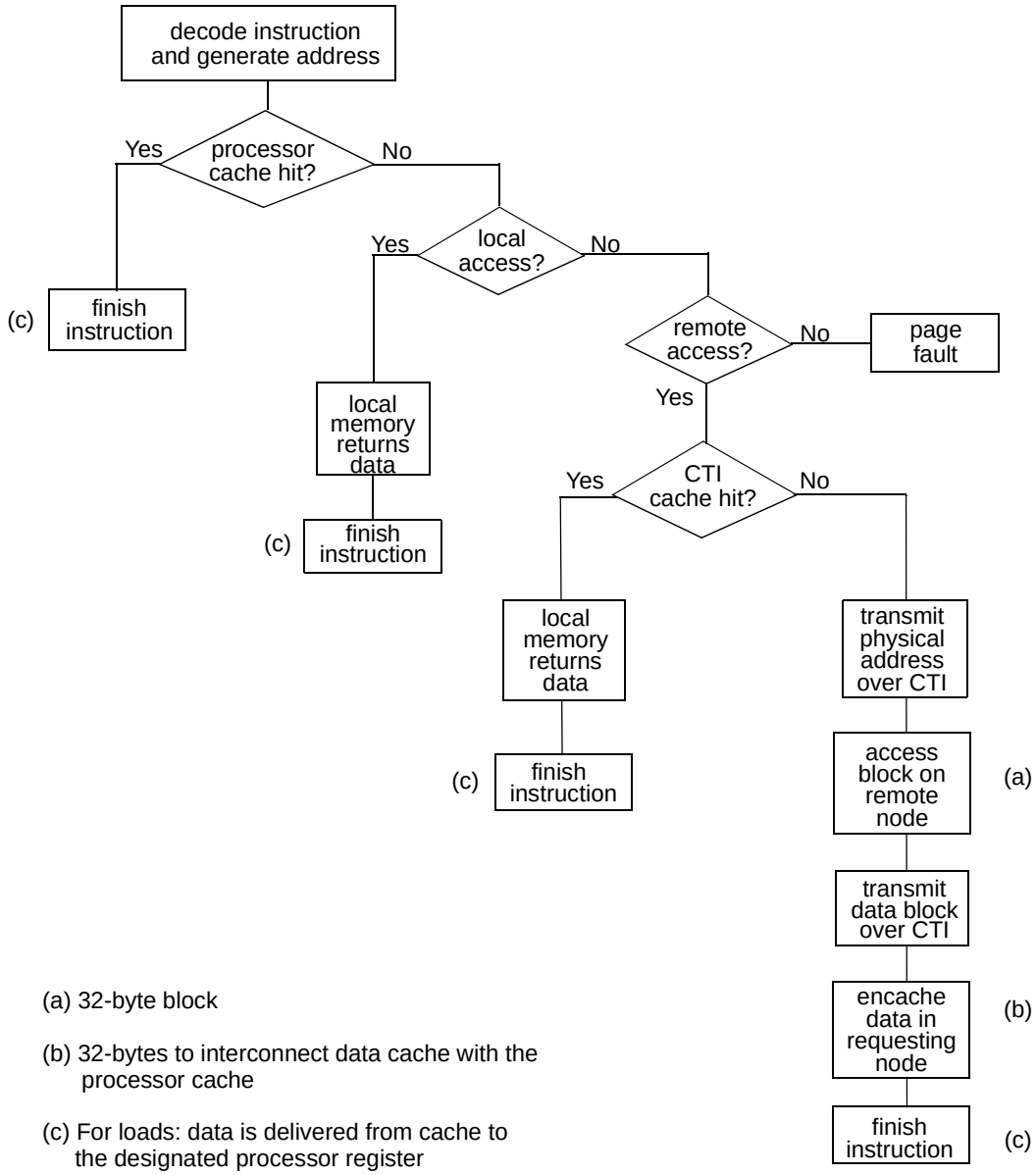
Figure 4 illustrates the various stages that are required for memory reference instructions. It depicts processing of either a load or store instruction. The processor decodes the instruction and generates the appropriate address. For a load instruction, if the address produces a hit in the processor 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 V2500 server remote memory access, the CTI cache is used in addition to the processor caches.

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

The chart in Figure 4 on page 16 illustrates how each type of memory reference experiences different latency, depending on the type.

Figure 4 Hardware processing of a load or store instruction



(a) 32-byte block

(b) 32-bytes to interconnect data cache with the processor cache

(c) For loads: data is delivered from cache to the designated processor register

For stores: data is written into processor cache from the designated processor register

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 node in the case of the V2500 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, V2500 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.
- The cache line must be "owned" exclusively by the processor in order to write data (store) into a cache line. 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 node 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.

2 Physical address space

This chapter describes the V2500 server physical address space, including coherent memory, core logic, and CSR address regions.

Physical addresses

The PA-8500 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-8500 chip, however, only 40 bits of the 64-bit physical address are implemented.

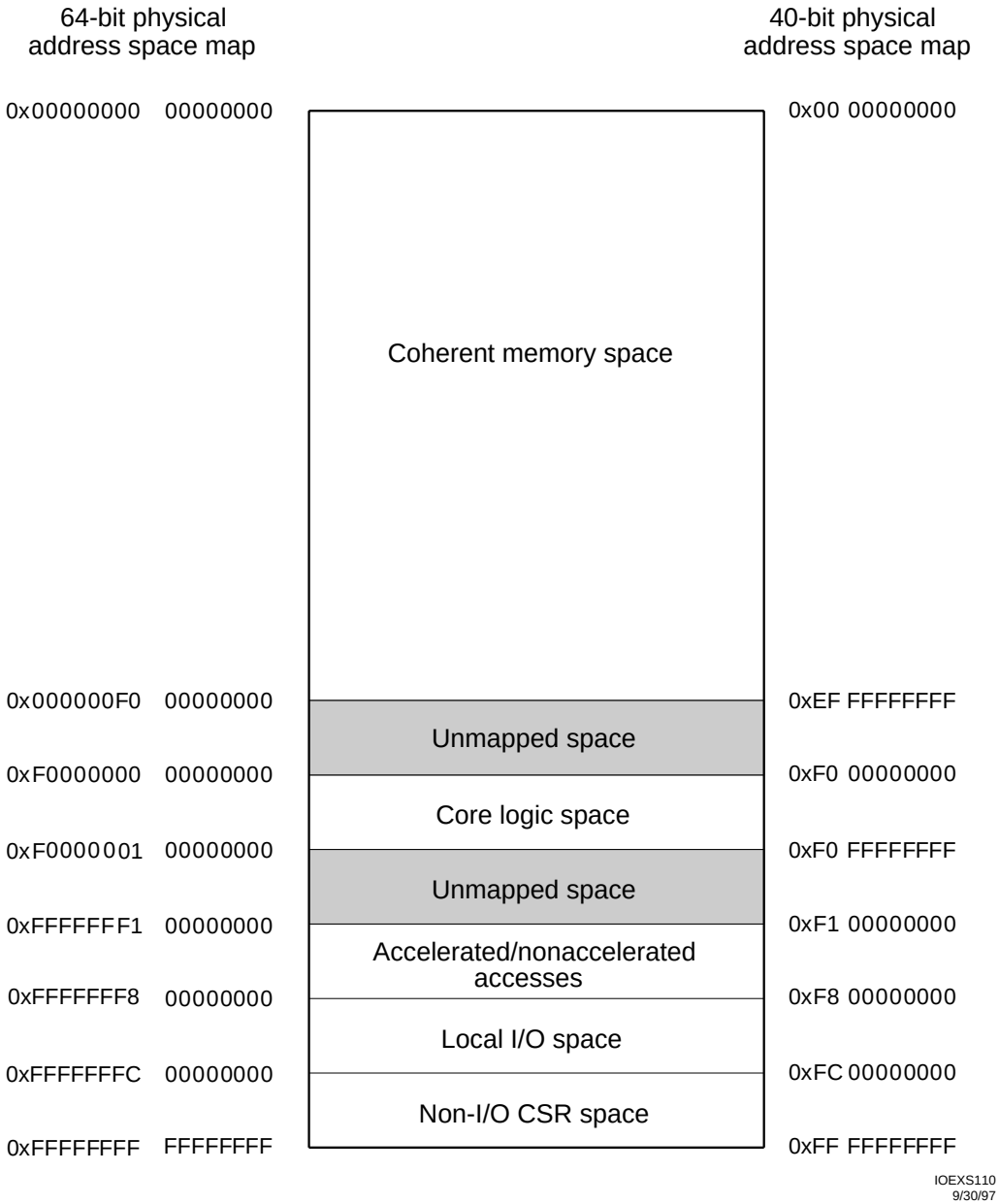
The I/O system uses controllers that have fewer than 40 address bits. The mapping of I/O addresses to the corresponding 40-bit physical address space occurs in the SAGA I/O subsystem.

Processors have four addressable physical address regions. These are:

- Coherent memory space—Memory 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 system core logic function and is accessible to all processors within the system.
- Local I/O space—The space occupied by the PCI buses and prefetch and context RAM CSRs associated with input and output devices within the system.
- Non-I/O CSR space—The space occupied by the group of all other CSRs within the system.

Figure 5 shows how the address space is partitioned.

Figure 5 Physical address space partitioning



The left side shows the PA-8500 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.

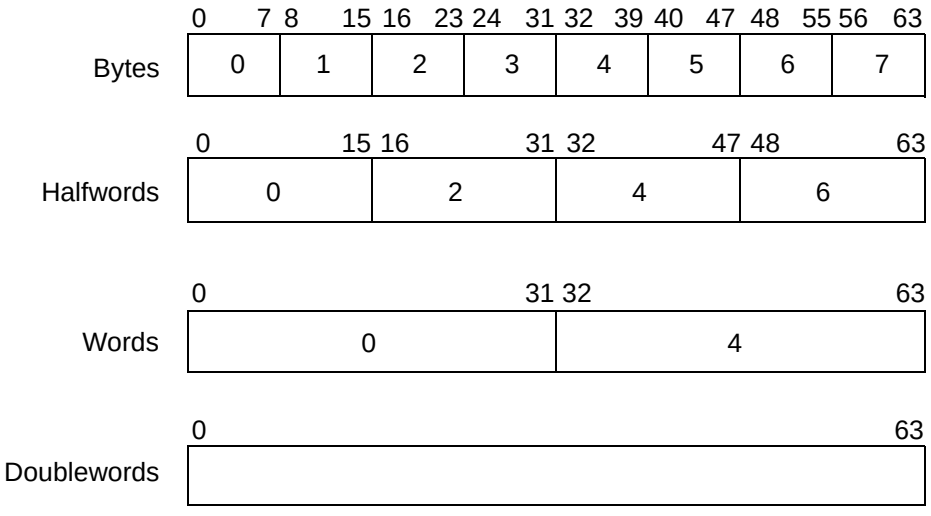
Node addressing

The physical memory within the node 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 6) 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.

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, an unaligned reference trap occurs.

Figure 6 Physical memory addressing and storage units

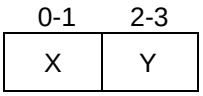


Node CSR space is referenced using bytes, halfwords, words, and double words with a strong preference for double-word accesses. Internode CSR accesses are double-word in size.

Node Identifiers

Each node of a multinode V2500 system has a unique node identifier (Node ID) of four bits. Figure 7 shows the Node ID format.

Figure 7 **Node identifier**



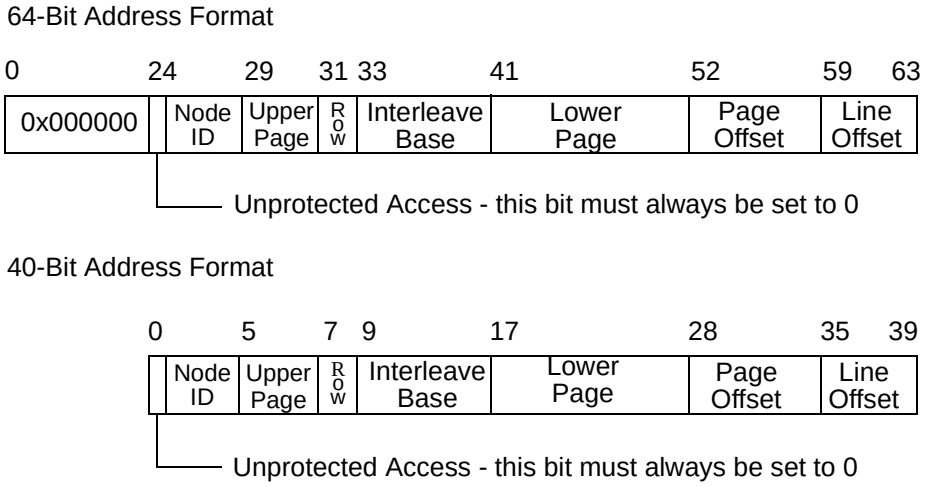
The X, and Y subfields of the Node ID correspond to the two dimensions used for configuring multi-node topologies.

A multinode V2500 system may be configured in different ways to optimize the system for its particular use. The configurations allow nodes to be placed on the vertices of a 4x4 mesh.

Coherent memory space

As shown in Figure 5, coherent memory occupies the largest amount of physical address space. Figure 8 shows both the 64-bit and 40-bit physical address formats.

Figure 8 Coherent memory space address formats



The field definitions are as follows:

- *Node ID*—Indicates the number of the node within the V2500 system.
- *Upper Page*—Selects a 1K block of pages (one of four) in a row.
- *Row*—Selects one of four rows of memory.
- *Interleave Base*—Specifies the first line of a span of interleaved memory lines
- *Lower Page*—Selects the page of memory. This field and the *Upper Page* field together select one of 8K pages.
- *Page offset*—Locates a line of memory within the selected page.
- *Line offset*—Locates the byte of memory within the selected line.

Coherent memory layout

Memory physically resides on memory boards, with each board having a memory controller (MAC). Each board has eight memory buses with up to two memory DIMMs per bus. Each memory DIMM has up to two rows of SDRAM memory chips. Each memory board is associated with a pair of CTI rings (X and Y) for remote memory accesses. Coherent memory is further divided into memory lines 32-bytes in size. Memory lines are accessed using interleaving to minimize access latency to memory. Memory interleave spans from one to four even-and-odd memory board pairs. Memory interleave spans from one to four even-and-odd memory board pairs.

Memory boards are populated with memory DIMMs, up to 16 DIMMs per board. Each DIMM can have one or two rows of SDRAM chips, with each DIMM constructed with 16-Mbit, 64-Mbit or 128-Mbit.

Multinode systems are only allowed to use certain combinations of memory boards, buses and banks. Each node must contain the same number of memory boards, and each board in the system must use the same interleave values for each row of memory.

Addressing a byte of memory

Figure 9 represents a fully populated system with 32 Gbytes of physical memory. It also shows how a byte of memory is addressed.

NOTE

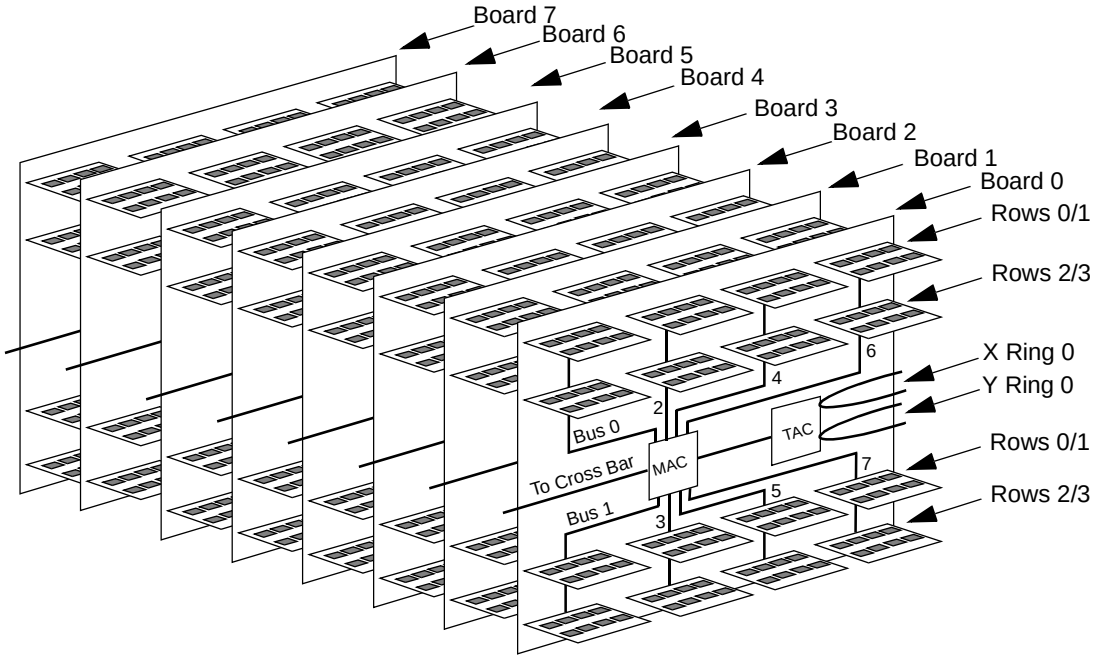
Figure 9 represents only the concept of how memory is configured in the system. It does not depict the physical implementation.

The eight memory boards (MB) are at the top of the drawing. Each board has 16 DIMMS and each DIMM is loaded with memory chips on both sides.

Each memory board has eight buses comprised of two DIMMs in the vertical direction. Also, each bus has four rows that span buses in the horizontal direction. The top and bottom of each DIMM 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 DIMMs located on the bottom of the memory board in the horizontal direction. If the memory chips were 64-Mbit SDRAMs, each board would contain two Gbytes of memory.

A ring is associated with each memory board for a total of eight rings in both dimensions (X and Y).

Figure 9 Conceptual layout of physical memory of a fully populated system



As shown in the physical address in Figure 8 and the conceptual memory layout in Figure 9, a byte of memory is accessed as follows:

1. The Interleave Base field specifies the board, bus and bank at which the first line of a span of interleaved memory lines resides. Subsequent memory lines are interleaved across the configured memory banks on a board first, then bus, and finally SDRAM bank basis..
2. The Row field specifies which of the four rows of SDRAMs the selected bank resides. The selected row spans all buses which the interleave covers.
3. The Upper Page and Lower Page fields select one of the 8K pages within a row.

4. The Page Offset field selects one of 128 memory lines in the page.
5. The Line Offset field selects the appropriate byte in the line.

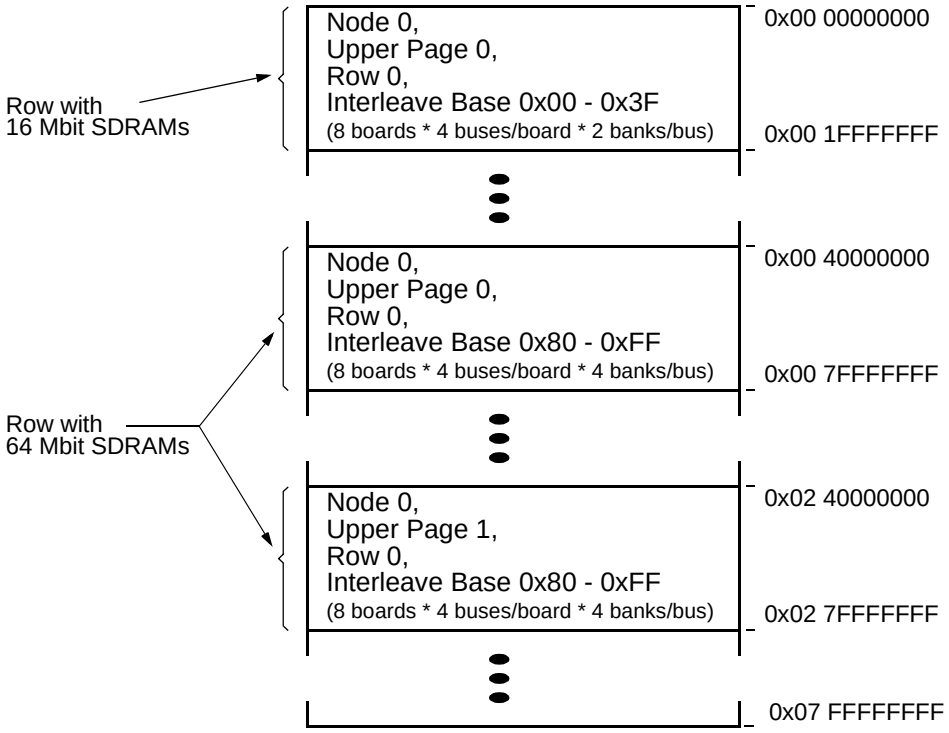
Each row contains four Gbytes with 64-Mbit SDRAMs. Within a row, 256 subpartitions exist, one for each memory board-bank combination (eight memory boards with 32 banks per board). Each subpartition is 64 Mbytes in size.

Memory interleaving

Three bus interleave options exist. These are buses zero through three, buses four through seven, and all buses (zero through seven). These options exist to allow increased flexibility of memory configurations. The options allow buses zero through three to be populated independently from buses four through seven. If all buses of a row are populated identically, then the interleave can span all eight buses, otherwise the interleave will span the lower four buses independently from the upper four buses.

Figure 10 shows an example configuration where 16-Mbit DIMMs are populated in row zero, buses zero through three, and 64-Mbit DIMMs are populated in row zero, buses four through seven. With this configuration, there are two separate interleave spans. SDRAM types can not be mixed within an interleave span.

Figure 10 Example Coherent Memory Space Layout



The most significant bit of the Interleaved Base field specifies whether an access is to buses zero through three or buses four through seven when a four bus interleave span is being referenced. The example in Figure 10 illustrates this by using Interleave Base field values of 0x80 through 0xFF for the 64-Mbit SDRAM row spanning buses four through seven.

Memory interleaving distributes consecutive lines of memory across as many banks as possible. The V2500 server supports the following memory interleave options:

- One, two or four memory board pairs
- Four or eight buses
- Two or four SDRAM banks

Memory interleave generation

Coherent memory (and thus CTI ring) interleaving is performed on all memory references. A memory board, MAC, and CTI rings (X and Y) are physically associated. To optimize memory bandwidth, memory boards must be installed on the MACs in pairs, even and odd. There is a maximum of four even-odd pairs of memory boards in a node.

A memory board has either four or eight memory buses with DIMMs installed. The four memory bus configuration is a reduced bandwidth configuration.

Figure 11 shows the mechanism for interleave. The 40-bit physical address provides the basis from which the memory blocks and memory bank are selected.

The System Configuration CSR (see the section “System Configuration register” on page 52) specifies the number of boards the interleave is to span (Memory Board Interleave Span field), whether four or eight bus interleaving is to be used on a per row basis (Memory Bus Interleave Span field), and whether two or four SDRAM bank interleave is to be used on a per DIMM basis (Memory Bank Interleave Span field).

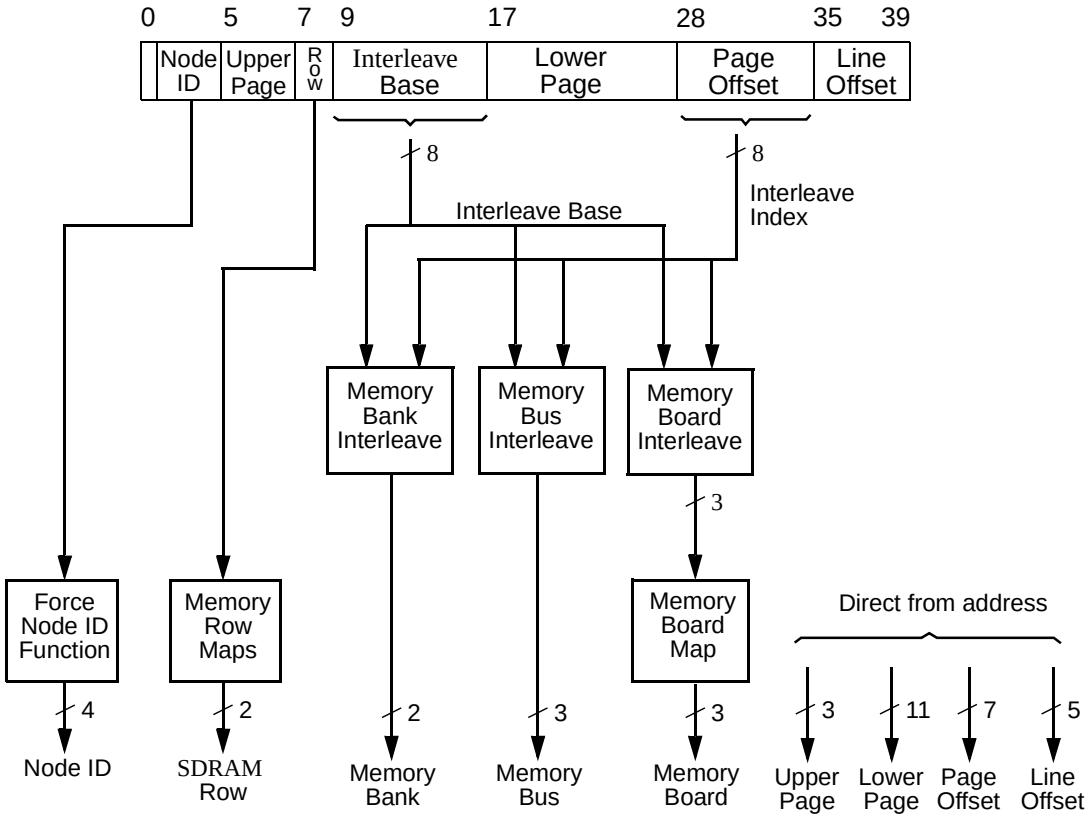
The Memory Row Configuration CSR verifies that all memory references are to memory which physically exists.

The MAC online field of the System Configuration register checks that a valid physical memory board value is generated for a memory address. An invalid memory board value results in an HPMC.

Physical address space

Coherent memory space

Figure 11 40-bit coherent memory address generation



Force node ID function

In multinode systems, the Force Node ID function allows node memory to be programmed in ranges as shown in Table 1. The default value is 128 Mbytes.

Table 1 **Force node ID programmable ranges**

Memory range	Memory size
0 - 0x1FFFFF	32 Mbyte
0 - 0x3FFFFF	64 Mbyte
0 - 0x7FFFFF	128 Mbyte

It allows the operating system to be loaded into each node without having to specify the Node ID.

The PAC verifies that the processor has issued a request within the previously specified address range and forces the local node ID into the address. When a processor performs coherency operations in this address range, the PAC forces a node ID of zero into the address prior to issuing the operation to one of the four processors.

The PAC generates the node ID portion of an address by selecting from either the Node ID field of the 40-bit PA-8500 physical address or the Node ID field of the System Configuration CSR register. The System Configuration register Node ID field is selected when the System Configuration register Force Node ID Region Size field is non-zero and the Upper Page, Row, and four, five or six most significant bits of Interleave Base field are zero.

The original node ID is checked to be the value zero prior to forcing it to the value of the local node. The check ensures cache coherency is maintained. The memory blocks checked depend on the Force ID setting as shown in Table 2.

Table 2 **Force node ID memory coherence check settings**

Memory size	Memory blocks checked
32 Mbyes	0, 1
64 Mbytes	0 - 3
128 Mbytes	0 - 7

If a processor accesses these memory blocks and the original node ID is not zero, a high-priority machine check trap occurs.

Memory board, bus and bank index selection

The Interleave Base field specifies the starting memory line bank within an interleaved span of memory banks. Increasing memory lines rotate through the span of memory banks on a memory board first, then a bus, and finally an SDRAM bank basis. Therefore, once the Interleave Base field has defined the starting memory bank, the Page Offset field bits specifies where in the rotation of spanning banks each memory line resides.

The method used to select the board, bus, and SDRAM bank base values from the Interleave Base field minimizes the number of holes in memory by using the physically contiguous least significant bits of the field. This results in the location of the memory board, bus, and bank index base values within the Interleave Base field being dependent on the configuration of the row being accessed. Table 3 defines the location of the base values for the various memory row configurations. The abbreviation IB is used to indicate the Interleave Base field of the physical address. The bits of the Interleave Base field which are not selected must be checked to have the value zero.

Table 3 Memory interleave base selection

Interleave Configuration			Base Selection		
Memory Board Even/Odd Pairs	**Memory Buses**	**Memory Banks**	**Bank Base**	**Bus Base**	**Board Base**
1	4	2 4	IB<4> IB<3:4>	IB<0, 5:6>	IB<7>
	8	2 4	IB<3> IB<2:3>	IB<4:6>	
2	4	2 4	IB<3> IB<2:3>	IB<0, 4:5>	IB<6:7>
	8	2 4	IB<2> IB<1:2>	IB<3:5>	
4	4	2 4	IB<2> IB<1:2>	IB<0, 3:4>	IB<5:7>
	8	2 4	IB<1> IB<0:1>	IB<2:4>	

The board, bus, and SDRAM bank indexes are obtained from the Page Offset field and least significant bit of the Lower Page field. The indexes are selected in a similar manner as the interleave base values. The index values are selected from the least significant bits of the field in order to maximize interleave span for a region of memory. Table 4 defines the location of the index values for the various memory configurations. The abbreviation PO is used to indicate the Page Offset field of the physical address, and LP is used to indicate the Lower Page field

Table 4 Memory interleave index selection

Interleave Configuration			Base Selection		
Memory Board Even/Odd Pairs	**Memory Buses**	**Memory Banks**	**Bank Base**	**Bus Base**	**Board Base**
1	4	2 4	PO<3> PO<2:3>	PO<4:5>	PO<6>
	8	2 4	PO<2> PO<1:2>	PO<3:5>	
2	4	2 4	PO<2> PO<1:2>	PO<3:4>	PO<5:6>
	8	2 4	PO<1> PO<0:1>	PO<2:4>	
4	4	2 4	PO<1> PO<0:1>	PO<2:3>	PO<4:6>
	8	2 4	PO<0> LP<10>,PO<0>	PO<1:3>	

Memory board interleave pattern

The memory board generated for interleave cases of one, two, and four board pairs are given in Table 5, Table 6, and Table 7, respectively. The tables show the memory board number with respect to the board base and board index values

Table 5 Memory board interleave pattern for one board pair

Board base	Board Index	
	0	**1**
0	0	1
1	1	0

Table 6 Memory board interleave pattern for two board pairs

Board base	Board Index			
	0	**1**	**2**	**3**
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Table 7 Memory board interleave pattern for four board pairs

Board Base	Board Index							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**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 bus interleave pattern

The memory bus generated for interleave cases of four and eight buses are given in Table 8 and Table 9, respectively. The tables show the interleaved memory bus for each bus base and bus index value. When interleaving across four buses, Bus Base values of 0 to 3 interleave on the lower buses and Bus Base values of 4 to 7 interleave on the upper buses.

Table 8 Memory bus interleave pattern for four buses

Bus Base	Bus Index			
	0	**1**	**2**	**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 9 Memory bus interleave pattern for eight buses

Bus Base	Bus Index							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**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

Bus Base	Bus Index							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
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 configurations allow for either two or four memory banks per SDRAM. Sixteen-Mbit SDRAMs have two internal banks, all other supported SDRAM sizes have at least four internal memory banks.

The memory bank generated for interleave cases of two and four banks is given in Table 10 and Table 11, respectively. The tables show the interleave memory bank generated for each bank base and index value.

Table 10 Memory bank interleave pattern for two banks

Bank Base	Bank Index	
	0	**1**
0	0	1
1	1	0

Table 11 Memory bank interleave pattern for four banks

Bank Base	Bank Index			
	0	**1**	**2**	**3**
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

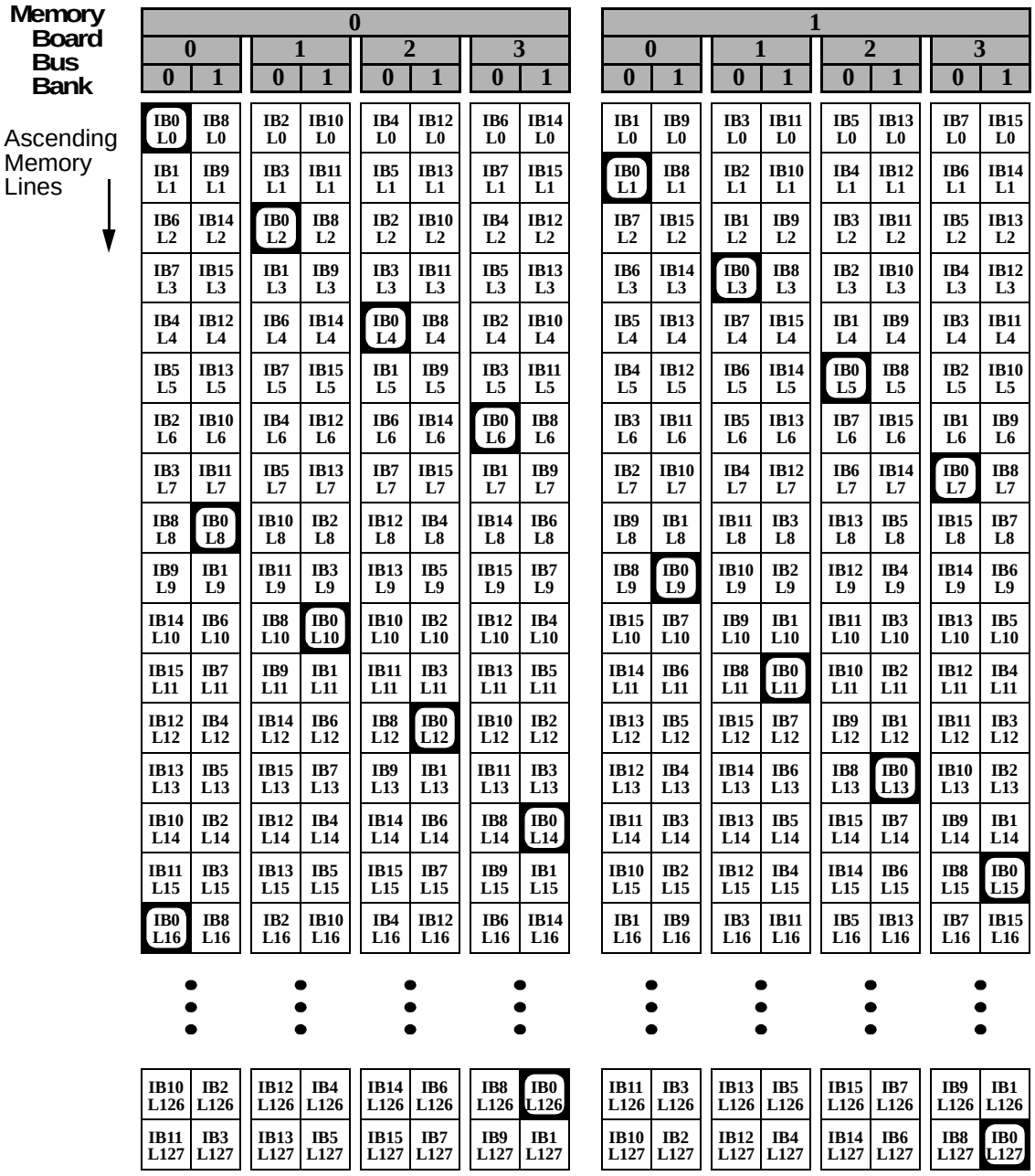
Memory board, bus bank interleave pattern

Memory lines are interleaved across the largest power-of-two memory banks that exist in each memory row. An V2500 system has up to eight memory boards, up to eight buses per memory board, and up to four SDRAM banks per bus, resulting in a maximum of 256-way memory line interleaving. The minimally configured system has two memory boards and uses only four memory buses per board and two SDRAM banks per bus for 16-way interleaving. Figure 12 shows the interleave pattern for a system with the minimum memory interleave.

Physical address space

Coherent memory space

Figure 12 Example memory line interleave pattern



Notice the pattern shown is for 4096-byte page of memory (128 memory lines per page). The memory configuration is two memory boards, four buses, and two SDRAM banks

CTI cache layout

In multinode systems, each node has a CTI cache to decrease the access latency of frequently accessed remote-node coherent memory lines. The size of the CTI cache is configurable from two Mbytes to 16 Gbytes. For single node systems, the CTI cache may be disabled. The architecture allows different sizes of CTI cache to be used on different nodes.

The ten bits shown in Figure 13 can be forced individually to configure the size and location of the CTI cache. The size of the CTI cache is determined by the number of bits forced to a value. The location of the CTI cache is determined by the specific value which the bits are forced. Table 12 shows the number of bits which must be forced for each CTI cache size. Note that bits of the Upper Page field must be forced to zero when the row(s) of memory in which the CTI cache resides are installed with SDRAMS that do not require those bits.

Figure 13 CTI cache bits within the 40-bit coherent memory address format

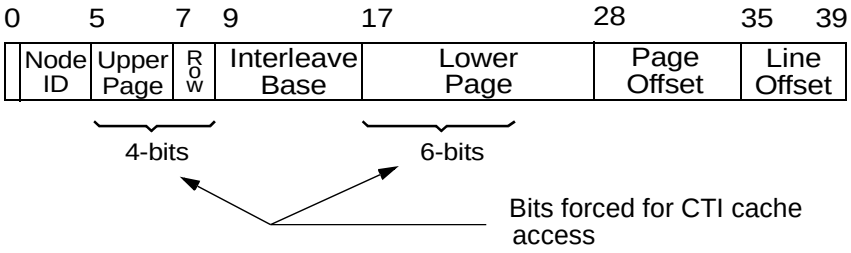


Table 12 **CTI cache size options**

Number of bits forced	Size of CTI Cache (in Mbytes)				
	Banks Cache is Interleaved across				
	16 Banks	32 Banks	64 Banks	128 Banks	256 Banks
1	1024	2048	4096	8192	16384
2	512	1024	2048	4096	8192
3	256	512	1024	2048	4096
4	128	256	512	1024	2048
5	64	128	256	512	1024
6	32	64	128	256	512
7	16	32	64	128	256
8	8	16	32	64	128
9	4	8	16	32	64
10	2	4	8	16	32

All nodes in a multinode complex must have identical memory board, bus and bank sizes. Failing this, all nodes must be in one of the configurations listed in Table 13.

Table 13 **Allowed nonmatched multinode configurations**

Boards	Buses	Banks
8	8	4
8	8	2
8	4	4
8	4	2
4	8	4

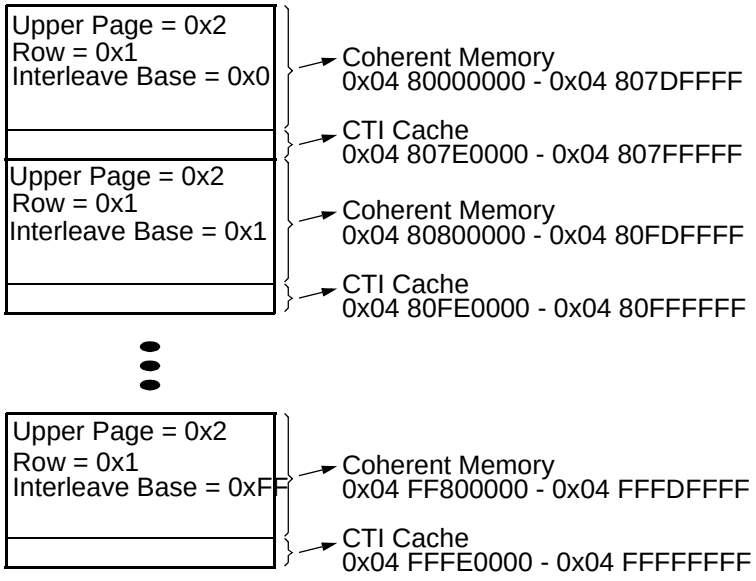
Boards	Buses	Banks
4	8	2
4	4	4
2	8	4

NOTE

The CTI cache must be programmed to reside in a row (or rows) that have all eight buses filled. The interleave does not have to span the eight buses, but there must be memory present on all eight buses for the rows where the CTI cache resides.

Figure 14 illustrates the layout for a CTI cache with the Force Mask set to 0x3FF and the Force Value set to 0x27F. That is, the Upper Page field forced to 0x2, the Row field forced to 0x1, and the most significant 6-bits of the Lower Page field forced to 0x3F. Notice that the configuration for this example assumes memory row one contains the maximum configurable number of banks (256).

Figure 14 Coherent memory space layout with CTI cache



As shown in Figure 14, the CTI cache occupies the upper 128 Kbytes of memory of each memory bank across which it is interleaved. In the example, the interleaving is 256-way so that the aggregate CTI cache size is 32 Mbytes (256-way, 128 Kbytes per bank). 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.

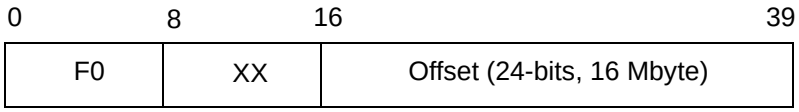
The specific checks made are:

- The appropriate MAC online bit is set in the source PAC System Configuration register.
- The Exist bit is set for the appropriate DIMM 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 the system. 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**



Core logic space is further partitioned for EEPROM, SRAM, and CSR Space. Table 14 shows the address ranges for each of these partitions.

Table 14 **Core logic space partitions**

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

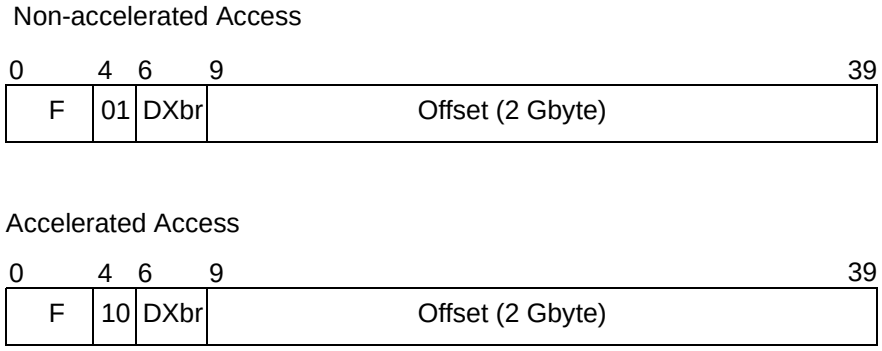
Processor-dependent code (PDC) space is accessed using the core logic bus attached to each processor. A PDC space access is not routed through the Hyperplane crossbar.

Local I/O space

A processor can directly access all I/O space in the system using the I/O controllers.

Figure 16 shows the format for both accelerated and non-accelerated Local I/O Space accesses. Non-accelerated accesses prevent future write access from being issued until the current one has completed (i.e. one write access outstanding at a time). Accelerated accesses do not prohibit write access overlap, resulting in pipelined write accesses.

Figure 16 40-bit local I/O space format



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

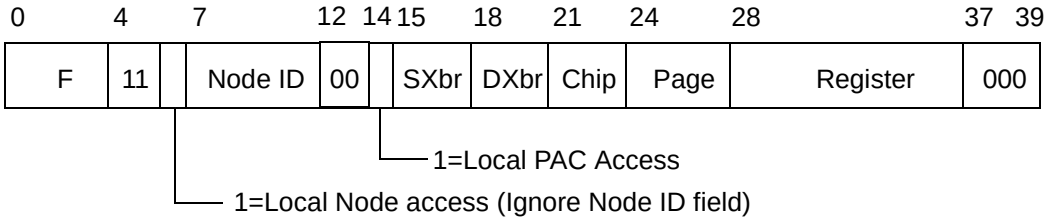
- *DXbr* field (bits 6:9)—Specifies to which of the eight Hyperplane crossbar ports (connected to the PAC chip) the request is to be routed.
- *Offset* field (bits 33:39)—Specifies the offset into I/O space. In this space, all PCI configuration, I/O CSRs and I/O memory space must be allocated. See the section “Host-to-PCI address translation” on page 122 for more information.

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 SAGA 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. See Figure 16.

Non-I/O CSR space

Non-I/O CSRs reside within the processors, PACs, MACs, TACs, and SAGAs.

Figure 17 Non-I/O CSR space format



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

- *Local node access bit* (bit 6)—Indicates that the access is to the local node. When the Local Node Access bit is set, the Node ID field is ignored for purposes of intranode 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 Hyperplane crossbar on the source node. This field is ignored for local node accesses, but is needed to allow systems that do not have full connectivity of all rings to send requests to remote nodes.
- *DXbr* field (bits 18:19)—Specifies which of the eight cross bar ports the request is to be routed on the destination node.
- *Chip* field (bits 21:23)—Routes the packet to the appropriate chip at a crossbar port. Table 15 shows the values of the *Chip* field.
- *Page* field (bits 28:36)—Separates groups of CSRs into similar usage spaces.

Table 15 Chip Field Values

***Chip* Field Value**	**Destination Chip for Packet**
0	PAC
1	I/O ASIC
2	PCXW 0 (Runway Bus 0)
3	PCXW 1 (Runway Bus 1)
4	MAC
5	TAC
6	PCXW 2 (Runway Bus 0)
7	PCXW 3 (Runway Bus 1)

Accelerated CSR access

Bits in the 40-bit physical address are used to determine whether a CSR write access should be issued as Non-Accelerated or Accelerated.

Non-accelerated accesses prevent future write accesses from being issued until the current one has completed (i.e. one write access outstanding at a time). Accelerated accesses do not prohibit write access overlap, resulting in pipelined write accesses. Table 16 shows the 40-bit physical address ranges that control the access method.

Table 16 Accelerated vs. Non-Accelerated Addresses

40-Bit Physical Address	Local Node Access	Accelerated Access
0xFC xxxx xxxx 0xFD xxxx xxxx	No	No
0xFE xxxx xxxx	Yes	Yes
0xFF xxxx xxxx	Yes	No

CSR access

There are three packet routing methods used for accessing CSRs:

- Runway-local
- PAC-local
- Node-local
- Global

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

Runway-local access

Runway-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. Table 17 identifies which addresses are used for runway-local accesses.

Table 17 Runway-local access addresses

Address	Description
0xFFFFFA0xxx	Directed processor CSR address first processor on Runway Bus
0xFFFFFA6xxx	Directed processor CSR address for second processor on Runway Bus
0xFFFFFCxxxx	Broadcast processor CSR address
0xFFFFFDxxxx	Broadcast processor CSR address
0xFFFFFExxxx	Broadcast processor CSR address
0xFFFFFFxxxx	Broadcast processor CSR address

NOTE The Runway On-line bits of the PAC are not checked for runway-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 Hyperplane crossbar. Table 19 shows which fields must be specified for PAC-local addressing.

Table 18 Field values for PAC-local access

Field	Value
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 an PAC. All processor-specific PAC CSRs are identified as having bit 2 of the *Page* field set. When the PAC detects a processor-specific page, it forces bits 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.

Node-local accesses

Node-local accesses are used to access CSRs that reside in the local node. 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 On-line field of the source PAC System Configuration CSR is checked to ensure the destination MAC is on-line. A high-priority machine check trap will result if the destination MAC is not on-line.

If the access is to an PAC, SAGA, 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. If the destination MAC is not online, an HPMC results.

Table 19 Field specifications for system access

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

Remote access

Remote accesses are accesses to CSRs that reside in any node of multinode systems. All local node CSRs and many remote node 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 node MAC specified by the SXbr field. The local node MAC checks if the Node ID is for the local node.

If the request is for the local node, the local node MAC uses the DXbr and Chip fields to determine the local node 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 node ID, the request is sent to the MAC of the remote node using a CTI ring. The remote node MAC routes the request using the DXbr and Chip fields to the appropriate remote node destination. The TAC of the remote node 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 node), the SXbr and DXbr fields must have the same value. Table 20 specifies the fields for remote addressing:

Table 20 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 to send a request to a CSR in a controller that is not online. Online bits are implemented for processors, PACs, MACs, and SAGAs. Memory uses existence bits.

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

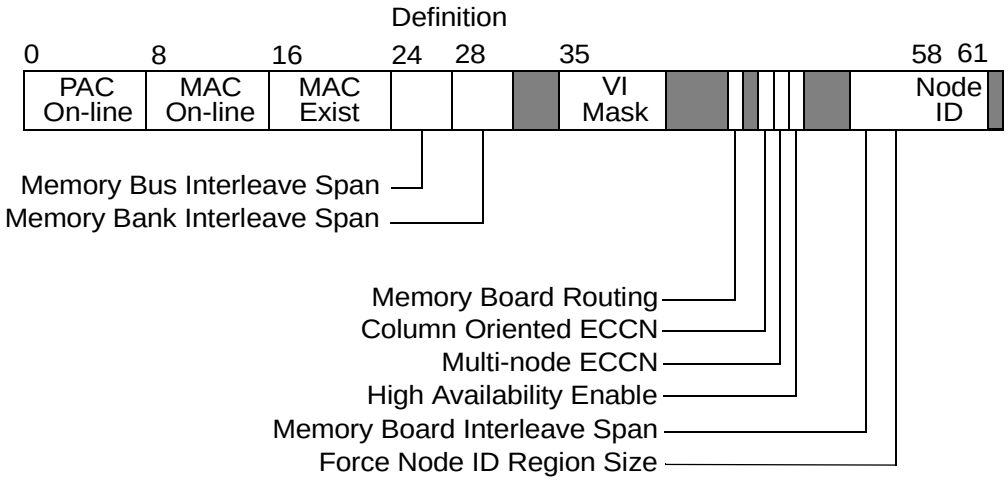
- Requests with a response to a CSR covered by an online bit result in an error response being returned to the processor. The processor issues a high-priority machine check interrupt.
- Requests without a response to a CSR covered by an online bit result in a time-out when the next synchronization operation 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 18 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 18 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.
- *Memory Bus Interleave Span* field (bits 24:27)—Indicates whether bus interleaving should span four or eight buses. An independent enable bit is provided for each of the four rows. Table 21 defines the association of enable bit versus memory row. A value of zero in the bit position specifies four bus interleave and a value one specifies eight.

Table 21 Memory Bus Interleave Span bit positions

Bit position	Row to which enable applies
24	Row 0
25	Row 1
26	Row 2
27	Row 3

- *Memory Bank Interleave Span* field (bits 28:31)—Indicates whether bank interleaving should span two or four banks. An independent enable bit is provided per memory DIMM for the Upper and Lower buses. Table 22 defines the association of enable bit versus memory DIMM. A value of zero in the bit position specifies two bank interleave and a value one specifies four.

Table 22 Memory Bank Interleave Span bit positions

Bit position	DIMM to which enable applies
28	Rows 0 and 1, Lower Buses (0-3)
29	Rows 0 and 1, Upper Buses (4-7)
30	Rows 2 and 3, Lower Buses (0-3)
31	Rows 2 and 3, Upper Buses (4-7)

- *VI mask* field (bits 35:41)—Specifies the Virtual Index bits generated by the processor that are masked (forced to zero).

Table 23 VI Mask field valuess

Field value	PA-8500 Data Cache Size
0x00	32 K Byte
0x01	64 K Byte
0x03	128 K Byte
0x07	256 K Byte

Field value	PA-8500 Data Cache Size
0x0F	512 K Byte
0x1F	1 Mega Byte
0x3F	2 Mega Byte
0x7F	4 Mega Byte

- *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.
- *Column oriented ECCN, Multinode ECCN, Enable high availability checks* bits (bits 48, 49, 50)—Control the high availability modes. These bits are currently not used in V2500 servers.
- *Memory board interleave* field (bits 54:55)—Specifies the number of even/odd memory board pairs which normal interleaving should span. Table 24 shows the possible values for the field.

Table 24 Memory board interleave Span field values

Field value	Interleave span
0	One Memory Board Pair (two MACS)
1	Two Memory Board Pairs (four MACS)
2	Reserved
3	Four Memory Board Pairs (eight MACS)

- *Force Node ID Region Size* field (bit 56:57)—Indicates whether a region of memory that can be accessed by specifying a node zero address is enabled. Table 25 shows the possible values for the field

Table 25 Force Node Id Region Size field values

Field value	Region Size
0	Disabled
1	32 Mbytes
2	64 Mbytes
3	128 Mbytes

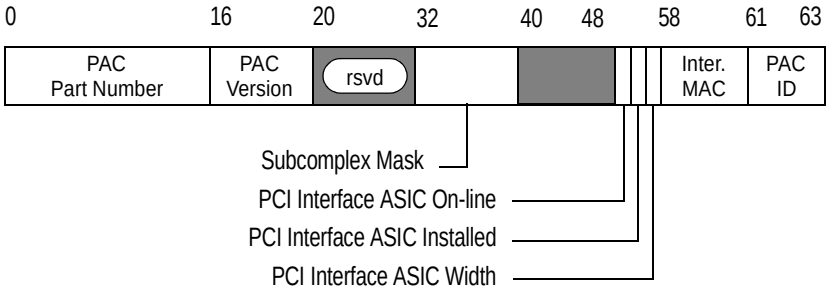
- *Node ID* field (bits 58:61)—Specifies the node identification number for the node.

PAC Configuration register

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

Figure 19 shows the format of the PAC Configuration register. All fields of the register are read by a read access.

Figure 19 PAC Configuration register definition



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

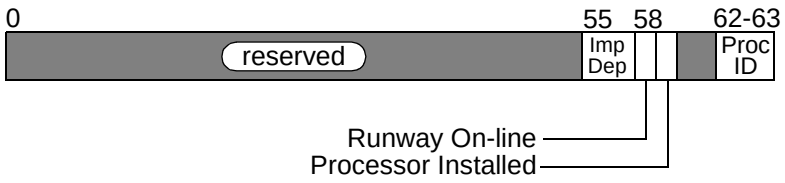
- *PAC part number* field (bits 0:15)—Specifies the part number for the PAC. A write is ignored and a read returns the hard-wired value.

- *PAC version* field (bits 16:19)—Specifies the version for the PAC. A write is ignored and a read returns the hard-wired value.
- *Subcomplex Mask* field (bits 32:39)—Specifies which PACs within the node belong to the same subcomplex as the PAC associated with the configuration register. Each bit in the mask when set specifies a PAC which should receive a broadcasted transaction. A PAC will propagate a received broadcasted transaction to each of its processors.
- *PCI Interface ASIC online* bit (bit 55)—Set by software to allow CSR accesses to the SAGA. The bit is cleared by reset.
- *PCI Interface ASIC installed* bit (bit 56)—Specifies whether an SAGA ASIC is connected to the PAC. A value of one indicates an SAGA is installed. This bit is read only.
- *PCI Interface ASIC width* bit (bit 57)—Specifies whether a 32-bit or 16-bit interface exists between the SAGA 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 system 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

There are four CSRs of this format on each PAC that are used to specify information which may be different for each processor attached to the PAC.

Figure 20 PAC Processor Configuration register definition



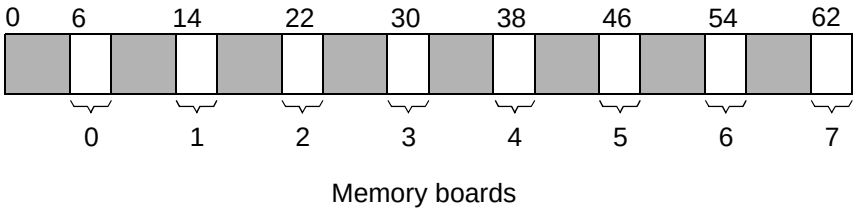
The bits and fields in the PAC Processor Configuration register are defined as follows:

- *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.
- *Runway On-line* bit (bit 58)—Indicates that the runway bus to which the processor attached is assessable. The PAC checks that this bit is set prior to issuing any CSR operation to the processor or allowing its requests to be issued to the rest of the system. Reset initializes the value to that of the *Processor Installed* bit. Notice that this bit is common for both processors attached to a runway bus. This implies that writing to either of the processor's Processor Configuration register effects both registers.
- *Processor Installed* bit (bit 59)—Indicates that the processor is installed. The value of this bit comes directly from a pin on the PAC. Writing to this bit is ignored. A read access returns the value of the input pin.
- *Processor Identification* field (bits 62:63)—Specifies the identification number for the physical processor. A writing to this field is ignored.

PAC Memory Board Configuration register

Each PAC has a Memory Board Configuration register that specifies the memory block to memory board mapping. Figure 21 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 21 PAC Memory Board Configuration register definition



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

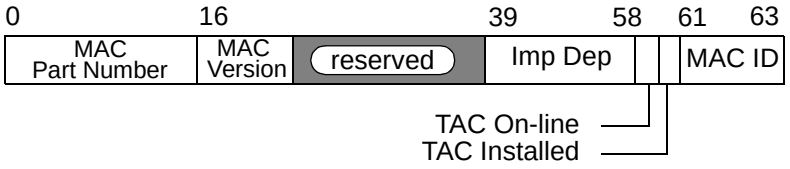
The memory board fields specify the most significant two bits of the physical memory board. The least significant bit of the memory block index is the least significant bit of the physical memory board. This forces even memory blocks to be mapped to even memory boards and odd memory blocks to odd memory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

MAC Configuration register

Each MAC has a Configuration register that contains specific information about the MAC on which it resides. Each MAC can be configured differently.

Figure 22 shows the format of the register. All fields of the register are read by a read access.

Figure 22 MAC Configuration register definition



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

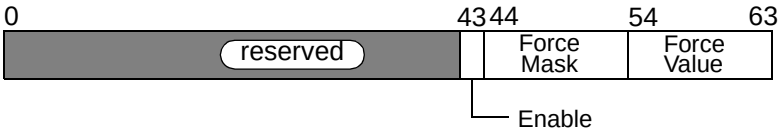
- *MAC part number* field (bits 0:15)—Specifies the part number for the MAC chip. A write is ignored and a read returns the hard wired value.
- *MAC version* field (bits 16:19)—Specifies the version for the MAC chip. A write is ignored and a read returns the hard wired value.
- *Implementation dependent* field (bits 39:58)—Used by low-level implementation dependent software. The value in this field should not be modified during normal operation.
- *TAC On-line* bit (bits 43:54)—Indicates that the TAC associated with this MAC is on-line. Reset clears this bit.
- *TAC Installed* bit (bit 60)—Specifies whether a TAC is connected to the MAC. A value of one indicates 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 Region register

There are three registers of this format on each MAC that define regions of memory. The three regions of memory are Unprotected Memory, Protected CTI Cache, and Unprotected CTI Cache. Protected memory is all physical memory not covered by these three regions.

Systems without high availability only use the protected CTI cache region register. When high availability mode is disabled the entire system is treated as a single cache coherent node. As a single cache coherent node, all nodes can access protected memory of any node. In this mode unprotected memory is not accessible since the Unprotected Access bit of the physical address is not available.

Figure 23 Memory region register definition



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

- *Enable* bit—Enables access to the memory region which the CSR register specifies. Reset clears the bit.
- *Force Mask* and *Force Value* fields—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 ten bits of the physical address are the two-bit *Upper Page* field, the two-bit Row field, and the six most significant bits of the *Lower Page* field. The most significant two bits of the 10-bit *Force* fields apply to the *Upper Page* field, and the least significant six bits apply to the *Lower Page* field.

Unprotected Memory register

The fields of the Unprotected Memory Region register check that requests from other nodes have access permission to the address of the request. An access violation exists if a request that arrived on a CTI link attempts to access memory outside the region defined by the Unprotected Memory Region register.

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 node memory.

When the physical address accesses a remote node and the *Unprotected Access* bit of the physical address is zero, the protected 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 node memory.

When the physical address accesses a remote node and the *Unprotected Access* bit of the physical address is one, the unprotected 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 attempting to access the unprotected CTI cache memory region.

Memory region access checking summary

Table 26 summarizes the memory region access violation checks performed by the MAC. The memory types are defined as follows:

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

- Protected CTI Cache—Memory covered by the Protected 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.

The access types are defined as follows:

- Protected Access—All memory accesses with high availability functionality disabled are protected accesses (as if the entire system was one ECCN). Also, all memory accesses with the Unprotected Access bit of the physical address set to zero when high availability functionality is enabled.
- Unprotected Access—All memory accesses with the Unprotected Access bit of the physical address set to one when high availability functionality is enabled.
- Diagnostic Access—All memory accesses initiated by the MAC diagnostic access operations.
- CTI Cache Flush Entry—Processor issued CTI cache flush entry operations.
- Non-local ECCN Protected Access—All accesses made from a node that is not in the ECCN of the destination node which have the Unprotected Access bit of the physical address set to zero when high availability functionality is enabled.
- Non-local ECCN Unprotected Access—All accesses made from a node that is not in the ECCN of the destination node which have the Unprotected Access bit of the physical address set to one when high availability functionality is enabled.
- Non-local ECCN Unprotected Access—All accesses made from a node that is not in the ECCN of the destination node which have the Unprotected Access bit of the physical address set to one when high availability functionality is enabled.

  The first four access types (Protected Access, Unprotected Access, Diagnostic Access, and CTI Cache Flush Entry) originate from a processor on the same node as the MAC making the access check. The last three memory types originate from a remote node and are transferred to the MAC from a TAC.

Table 26 Memory Region Access Checking Summary

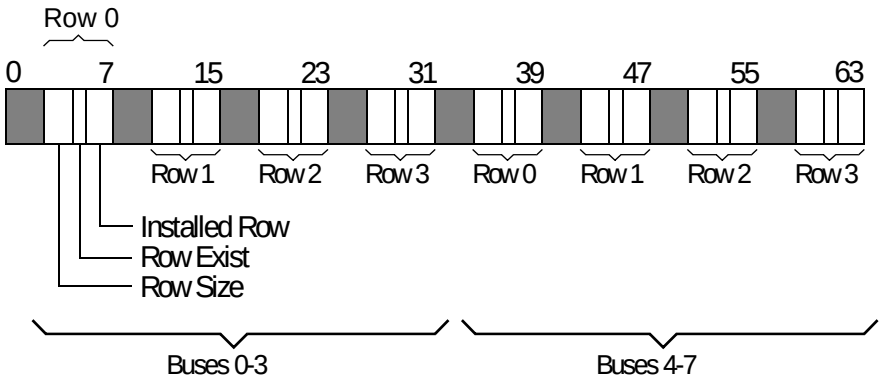
Access Type	Memory Type				
	Main Memory		CTI Cache		Non-existent Memory
	Protected	Unprotected	Protected	Unprotected	
Protected Access	OK	Error	Error	Error	Error
Unprotected Access	Error	OK	Error	Error	Error
Diagnostic Access	OK	OK	OK	OK	Error
CTI Cache Flush Entry	Error	Error	OK	OK	Error
CTI Protected Access with High Availability Disabled	OK	Error	OK	Error	Error
CTI Protected Access with High Availability Enabled	Error	Error	Error	Error	Error
CTI Unprotected Access	Error	OK	Error	OK	Error

Memory Region Access Checking Summary

Associated with each MAC are eight memory buses, each having up to four rows of SDRAMs. A table is used to map the row specified in the physical address to a physically installed row of SDRAMs. The upper and lower buses of memory are configured independently by a MAC controller.

Each MAC has a register to specify the memory row mapping. Figure 24shows the format of the register. All fields are written by a write access, and read by a read access. Reset has no effect. Writing to reserved bits is ignored and reading to reserved bits return the value zero.

Figure 24 MAC Memory Row Configuration register definition



The two-bit *Row* field of a physical address and the target are used to index one of the eight sets of *Row* fields of a MAC Memory Row Configuration CSR. The target can be either a lower bus (buses 0-3) or an upper bus (4-7).

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

- *Row Size* field (2 bits each)—Indicates the size of the SDRAM row.

Table 27 Row Size field values

Field value	Row size
0	16 Mbyte (2 Mbit x 8 SDRAMs)
1	64 Mbyte (8 Mbit x 8 SDRAMs)
2,3	128 Mbyte (8 Mbit x 4 or 16 Mbit x 8 SDRAMs)

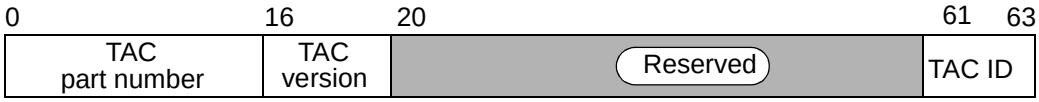
- *Row Exist* field (1 bit each)—Indicates that physical memory exists at the indexed row of memory. The field checks for the existence of physical memory.

- *Installed Row* field (2 bits each)—Maps the row specified by the access address to the physical memory row. The mechanism only allows rows from a lower bus (0-3) to be mapped to other lower bus rows, and rows from upper buses (4-7) to other upper bus rows.

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 25. All fields of the register are read by a read access.

Figure 25 TAC Configuration register definition



The 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.
- *TAC identification* field (bits 61:63)—Specifies the identification number for the physical TAC. The value is written by software.

3 Cache Management

This chapter presents the concepts and mechanisms associated with cache management on an V2500 System.

Cache concepts

V2500 supports caching to minimize the effect of access latency to local and remote node memory. The following sections describe the PA-8500 caches, the CTI cache, Cacheability, and Address Aliasing.

PA-8500 caches

Each processor in an V2500 system has a data cache and an instruction cache. The operation of copying information from memory into a cache is referred to as move-in. Whenever a memory item is moved into the cache, that processor may modify it there. If another processor references the same item while the copy is still in the first processor cache, the original referenced memory item and the copy must be identical to maintain coherency. The V2500 cache management mechanism maintains this coherency.

The PA-8500 processor supports speculative execution of code as one of its performance acceleration features. Speculative execution is enabled whenever virtual address translation is being performed. Speculative execution allows data cache move-in of any coherent memory line which exists in a page of memory that has its virtual-to-physical address translation in the processor's TLB. Additional restrictions are that virtual-to-physical translation must be enabled (PSW D-bit set) and that the TLB entry must have the U-bit (uncacheable bit) cleared to allow a data cache move-in (direct or speculative load).

The PA-8500's instruction cache and data cache each have a 32 byte line size.

CTI cache

Each node maintains a cache of memory references received from other nodes. This cache is referred to as the CTI cache. Any data that has been moved into a processor cache and is still resident in the processor cache is also "encached" in that node's CTI cache. Consequently, the CTI cache directory information can be used to locate any global data currently encached by the node.

The CTI cache is physically indexed and tagged with the global physical address (40-bit physical address). Since the cache is physically indexed, there are no aliasing issues.

The V2500 system ensures cache coherence between multiple nodes (two or more nodes that map the same global address will get a consistent view). This is done by maintaining a linked sharing list that contains a list of all the nodes sharing each cache line, or the node that exclusively owns the cache line. Within every node a record is kept of which processors have encached each line in the CTI cache so that incoming coherency requests (read shared or private for a remote node's use) can be forwarded to the appropriate processors.

The CTI cache line size is 32 bytes.

Cacheability

Memory references eligible for cache move-in are cacheable references. Some memory references may not legally cause a move-in; these references are referred to as uncacheable. There are two types of memory regions which are non-cacheable, these are:

- Coherent memory pages which are virtually mapped with a TLB entry with the U-bit set
- Cache coherence between processors

The cache-coherent part of a multiprocessor system behaves as if there were a single data cache and a single instruction cache (logically). If there are multiple data caches, they must cross interrogate for current data and must broadcast purges and flushes (except for FDCE and FICE). Purges and flushes do not cause TLB faults on other processors.

In the cache-coherent part of a multiprocessor system, all data references must be satisfied by data obtained using cache coherence checks. This data must have remained coherent since it was moved in. Implementations with write buffers must also check buffer contents on cache coherence checks in order to ensure proper ordering of storage accesses.

Address aliasing

Normally, a virtual address does not translate to two different physical addresses. The operating system ensures that this does not happen (through correct management of TLBs and PDIRs, see the PA-RISC 2.0 Architecture Reference Manual and the Exemplar Programming Guide).

Several virtual addresses, or a virtual address and a physical address, can in special situations map to the same physical memory. Such mappings present a consistent view of memory only when the current read operation produces the same data that was stored last.

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 may or may not 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. The actual indexing function is processor implementation-specific.

When the conditions required for keeping the caches of a system coherent are not met, hardware alone can not keep an application's view of memory consistent. The O/S software may use cache flushing to keep the view of memory consistent.

Cache operations

The V2500 system supports flush and prefetch operations on the processor's caches and the node's CTI cache. This section describes these operations.

PA-8500 cache operations

The PA-8500 has both instruction and data caches. These caches may be flushed by specifying either a specific memory line address, or by specifying the specific cache entry to flush. The difference between the two methods is whether the cache tag is checked for a match. When a memory line's address is specified, the tag is compared. Only if the tag matches does the cache's entry get flushed. When a cache entry is specified that entry is flushed, independent of the cache line's tag value.

Each processor cache operation is issued by executing a single processor instruction. These instructions are PDC, FDC, FIC, FDCE, or FICE instructions. The instructions are not privileged allowing use by user applications.

The instructions which flush based on a memory line address are PDC, FDC and FIC. These instructions are broadcast to other processors within a node that may be sharing the same cache lines. These instructions, therefore, have global effects within a node.

The PA-8500 implemented the date cache purge instruction (PDC) as a data cache flush operation. As a result, PDC and FDC have identical functionality. Both operations flush cache lines from all processors on the local node and from the node's CTI cache. FDC and PDC instructions write data from dirty cache lines back to memory.

The instructions which flush a cache entry independent of the tag value are FDCE (Flush Data Cache Entry) and FICE (Flush Instruction Cache Entry). These instructions flush an entry from the executing processor's caches only. If the cache line in the data cache is dirty, it will be written back to local memory or the CTI cache. However, the CTI cache is not flushed.

Cache flush instructions must be followed by a sync instruction to ensure that all flushes have made it to memory.

CTI cache operations

V2500 has four operators for CTI cache management. These operators are:

- CTI cache global flush
- CTI cache Flush Entry
- CTI cache prefetch for read
- CTI cache prefetch for write

These operators are used to manipulate the contents of the CTI cache without directly effecting the contents of the processor's caches. Indirectly, a prefetch operation may force an entry out of the CTI cache, forcing the memory line out of a processor cache as well.

CTI cache global flush

This operator is used to flush the memory line for a specific address out of all CTI caches in the entire system. Once completed, the operation is defined to have the memory line be at memory and not encached in any processor's data cache or any node's CTI cache.

CTI cache flush entry

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

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 prefetch for read

This operator is used to accelerate a memory line in to the CTI cache of the local node. The CTI cache will contain the memory line for read access (not write accessible) after the operation is complete.

CTI cache prefetch for write

This operator is used to accelerate a memory line to the CTI cache of the local node. The CTI cache will contain the memory line for read and write access after the operation is complete.

Cache operation summary

The following table gives the result of executing the various flush, purge and prefetch primitives for differing initial conditions.

Table 28 Cache operation summary

Instruction	Current state	Mem type	Resulting processor cache state	Resulting CTI cache state	Other nodes state
FDCE, FDC, FIC, FICE, PDC	clean, in data or instruction cache	local	no longer encached	not applicable	not applicable
FDC, FDCE, PDC	dirty, in data cache	local	data written to memory, no longer encached	not applicable	not applicable
FDC, FIC, PDC	not encached in processor	local	no longer encached on any processor in node	not applicable	not applicable
FDCE, FICE, FIC	clean, in processor data or instruction cache	global	data no longer encached	unchanged, data may still be encached	may be encached
FDCE	dirty, in processor data cache	global	data written to CTI cache; no longer encached	dirty; encached	not encached (current node is exclusive owner)
FDC, PDC	clean, in processor data cache	global	data no longer encached	data no longer encached	may be encached

Instruction	Current state	Mem type	Resulting processor cache state	Resulting CTI cache state	Other nodes state
FDC, PDC	dirty, in processor data cache	global	data written to memory; no longer encached	no longer encached	memory consistent; no longer encached
FDC, PDC	data not in processor cache, but present in CTI cache	global	unchanged (not encached)	written back if dirty; no longer encached	may be encached if clean
FDCE, FICE	data not in processor cache, but present in CTI cache	global	unchanged (not encached)	unchanged	unchanged
CTI Cache Flush Global	any state	global	written to memory if dirty; no longer encached	no longer encached, written to memory if dirty	no longer encached, written to memory if dirty
CTI Cache Flush Entry	any state	global	written to memory if dirty; no longer encached	no longer encached, written to memory if dirty	unchanged
CTI Read Prefetch	not encached in node	global	not encached	encached, shared	may be encached
CTI Write Prefetch	not encached in node	global	not encached	encached, exclusive	not encached
Instruction	Current state	Mem type	Resulting processor cache state	Resulting CTI cache state	Other nodes state

Cache operation interfaces

This section presents the hardware interfaces available to software to issue the cache management operations.

PA-8500 cache interfaces

All processor cache operations are issued with a single PA-8500 processor instruction. These operations include Flush Data Cache (FDC or PDC), Flush Data Cache Entry (FDCE), Flush Instruction Cache (FIC), and Flush Instruction Cache Entry (FICE).

CTI cache interfaces

The PA-RISC architecture (1.0 and 2.0) does not support the concept of a CTI Cache. Thus, there are no PA-RISC architected instructions for issuing the CTI cache operations. The V2500 system provides interfaces to these operations using non-PA-RISC-architected mechanisms. User applications are not allowed to directly use these mechanisms to ensure compatibility with previous and future generation platforms which may choose to implement the interface mechanisms differently. Access to these mechanisms is provided through the use of the Architectural Interface Library (AIL). The library routine interfaces are consistent on all Exemplar platforms, allowing applications to be portable.

CTI cache AIL routines

AIL interface routines are provided to implement a global cache flush (cache_flush) and CTI cache prefetch (read_prefetch and write_prefetch). The definition of these interface routines are as follows:

```
void cache_flush(virt_addr);
void *virt_addr;

void read_prefetch(virt_addr);
void *virt_addr

void write_prefetch(virt_addr);
void *virt_addr;
```

`Cache_flush` is a CTI-wide cache line flush; after its execution the referenced cache line is not cached anywhere in the system until a subsequent reference (direct, prefetch or speculative). `Read_prefetch` accelerates a cache line into the CTI cache for read access by the local node; it can be shared with other nodes. `Write_prefetch` accelerates a cache line into the CTI cache for local node read or write use; it is exclusively owned by the local node.

There are two interface mechanisms which exist to issue the CTI cache operations. The first mechanism issues the operations by using a single PA-8500 instruction. The second method issues the operations through the use of CSR read and write operations. These two mechanisms are explained in the following sections.

Instruction method of issuing cache management operations

The PA-8500 processor implements an instruction which allows an operation to be passed directly to the PAC. Fields of the instruction allow specification of which operation to perform and whether read and/or write protection checking should be performed. The instructions are enabled using a field in one of the PA-8500's Remote Diagnostic Registers. When enabled, the instructions are allowed to executed at any privilege level. The AIL routines may execute these instructions to issue the CTI cache operations. A user application may also execute these instructions, resulting in functional, but non-portable code.

There are three CTI cache instructions. The formats for these instructions are shown on the following pages.

CTI cache flush global instruction

The CTI Cache global instruction is decribed below.

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 1 or 2. Table 29 defines the assembly language h-field values.

The PSW D-bit (Data address translation enable) determines whether a virtual or absolute address is used.

Table 29 CTI cache flush global hint field values

Description	h
No read or write access protection check	1
Read access protection check	2
Read and write access protection check	3

Operation:

```
If (RDR enabled) {
space ¨ space_select(s,GR[b],SHORT);
offset ¨ GR[b];
Global_flush(space, offset);
} else
Illegal_instruction_trap
```

Exceptions: Non-access 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

The CTI cache prefetch read instruction is described below.

CTI Cache Prefetch Read — NCPR

Format:
NCPW (s,b)

01	b	04	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 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 30 defines the assembly language h-field values.

Table 30 CTI cache prefetch read hint field values

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: Non-access data TLB miss fault
llegal_instruction_trap
Data memory access rights trap

Restrictions: The base register must be 32-byte aligned.

CTI cache prefetch write

The CTI cache prefetch write instruction is described below.

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 31 defines the assembly language h-field values.

Table 31 CTI cache prefetch write hint field values

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:	`Operation:If (RDR enabled) {`
	`space ¨ space_select(s,GR[b],SHORT);`
	`offset ¨ GR[b];`
	`CTIcache_prefetch_read(space, offset);`
	`} else`
	`Illegal_instruction_trap`

Exceptions:	Non-access data TLB miss fault llegal_instruction_trap Data memory access rights trap
Restrictions:	The base register must be 32-byte aligned.

CSR method of issuing cache management operations

An alternate method of issuing the CTI cache operations exists that uses a sequence of instructions that use PAC CSR reads and writes. This method is a lower bandwidth approach, but does not require any non-architected support from the PA-8500 processor.

Operation CSRs

The three operations (CTI cache flush global, CTI cache prefetch read, and CTI cache prefetch write) are issued by writing the 40-bit physical address where the operation is to be performed to a PAC CSR. There is a different CSR for each of the three operations. The names of the CSRs for Processor 0 are:

- Processor 0 CTI Cache Flush Global
- Processor 0 CTI Cache Prefetch Read
- Processor 0 CTI Cache Prefetch Write

Three CSRs with similar names exist for Processors 1, 2 and 3.

Instruction sequence

The sequence of instructions which will issue a CTI cache global flush is as follows:

```
loop
LDI1,%t1
STD%t1,(CSR_OP_CNTX); Arm CSR Operations
LPAncfg_addr,%t2
STD%t2,(CSR_NCFG); Issue NC Global Flush
LDD(CSR_OP_CNTX),%t4
BB,*>=%t4,62,loop; Check if triggered
```

The steps of the sequence are:

1. Arm the operation by writing to the PAC Operation Context register's Armed bit.
2. Perform virtual to physical address translation
3. Issue the CTI Cache operation with the physical address provided as the data of the write.

4. Check the PAC Operation Context register Triggered bit to make sure the operation was issued. If the Triggered bit is not set then the operation must be restarted.

The sequence uses the Armed and Triggered mechanism of the PAC to check whether a processor interrupt (internal or external) occurred between the load physical address (LPA) instruction and the store to the operation CSR. The write to the operation CSR will issue the operation to the remainder of the system provided that the Armed bit is set. When the Armed bit is set, writing to the operation CSR causes the Armed bit to be cleared and the Triggered bit to be set.

If an interrupt does occur during the sequence, the operating system's software interrupt handler should clear the Armed bit. With the Armed bit cleared, the write to the operation CSR will not issue the operation to the remainder of the system and the branch at the bottom of the instruction sequence will cause the entire sequence to be retried. This is required to ensure that a valid virtual-to-physical address translation is passed to the operation CSR.

Cache management CSRs

CSRs may be used to issue cache management operations.

The CSR based cache management operations use only loads and stores to CSR space to issue the operations. The operations are issued by performing the following steps:

1. Arming the operation by setting the Armed bit in the Operation Context CSR.
2. Writing to the appropriate Cache Management Operation CSR address to perform the operation with the data being the 40-bit physical address of the target memory line.
3. Reading the Triggered bit in the Operation Context CSR to verify that the operation succeeded.

Cache management operations

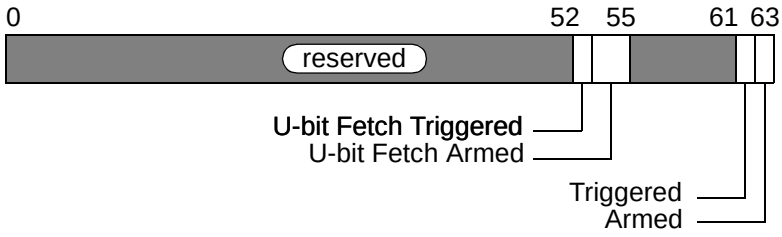
This section describes the CSR registers which are required to issue cache management operations. The following CSRs reside in the PAC. The CSRs include:

- Operation Context register
- Operation Address register
- Cache Flush Global address
- Cache Prefetch for Read address
- Cache Prefetch for Write address

PAC Operation Context register

Each PAC has four Operation Context CSRs, one for each processor. 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. shows the format of the PAC CSR Operation Context register.

Figure 26 PAC Operation Context register



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

- *U-bit Fetch Triggered* field (bit 53)—Indicates that a read short to a coherent memory line was observed from the processor associated with this CSR while the U-bit Fetch Armed field was non-zero. The bit is cleared by reset.
- *U-bit Fetch Armed* field (bits 54:55)—Enables translation of a read short to coherent memory space into a non-coherent semaphore operation. The first read short, to coherent or non-coherent space, will cause the U-bit Fetch Armed field to be cleared (disabled). Only if the read short is to coherent memory space is the operation translated to a non-coherent semaphore and the U-bit Fetch Triggered bit set. Note that TLB purges have no effect on this field. The field is cleared by reset. Table 32shows the values used to enable non-coherent semaphore operations.

Table 32 U-bit Fetch Armed field values

Field Value	Non-coherent semaphore operation
0	Disabled
1	Fetch and Increment
2	Fetch and Decrement
3	Fetch and Clear

- *Triggered* field (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.

- *Armed* field (bit 63)—Set by software to arm the functionality of specific PAC processor CSRs. The PAC CSRs armed by this bit are:
  - Fetch and Increment address
  - Fetch and Decrement address
  - Fetch and Clear address
  - Non-coherent Read address
  - Non-coherent Write address
  - Coherent Increment address
  - CTI Cache Global Flush address
  - CTI Cache Prefetch for Read address
  - CTI Cache Prefetch for Write address

Each of these CSRs and the effect on them by the Armed bit is discussed in various chapters of this document. The Armed bit is set by software and is cleared by either hardware or software.

Table 33 shows the Armed and Triggered bit transitions which hardware controls when software accesses one of the operation address CSRs.

Table 33 PAC Operation Context register transitions when Operation address accessed

State Transition when Operation 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 34 shows the Armed and Triggered bit transitions which hardware controls when a TLB invalidate transaction is detected.

Table 34 PAC Operation Context register transitions when TLB invalidate issued

State Transition when TLB Invalidate detected			
Present Value		Next Value	
Triggered	Armed	Triggered	Armed
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Context State Save/Restore

The PAC Operation Context register has an Armed and Triggered mechanism to protect an operation being issued using the multistep CSR approach from being corrupted by a processor interrupt (external, DTLB miss, etc.) that disrupts the sequence. The mechanism works by having all interrupt handlers examine the PAC Operation Context register and take appropriate action. The following situations will 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 being allowed to start. The Triggered bit is cleared to prohibit software from believing the operation was issued successfully.

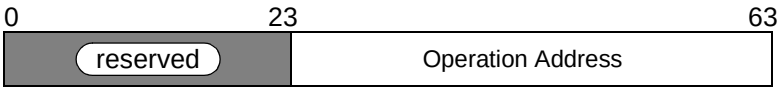
If the thread which is interrupted is to be context switched, then the modified Armed and Triggered bits must be saved and later restored when the thread is to resume. Note that no other CSR context must be saved/restored for the CSR method of issuing operations to function properly.

PAC Operation Address registers

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

The format of the PAC Operation Address register is shown in Figure 27. The field of the register is read by a read access.

Figure 27 PAC Operation Address register definition



The PAC Operation Address register (bits 24:63) is written to by software with a write to the address of this CSR or by writing to any of the following CSR addresses:

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

The Operation Address field specifies the address for each of the following CSR operations:

- Fetch and Increment address
- Fetch and Decrement address
- Fetch and Clear address
- Non-coherent Read address
- Non-coherent Write address
- 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 32 to 128 Mbytes (depending in the Force Node ID value in the PAC System Configuration register) of physical memory on node zero are forced to reference the local node.

To perform this mapping, the hardware compares the Upper Page, Row, and most significant six bits of the Interleave Base 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 node.

Cache Management Operation addresses

Each PAC has three Cache Management addresses per processor. The three operation addresses are for: CTI Cache Flush Global, CTI Cache Prefetch for Read, and CTI Cache Prefetch for write. Writes to these addresses trigger the respective cache management operation.

If the Armed bit in the PAC Operation Context register is set, a write to one of the addresses results in the cache management operation being sent to memory. The address of the target memory line is specified as the data of the double word store instruction which issues the operation. As part of the cache management 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.

4 Synchronization

There are two types of processor synchronization which is supported by the V2500 architecture. The first type is used to enforce exclusive access of a shared data structure. A data structure that is shared by multiple processors must have a mechanism that allows only a single processor to atomically modify its data. V2500 servers supports coherent semaphore operations to achieve exclusive access.

The second type of processor synchronization is called barrier synchronization. This type is used to inhibit any processor from beginning the next section of a program until all processors have completed the previous section of the program.

The following sections present the V2500-supported coherent semaphore operators (used for mutual access of data structures), non-coherent semaphore operators (used primarily for barrier synchronization operations), barrier synchronization, and PA-8500 TLB entry U-bit functionality.

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.

NOTE The LDCW and LDCD instructions ignore the TLB-U bit for all modes of operation. This implies that the user may accelerate a non-coherent memory line into the data cache of a processor by incorrectly using the LDCW or LDCD instructions.

The function performed by the LDCW and LDCD instructions are effected by Remote Diagnostic register 30, DIAG_KITTYHAWK bit. When this bit is set to one, the LDCW and LDCD instructions ignore the cache hint field and always accelerate the memory line to the processor's data cache. Otherwise when the bit has the value zero, the hint bits are used to determine whether the instruction accelerates the memory line into the processor's data cache to be cleared, or whether the MAC is expected to perform the clear function.

Noncoherent semaphore operators

The V2500 server supports 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 section “PA-8500 TLB Entry U-bit” on page 103). They do not accelerate the semaphore into the data cache.

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

- 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 instruction reads 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.

NOTE The Exemplar Architecture defines all semaphore variables to be 16-byte aligned and that Semaphore operations to nonaligned variables produce undefined results. On non-coherent semaphore operations can be issued to any memory word for word sized operations and any double word for double-word-sized operations.

AIL routines for non-coherent semaphores

All non-coherent semaphore operations should be accessed using the Architectural Interface Library (AIL) to maintain portability across the V-Class family of products. The AIL library implements the non-coherent

semaphore operations with a sequence of CSR operations. V2500 servers support two methods to issue non-coherent semaphore operations. The methods are described in the following sections.

First method of issuing non-coherent semaphore operations

All non-coherent semaphore operations are issued using accesses to PAC-local CSRs. The mechanism requires a sequence of instructions to issue an individual operation.

The sequence of instructions for a `fetch_and_inc32` is as follows:

```
PROBEWfetch_addr;Check protection
loop
LDI1,%t1
STD%t1,(CSR_OP_CNTX);Arm Operation
LPAfetch_addr,%t2
STD%t2,(CSR_OP_ADDR);Fetch address
LDW(CSR_FETCH_INC),%t3;Issue fetch
LDD(CSR_OP_CNTX),%t4
BB,*>=%t4,62,loop;Check trigger
```

Implementing noncoherent semaphore operations requires the following sequence of instructions using PAC CSRs:

1. Check write access privilege for the semaphore address.
2. Arm the operation by writing to the PAC Operation Context register Armed bit.
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.

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.

Second method of issuing non-coherent semaphore operations

A second method was added to the PAC in order to avoid the possibility of a user causing an HPMC to be sent to a processor.

All non-coherent semaphore operations are issued using accesses to PAC local CSRs. The mechanism requires a sequence of instructions to issue an individual operation.

The sequence of instructions for a fetch_and_inc32 is as follows:

```
loop
     LDI  0x100,%t1
     STD  %t1,(CSR_OP_CNTX);Arm Operation
     LDW  fetch_addr,%t2;Issue operation
     LDD  (CSR_OP_CNTX),%t4
     BB,*>=%t4,55,error;Check armed
     BB,*>=%t4,53,loop;Check trigger
```

The steps of the sequence are:

1. Arm the operation by writing to the PAC Operation Context register *U-bit Fetch Armed* field.
2. Read back the PAC Operation Context register (or any other PAC register). This step is required to prevent a race condition between the arming of the operation and the fetch operation.
3. Issue the `fetch` operation. The value read is the return value for the semaphore operation. The size of the load instruction, word, or double word indicates the size of the semaphore operation.
4. Check the PAC Operation Context register *U-bit Fetch Armed* field to determine if the operation is still armed. If it is, the TLB-U bit was not set for the page of memory that was the target of the semaphore.
5. Check the PAC Operation Context register *U-bit Fetch Triggered* bit to make sure the operation was issued. If the *U-bit Fetch Triggered* bit is not set, then the operation must be restarted.

Barrier synchronization

Not all threads in a multithread process complete at the same time. Multi-threaded applications usually require synchronization of threads when all threads must complete a section of computation prior to any thread starting the next section. A barrier is placed between the two sections of computation. The threads *hit* the *barrier* and are held until all threads are ready to continue.

A barrier synchronization counting 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 release semaphore signaling the other processors that the threads are synchronized.

Barrier synchronization operations are accessed through AIL routines. The

The operation of the barrier synchronization AIL routine, `barrier_sync`, is functionally equivalent to the following:

```
void
barrier_sync (unsigned int *semaphore,
          unsigned int semaphore_init,
          unsigned int *release,
          unsigned int release_value)
{
     if (fetch_and_dec32(*semaphore) - 1 == 0) {
          *semaphore = semaphore_init;
          *release = release_value;
     } else
          while (*release == release_value);
}
```

The operation of the barrier synchronization AIL routine, `barrier_sync2`, is functionally equivalent to the following:

```
void
barrier_sync2 (int *semaphore,
          unsigned long long *release,
          unsigned long nthreads)
{
     unsigned long long tmp;

     tmp = *release;
     if(fetch_and_inc32(semaphore)+1==nthreads){
          *semaphore = 0;
          coherent_inc64(release);
     }
     /* coherent_inc64() is a weakly ordered
      * operation
      */
     while (*release == tmp);
}
```

The V2500 server implements the `coherent_inc64()` function in hardware for improved performance of the barrier release operation. The function coherently increments a double word in memory.

The operation is performed in the CTI cache of each node that has encached that memory line. Once the double word of the memory line is incremented in a CTI cache, all processors with that memory line cached invalidate the memory line from their data caches. Invalidating the memory line from the data caches forces each processor to re-access the CTI cache, obtaining the newly incremented value. The newly incremented value does not compare as equal to the original release value, and the barrier synchronization operation releases the thread that issued it.

NOTE

The barrier release variable must be the first double word of a memory line, and the remainder of the memory line should not be used (for performance reasons). The TLB U-bit must *not* be set for the page containing the barrier release memory line.

The semaphore variable must be the first word of a memory line. The TLB U-bit must be set for the page containing the semaphore memory line.

Issuing the coherent_inc64 operation

The `barrier_sync2` AIL routine described in the previous section requires the `coherent_inc64`.

There are two methods that perform the `coherent_inc64` operation. The first is to use a single PA-8500 instruction; the second uses a sequence of CSR accesses to issue the operation.

coherent_inc64 instruction

The coherent increment double instruction is described below.

Coherent Increment Double **CINCD**

Format:
CINCD **(s,b)**

01	b	00	s	FA	0	rv
6	5	5	2	8	1	5

Description: The memory system increments the double word specifed by the effective address generated by the instruction. If the memory line was being held for private use (cached for write) prior to the instruction, then the encached line is written back to memory. The line may be cached in the CTI cache of other nodes for read access after the instruction is performed. The offset is formed from the base register, b.

The instruction performs read and write access protection checking and is not executed specutively.

The PSW D-bit (Data address translation enable) determines whether a virtual or absolute address is used.

The fields have the same meaning as the FDC instruction of the PA-RISC 2.0 Architecture. The im5 field of the FDC instruction is set to zero for this instruction. The PAC uses im5 field with a zero value ito detect a CINCD operation.

Operation:

```
space ¨ space_select(s,GR[b],SHORT);
offset ¨ GR[b];
Coherent_increment_double(space, offset)
```

Exceptions: Non-access data TLB miss fault
Data memory access rights trap

Restrictions: The base register must be 32-byte aligned.

CSR method of issuing coherent_inc64

An alternate method of issuing the coherent_inc64 operation exists that uses CSR operations rather than a single PA-8500 instruction. The CSR method requires a sequence of instructions that accesses PAC CSRs.

This method takes more execution time to setup and issue the operation than the single instruction method but does not require any non-architected PA-8500 support.

The following sequence of instructions provides an alternate method of issuing the coherent_inc64 function:

```
     PROBEW cincd_addr        ;Check protection
loop
     LDI    1,%t1
     STD    %t1,(CSR_OP_CNTX) ;Arm Operation
     LPA    cincd_addr,%t2
     STD    %t2,(CSR_CINCD)   ;Issue Coh.Inc.
     LDD    (CSR_OP_CNTX),%t4
     BB,*<  %t4,62,loop       ;Check trigger
```

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 *Triggered* 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 addresses

The PAC has multiple registers and several addresses to implement CSR-based semaphore operations. The two types of registers, the Operations Context register and the Operation Address register, are detailed in the chapter "Cache Management." The addresses are discussed in the following sections.

PAC Fetch Operation addresses

Each PAC has six Fetch Operation addresses, three for each processor pair. Reading 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. As part of the fetch operation, the *Armed* bit is cleared and the *Triggered* bit is set. The address contained in the Fetch Operation Address register is used as the address for the fetch operation, 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.

The size of the read access of the fetch operation address determines the size of the operation, 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. As part of the noncoherent operation, the *Armed* bit is cleared and the *Triggered* bit is set. The address contained in the Fetch Operation Address register is used as the address for the non-coherent access.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. For a Noncoherent Write Operation, if the *Armed* bit is not set, the PAC drops the noncoherent write.

The size of the read or write access to the CSR Operation address determines the size of the operation, 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 addresses

Each PAC has four Coherent Increment addresses, one for each processor. Writes to these addresses trigger coherent increment operations.

If the *Armed* bit in the Operation Context register is set, a write to one of the addresses results in a coherent increment operation at memory. The address of the double word that is incremented is obtained from the data of the double word store instruction that is used to issue the operation. As part of the coherent increment operation, double word memory 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.

PA-8500 TLB Entry U-bit

Each PA-8500 TLB entry contains a bit that controls whether an access to coherent memory space should accelerate the memory line into its data cache. The V2500 server uses this bit to inhibit noncoherent operations from being moved into data cache.

Table 35 lists the supported semaphore operators and the associated PA-8500 instructions used to issue the operations for the accessed memory page.

Table 35 Semaphore operation instructions

TLB entry U-bit value	PA-8500 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	Noncoherent Load 32-bit
	LDD	Noncoherent Load 64-bit
	STW	Noncoherent Store 32-bit
	STD	Noncoherent Store 64-bit

An entire memory page must be used in a consistent manor with respect to coherent or noncoherent operations. Mixing of coherent and noncoherent accesses to a memory line result in errors being returned to the issuing processor.

5 Interrupts

This chapter discusses the interrupt mechanism of the V2500 server.

The *PA-RISC 2.0 Architecture* manual presents a detailed discussion of the interrupt mechanism implemented for the PA-8500 processor, and that material is not presented in this book. Instead, this chapter discusses interrupts registers unique to the V2500 server.

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 instruction.

When responding to an interrupt, the processor behaves as if a single instruction were fetched and executed, not pipelined. Any interrupt conditions raised by that instruction are handled immediately. 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. The external device sets a bit in the External Interrupt Request register (EIRR). 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

The PA-8500 processor is sent an external interrupt by writing to its External Interrupt Request Register (EIRR). Provided that the PA-8500's external interrupts are enabled (processor status word's I-bit) and that the External Interrupt Enable Mask (EIEM) register has not masked the particular interrupt being sent, the processor current execution of instructions will be interrupted to deal with the external interrupt.

The processors send interrupts to other processors. Also, hardware detected events may send an interrupt to a processor. Independent of the source of the interrupt, the interrupt is sent to the destination processor by writing to that processor's External Interrupt Request Register.

The V2500 server interrupts occur from several sources which include:

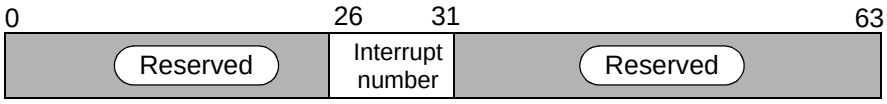
- Other processors
- I/O subsystem
- Memory subsystem
- Time of Century counter loss of synchronization
- Utilities board

The EIRR is written to using a double word store.

All fields of the register are undefined when read.

The format of the EIRR is shown in Figure 28.

Figure 28 PA-8500 External Interrupt Request register definition

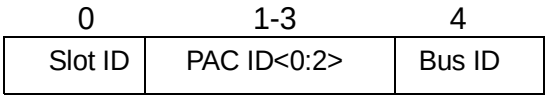


The *Interrupt Number* field (bits 26:31) specifies the external interrupt to be set in the EIRR. The value of the interrupt, 0-63, is encoded in the six-bit field.

Five-bit processor identifier

Each PA-8500 processor within a node has a unique five-bit identifier that to specifying which processor an interrupt should be sent. The identifier is five bits wide (specifying up to 32 processors).

Figure 29 Five-bit processor identifier



The fields in the Processor Identifier are defined as follows:

- *Slot ID* bit (bit 0)—Identifies which of the two processors on a runway bus is being referenced.
- *PAC ID<0:2>* field (bits 1:3)—Identifies which of the eight PACs within a node is being referenced.
- *Bus ID* field (bit 4)—Identifies which of the two runway buses of a PAC is being referenced.

The 2-bit *Proc. ID* field of the Processor Configuration register on each processor determines which of the four processors physically attached to a PAC it is. The most significant bit of the *Proc. ID* field specifies the processor Slot ID, and the least significant bit specifies the processor Bus ID.

The processors attached to PAC 0 are numbered 0, 1, 16, and 17, with processors 0 and 16 sharing one runway bus, and processors 1 and 17 sharing the other. The remaining PACs within a node are numbered in a similar manor.

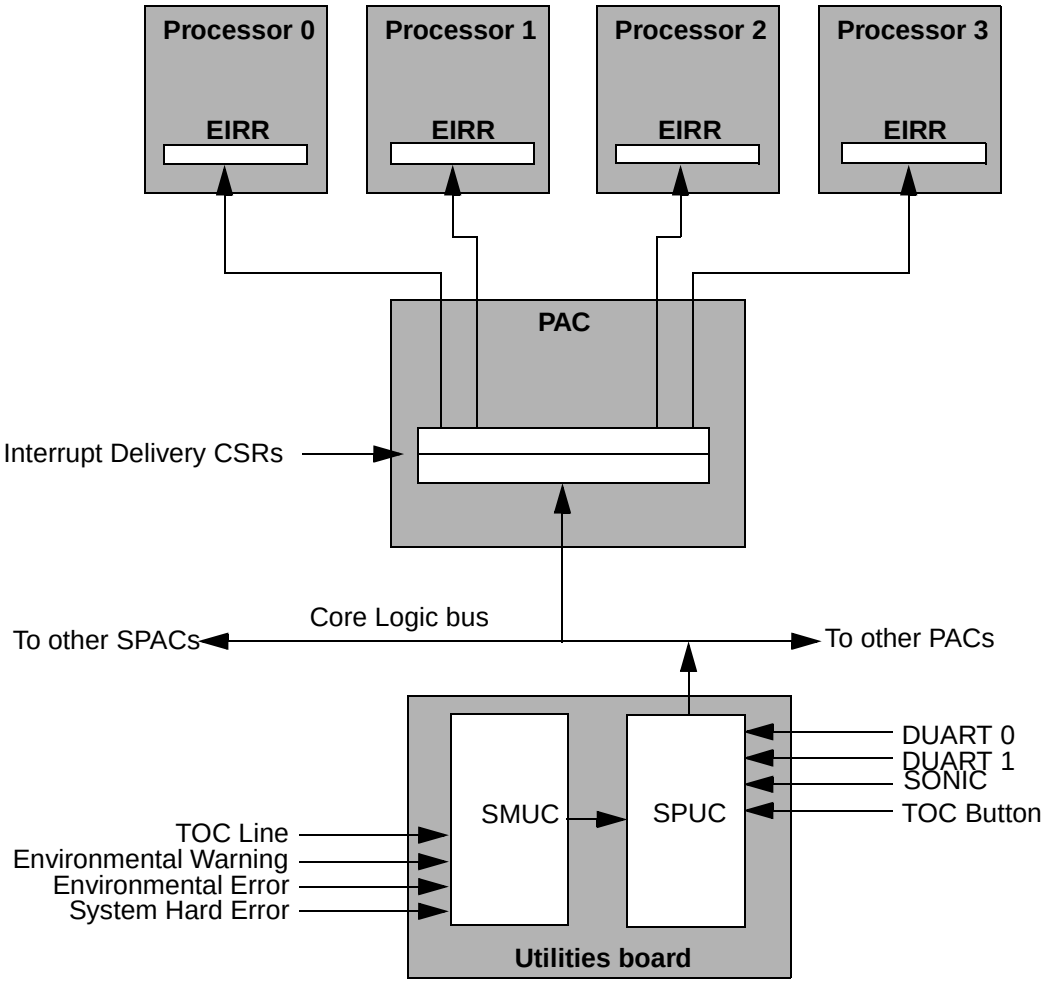
Utilities board interrupts

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

- Environmental conditions
- Transfer of control (TOC)
- External communications
- System warnings and failure

Figure 28 shows how these interrupts are presented to the processor.

Figure 30 Core logic interrupt system



The Utilities board provides interrupt information to all PACs in the system. Each PAC determines if one of its four processors is enabled to handle the pending interrupt.

The Utilities board accepts eight separate interrupt sources listed in Table 36.

Table 36 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

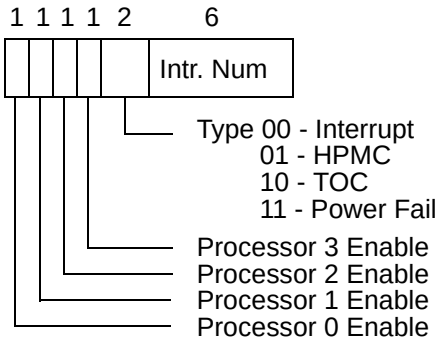
The Utilities board processes interrupts as follows:

- Interrupts are latched into the Interrupt Status register in the MUC.
- Interrupts can also be forced into the Force Interrupts register for testing purposes.
- 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.

PAC interrupt logic

Each PAC receives an eight-bit mask from the PUC (using 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 31 shows the interrupt delivery data.

Figure 31 **PAC interrupt delivery information**

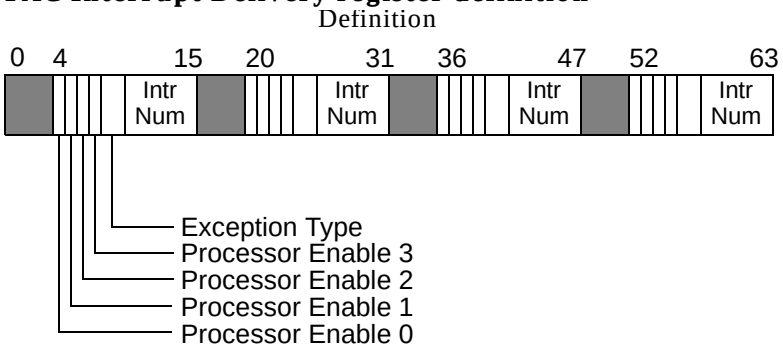


The information contains individual enables for each of the four processors connected to a PAC, the type of exception, and the interrupt number or an interrupt exception type.

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 32 **PAC 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 exception:

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

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

The Utilities board interrupts map to the core logic interrupt delivery registers as shown in Table 37. All fields are written to by a CSR write and read using a CSR read. Reset has no effect on the register.

Table 37 Core logic interrupt delivery registers

Utilities board interrupt source	Register and bits
Duart Channel #0	Register 0, bits 4:15
Duart Channel #1	Register 0, bits 20:31
SONIC controller	Register 0, bits 36:47
Transfer of Control Button	Register 0, bits 52:63
Transfer of Control Line (from test station)	Register 1, bits 4:15
Environmental Warning	Register 1, bits 20:31
Environmental Error	Register 1, bits 36:47
System Hard Error	Register 1, bits 52:63

PUC interrupt logic

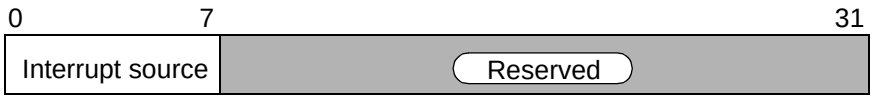
The following PUC interrupt registers comprise the PUC interrupt logic:

- 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 37. Figure 33 shows the definition of the register.

Figure 33 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.

Each individual bit of the Interrupt Status register can be cleared without affecting the other bits, even when the register is receiving an interrupt. 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 ASIC.

Table 38 shows the bit field assigned to each core logic interrupt source.

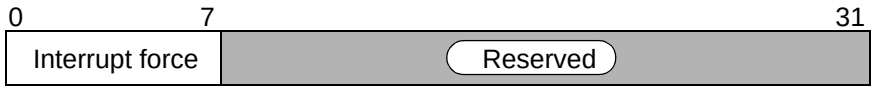
Table 38 **PUC Interrupt register field definitions**

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

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 34 shows the definition of the register.

Figure 34 **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 or not. A reset clears the register. Table 38 shows the interrupts assigned to each core logic interrupt force bit.

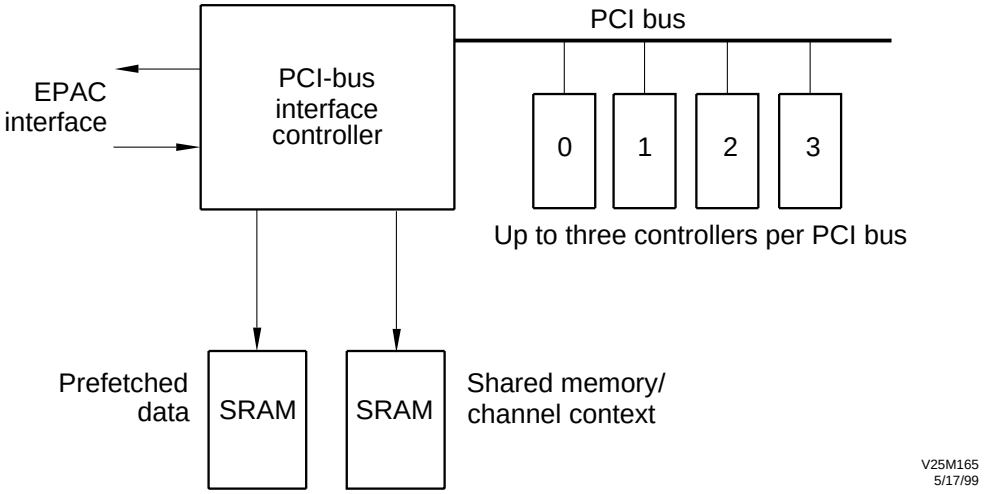
6 I/O subsystem

The I/O subsystem connects the system to its peripheral devices using the industry standard peripheral component interface (PCI) bus. A fully configured system provides up to eight, 64-bit PCI buses, one for each PAC. Each bus supports up to four controllers for a maximum of 28 PCI controllers for the system.

Overview

The I/O subsystem transfers data coherently to and from the system main memory, eliminating the need for flushing the processor caches. Figure 35 shows a block diagram of the I/O subsystem based on the PCI-bus Interface Controller (SAGA).

Figure 35 I/O system block diagram



The SAGA 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 SAGA per PAC. Each SAGA has a pair of unidirectional links to the associated PAC. Each SAGA has two physical SRAM banks, one for SAGA data prefetch and one for PCI controller-shared memory.

Logical I/O channel

The SAGA 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 36 depicts the logical I/O channel concept, and Figure 37 shows the PCI bus command and address format.

Figure 36 Logical I/O channel model

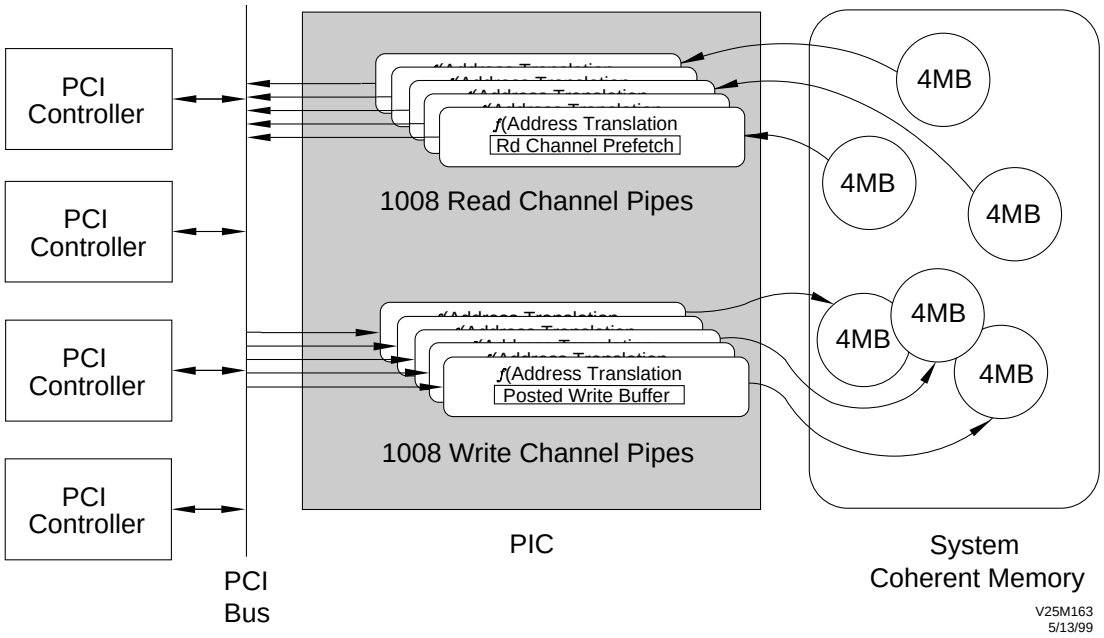
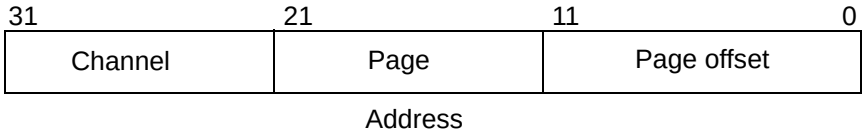


Figure 37 **PCI bus command and address**



The 10 most significant bits of the PCI address define the logical channel number, providing a total of 1,024 logical channels. Channels 1008-1023 are reserved, 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. Best performance is usually realized, however, with a single I/O transfer per channel. A channel can not be used by multiple controllers at the same time.

Channel initialization

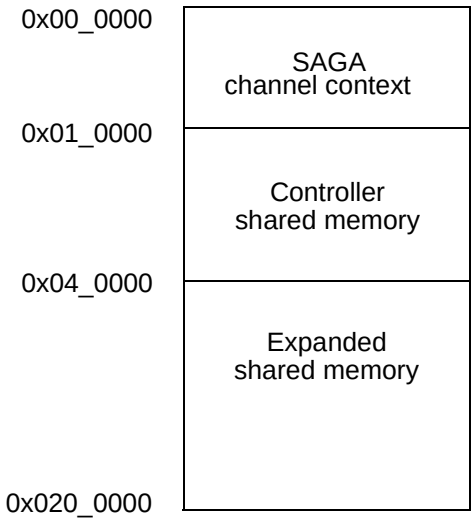
Before a processor initializes an I/O operation, it must set up a channel for the appropriate controller by writing to the SAGA Channel Builder register.

The build consists of a single write to the Channel Builder register. See the section “SAGA Channel Builder register” on page 142. The SAGA initializes all the external SRAM channel context state and prefetches any needed data and TLB entries.

Channel context and shared memory SRAM

The SAGA 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 for both controller and expanded is available for the remainder. The SAGA supports up from 256 Kbytes to 2 Mbytes of external CCSRAM. See Figure 38.

Figure 38 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 access 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 SAGA Channel Builder register.

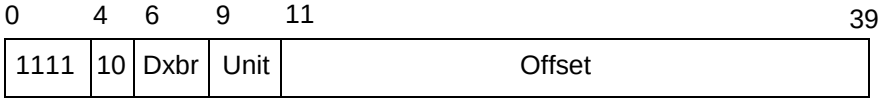
Shared memory

The SAGA provides a locally 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*, however, from both PCI memory space and processor I/O space.

Host-to-PCI address translation

The 40-bit system address map, shown in Figure 39, reserves 16 Gbytes from 0xF8 0000 0000 to 0xFB FFFF FFFF for host access to PCI devices.

Figure 39 **I/O address space format**



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

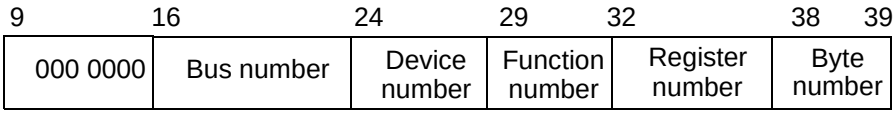
- *Dxbr* field bits (6:9)—Specifies one of eight PACs
- *Unit* field bits (9:10)—Specifies one of four I/O units (SAGAs) connected to a PAC.
- *Offset* field bits (11:39)—Specifies 29-bit SAGA mapping.

PCI configuration space

The PCI specification establishes three PCI address spaces: configuration, I/O, 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 SAGA to set up PCI I/O and memory space requirements in the system address map. Figure 39 shows the PCI configuration address format.

Figure 40 PCI I/O configuration space format



The fields for the 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 SAGA. 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 SAGA 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 SAGA combines the least significant bits of the system 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.

I/O space-to-PCI map

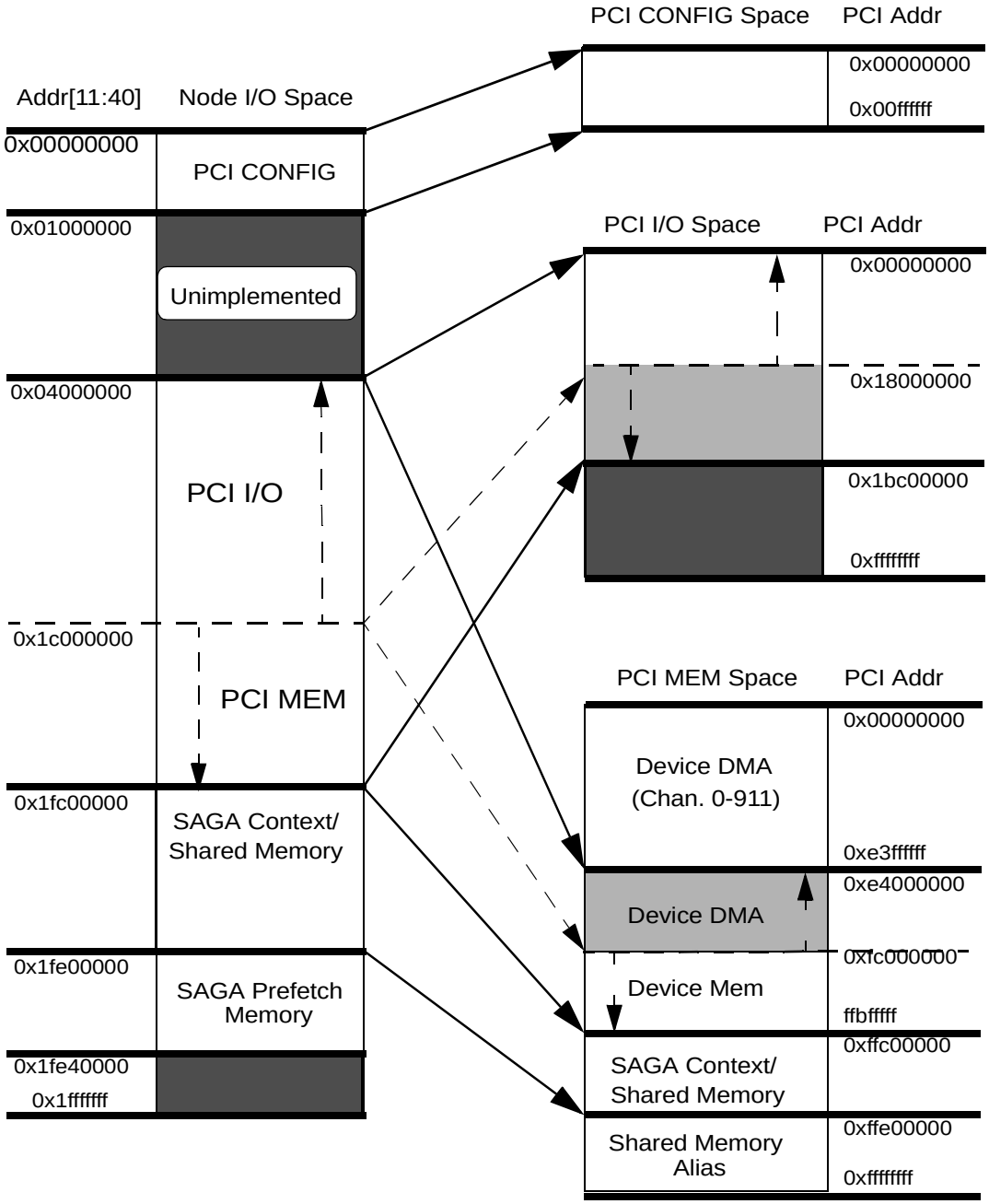
As shown in Figure 41, the SAGA maps its partition of I/O space into the three PCI spaces. It also reserves an area for diagnostic windows into the external SAGA context/shared memory and external SAGA prefetch memory.

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

The SAGA can generate PCI addresses, increasing from 0x0000 0000 in PCI I/O space and decreasing downward from 0xFFBF 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.

Figure 41 I/O space to PCI space mapping



PCI-to-host memory address translation

There are two types of address translation: physical and logical. An Address Translation Enable bit (ATE) for each channel determines the address translation between PCI and system coherent memory. SAGA only supports 32-bit PCI addressing.

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.

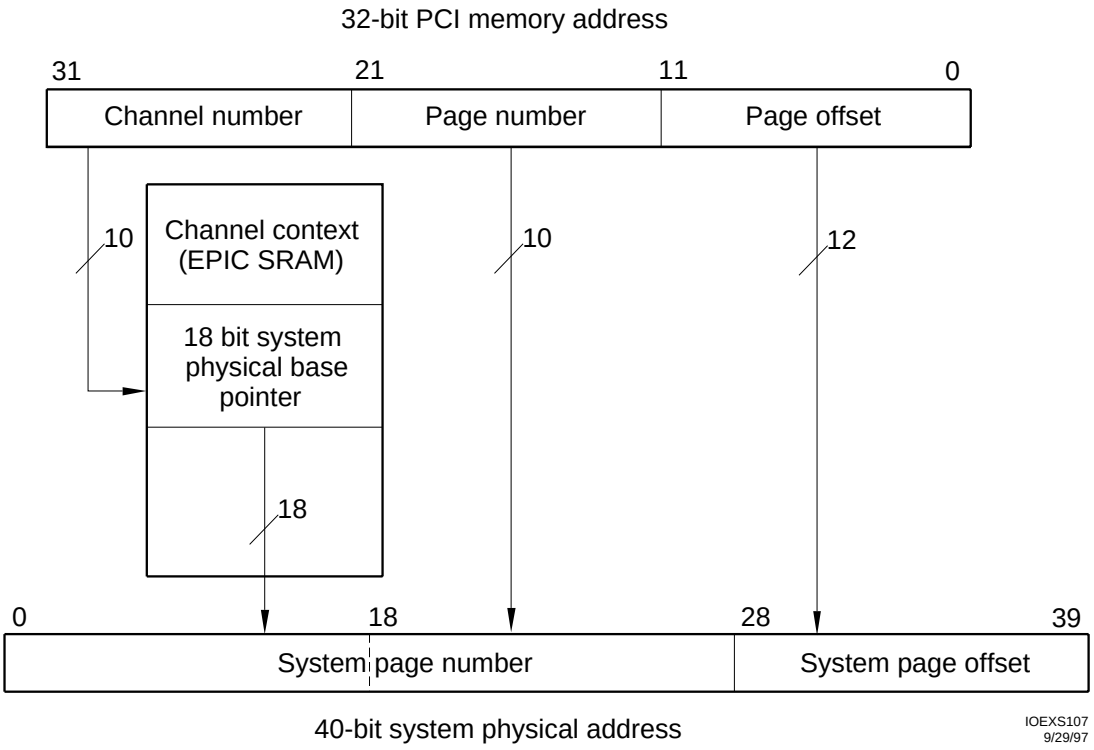
Most modern I/O controllers use part of the host memory for storing control and status blocks. Typically, these are accessed using word accesses over the PCI bus. Since the main memory access latency is relatively large, part of the channel context SRAM is used for storing the control and status structures.

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 system address.

Figure 42 Physical mode address translation



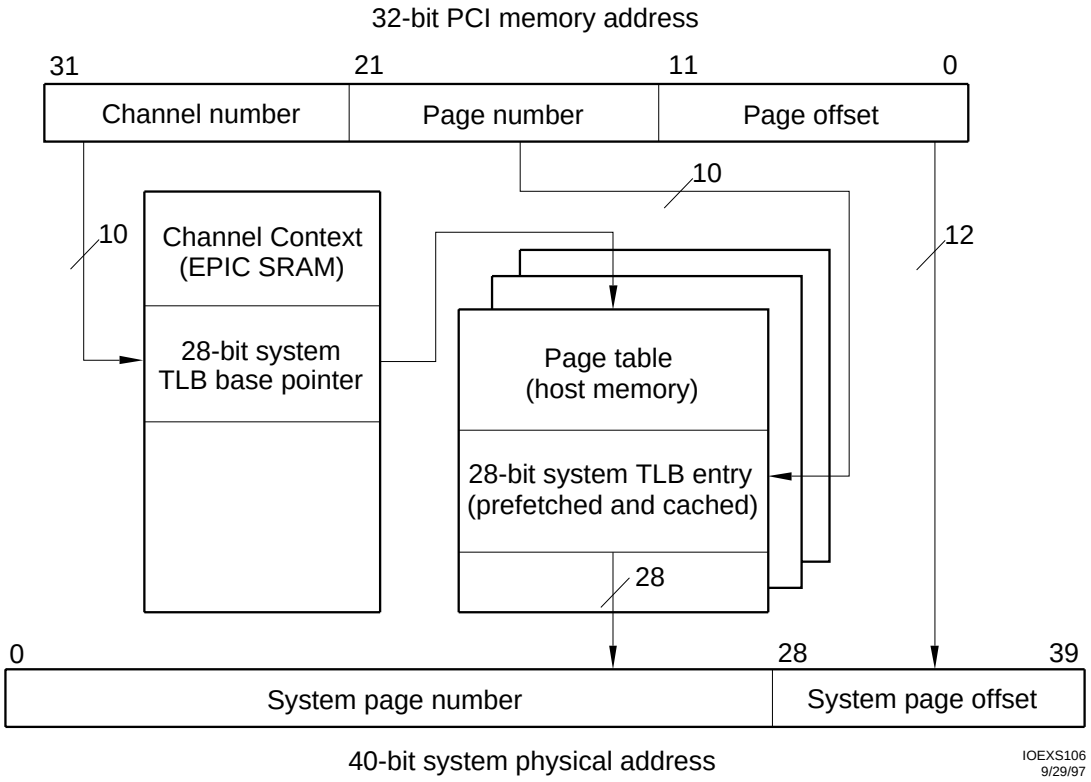
Some I/O transfers, specifically remote receive transfers with many small I/O streams, need to be handled in a nondeterministic order. If each transfer were located in its own channel, software could run out of channels. If the transfers are packed into a single logical channel, the TLB miss overhead 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 system address is using a logical translation mode channel. For logical translations, a translation table is used to generate the 40-bit system address from the 32-bit PCI address.

Figure 43 Logical mode address translation



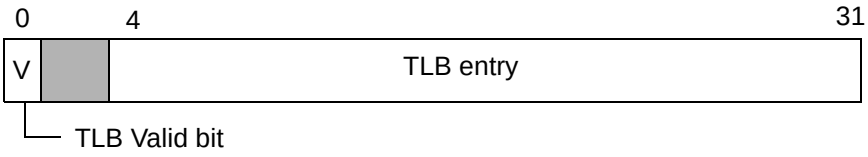
The logical address translation mode is based on a translation lookaside buffer much like the processor TLB. The translation table converts the PCI bus addresses into system addresses on a page (four-Kbyte) basis, and therefore, are aligned on page boundaries. A TLB base pointer points to the page of TLB entries in system memory. The PCI page

number indexes 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.

I/O TLB entry format

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

Figure 44 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 43.

PCI memory read transfers

To handle long and variant system memory latencies, the SAGA 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 SAGA to:

- Compensate for start-up latencies from memory
- Maintain a minimal prefetch depth that matches the memory latency needed for the controller 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. I/O transfers are not included, however, in system memory sharing lists. Therefore, if the data is modified later, the SAGA prefetched data will be stale. This fact dictates that direct memory accesses (DMA) from system memory be used only for buffered data that is defined prior to the SAGA issuing any prefetches for that data. To purge prefetched data from the SAGA prefetch buffers, the channel must be either reinitialized or rebuilt through the Channel Builder register.

To accommodate these prefetch techniques, the SAGA 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 system accesses when a controller switches from one channel to another.

There is one channel prefetch buffer per channel for a total of 1,008 channel prefetch buffers. The amount of storage space needed to cover the start-up latency for system accesses 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 with minimal stalls on the PCI bus.

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 SAGA a new address stream to the time the SAGA provides the first data word.

The SAGA provides a Channel prefetch enable (P) bit to hide controller start-up memory latencies. When enabled (P is set to 1), the SAGA Channel Builder register prefetches data at channel initialization time. The prefetched data is stored locally in the channel space of the SAGA external SRAM. Therefore, when a controller presents the SAGA with an address mapping into this channel, the data is already local to the SAGA, 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 (R) bit. When this bit is enabled (R set to 1) and one channel is swapped for another, the SAGA 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 SAGA can be connected to controllers with different bandwidth requirements. The SAGA must ensure that each controller has fair access to the system memory.

The SAGA consumption-based prefetch algorithm keeps the prefetch request rate matched to the amount of data that a controller is consuming from the SAGA. Each time a line of data is transferred across the PCI interface to a controller, a prefetch is scheduled for that SAGA 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 SAGA from over prefetching for a particular controller.

Stall prefetch

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

PCI memory write transfers

The SAGA has one independent write buffer 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 SAGA 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 processors.

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 SAGA. The SAGA provides a Write Purge Partial (W) mode bit per channel that defines how the SAGA 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 SAGA must perform a `Dflush_Alloc/Write_Mask` flow. The SAGA first issues a `Dflush_Alloc` to flush the line back to memory, locking down the line. When `Dflush_Alloc` is complete, the SAGA 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, there is no guarantee that the nonwritten portion of the cache line in memory is coherently maintained. Setting this bit provides accelerated partial line transfers to system coherent memory, making this mode suitable for transfers like those to kernel buffers but not suitable for I/O transfers directly to user space.

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

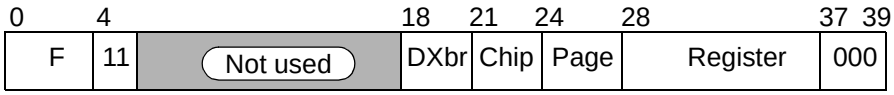
- SAGA Chip Configuration
- SAGA PCI Master Configuration
- SAGA PCI Master Status
- SAGA Channel Builder
- SAGA Interrupt Configuration
- SAGA Interrupt Source
- SAGA Interrupt Enable
- PCI Slot Configuration
- PCI Slot Status
- PCI Slot Interrupt
- PCI Slot Synchronization

SAGA CSR address decoding

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

The SAGA CSRs may be accessed by addressing, the mapping of which is shown in Figure 45.

Figure 45 SAGA CSR 40-bit address format



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

- *DXbr* field (bits 18:19)—Specifies which of the eight cross bar ports the request is to be routed.
- *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.

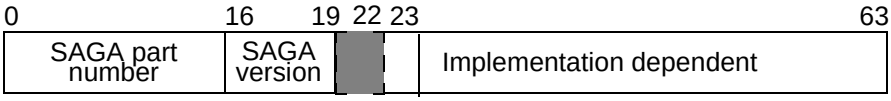
SAGA CSR definition

This section describes the SAGA CSRs.

SAGA Chip Configuration register

The SAGA contains one SAGA Chip Configuration register on each SAGA. It specifies configuration information.

Figure 46 SAGA Chip Configuration register definition



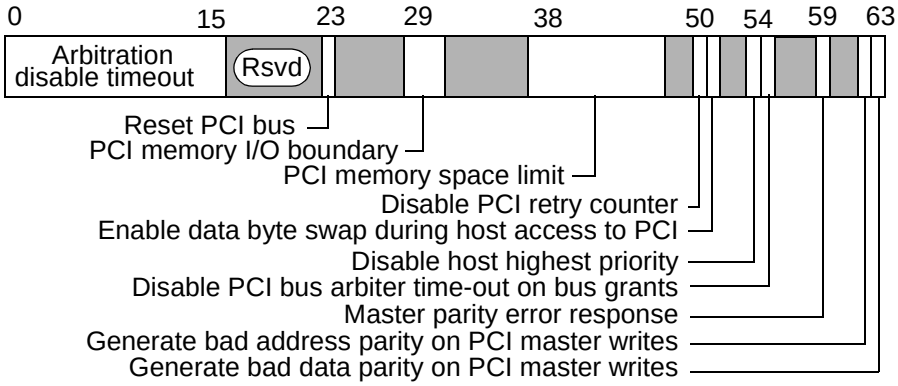
The fields of the SAGA Chip Configuration register are defined as follows:

- *SAGA part number* field (bits 0:15)—Specifies the part number for the SAGA chip. A write is ignored and a read returns the hardwired value.
- *SAGA version code* field (bits 16:19)—Specifies the version of the SAGA chip. A write is ignored and a read returns the hardware value.
- *Multiple Delayed Transaction* bit (bit 23)—Specifies the support level for PCI 2.1 multiple delayed transactions.
- *Implementation dependent* field (bits 24:63)—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 SAGA that provides configuration information about the PCI Master interface. The format of the register is shown in Figure 47. All reserved fields are read as zero, and writes are ignored. All implemented fields are read with the last value written.

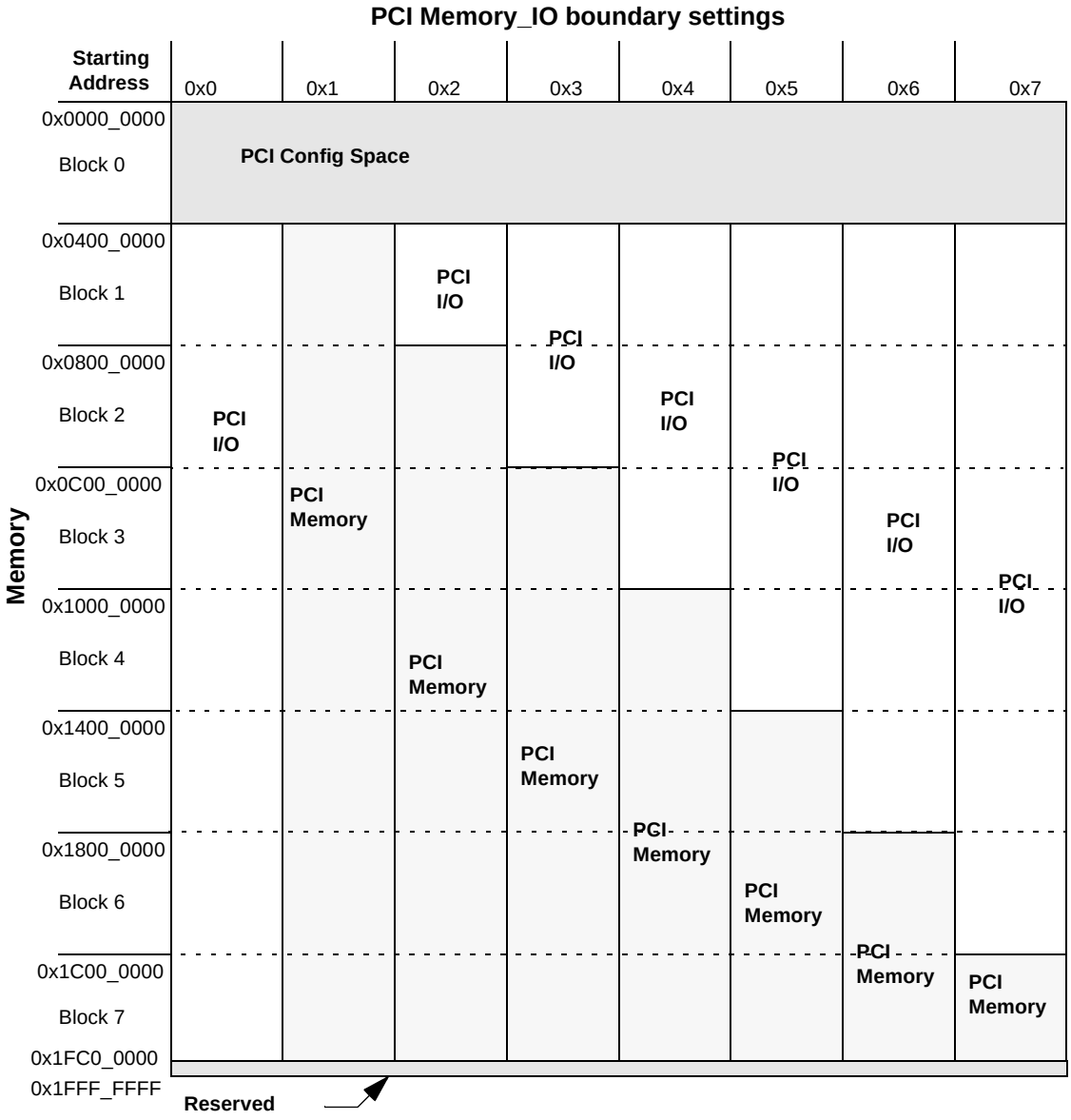
Figure 47 **PCI Master Configuration register definition**



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

- *Arbitration disable timeout* field (bits 0:15)—Defines the PCI arbitration disable timeout threshold. This field defines the controller's arbitration disable time-out from the read and write managers. The incrementer for this timer is in units of 256 PCI clocks.
- *Reset PCI bus* field (bit 23)—Resets the PCI bus.his output triggers external hardware to reset the PCI bus.
- *PCI Memory_I/O Boundary* field (bits 29:31)—Controls the amount of Node Local-I/O space mapped to PCI MEM and PCI I/O address space in contiguous 64-Mbyte blocks. Figure 48 shows the PCI memory space setting.

Figure 48 PCI Memory space setting

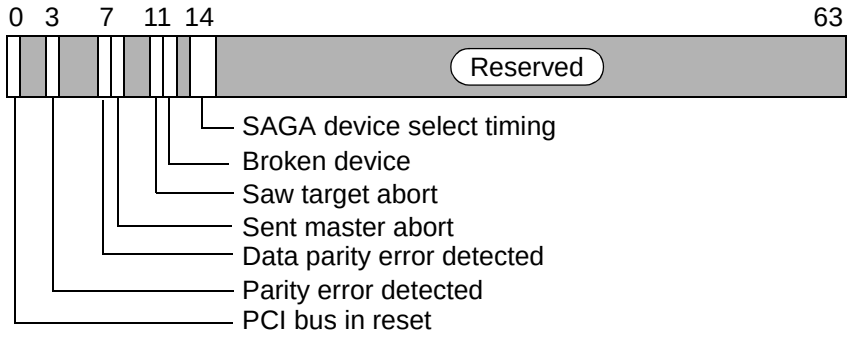


- *PCI memory space limit* field (bits 38:47)—Resets to 0x3f0 or 1008 decimal. The SAGA reserves the upper 60-Mbytes (channels 1008-1022 inclusive) for PCI controller shared memory address space and the highest four Mbytes (channel 1023) for SAGA context SRAM. The SAGA does not respond to PCI Controller DMA from channels 1,022 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 SAGA performs its normal operation when it detects a parity error as bus master.
- *Generate bad address parity on PCI master writes* bit (bit 62)—Forces bad address parity out to PCI (for diagnostic use only).
- *Generate bad data parity on PCI master writes* bit (bit 63)—Forces bad data parity out to PCI (for diagnostic use only).

PCI Master Status register

Each SAGA has one PCI Master Status register that provides status information for the PCI Master interface. All fields are cleared by a reset except the SAGA Device Select Timing field; it is hardwired to the value one.

Figure 49 PCI Master Status register definition



The bits in the SAGA 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 SAGA detected a parity error on incoming read data while the SAGA was bus master.This bit is set regardless of Master Parity Error Response bit in the PCI Master Control CSR. Returns an error response to the requestor and sets the Master error in Error Cause CSR.
- *Data parity error detected* bit (bit 7)—Indicates that a parity error was detected on the bus while SAGA was PCI bus master. SAGA was PCI bus master, asserted PCI_PERR_ or received PCI_PERR_ during its bus tenure, and the Master Parity Error Response bit in the PCI Master Control CSR was set. Returns an error response to the requestor and sets the Master error in Error Cause.
- *Sent master abort* bit (bit 8)—Indicates that the SAGA was master and sent a Master Abort (no target claimed the bus cycle). The Host receives an error response. The Host will receive an error response to it's request. In the case of an address parity error, one or more targets should signal SERR_. Returns an error response to the requestor and also sets the Master error in Error Cause.
- *Saw target abort* bit (bit 11)—Indicates that the SAGA was master and received a Target Abort. The Host will receive an error response. The Host will receive an error response to it's request. In the case of an address parity error claimed by a target, the target should

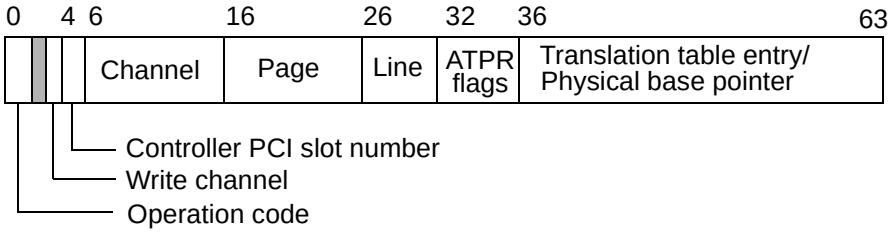
terminate with a target abort and signal SERR_ to SAGA. Returns an error response to the requestor and sets the Master error in Error Cause.

- *Broken device* bit (bit 12)—Indicates that the SAGA PCI interface received a grant and an Idle bus for 16 clocks but did not run a bus cycle. The SAGA returns an error response to the requestor and sets the Master error in the Error Cause register.
- *SAGA device select timing* field (bits 14:15)—Sets medium-speed address decode on PCI. This field is read-only.

SAGA Channel Builder register

The SAGA Channel Builder register on each SAGA sets up channel context in preparation for an I/O operation. The format of the register is shown in Figure 50. Writing to this register stores the value to all fields, and reading it returns the last written value. A reset clears all fields. 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.

Figure 50 SAGA Channel Builder register definition



The fields and bits in the SAGA 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 when set or a memory read channel when 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.

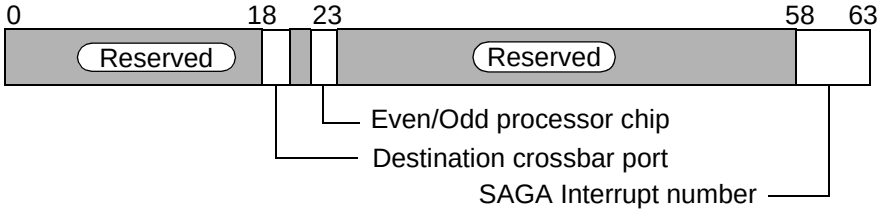
- *Page number* field (bits 16:25)—Indicates the 10-bit PCI page number.
- *Line number* field (bits 26:31)—Indicates the line number inside the page from which to start prefetch (only applies if a read channel, i.e. the Write Channel bit = 0).
- *A - Address translation enable* bit (bit 32)—Indicates 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 previously encached 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 SAGA 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 the 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 TLE 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, pointing to a four-Mbyte physically contiguous block of memory.

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

SAGA Interrupt Configuration register

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

Figure 51 SAGA Interrupt Configuration register definition



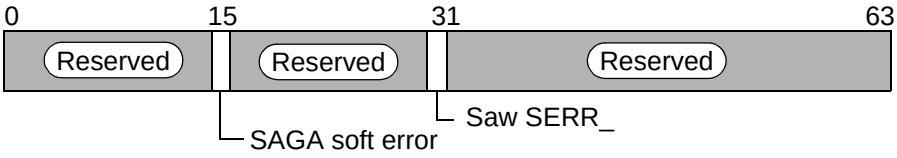
The fields and bits in the SAGA Interrupt Configuration register are defined as follows:

- *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.
- *SAGA Interrupt number* (bits 58:63)—Indicates the processor External Interrupt register interrupt bit to be set.

SAGA Interrupt Source register

Each SAGA has one Interrupt Source register that holds pending SAGA 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 SAGA Interrupt Enable register, then SAGA sends an interrupt. If the interrupt enable is written to a one value while the interrupt is pending in the Interrupt Source register, the SAGA generates an interrupt following the response to the register write.

Figure 52 SAGA Interrupt Source register definition



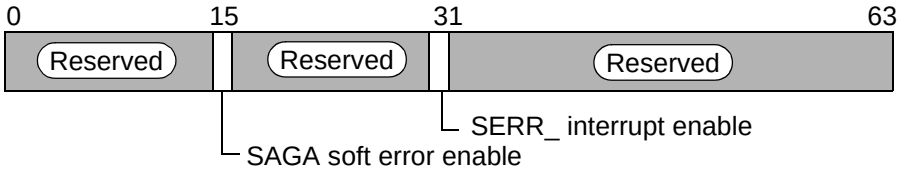
The bits in the SAGA Interrupt Source register are defines as follows:

- *SAGA 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 SAGA received an SERR_ on the PCI bus.

SAGA Interrupt Enable register

SAGA Interrupt Enable register has a bit for every source interrupt in SAGA Interrupt Source. A one value in any SAGA Interrupt Enable bit causes an interrupt when the corresponding source event in SAGA Interrupt Source occurs. This register resets to zero (all interrupts disabled). The format for this register is shown in Figure 53.

Figure 53 SAGA Interrupt Enable register definition



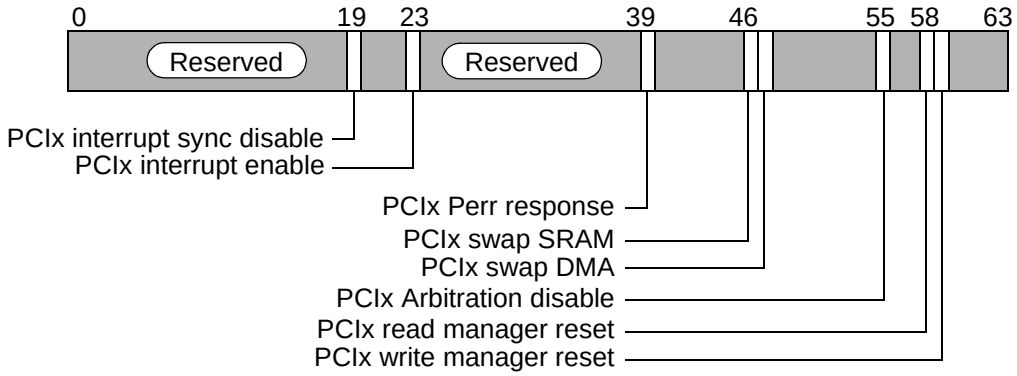
The bits in the SAGA Interrupt Enable register are defined as follows:

- *SAGA 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. A value of one enables the interrupt. (SERR_ always sets PCI Master Status CSR Saw Serr regardless of this bit).

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 54 PCI Slot Configuration register definition



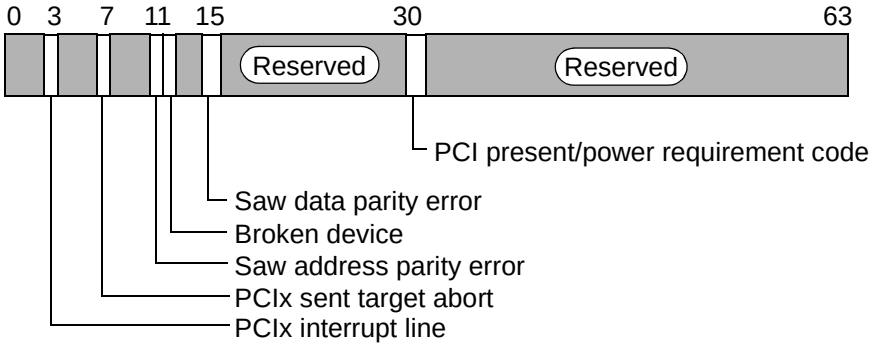
The bits in the PCI Slot Configuration register are defined as follows:

- *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 PCIx shared memory transfers.
- P*CIx_Swap DM*A bit (bit 47)—Enables byte swapping on PCIx DMA transfers.
- *PCIx Arb Disable* bit (bit 55)—Disables bus arbitration for PCIx.
- *PCIx Read Manager Reset* bit (bit 58)—Resets SAGA Read manager x.
- *PCIx Write Manager Reset* bit (bit 59)—Resets SAGA Write manager x.

PCI Slot Status register

Each SAGA has four PCI Slot Status registers that specify the status of slot specific events. Bits in these registers are set when SAGA is the target of one of the four controllers and a status event occurs. All writable fields are reset to the value zero.

Figure 55 PCI Slot Status register definition



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

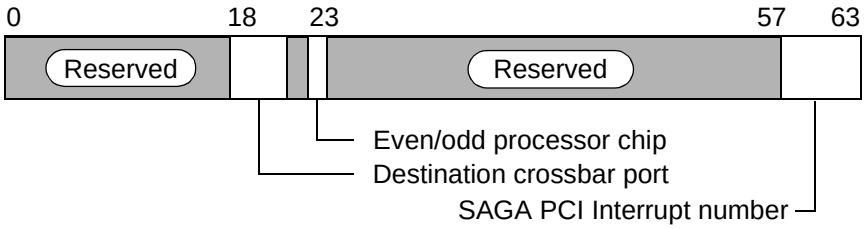
- *PCIx interrupt line* bit (bit 3)—Indicates the current state of the PCIx INTA_ line (read only).
- *PCIx sent target abort* bit (bit 7)—Indicates that the SAGA sent this slot a Target Abort bus cycle termination. This bit does not set the PCI Controller x bit in the Error Cause register.
- *Saw address parity error* bit (bit 11)—Indicates that the SAGA detected an address phase parity error on a transfer from this slot. The SAGA 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. The bus arbiter flags it as broken until the device removes it's request.This bit sets the PCI Controller x bit in the Error Cause CSR.
- *Saw data parity error* bit (bit 15)—Indicates the SAGA (as a target) detected a PCI data phase parity error on incoming (write) data from this slot. This bit does *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 SAGA has four PCI Slot Interrupt Configuration registers that specify the interrupt number and processor when an interrupt occurs on the corresponding PCI slot. The SAGA forwards the interrupt by writing the interrupt number to a local processor EIRR register. Because the SAGA can only send interrupts to one of the 16 processor EIRR registers on the system, only four bits of the address are programmable. All programmable fields are reset to zero.

Figure 56 PCI Slot Interrupt Configuration register definition



The fields in the SAGA PCI Slot Interrupt Configuration register are defined as follows:

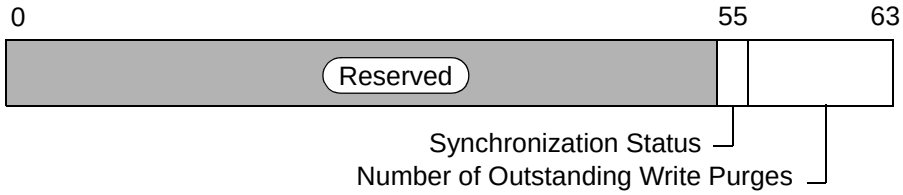
- *Destination crossbar port* field (bits 18:20)—Determines to which crossbar port (and therefore which PAC) the interrupt will be sent.
- *Even/Odd processor chip* field (bits 23)—Specifies which of the two processors for the given PAC the interrupt is to be sent.
- *Interrupt number* field (bits 58:63)—Specifies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

PCI Slot Synchronization register

The SAGA 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 reads 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 57.

Figure 57 PCI Slot Synchronization register definition



The fields and bits in SAGA PCI Slot Synchronization register are defines as follows:

- The *Synchronization status* bit (bit 55)—Specifies the completion status of the device write manager for the corresponding slot.
- *Number of Outstanding Write Purges* field (bits 56:63)—Specifies the number of outstanding write purges left to be returned before the device is synchronized. This field is informational only and should not be used in determining sync completion.

Byte swapping

In order to address different byte ordering between the PCI bus and the rest of the system, the SAGA 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 section “PCI Slot Configuration register” on page 146).
- PCI read and write of shared memory on a per-controller basis via the PCI Slot Configuration register
- Host read and write of PCI I/O, Memory, and Configuration space via the PCI Master Configuration register See the section “PCI Master Configuration register” on page 137).

7 Performance monitors

This chapter discusses the hardware used to determine the performance of the system. Some performance factors include:

- Parallel program efficiency
- Communications costs
- I/O bandwidth
- Cache-hit rate

Performance factors

The performance of applications run on the V2500 server depends upon the factors already stated. The V2500 server includes hardware to measure each of the principal factors. The measurements indicate the overall results and provide data that enables programmers to identify changes to algorithms that improve overall performance. The process of measuring performance is intrusive and can impact system performance.

Some of the important factors and concepts for their measurement are:

- 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
- I/O performance—The largest I/O factor is data transfer rates in the disk subsystem. Of particular interest are peak measurements, as most I/O is done in a burst mode. Also, information about disk access patterns should be available.
- Communication costs—Concerns in communication include the following:
  - Memory usage
  - Memory access patterns of particular code sections
  - Communication costs between threads
- Cache-hit rate—Algorithms must optimize cache use to perform well. Consequently, the V2500 server provides 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.

Performance monitor hardware

The V2500 server provides registers to record events and enable performance measurement. These registers include:

- Processor interval timer
- Per-processor latency counter
- PAC CTI cache-hit-rate counter
- Time-of-century clock (TIME_TOC)

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.
- Implementing thread timer register (TTR) in software. The operating system allows user-read access to this timer in order to use the TTR.

Per processor latency counters

V2500 system monitors all coherent read accesses made by its processors. The information monitored includes:

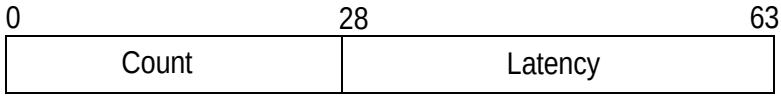
- Total coherent read latency
- Total coherent read count

Total latency for read requests is accomplished by incrementing a counter each cycle by the number of outstanding reads. The value in the counter can be divided by the total number of reads to provide the average access time for read requests.

Latency counter

The latency counter is a 40-bit read-write register that 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 58 shows the bit definition of the Performance Monitor Latency Pn register counter (where n is the processor number).

Figure 58 PAC Performance Monitor Latency register definition



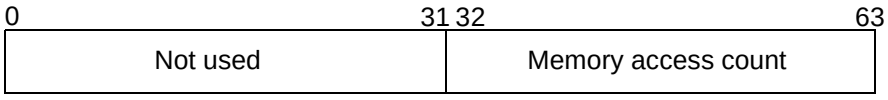
The fields in the Processor Read Latency register are defined as follows:

- *Count* field (bits 0:27)—Accumulates the count of all coherent read requests. The field can roll over no quicker than every 13 seconds (2^{28} * 6 / 120 Mhz).
- *Latency* field (bits 28:63)—Accumulates the total latency of all coherent read requests. The number of coherency read requests the processor has outstanding increments the counter each cycle . The field can roll over no quicker than every 57 seconds. (2^{36} / 120 Mhz / 10).

Event counters

The PAC has a 32-bit read-write memory access event counter shown in Figure 59.

Figure 59 PAC Performance Monitor Memory Access Count Pn register definition



Per PAC CTI cache hit rate counters

A V2500 system monitors all CTI cache read accesses made at the memory controllers (MAC). The information monitored includes:

- Total CTI cache read accesses.
- Total CTI cache read hits.

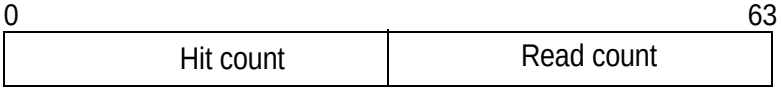
The CTI cache hit rate is computed by dividing the total CTI cache read hits by the total CTI cache read accesses.

Each PAC has a pair of CTI cache monitoring counters that includes one CTI cache read access counter and one CTI cache read hit counter. The counters are grouped and concatenated into a 64-bit register, thereby requiring only one access read the pair of counters. The following section describes the register.

There is one CTI Cache Hit Rate register per PAC that determines the CTI cache hit rate of the accesses issued by the four processors attached to that PAC. The counters are each 32-bits wide and are paired in a 64-bit register.

The format of the CTI Cache Hit Rate register is shown in Figure 60. The fields of the register are read by a read access and written by a write access.

Figure 60 PAC CTI Cache Hit Rate register definition



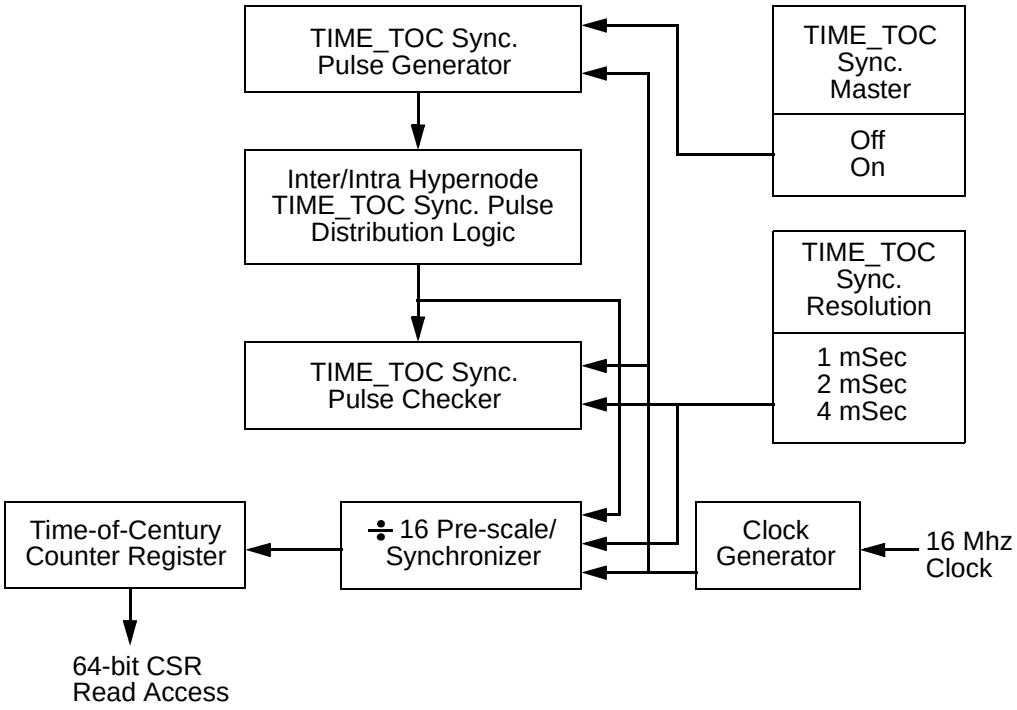
The fields in the PAC CTI Cache Hit Rate register are defined as follows:

- Hit Count field (bits 0:31)—Accumulates the number of CTI cache read requests which hit. The count can roll over no quicker than every 64 seconds (232 * 6 cycles / 120 Mhz / 2 proc).
- Read Count field (bits 32:63)—Accumulates the number of CTI cache read requests. The count can roll over no quicker than every 64 seconds (232 * 6 cycles / 120 Mhz / 2 proc).

Time-of-Century clock

The V2500 server Time-of-Century clock (TIME_TOC) can be used to time-stamp trace data stored within the system. It also provides time-stamping of transmitted messages. The receiving processor can determine the transmission time by subtracting time-stamp from the current time. Each PAC has a 64-bit TIME_TOC register accessed with a single 64-bit read.

Figure 61 Time-of-century clock hardware



Clock generator

The CUB core logic generates a 16-Mhz clock for the each TIME_TOC register. 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.

TIME_TOC synchronization pulse generation and distribution

A single PAC acts as the TIME_TOC synchronization master of the system (the outputs of all other PACs are in a high-impedance state). The master PAC sends its synchronization pulse to the nonmaster PACs on the master synchronization node. The synchronization pulse is distributed to the other nodes by setting a bit in the header of CTI packets. The amount of time required to distribute the synchronization pulse is a function of system size.

Each TAC has a CSR which specifies the source for the incoming synchronization pulse (sync. signal, CTI incoming X link, or CTI incoming Y link). The CSR also specifies whether the sync. pulse should be propagated to the CTI out going X link and/or the CTI out going Y link.

TIME_TOC synchronization pulse checker

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. Table 39 shows the check range for each of the supported resolutions.

Table 39 Time-of-Century synchronization check range

Resolution	Check Range (16 μSec clocks)
1 μSec	256 ± 7
2 μSec	256 ± 15
4 μSec	256 ± 31
Resolution	Check Range (16 μSec clocks)

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

Pre-Scale/Synchronizer

The pre-scale logic performs a divide by 16 on the SPAC 16-millisecond clock resulting in a 1 micro second period signal. This signal is used to enable incrementing the Time-of-Century Counter register.

The Time-of-Century Counter register synchronization function is performed by rounding up/down the pre-scale value when a sync. pulse arrives. The amount of rounding is a function of the TIME_TOC resolution. shows the function applied for rounding with the three supported resolutions.

Table 40 Time-of-Century synchronization check range

1 µSec Resolution Rounding Applied to bits Pre-scale<0:3>	
Original Value	Rounded Value
0-7	0
8-F	F

2 µSec Resolution Rounding Applied to bits {TIME_TOC<63>:Pre-scale<0:3>}	
Original Value	Rounded Value
00-0F	00
10-1F	1F

4 µSec Resolution Rounding Applied to bits {TIME_TOC<62:63>:Pre-scale<0:3>}	
Original Value	Rounded Value
00-1F	00
20-3F	3F

When the Sync Mode field of the Time-of-Century Configuration register has the value zero (by reseting or explicitly writing to the field), the pre-scale register is set to zero and inhibited from incrementing until the first sync pulse is received.

Holding the pre-scale value to zero allows all TIME_TOC associated initialization to proceed independently. Once all initialization is complete, then synchronization pulse generation in the master PAC is enabled.

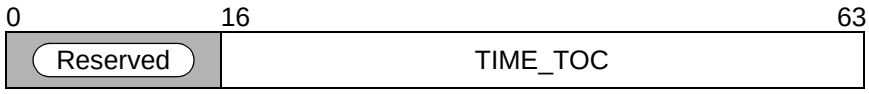
The Time-of-Century Counter register increments its 48-bit value when the pre-scale register reaches the value 0xF.

PAC Time-of-Century Counter register

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

The Time-of-Century register increments its 48-bit value when the pre-scale register reaches the value 0xF. The format of the PAC Time-of-Century register is shown in Figure 62.

Figure 62 Time-of-Century Clock register definition

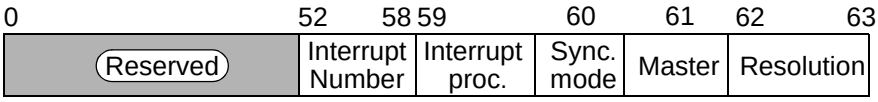


The *TOC* field (bits 16:63) increments each time the prescale logic has the value of 0xF. The register is accessible using a 64-bit CSR read or write. Reset does not effect this register. The least significant two bits may be rounded up or down when a synchronization pulse is received, depending on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

PAC Time-of-Century Configuration register

The format of the PAC Time-of-Century configuration register is shown in Figure 63.

Figure 63 PAC Time-of-Century Configuration register definition



The bits and fields of the PAC Time-of-Century Configuration register are defined as follows:

- *Interrupt number* field (bits 53:58)—Specifies the interrupt number sent 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. An interrupt can only be sent to one processor of each processor bus, and the Interrupt Processor bit specifies which runway bus the interrupt is to be sent. The value of zero specifies that the interrupt be sent to runway bus zero, processor zero. Similarly, a value of one specifies that the interrupt is to be sent to runway bus one, processor zero. The interrupt can not be sent to processor one of a runway bus. The field is not initialized by reset.
- *Synchronization mode* bit (bit 60)—Indicates whether the synchronization pulse starts incrementing or synchronizing the Time-of-Century and prescale registers. At reset the field is cleared and indicates that the next synchronization pulse received will start incrementing the Time-of-Century 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-of-Century 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-of-Century register resolution. The field is reset to the value zero (TIME_TOC off). Table 41 shows the supported resolutions.

Table 41 **Time-of-Century resolutions**

Value	Resolution
0	TIME_TOC off
1	1 microsecond
2	2 microseconds
3	4 microseconds

A value of zero (TIME_TOC off) disables TIME_TOC synchronization pulse checking.

TAC Time-of-Century Configuration register

The format of the TAC Time-of-Century Configuration register is shown in Figure 64.

Figure 64 **TAC Time-of-Century Configuration register definition**

The fields of the TAC Time-of-Century Configuration register are defined as follows:

- *Source* field (bits 60:61)—Specifies which synchronization pulse input (sync. signal, X incoming link, or Y incoming link) should be propagated to the enabled synchronization pulse output. Table 42 shows the *Source* field selection values.

Table 42 **Time-of-Century synchronization pulse source**

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

The Source field is cleared by reset.

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

- *X link* field (bit 62)—Enables a synchronization pulse from the source selected by the *Source* field to be propagated to the X CTI out going link.
- *Y link* field (bit 63)—Enables a synchronization pulse from the selected source to be propagated to the Y CTI out going 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 is cleared. With this field 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.

8 System utilities

Each V2500 server has a section of hardware known as the Exemplar Core Utilities board (CUB) located on the MIB. On the CUB are two FPGAs: the Exemplar Processor Utilities (PUC) and the Exemplar Monitoring Utilities (MUC). The PUC provides the CUB a means to send interrupts and error messages to the processors and to receive control messages from the processors. The MUC performs all environmental monitoring. The CUB board connects to all PACs through the core logic bus.

Utilities board

The CUB, or Utilities board, handles all system housekeeping chores. It connects directly to the MIB where it attaches to the core logic bus, the environmental sensors, and other test points. It interfaces to the liquid crystal display (LCD), the optional teststation (an ethernet connection), and other external devices. Figure 65 shows the Utilities board functional layout.

The heart of the CUB 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 teststation can be connected via these links to run diagnostics and configure the system.

The MUC latches system interrupts, most of which are from environmental sensors located throughout the system. 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.

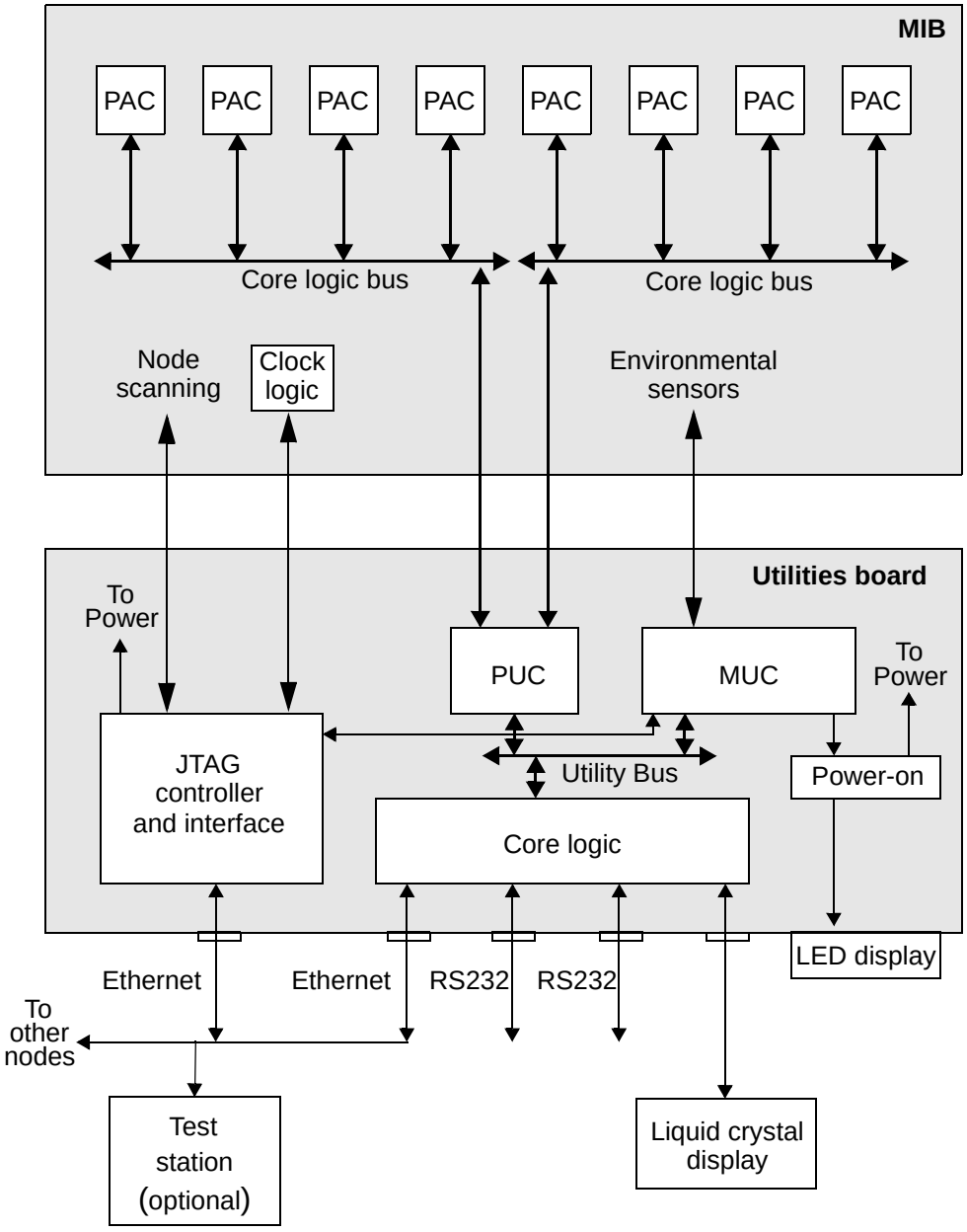
The PUC provides the core logic an interface to the core logic bus. There are actually two buses; each one connects up to four PACs. The PUC communicates to the PACs using data packets.

The JTAG (Joint Test Action Group) interface supports a teststation and a mechanism to fan out JTAG to all the boards in the system. It is used only for testing.

The V2500 server uses a test method called scanning to test boards and other hardware units. With the teststation connected to the ethernet between nodes, you can test any part of the system.

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 teststation can also read and write every CSR in the system.

Figure 65 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, the OBP interface firmware, spp_pdc, and power-on self test software (POST) (see the chapter "Booting," for more information). This EEPROM memory is four MBytes, configured as one-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.

Real Time Clock

The core logic has a battery backed real time clock used to determine the date and time when the system boots. The real time clock is physically part of the NVRAM device.

DUART

The CUB 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 connected to a modem for field service.

SRAM

SRAM is needed to support the simple core system functions. When the system powers up, the processors operate out of this SRAM. They run self test software to test and configure the rest of the system. Once the system is fully configured, the processors execute out of main memory. The SRAM is byte addressable and is 256 KBytes, configured as 64K 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 system.

LEDs and LCD

LEDs display environmental information, such as the source of an environmental error that caused the CUB to power down the system.

The LCD is driven by one of the processors via the CUB. 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 MIB and allows the system to read any COP in the system.

PUC

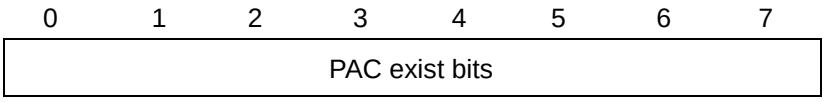
The PUC applies interrupts and error messages to the processors and receives control messages from the processors. It has two 18-bit, bidirectional buses. Each interface connects up to four PACs. The PUC provides core logic bus arbitration for the eight PACs.

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

PUC Processor Agent Exist register

The Processor Agent Exist register indicates which PACs exist in the system. 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 66 PUC Processor Agent Exist register definition

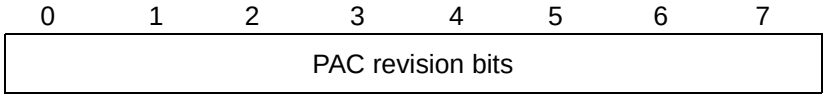


A value of one on any bit indicates that the respective PAC 0-7 is active and exists.

PUC Revision register

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

Figure 67 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 CUB. It attaches to the core logic bus so that processors can monitor the system by accessing these CSRs.

The MUC works in conjunction with a hardware section on the CUB known as the power-on circuit. This circuit controls powering up the entire system. It operates when the rest of the system 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 CUB to power down the system.

The teststation 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-volt failure
- 48-volt maintenance
- Ambient air temperature sensing
- Power-on

Table 43 **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-volt 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
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 conditions 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 system until the conditions are corrected. It also detects environmental errors such as 48-volt 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 maintenance are also detected by the power-on circuit. It applies these to the MUC, which then sends an environmental warning interrupt to the system 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.

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 milliseconds, 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 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 CUB 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 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 44 shows the LED display error codes.

Table 44 Environmental LED display

ATTN bit	LED Display	Description
1	00	CUB 3.3-volt error (highest priority)
1	01	ASIC Install 0 (MIB)
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-volt error, NPSUL fail, PWRUP=0-9
1	12-1B	48-volt error, NPSUR failure, PWRUP=0-9
1	1C-25	48-volt error, NPSLL failure, PWRUP=0-9
1	26-2F	48-volt error, NPSLR failure, PWRUP=0-9
1	30-39	48-volt error, no supply failure, PWRUP=0-9
1	3A	48-volt 7yo-yo error
1	3B	MIB power failure (MIBPB)
1	3C	Clock failure
1	3D-3F	Not used (3)
1	40-47	MB0-MB7 power failure
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

ATTN bit	LED Display	Description
1	5C-61	Fan failure (UR,UM,UL,LR,LM,LL)
1	62	Ambient hot
1	63	Overtemp MIB
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 the power-on circuit and the MUC.

CUB 3.3-volt error

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

ASIC installation error

Each ASIC has *install* lines to prevent power-up if an ASIC is installed incorrectly (such as an PAC installed in an RACs position). If an ASIC is improperly installed, the CUB 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 CUB 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 that detects board overheating. The sensors are bussed together into four system quadrants plus the MIB 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.

MIB power failure

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

48-volt maintenance

There are up to 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-volt 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. 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 teststation. 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

Described in the following sections are functions the CUB performs to control the system 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 CUB 5-volt supply reaches an acceptable level, the power-on circuit starts powering up the other dc-to-dc converters of the system in succession.

The power-on circuit does not power up the system if an ASIC is installed incorrectly (see the section “ASIC installation error” on page 173) or if an FPGA is not configured (see the section “FPGA configuration and status” on page 174). 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.

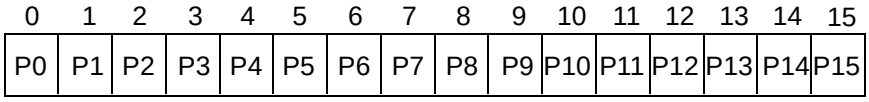
MUC CSRs

This section describes some of the MUC CSRs.

Processor Report register

There are two Processor Report registers. They indicate the processors that are working in the system. One register handles processors 0-15 and the other handles processors 16-31. Each processor reports by writing to this register and setting the bit corresponding to the processor number.

Figure 68 Processor Report register definition

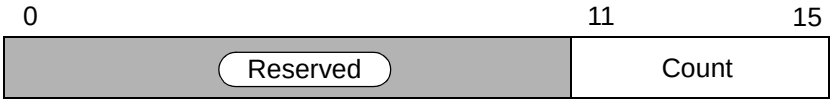


P0-P15 comprise a fully readable and writable field. The bits are cleared on reset. Once a bit is written to a one value, it remains set until cleared by reset. Writes of a zero value do nothing. The bit, P*x*, set to a one value, indicates that processor *x* has reported in working.

Processor Semaphore register

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

Figure 69 Processor Semaphore register definition

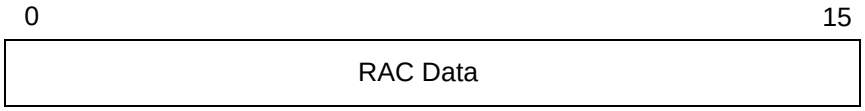


Count is cleared on reset. Writes load any value. Reads return the value of Count and then increment Count atomically.

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 70 RAC Data register definition

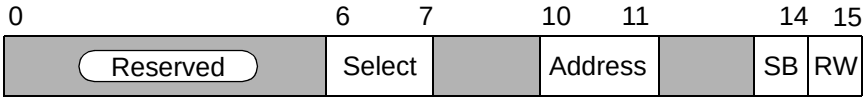


RAC Data bits comprise a fully readable and writable field. After an RAC read operation, the RAC Data register holds the data. After the RAC write operation, the data is stored in the RAC register, and RAC Data is undefined.

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 71 RAC Configuration Control register definition



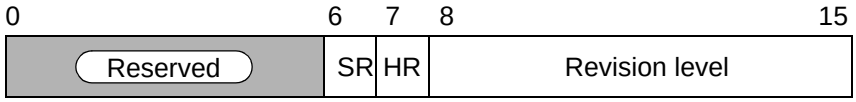
The fields and bits of the RAC configuration Control registers are defined as follows:

- *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 72 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 shown in Table 45. The combination of SR and HR bits in the write mode indicate the resets given in Table 46.

Table 45 **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 46. The only difference between a hard and soft reset is the action taken by the software upon reading the codes.

Table 46 **Reset register write codes**

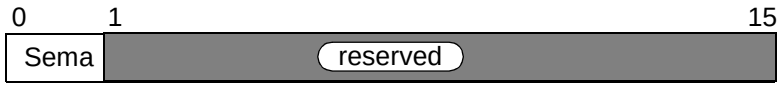
SR HR - Write	Action taken
X 1	Hard reset
1 0	Soft reset

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

General semaphore register

There are four General Semaphore register available on a SMUC used to provide general purpose critical code region locking by level software. This is an atomic Read-and-Set register.

Figure 73 **General semaphore register definition**



The General Semaphore register Sema field (bit 0) indicates the current state of the semaphore. A read access returns the current value of the Sema field and then sets the bit atomically. A write will clear the bit (independent of the value being written). The field is cleared on reset.

JTAG interface

The JTAG interface supports a teststation and a mechanism to fan out JTAG to all the boards in a system. It is used only for testing.

The JTAG functions are described in the following sections.

Teststation interface

The teststation can be a PA-RISC based workstation. The interface to the teststation is an ethernet AUI port for flexibility in connecting to many workstations.

AC test

An ac test is performed by a Test Bus Controller (TBC) scanning in data to all boards in the system 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 system.

9 Booting

The CUB contains system booting functions. It connects to the system teststation to provide a means of initialization, diagnostic testing, and remote booting of the system.

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 the system, all controllers 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 identification number of each processor is read from a CSR. The registers can only be cleared by software.
- 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 disabled.
- 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.

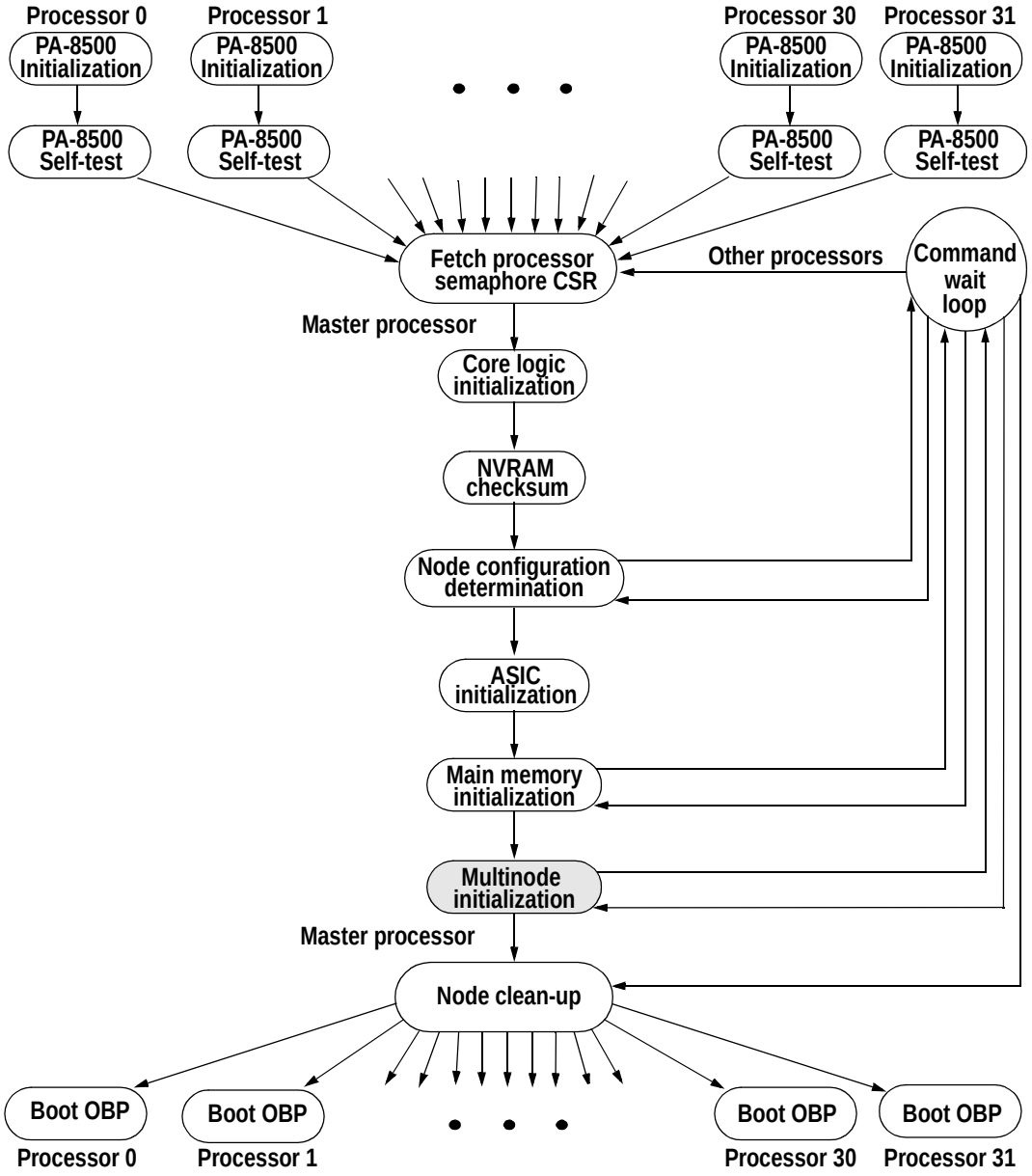
Power-On Self Test routine

When the system first powers up, all processors and supporting hardware must be initialized before the system proceeds with booting.

Upon power up, POST begins executing and brings up the system from an indeterminate state and then executes OBP. POST determines the system hardware configuration before running OBP. If POST encounters an error during initialization, it passes the appropriate error code to an LCD.

Figure 74 shows how POST initializes the processors up to booting of OBP.

Figure 74 POST program flow



Basic processor initialization and selftest

Upon reset, all processors and caches are initialized. If the NVRAM selftest variable is set, then all processors execute selftests.

Each processor determines its identification (ID) from the PAC. Also, each processor fetches the Processor Semaphore register in the PUC. Because register requests are queued, one processor will fetch this CSR before the others and becomes the booting, or *monarch*, processor. All others go into an idle loop, waiting for commands. The booting processor continues executing POST code from the EEPROM.

Core logic initialization

The core logic contains SRAM and DUARTs that support external terminal connection for the console and the LCD panel. POST initializes the SRAM, DUARTs, and all controller CSRs in the system.

Checksum verification of the core logic NVRAM

The PUC EEPROM contains the POST, OBP, system diagnostics code, and the spp_pdc code. In addition, these four routines use shared data structures in NVRAM that are validated by POST by reading and comparing checksums embedded in the structures.

System configuration determination

POST determines which and how many controllers 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 controller (ASIC) that does not respond to any CSR access is considered to be failed.

System ASIC initialization

POST sets every system controller (ASIC) to a known state. The state is based both on configuration parameters and the current hardware configuration.

PACs are reported in the PUC PAC-Exist register.

MACs and SAGAs are reported in the PAC Configuration register.

RACs are always all present (they are not sensed).

System main memory initialization

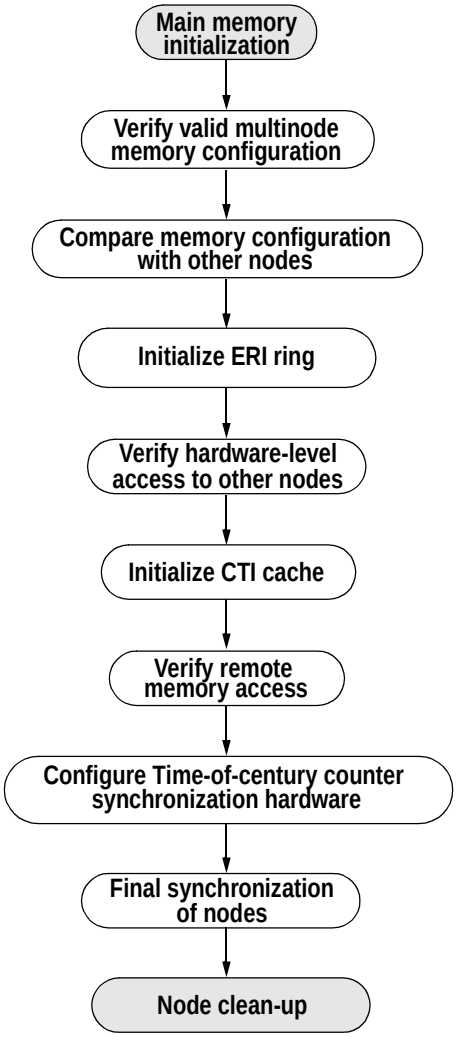
The processor reads the node ID from the COP EEPROM and loads it into the node_ID field of the System Configuration register.

Next, the monarch processor determines the memory configuration for all MACs. It determines the size, population, and installation of each DIMM on a memory board. The monarch processor assigns available processors to initalize the EMBs, and initializes memory and tags in parallel.

Multinode initialization

For multinode systems, POST initializes all other nodes after node 0. The task is shaded in Figure 74, because it is shown in greater detail in Figure 75. POST first verifies the system is configured for multinode operation. It then verifies that the memory configuration in each node is compatible. After POST initializes the CTI rings and cache in each node, it verifies that it can access memory in each node. POST then synchronizes the Time-of-Century counters and nodes.

Figure 75 POST multinode initialization flow



System clean up and OBP boot process

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.

HP-UX bootup

Once each processor in the system has completed initialization and selftest, it loads and executes OBP. The following is the sequence of events for booting the system starting with loading OBP (for every processor) and finishing with the system ready for use:

- The processor loads OBP—After initialization and selftest, each processor loads and begins executing OBP. OBP transfers its ROM image to SRAM, initializes the virtual mode, and turns on translation.
- OBP builds its device tree—It probes the system hardware.
- The processor loads spp_pdc from flash RAM—This firmware is layered over OBP and provides interface between OBP and the HP-UX kernel. spp_pdc must be loaded before OBP can perform any boot functions.
- OBP loads system boot loader—It opens the boot disk, loads a special system loading program, and closes boot disk.
- The processor executes spp_pdc—This firmware layer must be executing so that OBP can complete booting the system.
- The processor executes system boot loader—The loader starts in physical mode (32 bits) and performs the following tasks:
  - Relocates itself
  - Opens PCI devices through spp_pdc

    When spp_pdc calls OBP to perform PCI I/O transfers, OBP must turn on its virtual mode and then turn virtual mode off again when it returns control to spp_pdc. This means all buffers must already be equivalently mapped in OBP's virtual mode page tables.
  - Reads in the kernel using spp_pdc for I/O
  - Starts the kernel
- The kernel reads /etc/ioconfig—spp_pdc opens the boot device for I/O.
- The kernel boot I/O completes.

- spp_pdc closes the boot device.
- OBP turns off Virtual Mode—It removes PCI CSR virtual mode mapping.
- The kernel switches to its virtual mode
- The kernel relocates the system boot loader
- Kernel continues booting in one of two ways: normal boot and install boot.

Normal booting

For normal booting, the following additional tasks are performed:

- OBP loads the special system kernel loader into memory.
- The kernel loader loads /stand/vmunix or user-specified kernel.
- The kernel uses kernel loader for boot I/O to load /etc/ioconfig. Booting is complete.

Install booting

For install booting, the following sequence is performed:

- OBP loads the kernel loader into memory.
- The kernel loader loads `VINSTALL` LIF image.
- `VINSTALL` uses the kernel loader for boot I/O to load ramdisk `VINSTALLFS`
- `VINSTALL` completes booting, and the cold install GUI opens for the user.

10 Error handling

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. The cause 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 in nature, not being the result of a permanent hardware failure. The effect of these errors can many times be contained while the system continues 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 disabling 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 can be aborted by the operating system. If the process is a kernel or server, an operating system 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. 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 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, an indication that the data 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 can not be notified of the corruption.

A hard error is sent to the MUC and then the PUC may generate an interrupt, HPMC, or transfer of control (TOC) to one or all processors in the system. See “PUC interrupt logic” on page 114 for more information on PUC interrupts.

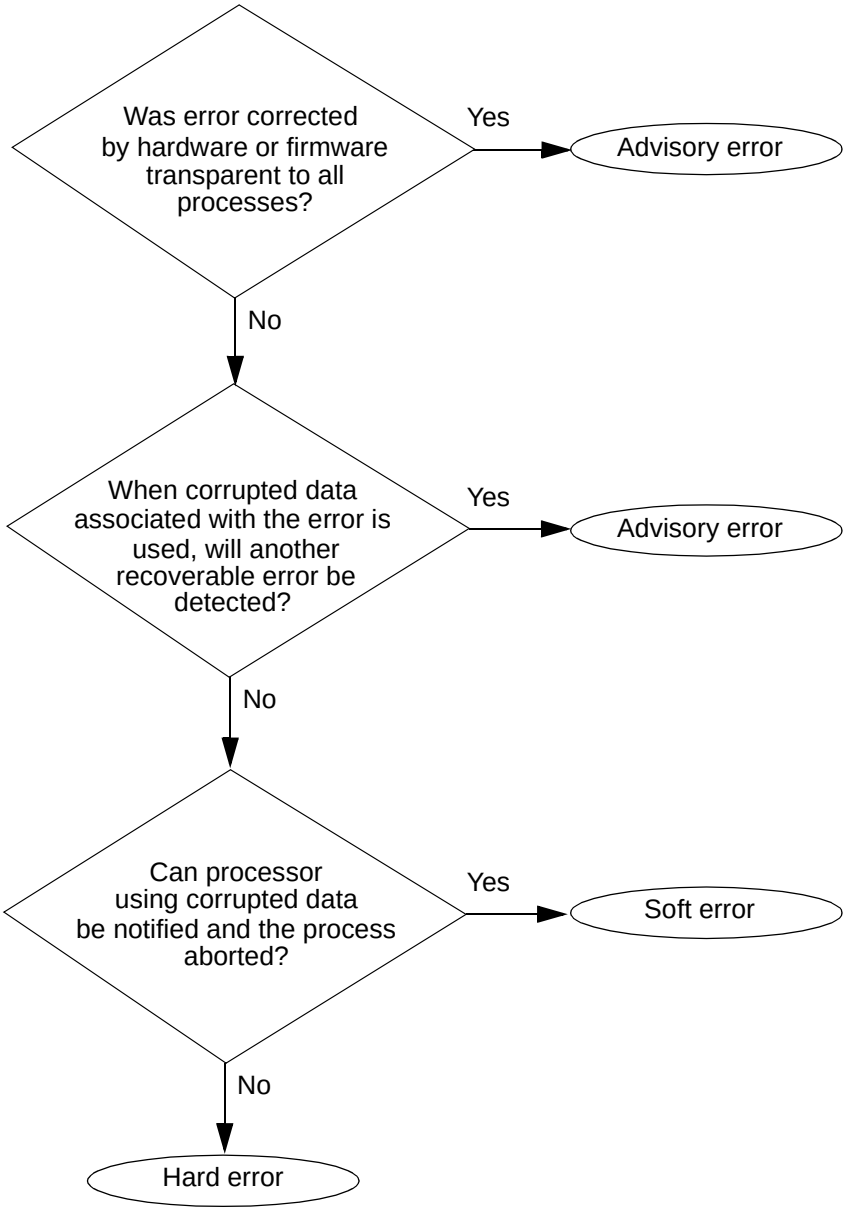
All error CSRs are locked (or frozen) when a hard error is detected so that additional errors caused by the propagation of the hard error are not logged. Usually, only one controller fails in the error condition. If more than one controller detects a hard error, however, the state in the MUC and clock phase information in each chip indicate the chip that first detected the hard error.

When a hard error occurs, the system must be rebooted. The failed hardware 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.

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.

Figure 76 illustrates the characteristics of the three error types: soft, advisory, and hard.

Figure 76 Determining error types



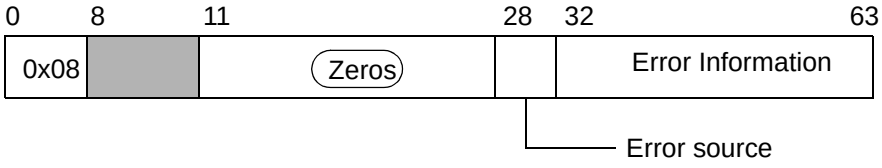
Error responses

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 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 SAGA (in the case of an I/O request). The processor logs the error information in its SADD_LOG register and the SAGA logs the error information in its internal CSRs. Error recovery software can then read these CSRs and take appropriate action.

Figure 77 shows the format for the processor SADD_LOG after an error response.

Figure 77 SADD_LOG after error response



The *Error Source* field (bits 28:31) indicates one of the sources as shown in Table 47.

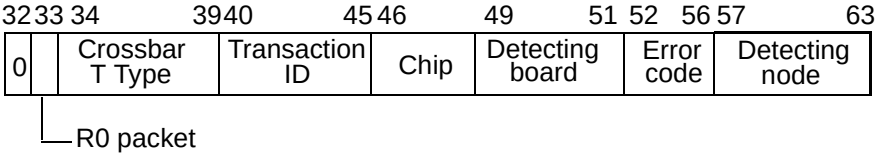
Table 47 SADD_LOG error source field definition

Error source field	Source
0x0	Hyperplane crossbar 0 input
0x1	Hyperplane crossbar 1 input
0x2	Hyperplane crossbar 0 output
0x3	Hyperplane 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 Hyperplane crossbar inputs (that is, external to the PAC), the Error Information field (bits 32:63) contains the information shown in Figure 78.

Figure 78 PAC error response information when received from either crossbar input



The subfields and bits 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 Hyperplane crossbar response.
- *Transaction ID* 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 node* field (bits 57:63)—Specifies which node that sent the error response.

If the error source field indicates the response was from the Hyperplane crossbar output logic (internal to the PAC), 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 the PAC Runway bus input logic, 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 *Error information* field (bits 32:63) contains the CSR error code.

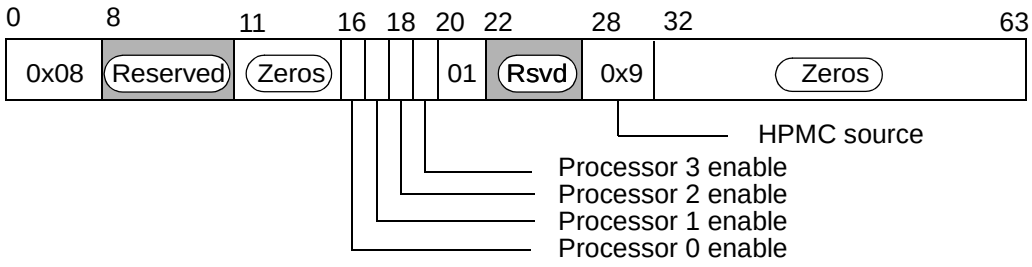
Hard error logging

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 a high-priority machine check. 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 79.

Figure 79 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, 1, 2 and 3 enable fields (bits 16:19)—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 of the following registers:

- Error Cause
- Error Address
- Error Information
- Error Configuration

These registers are accessible through load and store instructions and diagnostic scan.

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.

The 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.

The Error Address register contains the address of any error that can be isolated to a specific address. This register is loaded or held under the same error conditions as the 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.

The 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.

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. 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 Hyperplane crossbar destined for a processor, it sends a directed error, followed (in most cases) by a dummy response 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. There is no Error Address register in the PAC. When the PAC receives an error response destined for the SAGA, it forwards it as an error response packet on the SAGA 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 Hyperplane 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 by way of 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 Hyperplane crossbar packets between the PAC and the MAC, checking 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 contains memory error detection and correction hardware that corrects and logs single-bit errors. When a single-bit error occurs, an interrupt is sent to a designated processor. Single-bit errors on memory data persist in memory and must be scrubbed by error handling software. When the MAC 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 multinode systems, hypernodes connect through the X- and Y-ring CTIs. The STAC 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 STAC 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.

A CSR map

Table 48 lists the CSRs in the V2500 server.

Table 48 V2500 server CSR map

40-Bit physical address	CSR space	CSR register name
0xF0 0000 0000 - 0xF0 FFFF FFFF	Core Logic	
0xF0 xx00 0000 - 0xF0 xx7F FFFF	PDC EEPROM	V2500 implements 4 MBytes (0x0-0x1FFFFF)
0xF0 xx80 0000 - 0xF0 xxEF FFFF	SRAM	V2500 implements 256 KBytes (0x820000-0x81FFFF)
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
0xF0 xxC1 0000 - 0xF0 xxCF FFFF	MUC	MUC CSR space
0xF0 xxC1 0000	Half Word Access	Processor Report 0 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

40-Bit physical address	CSR space	CSR register name
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 xxC1 0034		General Semaphore 0 Register
0xF0 xxC1 0038		Processor Report 1 Register
0xF0 xxC1 003C		General Semaphore 1 Register
0xF0 xxC1 0040		General Semaphore 2 Register
0xF0 xxC1 0044		General Semaphore 3 Register
0xF0 xxD0 0000 - 0xF0 xxD2 FFFF	DS1646	Nonvolatile Ram and Real Time Clock
0xF0 xxD1 0000 - 0xF0 xxD1 FFF7	Byte, Half, Word or Double Word Access	Nonvolatile SRAM
0xF0 xxD1 FFF8		RTC Control register
0xF0 xxD1 FFF9		RTC Seconds register
0xF0 xxD1 FFFA		RTC Minutes register
0xF0 xxD1 FFFB		RTC Hour register
0xF0 xxD1 FFFC		RTC Day register
0xF0 xxD1 FFFD		RTC Date register
0xF0 xxD1 FFFE		RTC Month register

40-Bit physical address	CSR space	CSR register name
0xF0 xxD1 FFFF		RTC Year register
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

40-Bit physical address	CSR space	CSR register name
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
0xF0 xxD3 006C		Temporary Buffer Word Count 0 register
0xF0 xxD3 0070		Temporary Buffer Word Count 1 register
0xF0 xxD3 007C		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

40-Bit physical address	CSR space	CSR register name
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 Port 0	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
0xF0 xxD4 6010		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 Port 1	Serial Port 0
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

40-Bit physical address	CSR space	CSR register name
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 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	Node-local Nonaccelerated I/O	
0xF8 0000 0000 - 0xFB FFFF FFFF	Node-local Accelerated I/O	
0xFC 0000 0000 - 0xFC 003F FFFF	Node 0	Globally accessible CSRs physically residing on Node #0
0xFC 0000 0000 - 0xFC 0000 FFFF	PAC 0	PAC CSR space
0xFC 0000 0000	PAC 0, Page 0	System Configuration register
0xFC 0000 0008	Double-word Access	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

40-Bit physical address	CSR space	CSR register name
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 0200		PAC CTI Cache Hit Rate register
0xFC 0000 0300		Time-of-Century Configuration register
0xFC 0000 0308	PAC0, Page 2	Time-of-Century Count register
0xFC 0000 2000	PAC 0, Page 2 Processor 0 Specific	Processor 0 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		CPU #0 CTI Cache Flush Global Address
0xFC 0000 2070		CPU #0 CTI Cache Prefetch for Read
0xFC 0000 2078		CPU #0 CTI Cache Prefetch for Write
0xFC 0000 2200		Processor 0 Performance Monitor Memory Access Count 0 register

40-Bit physical address	CSR space	CSR register name
0xFC 0000 2208		Processor 0 Performance Monitor Memory Access Count 1 register
0xFC 0000 3000	PAC 0, Page 3 Processor 1 Specific	Processor 1 Configuration register
0xFC 0000 3010	Double Word Access	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		CPU #1 CTI Cache Flush Global Address
0xFC 0000 3070		CPU #1 CTI Cache Prefetch for Read
0xFC 0000 3078		CPU #1 CTI Cache Prefetch for Write
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 6000	SPAC 0, Page 6 Processor 2 Specific	Processor 2 Configuration Register
0xFC 0000 6010		Processor 2 CSR Operation Context Register
0xFC 0000 6018		Processor 2 CSR Operation Address Register

40-Bit physical address	CSR space	CSR register name
0xFC 0000 6020		Processor 2 Fetch and Increment Address
0xFC 0000 6028		Processor 2 Fetch and Decrement Address
0xFC 0000 6030		Processor 2 Fetch and Clear Address
0xFC 0000 6038		Processor 2 Non-Coherent Read Address
0xFC 0000 6040		Processor 2 Non-Coherent Write Address
0xFC 0000 6060		Processor 2 Coherent Increment Address
0xFC 0000 6068		CPU #1 CTI Cache Flush Global Address
0xFC 0000 6070		CPU #1 CTI Cache Prefetch for Read
0xFC 0000 6078		CPU #1 CTI Cache Prefetch for Write
0xFC 0000 6200		Processor 2 Processor Read Latency Register
0xFC 0000 6208		CPU2 Performance Monitor Configuration
0xFC 0000 7000	SPAC 0, Page 7 Processor 3 Specific	Processor 3 Configuration Register
0xFC 0000 7010	Double Word Access	Processor 3 CSR Operation Context Register
0xFC 0000 7018		Processor 3 CSR Operation Address Register
0xFC 0000 7020		Processor 3 Fetch and Increment Address
0xFC 0000 7028		Processor 3 Fetch and Decrement Address
0xFC 0000 7030		Processor 3 Fetch and Clear Address
0xFC 0000 7038		Processor 3 Non-Coherent Read Address
0xFC 0000 7040		Processor 3 Non-Coherent Write Address
0xFC 0000 7060		Processor 3 Coherent Increment Address
0xFC 0000 7068		Processor 3 CTI Cache Flush Global Address

40-Bit physical address	CSR space	CSR register name
0xFC 0000 7070		Processor 3 CTI Cache Prefetch for Read
0xFC 0000 7078		Processor 3 CTI Cache Prefetch for Write
0xFC 0000 7200		Processor 3 Read Latency Register
0xFC 0000 7208		CPU3 Performance Monitor Configuration
0xFC 0001 0000 - 0xFC 0001 FFFF	SAGA	SAGA CSR space
0xFC 0001 0000	SAGA 0, Page 0	System Configuration register (Reserved on SAGA)
0xFC 0001 0008	Double Word Access	SAGA Chip Configuration register
0xFC 0001 0010		PCI Master Configuration register
0xFC 0001 0018		PCI Master Status register
0xFC 0001 0020		SAGA Channel Builder register
0xFC 0001 0080		SAGA Error Cause register
0xFC 0001 0088		SAGA Error Configuration register
0xFC 0001 0090		SAGA Error Address register
0xFC 0001 0098		SAGA Error Info register
0xFC 0001 00A0		SAGA Interrupt Configuration register
0xFC 0001 00A8		SAGA Interrupt Source register
0xFC 0001 00B0		SAGA 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

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		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	CPU 0	Node 0, CPU 0 CSR Space
0xFC 0002 0000	Word or Double Word Access	External Interrupt Request Register
0xFC 0003 0000 - 0xFC 0003 FFFF	CPU 1	Node 0, CPU 1 CSR Space
0xFC 0003 0000	Word or Double Word Access	External Interrupt Request Register
0xFC 0006 0000 - 0xFC 0006 FFFF	CPU 2	Node 0, CPU 2 CSR Space
0xFC 0006 0000	Word or Double Word Access	External Interrupt Request Register
0xFC 0007 0000 - 0xFC 0007 FFFF	CPU 3	Node 0, CPU 3 CSR Space

40-Bit physical address	CSR space	CSR register name
0xFC 0007 0000	Word or Double Word Access	External Interrupt Request Register
0xFC 0004 0000 - 0xFC 0004 FFFF	MAC	MAC 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 CTI Cache Memory Region Register
0xFC 0004 0038		Unprotected CTI 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 Error Configuration register 0
0xFC 0004 00A0		MAC Error Configuration register 1
0xFC 0004 00B0		MAC Error Interrupt register
0xFC 0004 0200		Diagnostic Address register
0xFC 0004 0208		Diagnostic Data register
0xFC 0004 0210		Diagnostic Data Register
0xFC 0004 0218		Diagnostic Read Memory Tag Address
0xFC 0004 0220		Diagnostic Read Memory Data address
0xFC 0004 0228		Diagnostic Write Memory Data address
0xFC 0004 0230		Diagnostic Read Memory ECC address
0xFC 0004 0238		Diagnostic Write Memory ECC address

40-Bit physical address	CSR space	CSR register name
0xFC 0004 0240		Diagnostic Initialize Memory address
0xFC 0004 0248		Diagnostic Scrub Memory address
0xFC 0005 0000 - 0xFC 0005 FFFF	Node 0, TAC 0	Node 0, TAC 0 CSR Space
0xFC 0005 0000	TAC 0, Page 0	System Configuration Register
0xFC 0005 0008	Double Word Access	TAC Chip Configuration Register
0xFC 0005 0010		TAC Ring Configuration Register
0xFC 0005 0018		TAC Chip Status 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		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	Hyperplane crossbar Port 1	
0xFC 0010 0000 - 0xFC 0017 FFFF	Hyperplane crossbar Port 2	
0xFC 0018 0000 - 0xFC 001F FFFF	Hyperplane crossbar Port 3	
0xFC 0020 0000 - 0xFC 0027 FFFF	Hyperplane crossbar Port 4	

40-Bit physical address	CSR space	CSR register name
0xFC 0028 0000 - 0xFC 002F FFFF	Hyperplane crossbar Port 5	
0xFC 0030 0000 - 0xFC 0037 FFFF	Hyperplane crossbar Port 6	
0xFC 0038 0000 - 0xFC 003F FFFF	Hyperplane crossbar Port 7	
0xFC 1000 0000 - 0xFC 103F FFFF	Node 1	
0xFC 2000 0000 - 0xFC 203F FFFF	Node 2	
0xFC 3000 0000 - 0xFC 303F FFFF	Node 3	
	.	
	.	
	.	
0xFC F000 0000 - 0xFC F03F FFFF	Node 15	

Index

D

W