VAX 7000/10000
KA7AA CPU Technical Manual

Order Number EK—KA7AA—TM.001

The KA7AA is a VAX CPU module designed for the LSB platform. It is based on the NVAX+ microprocessor and is used in the VAX 7000 and VAX 10000 computer systems. It supports up to seven MS7AA memory modules in a uniprocessor configuration and one IOP module per system. Used in a single-processor system, the KA7AA module achieves a minimum scalar performance equivalent to that of more than 20 VAX 11/780 systems. A multiprocessor system supports up to six KA7AA CPU modules.

First Printing, December 1992

FCC NOTICE: The equipment described in this manual generates, uses, and may emit radio frequency energy. The equipment has been type tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such radio frequency interference when operated in a commercial environment. Operation of this equipment in a residential area may cause interference, in which case the user at his own expense may be required to take measures to correct the interference.

Contents

Chapter 3 Cache Subsystem

Chapter 4 LSB Bus Interface

Chapter 5 Console Overview

Chapter 6 I/O Operations

Chapter 7 CPU Module Registers

Chapter 8 Initialization

Chapter 9 Error Handling

Examples

Figures

Tables

Preface

Intended Audience

This manual is written for developers of system and application software based on the LSB platform of VAX computer systems. It assumes machine level programming knowledge and familiarity with the OpenVMS VAX operating system.

Document Structure

The material is presented in nine chapters.

- Chapter 1, **CPU Module Overview,** presents an overall introduction to the KA7AA CPU module.
- Chapter 2, **CPU Chip,** describes the functions and operations of the central processor of the KA7AA CPU module. It discusses such topics as addressing, memory management, exceptions and interrupts, functional partitions of the CPU chip, and internal processor registers.
- Chapter 3, **Cache Subsystem,** describes the elements and operations of the cache hierarchy, which includes the virtual instruction cache, the primary cache, and the backup cache.
- Chapter 4, **LSB Bus Interface,** describes the functions and operations of the LEVI gate arrays that provide the CPU module interface to the LSB bus. It discusses processor- initiated and LSB bus- initiated transactions, LEVI address and data paths, and the LEVI controllers.
- Chapter 5, **Console Overview,** gives a brief description of the various elements that comprise the console. It also describes the Gbus registers, which perform console control, diagnostic, and interrupt- related functions.
- Chapter 6, **I/O Operations,** describes the mailbox data structure, the operation of the mailbox, interrupt handling, and the I/O registers.
- Chapter 7, **CPU Module Registers,** lists the LSB required and CPU-specific registers, and provides bit- level functional descriptions of each register.
- Chapter 8, **Initialization,** gives an overview of the CPU module initialization, describes the methods and process of initialization, system configuration, and bootstrapping of the operating system.
- Chapter 9, **Error Handling,** describes how the KA7AA module handles various types of errors. It discusses error analysis and recovery, machine check exceptions, and sources of error interrupts. Parse trees included in this chapter help isolate errors to particular causes.

Conventions Used in This Document

Results and Operations

Results of operations termed UNPREDICTABLE must not be used by software.

Operations termed UNDEFINED do not cause the processor to hang, that is, reach a state from which there is no transition to a normal state of instruction execution. Nonprivileged software cannot invoke UNDEFINED operations.

Register and Bit Designations

Certain conventions are followed in register descriptions and in references to bits and bit fields:

- Registers are referred to with their mnemonics, such as **VMAR register.** The full name of a register (for example, **VIC Memory Address Register)** is spelled out only at the top of the register description page, or when the register is first introduced.
- Acronyms are used in register description tables to indicate the access type of the bit(s).

Acronym	Access Type
RC	Read to clear. The value is written by hardware and remains unchanged until read by software or microcode.
RO	Read only. May be read by software, microcode, or hardware. Written by hardware. Software or microcode writes are ignored.
R/W	Read/write. May be read and written by software, microcode, or hardware.
R0	Reads as zero. Read only. Writes are ignored.
WO	Write only. May be written by software or microcode. It is read by hardware. Reads by software or microcode return an unpredictable value.
W1C	Write 1 to clear. The value may be read by software or microcode. Software or microcode writes of 1 to the position cause hardware to clear the bit. Software or microcode writes of 0 do not modify the state of the bit.
W1S	Write 1 to set. May be read and written by software, microcode, or hardware. Set by software or microcode with a write of 1.

- Bits and fields are enclosed in angle brackets. For example, **bit <31>; bits <31:16>.** For clarity of reference, bits are usually specified by their numbers or names enclosed in angle brackets adjacent to the register mnemonic, such as **VMAR<31:11>** or **VMAR<ADDR>,** which are equivalent designations.

- When the value of a bit position is given explicitly in a register diagram, the information conveyed is as follows:

Bit Value	Meaning
0	Reads as zero; ignored on writes.
1	Reads as one; ignored on writes.
X	Does not exist in hardware. The value of the bit is UNPREDICTABLE on reads and ignored on writes.

- Fields (in registers or data structures) noted as must be zero (MBZ) must never be filled by software with a nonzero value. If the processor encounters a nonzero value in an MBZ field, a Reserved Operand Exception occurs.
- Fields (in registers or data structures) noted as should be zero (SBZ) should be filled by software with a zero value. A nonzero value in an SBZ field produces UNPREDICTABLE results.
- The entry in the Type column of a register description table may include the initialization values of the bits. For example, entry "R/W, 0" indicates a read/write bit that is initialized to zero.

Documentation Titles

Table 1 lists the books in the DEC 7000/10000 and VAX 7000/10000 documentation sets.

Table 1 DEC 7000/10000 and VAX 7000/10000 Documentation

Title	7000 Systems Order Number	10000 Systems Order Number
Installation Kit	EK–7000B–DK	EK–1000B–DK
Site Preparation Guide	EK–7000B–SP	EK–1000B–SP
Installation Guide	EK–700EB–IN	EK–100EB–IN
Hardware User Information Kit	EK–7001B–DK	EK–1001B–DK
Operations Manual	EK–7000B–OP	EK–1000B–OP
Basic Troubleshooting	EK–7000B–TS	EK–1000B–TS
Service Information Kit—VAX 7000	EK–7002A–DK	EK–1002A–DK
Platform Service Manual	EK–7000A–SV	EK–1000A–SV
System Service Manual	EK–7002B–SV	EK–1002A–SV
Pocket Service Guide	EK–7000A–PG	EK–1000A–PG
Advanced Troubleshooting	EK–7001A–TS	EK–1001A–TS

Table 1 DEC 7000/10000 and VAX 7000/10000 Documentation (Continued)

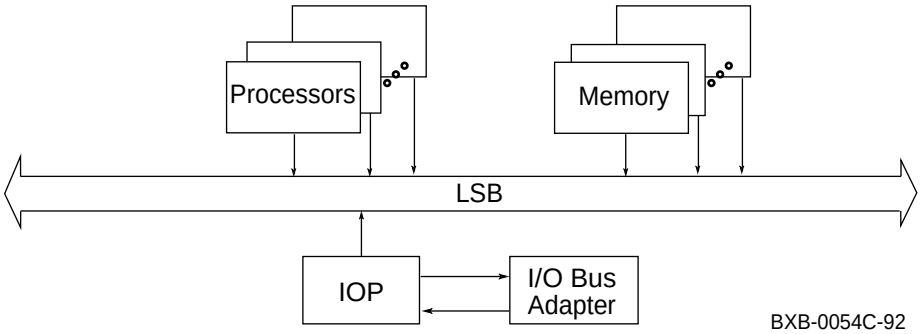
Title	7000 Systems Order Number	10000 Systems Order Number
Reference Manuals		
Console Reference Manual	EK–70C0B–TM	
KA7AA CPU Technical Manual	EK–KA7AA–TM	
KN7AA CPU Technical Manual	EK–KN7AA–TM	
MS7AA Memory Technical Manual	EK–MS7AA–TM	
I/O System Technical Manual	EK–70I0A–TM	
Platform Technical Manual	EK–7000A–TM	
Upgrade Manuals		
KA7AA CPU Installation Card	EK–KA7AA–IN	
KN7AA CPU Installation Card	EK–KN7AA–IN	
MS7AA Memory Installation Card	EK–MS7AA–IN	
KZMSA Adapter Installation Guide	EK–KXMSX–IN	
DWLMA XMI PIU Installation Guide	EK–DWLMA–IN	
DWMBB VAXBI PIU Installation Guide	EK–DWMBB–IN	
H7237 Battery PIU Installation Guide	EK–H7237–IN	
H7263 Power Regulator Installation Card	EK–H7263–IN	
Futurebus+ PIU Installation Guide	EK–DWLAA–IN	
BA654 DSSI Disk PIU Installation Guide	EK–BA654–IN	
BA655 SCSI Disk and Tape PIU Installation Guide	EK–BA655–IN	
Removable Media Installation Guide	EK–TFRRD–IN	

Chapter 1

CPU Module Overview

The KA7AA CPU module is designed around the NVAX+ microprocessor chip (P/N DC262) and communicates with main memory and I/O subsystems by way of the LSB bus. Figure 1- 1 shows how the KA7AA CPU module fits in a VAX computer system based on the LSB bus.

Figure 1-1 Block Diagram of a VAX 7000 or VAX 10000 System



The CPU module supports the VAX base instruction group of 242 instructions and associated data types, an address space of 4 gigabytes, and provides full memory management.

A floating- point accelerator and a three- level hierarchical instruction and data cache structure allow the CPU module to achieve a minimum scalar single- CPU performance of more than 20 times that of a VAX 11/780 on an average workload.

The CPU module includes VAX- compatible macrocoded console firmware. Microcode and on- board ROMs permit booting from supported devices and provide self- test diagnostics on power- up.

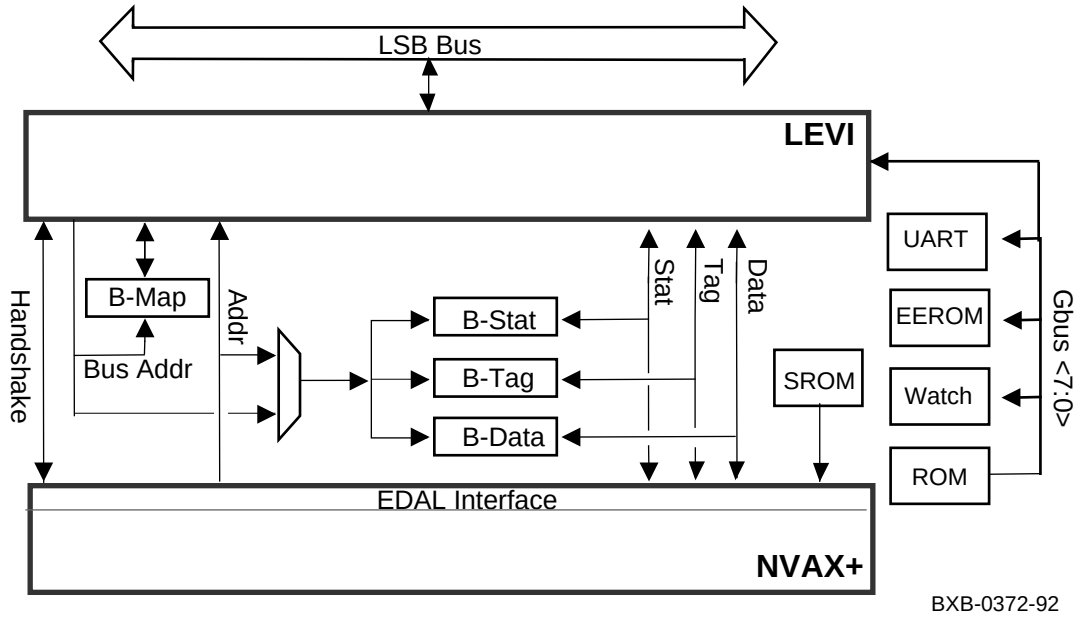
A multiprocessor VAX 7000 or VAX 10000 system can be configured with up to six CPU modules. All backplane slots except slot 8, which is dedicated to the IOP module, can be used interchangeably with memory modules, allowing an array of configurations tailored to specific application requirements. It is strongly recommended, however, that the first CPU module be placed in slot 0.

The KA7AA CPU module is comprised of three major sections:

- CPU chip (NVAX+)
- Backup cache (B- cache)
- LSB interface (LEVI)

Figure 1- 2 shows the major sections of the CPU module.

Figure 1- 2 KA7AA CPU Module Block Diagram



1.1 NVAX+ CPU Chip

The NVAX+ processor is a single- chip macropipelined implementation of the VAX architecture and the VAX base instruction group. It is partitioned into five functional units, or boxes, as listed in Table 1- 1.

Table 1- 1 NVAX+ CPU Chip Functional Units

NVAX+ Unit	Functional Description
Ibox	**Instruction box.** Prefetches, decodes, parses, and queues VAX instructions for execution.
Ebox	**Execution box**. Executes, together with the microsequencer, instructions received from the Ibox.
Fbox	**Floating-point accelerator box.** Executes floating- point and integer multiply VAX instructions received from the Ebox.
Mbox	**Memory management box.** Performs all virtual to physical address translations, for the Ibox and Ebox. The Mbox also handles Cbox requests for cache fills and invalidates for the primary cache.
Cbox	**Cache control box**. Initiates access to the backup cache (B- cache) and issues memory requests.

The NVAX+ CPU chip includes:

- Translation buffer: 96 entries, fully associative
- Virtual instruction cache (VIC): Two Kbytes, direct- mapped, virtually addressed, I- stream only
- Primary cache (P- cache): Eight Kbytes, 2- way set associative, physically addressed, write through, mixed I- stream and D- stream, 32- byte block and fill size
- Control store: 1600 61- bit microwords
- EDAL interface: Cache and memory subsystem interface

The macroinstruction pipelined design of the NVAX+ CPU chip allows significant parallel processing. NVAX+ pipelines macroinstruction decode and operand fetch with macroinstruction execution. Pipeline efficiency is increased by queuing up instruction information and operand values for later use by the execution unit. Thus, when the macropipeline is operating smoothly, the instruction unit (Ibox), which parses instructions and fetches operands, is running several macroinstructions ahead of the execution unit (Ebox). Branch predictions allow quicker execution of loops in software. Outstanding writes to registers or memory locations are kept in a scoreboard to ensure that data is not read before it has been written.

1.2 Backup Cache (B- Cache)

The external backup cache (B- cache) is a 4- Mbyte superset of the P- cache. It is a physically addressed, direct mapped, write back, mixed I- stream and D- stream cache with a block and fill size of 64 bytes. It consists of three sets of RAMs:

B- data
B- tag
B- stat

Each block of data (B- data) has a tag (B- tag) and three status bits (B- stat) associated with it. The status bits are Valid, Dirty, and Shared.

The B- cache has controllable data RAM access time. The CPU module supports cache access times of 2, 3, and 4 times the NVAX+ chip cycle time.

1.3 LSB Interface (LEVI)

The interface to the LSB bus is called LEVI, which consists of two chips: LEVI- A and LEVI- B. LEVI- A contains most LSB required registers, implements all command execution, LSB arbitration, and B- cache manipulation functions. It also contains a P- cache backmap (P- map) to allow the CPU to do invalidate filtering and to make intelligent update vs. invalidate decisions in response to LSB write traffic. LEVI- A uses an external RAM structure to implement a backmap of the B- cache to filter bus traffic from the B- cache while still maintaining cache coherence.

LEVI- B completes the 128- bit data path between the NVAX+ and the LSB bus. An 8- bit communication bus between LEVI- A and LEVI- B provides a path that allows LEVI- B to perform look- aside ECC checking on incoming memory traffic.

Chapter 2

CPU Chip

The NVAX+ CPU chip is the central processor of the KA7AA CPU module. It executes the VAX base instruction group and provides full VAX memory management. This chapter provides a summary of the architectural features and functional elements of the NVAX+ CPU chip. Sections include:

- Data types
- Instruction set
- Address space
- Memory management
- Exceptions and interrupts
- System control block
- Process structure
- Functional partitions
- General purpose registers
- Internal processor registers

For more information on some of these topics, consult the *VAX Architecture Reference Manual*.

2.1 Data Types

The NVAX+ CPU chip supports the following subset of VAX data types:

Byte
Word
Longword
Quadword
Variable- length bit field
Character string
Absolute queues
Self- relative queues
D_floating
F_floating
G_floating

The remaining VAX data types (octaword, H_floating, trailing numeric string, leading separate numeric string, and packed decimal string) are supported by macrocode emulation.

2.2 Instruction Set

The NVAX+ hardware supports the following instruction classes:

Integer arithmetic and logical
Address
Variable- length bit field
Control
Procedure call
Miscellaneous
Queue
Character string
Operating system support
D_floating
F_floating
G_floating

A subset of the VAX instruction group is emulated by NVAX+ microcode. See the *VAX Architecture Reference Manual* for detailed descriptions of the VAX instruction set.

2.3

2.4 Address Space

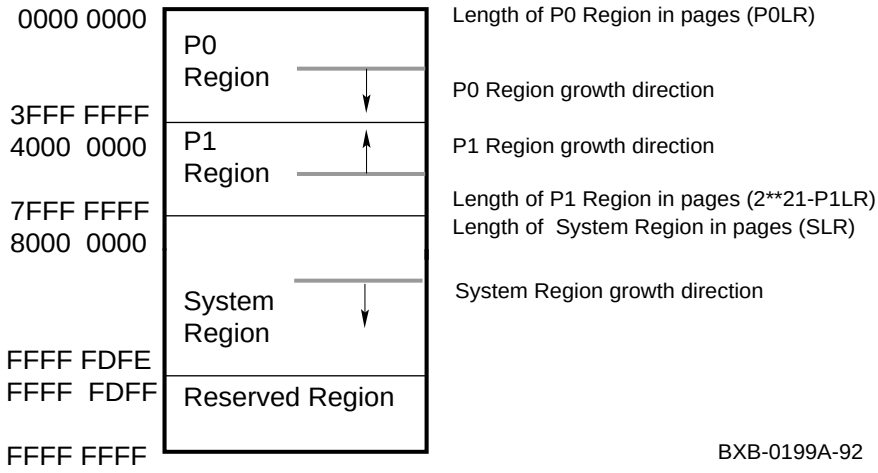
The memory space that a programmer uses is a single logical structure of contiguous pages. It is called “virtual memory” and is referenced in virtual address space. The memory space that the operating system allocates to a process is a structure of noncontiguous pages called “physical memory” and is referenced in the physical address space. The operating system uses the page mapping scheme because in a multiuser and multitasking environment no available block in physical memory may be large enough to accommodate a block of contiguous pages generated in virtual memory.

2.4.1 Virtual Address Space

Software can reference a 32- bit virtual address. This address width allows access to a virtual address space of 4 gigabytes (2**32 locations). The virtual address space is divided into two sections, process space and system space. The process space is further partitioned into two regions, P0 and P1. All the virtual address space except a reserved region in the system space is accessible to software. Figure 2- 1 shows the virtual address space layout.

NOTE: NVAX+ implements the VAX extended system region address space. The extension allows the chip to address 2^{22}-1 pages of system space.

Figure 2- 1 Virtual Address Space Layout



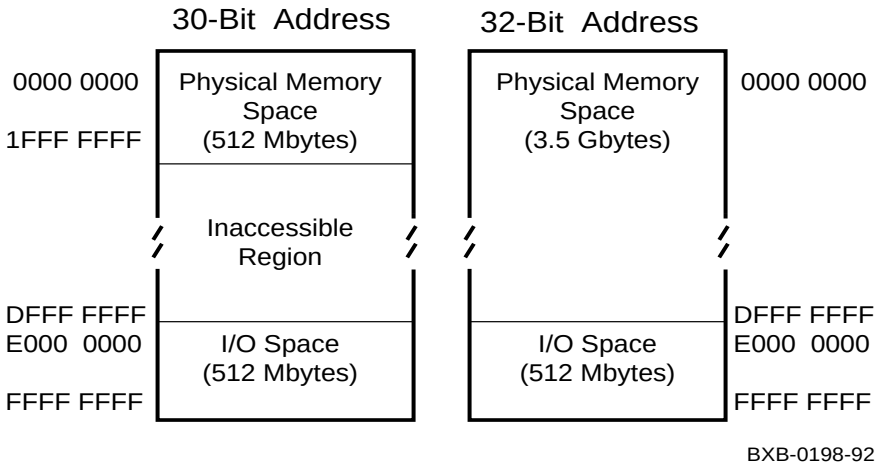
2.4.2 Physical Address Space

The NVAX+ CPU generates 32- bit physical addresses. This address width can access 4 gigabytes of physical address space. Memory space occupies the first seven- eighths (3.5 Gbytes) of the physical address space. I/O space occupies the last one- eighth (512 Mbytes) of the physical address space and can be distinguished from memory space by the fact that bits <31:29> of the physical address are all ones.

The NVAX+ CPU chip can also run in a 30- bit address mode if selected by the programmer. In this mode the NVAX+ CPU chip can reference only

one gigabyte of total address space, evenly divided into memory space and I/O space. Figure 2- 2 shows the physical address space layouts.

Figure 2- 2 Physical Address Space Layouts



During power- up, microcode configures the NVAX+ CPU chip to generate 30- bit physical addresses. Operating system initialization code can reconfigure the CPU module to generate either 30- bit or 32- bit physical addresses by writing to the Mode bit (<0>) in the PAMODE register (IPR231).

2.5 Memory Management

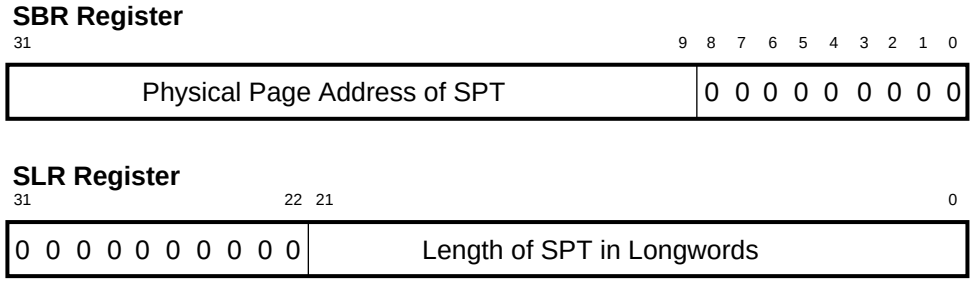
Memory management is the translation of virtual addresses (contiguous page addresses used by the programmer) into physical addresses (noncontiguous page addresses in physical memory space). The central processor extracts the virtual page number (VPN) from the virtual address (bits <29:9>) and generates a page frame number (PFN), which makes up bits <31:9> of the physical address. The physical address is then formed by appending bits <8:0> of the virtual address (address of the byte within the page) to the PFN.

2.5.1 System Space Address Translation

A virtual address in system space is mapped by the system page table (SPT), which is defined by the System Base Register (SBR) and the System Length Register (SLR). The SBR register contains the page- aligned physical address of the SPT. The SLR contains the size of the SPT in longwords, that is, the number of page table entries (PTE). The PTE addressed by the SBR register maps the first page of the system virtual address space, that is, virtual byte address 8000 0000 (hex). Figure 2- 3 shows the SBR and SLR registers.

NOTE: When the CPU module is configured to generate 30- bit physical addresses, SBR<31:30> are ignored.

Figure 2- 3 SBR and SLR Registers



BXB-0178-92

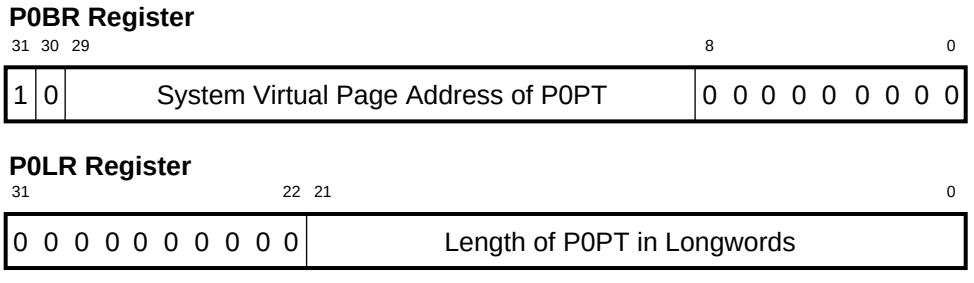
2.5.2 Process Space Address Translation

A virtual address with bit <31> = 0 is an address in the process virtual address space. Process space is divided into two equal sized, separately mapped regions. If virtual address bit <30> = 0, the address is in region P0. If virtual address bit <30> = 1, the address is in region P1.

2.5.2.1 P0 Region Address Translation

The P0 region of the address space is mapped by the P0 page table (P0PT), which is defined by the P0 Base Register (P0BR) and the P0 Length Register (P0LR). The P0BR register contains the system page- aligned virtual address of the P0 page table. The P0LR register contains the size of the P0PT in longwords, that is, the number of PTEs. The PTE addressed by the P0BR register maps the first page of the P0 region of the virtual address space, that is, the virtual byte address 0. Figure 2- 4 shows the P0BR and P0LR registers.

Figure 2- 4 P0BR and P0LR Registers



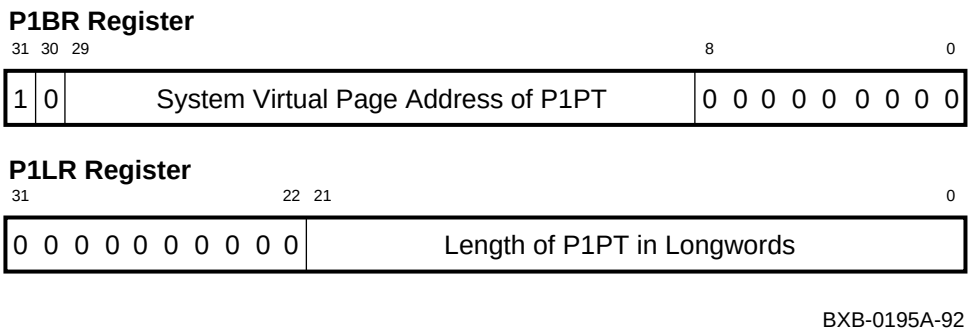
BXB-0195-92

2.5.2.2 P1 Region Address Translation

The P1 region of the address space is mapped by the P1 page table (P1PT), which is defined by the P1 Base Register (P1BR) and the P1 Length Register (P1LR). Because P1 space grows toward smaller addresses, and because a consistent hardware interpretation of the base and length register-

sis desirable, the P1BR register and the P1LR register describe the portion of P1 space that is not accessible. Note that the P1LR register contains the number of nonexistent PTEs. The P1BR contains the page- aligned virtual address of what would be the PTE for the first page of P1, that is, virtual byte address 4000 0000 (hex). The address in the P1BR register is not necessarily an address in system space, but all the addresses of PTEs are in system space. Figure 2- 5 shows the P1BR and P1LR registers.

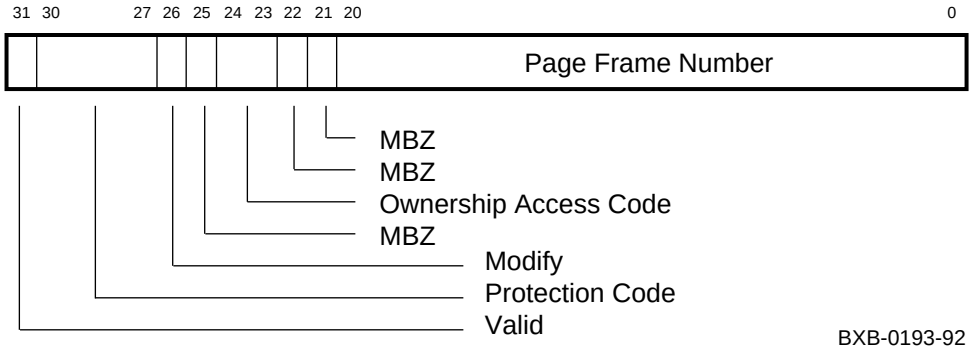
Figure 2- 5 P1BR and P1LR Registers



2.5.3 Page Table Entry Format

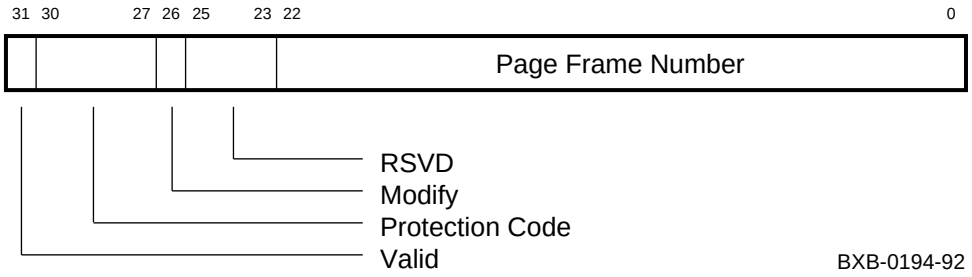
When the CPU module is configured to generate 30- bit physical addresses, it interprets PTEs in the 21- bit PFN format shown in Figure 2- 6.

Figure 2- 6 PTE Format (21- Bit PFN)



If the CPU module is configured to generate 32- bit physical addresses, it interprets PTEs in the 25- bit PFN format shown in Figure 2- 7.

Figure 2- 7 PTE Format (25- Bit PFN)



Note that bits <24:23> of the 25- bit PFN format are ignored by the NVAX+ CPU chip.

2.5.4 Translation Buffer

The NVAX + CPU chip implements a 96- entry, fully associative, fast access translation buffer to minimize memory accesses when repeatedly referencing the same memory pages. Each entry stores a PTE for translating virtual addresses to physical addresses in either process space or system space. The translation buffer caches the PTEs of recent virtual address translations. It is accessed in parallel with each memory reference.

Each translation buffer entry is divided into two parts: a 26- bit tag register and a 29- bit PTE register. The tag register is used to store the VPN of the virtual page that the corresponding PTE register maps. The tag register also contains a valid bit (TBV) that indicates a valid VPN in the tag.

During virtual- to- physical address translation, the contents of the 96 tag registers are compared with the VPN field (bits <31:9>) of the virtual address of the reference. If there is a match with one of the tag registers and the TBV bit indicates a valid entry, the translation buffer sends the correct physical address contained in the corresponding PTE register to the cache. If no match occurs, the PTE that maps the page is fetched from memory through the normal memory access stages (P- cache, B- cache, and then main memory) and the translation buffer is updated by replacing the entry at the location indicated by the replacement pointer.

Translation buffer entries are replaced using a not- last- used (NLU) algorithm. This algorithm guarantees that the replacement pointer is not pointing at the last translation buffer entry to be used. This is accomplished by rotating the replacement pointer to the next sequential translation buffer entry. Both D- stream and I- stream references can cause the NLU algorithm to execute.

2.5.5 Memory Management Control

Three processor registers control memory management:

- Memory Management Enable Register (MAPEN), IPR56
- Translation Buffer Invalidate Single Register (TBIS), IPR58
- Translation Buffer Invalidate All Register (TBIA), IPR57

Bit <0> of the MAPEN register enables memory management if set to a one and disables memory management if written with a zero. The MAPEN register is written to with a Move To Processor Register (MTPR) instruction. It is read with a Move From Processor Register (MFPR) instruction.

The TBIS register controls single- entry invalidation in the translation buffer. Entries that map a particular virtual address are invalidated by writing the virtual address to the TBIS register using the MTPR instruction.

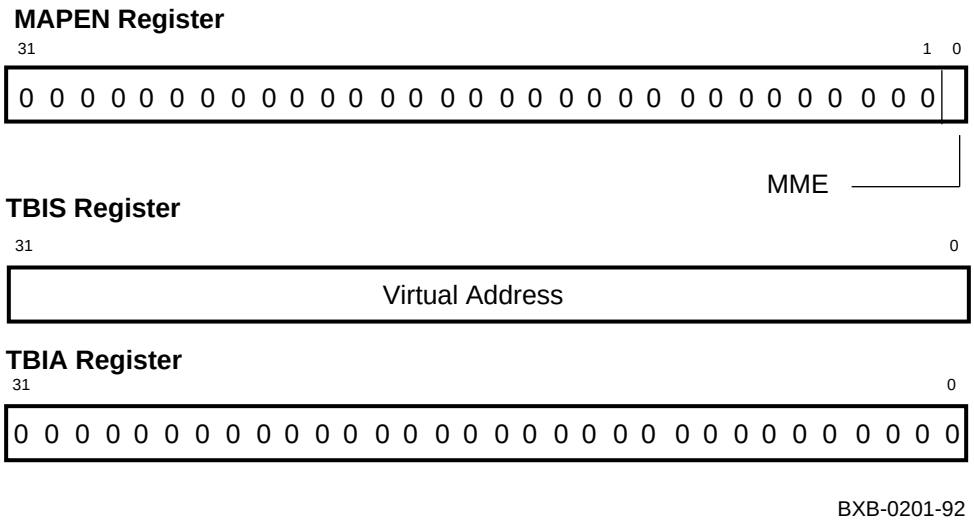
The TBIA register controls total translation buffer invalidation. Writing a zero into the TBIA register invalidates the entire translation buffer. Figure 2- 8 shows the three memory control registers.

CAUTION: All affected process pages must be invalidated in the translation buffer whenever software changes one of the following:

- *A valid PTE for the system or the current process region*
- *A system PTE that maps any part of the current process page table*

Otherwise, address translations point to wrong locations in memory.

Figure 2- 8 Memory Management Control Registers



The base and length of the P0, P1, and S0 page tables are changed by writing the appropriate address or length to the P0BR, P0LR, P1BR, P1LR, SBR, or SLR register. The entire translation buffer is flushed when a change is made to any of these six registers.

When a process context is loaded with the Load Process Context (LDPCTX) instruction, all translation buffer entries that map process space pages are automatically invalidated. System- space mappings are preserved.

To determine if the translation buffer contains a valid translation for a particular virtual page, write a virtual address within that page to TBCHK using an MTPR instruction. If the translation buffer contains a valid translation for the page, the condition code V bit (PSL<1>) would be set.

A full invalidation of the translation buffer, whether performed with an explicit write to the TBIA register, or as an implied clear due to writes to the MAPEN register or any base/length register, resets the NLU pointer to the first location in the translation buffer.

NOTE: The contents of the translation buffer are UNPREDICTABLE whenever memory management is disabled. The entire translation buffer contents should be flushed before memory management is enabled. The console firmware performs this function as the system is booted.

2.6 Exceptions and Interrupts

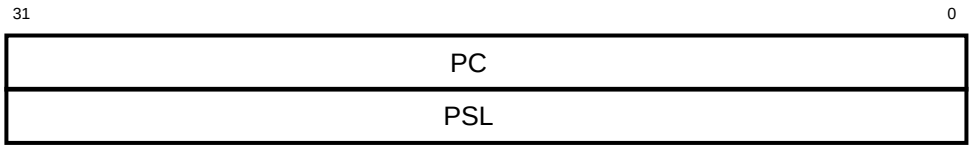
At certain times during the operation of a system, events may occur that break the explicit flow of instructions in the current process and require execution of external software routines. Two types of events can cause such a break: exceptions and interrupts.

An exception is an event related to the currently executing process and invokes a software routine in the context of the current process. Exception handlers are often system routines, not process routines.

An interrupt is an event caused by some activity outside the current process and invokes a software routine outside the context of the current process.

The CPU chip reports exceptions and interrupts by constructing a frame on the stack and then dispatching to the service routine through an event-specific vector in the system control block (SCB, Section 2.7). The minimum stack frame for any interrupt and exception is a program counter/processor status longword (PC/PSL) pair, as shown in Figure 2- 9.

Figure 2- 9 Minimum Stack Frame

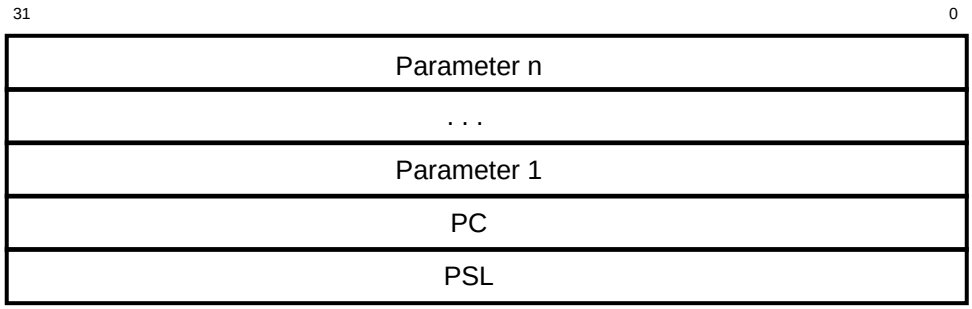


BXB-0188-92

The minimum stack frame is used for all interrupts. Certain exceptions expand the stack frame by pushing additional parameters on the stack above the PC/PSL pair, as shown in Figure 2- 10.

Parameters that are pushed on the stack above the PC/PSL pair depend on the type of exception.

Figure 2- 10 Expanded Stack Frame



2.6.1 Exceptions

There are three types of exceptions:

- Traps
- Faults
- Aborts

A **trap** occurs at the end of an instruction. Therefore, the PC saved on the stack is the address of the next instruction that would normally have been executed had the trap not occurred.

A **fault** occurs during the execution of an instruction and leaves the registers and memory in a consistent state, so that eliminating the fault condition and restarting the instruction gives correct results. The PC saved on the stack points to the instruction that faulted.

An **abort** occurs during the execution of an instruction and leaves the value of the registers and memory UNPREDICTABLE, so that the instruction cannot be restarted, completed, simulated, or undone. In most cases the NVAX+ microcode attempts to convert an abort into a fault by restoring the state preceding the start of the instruction that caused the abort.

This section lists VAX standard exceptions and discusses their implementation on the KA7AA CPU module. The standard exceptions are described in the *VAX Architecture Reference Manual*.

The VAX architecture recognizes six categories of exceptions. Table 2- 1 lists the types of exceptions in each category. Specific characteristics are found in the following exceptions implemented on the NVAX+ chip:

- Arithmetic
- Memory management
- Emulated instruction
- System failure

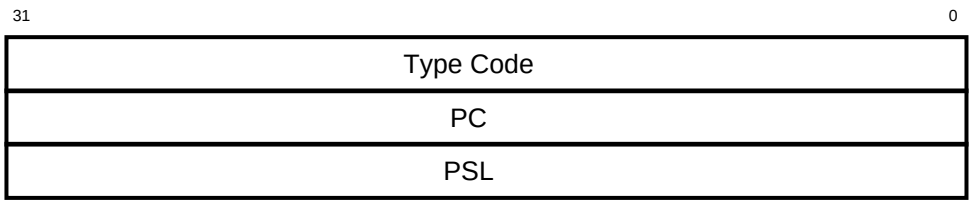
Table 2- 1 KA7AA Module Exceptions

Exception Category	Type
Arithmetic traps/faults	Integer overflow trap Integer divide- by- zero trap Subscript range trap Floating overflow fault Floating divide by zero fault Floating underflow fault
Memory management exceptions	Access control violation fault (ACV) Translation not valid fault (TNV) Modify fault (M)
Operand reference exceptions	Reserved addressing mode fault Reserved operand fault or abort
Instruction execution exceptions	Emulated instruction fault Extended function (XFC) fault Change- mode trap Breakpoint fault
Tracing exceptions	Trace fault
System failure exceptions	Kernel stack not valid abort Interrupt stack not valid fault Console error halt Machine check abort

2.6.1.1 Arithmetic Exceptions

Arithmetic exceptions are detected during the execution of integer or floating- point arithmetic instructions. Figure 2- 11 shows the arithmetic exception stack frame.

Figure 2- 11 Arithmetic Exception Stack Frame



BXB-0189-92

The exception is reported as either a trap or a fault, depending on the specific event. Table 2- 2 shows the encoding of arithmetic exceptions.

Table 2- 2 Arithmetic Exception Codes

Code (Hex)	Type	Exception
1	Trap	Integer overflow
2	Trap	Integer divide- by- zero
7	Trap	Subscript range
8	Fault	Floating overflow
9	Fault	Floating divide- by- zero
A	Fault	Floating underflow

2.6.1.2 Memory Management Exceptions

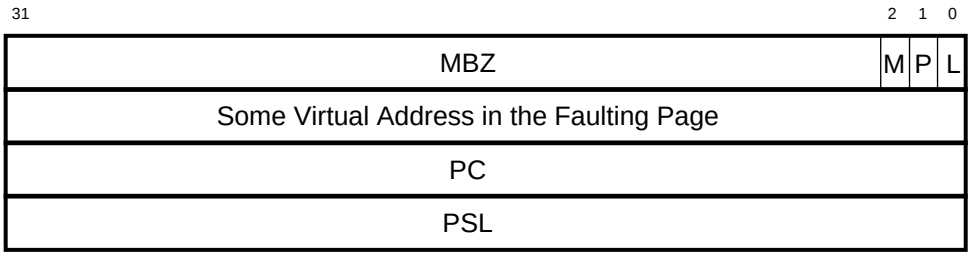
Memory management exceptions are detected during a memory reference and are always reported as faults. The memory management exceptions and the associated SCB vectors are listed in Table 2- 3.

Table 2- 3 Memory Management Exceptions

Exception	SCB Vector (Hex)
Access control violation (ACV)	20
Translation not valid (TNV)	24
Modify fault (M)	3C

All memory management exceptions push the same frame on the stack, as shown in Figure 2- 12.

Figure 2- 12 Memory Management Exception Stack Frame



BXB-0190-92

The M, P, and L bits of the parameter pointed to by the stack pointer reflect bits <2:0> of the MMESTS register, which are described in Table 2- 38.

2.6.1.3 Emulated Instruction Exceptions

The NVAX+ chip implements the VAX base instruction group in hardware and provides microcode support for macrocode emulation of certain other instructions. Two types of emulation exceptions depend on the state of PSL<FPD>. If FPD is zero at the beginning of the instruction, then the exception is reported through SCB vector C8 (hex) as a trap, with the stack frame shown in Figure 2- 13.

Figure 2- 13 Emulated Instruction Trap Stack Frame

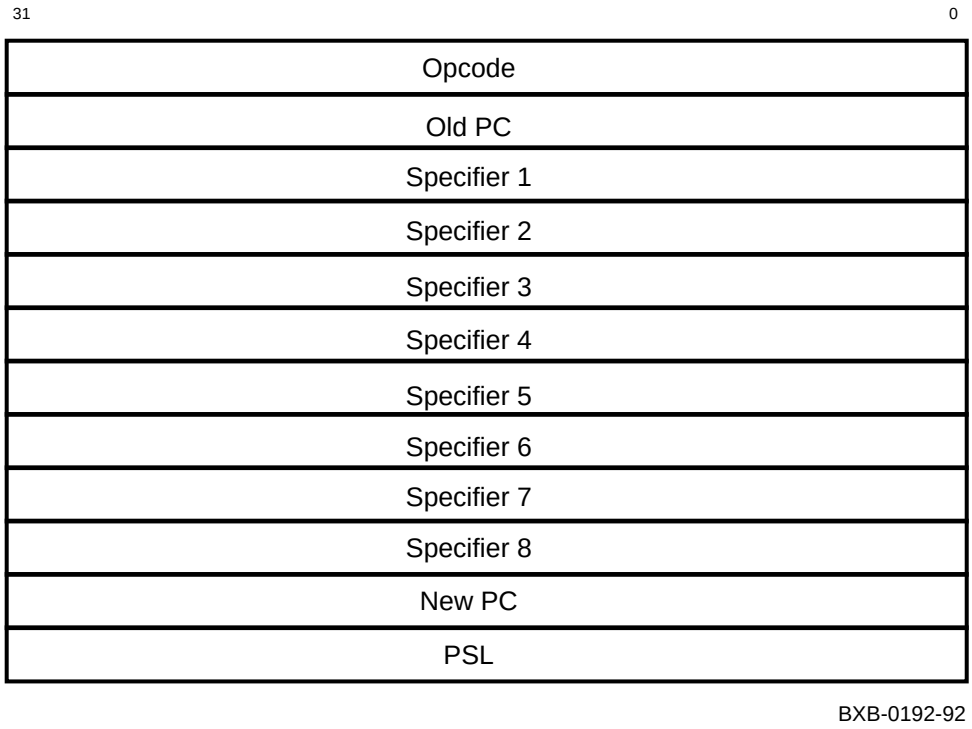


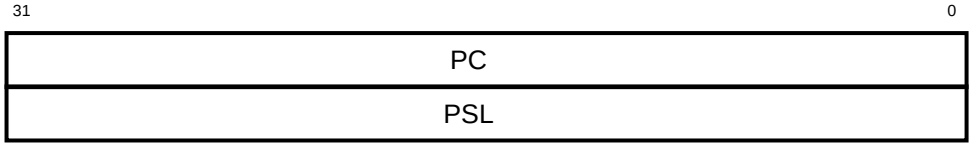
Table 2- 4 describes the emulated instruction stack frame parameters.

Table 2- 4 Emulated Instruction Trap Stack Frame Parameters

Parameter	Function
Opcode	Zero- extended opcode of the emulated instruction.
Old PC	Program counter of the opcode of the emulated instruction.
Specifiers	Address of the specified operand for specifiers of either access type write (.wx) or address (.ax). Operand value for specifiers of access type read (.rx). For read- type operands whose size is smaller than a longword, the remaining bits are UNPREDICTABLE. For those instructions that do not have eight specifiers, the remaining specifier longwords contain UNPREDICTABLE values.
New PC	Program counter of the instruction following the emulated instruction.
PSL	PSL saved at the time of the trap.

If PSL<FPD> (see Figure 2- 19) is one at the beginning of the instruction, then the exception is reported through SCB vector CC (hex) as a fault, with the stack frame as shown in Figure 2- 14. In this case, PC is the opcode of the emulated instruction.

Figure 2- 14 Emulated Instruction Fault Stack Frame



BXB-0188-92

2.6.1.4 System Failure Exceptions

A machine check exception is reported through SCB vector 04 (hex) when the NVAX+ chip detects an error condition. The frame pushed on the stack for a machine check indicates the type of error and provides internal state information that may help identify the cause of the error. Machine check exceptions are discussed at length in Chapter 9 (Error Handling).

In certain microcode flows, the NVAX+ microcode may detect an inconsistency in internal state, a kernel- mode HALT, or a system reset. In these instances, the microcode initiates a hardware restart sequence which passes control to the console program. This process is called console halt.

When a console halt occurs, the NVAX+ microcode saves the current CPU state, partially initializes the CPU, and passes control to the console program at the physical address contained in the CHALT register.

During a hardware restart sequence, the stack pointer is saved in the appropriate stack pointer (0 through 4), the current PC is saved in IPR42 (SAVPC register), and the current PSL, halt code, and validity flag are

saved in IPR43 (SAVPSL register). Console halts are discussed at length in Chapter 9 (Error Handling).

2.6.2 Interrupts

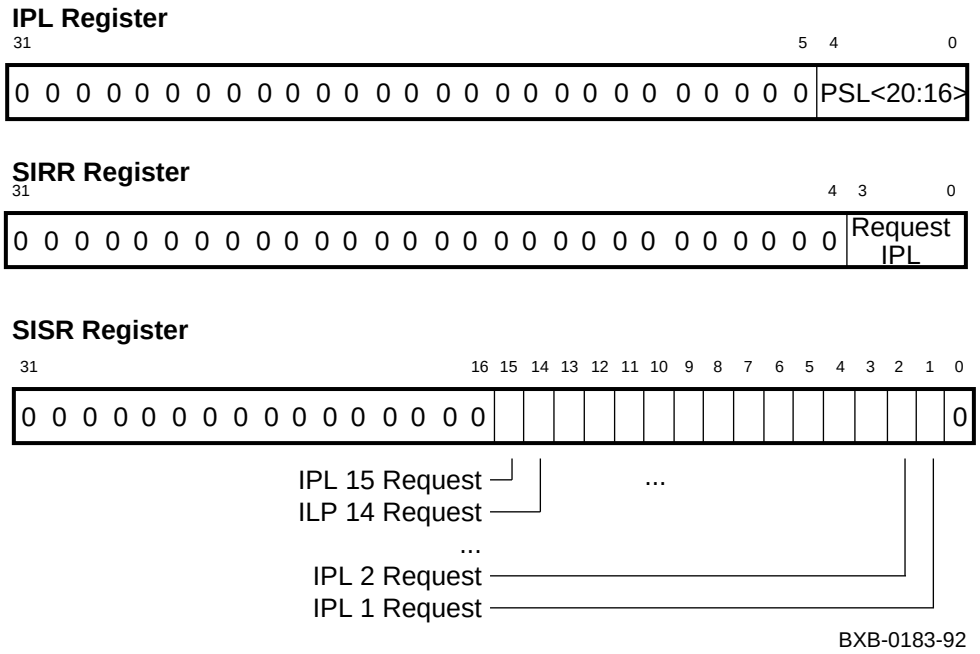
The interrupt section receives interrupt requests from both internal and external sources. When an interrupt request is generated, the NVAX+ chip compares the request with the current IPL of the CPU. If the new request is of higher priority, an internal request is generated. At the completion of the current instruction, or at selected points during the execution of interruptable instructions, a microcode interrupt handler is invoked to process the request. The microcode handler determines the highest priority interrupt, updates the IPL, pushes a PC/PSL pair on the stack, and dispatches to a macrocode interrupt handler through the appropriate location in the SCB.

The interrupt system is controlled by three IPRs:

- Interrupt Priority Level Register (IPL), IPR18
- Software Interrupt Request Register (SIRR), IPR20
- Software Interrupt Summary Register (SISR), IPR21

The IPL register is used for loading the interrupt priority level field (IPL<4:0>) into PSL<20:16>. The SIRR register is used for creating software interrupt requests. The SISR register records pending software interrupt requests at levels 1 through 15. Figure 2- 15 shows the three interrupt control registers.

Figure 2- 15 Interrupt Control Registers



2.6.2.1 External Interrupt Requests

NVAX+ inputs six external interrupt signals as IRQ_H<3:0>, HALT_H, and ERR_H. These signals request general- purpose interrupts at the following IPLs:

- HALT_H: The assertion of HALT_H causes the CPU to enter the console at IPL 1F (hex) at the next macroinstruction boundary. The interrupt is not gated by the current IPL, and always results in console entry, even if the IPL is already 1F (hex). Note that unlike normal interrupts, which cause a PC/PSL pair to be pushed on the interrupt stack, HALT_H interrupts store the current PC/PSL and halt code in the SAVPC and SAVPSL processor registers. Console halts are discussed at length in Chapter 9 (Error Handling).
- ERR_H: The assertion of ERR_H indicates that an error has been detected in the system environment. This results in the dispatch of the interrupt to the operating system at IPL 1D (hex) through SCB vector 60 (hex).
- IRQ<3:0>: Device interrupts resulting in dispatch of the interrupt to the operating system at IPL 14–17 (hex) through SCB vector D0, D4, D8, or DC (hex).

Table 2- 5 shows the external interrupt request levels and the associated SCB vectors. Interrupt routines at the specified SCB acknowledge the interrupt.

Table 2- 5 External Interrupt Requests

Interrupt Request	Request IPL (Hex)	Request IPL (Decimal)	Interrupt Condition	SCB Vector (Hex)
HALT_H	1F	31	Ctrl/P typed at the console, LCNR<NHALT> set, node reset, or system reset	Console
ERR_H	1D	29	Hardware error	60
IRQ<3>	17	23	Device interrupt	DC
IRQ<2>	16	22	Device interrupt	D8
IRQ<1>	15	21	Device interrupt	D4
IRQ<0>	14	20	Device interrupt	D0

2.6.2.2 Internal Interrupt Requests

The Cbox, Ibox, and Mbox report error conditions by asserting internal interrupt request signals. Table 2- 6 gives the interrupt requests caused by internal hard and soft errors.

Table 2- 6 Internal Interrupt Requests

Interrupt Request	Request IPL (Hex)	(Decimal)	SCB Vector (Hex)
H_ERR	1D	29	60
S_ERR	1A	26	54

The performance monitoring facility requests an interrupt at IPL 1B (hex) when the performance counters become half full. This request is serviced entirely by microcode. It is cleared by writing to the appropriate interrupt service routine.

When the interval timer period expires (ICCS<6> is set), the interrupt is dispatched to the operating system at IPL 16 (hex) through SCB vector C0 (hex).

An architecturally defined software interrupt request causes an associated bit to be set in the SISR register. The request is dispatched to the operating system. The IPL of the SCB vector is implied by the interrupt request. Table 2- 7 shows the association between an SISR register bit, request IPL, and SCB vector.

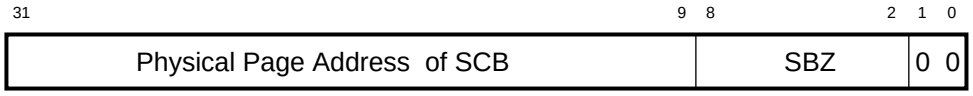
Table 2- 7 Software Interrupt Requests

SISR Bit Set	Request IPL (Hex)	(Decimal)	SCB Vector (Hex)
<15>	0	15	BC
<14>	0E	14	B8
<13>	0D	13	B4
<12>	0C	12	B0
<11>	0B	11	AC
<10>	0A	10	A8
<9>	09	09	A4
<8>	08	08	A0
<7>	07	07	9C
<6>	06	06	98
<5>	05	05	94
<4>	04	04	90
<3>	03	03	8C
<2>	02	02	88
<1>	01	01	84

2.7 System Control Block

The system control block (SCB) is a page containing vectors for servicing interrupts and exceptions. The SCB is pointed to by the 32- bit System Block Base Register (SCBB). Since the SCBB contains a 32- bit physical address, the SCB can reside anywhere in memory space. For optimum performance, the SCBB should contain a page- aligned address. Microcode forces longword alignment by clearing bits <1:0> of the new value before loading it into the SCBB register. Figure 2- 16 shows the format of the SCBB register.

Figure 2- 16 System Control Block Base Register

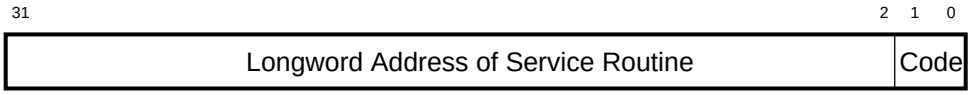


BXB-0185-92

NOTE: When When the CPU is in 30- bit physical address mode, SCBB<31:30> are ignored.

An SCB vector is an aligned longword in the SCB. The NVAX+ chip microcode dispatches interrupts and exceptions through the SCB vector, shown in Figure 2- 17.

Figure 2- 17 System Control Block Vector



BXB-0197-92

Table 2- 8 describes the bit functions of the SCB vector.

Table 2- 8 System Control Block Vector Bit Functions

Bit(s)	Function
<31:2>	Virtual address of the service routine for the interrupt or exception. The routine is longword aligned, as the microcode forces the lower two bits of the address to 00 (hex).
<1:0>	These two bits are decoded as follows:

SCB<1:0>	Function
00	The event is to be serviced on the kernel stack unless the CPU is already on the interrupt stack, in which case the event is to be serviced on the interrupt stack.
01	The event is to be serviced on the interrupt stack. If the event is an exception, the IPL is raised to 1F (hex).
10	Unimplemented; results in a console error halt.
11	Unimplemented; results in a console error halt.

Table 2- 9 shows the SCB layout.

Table 2- 9 System Control Block Layout

Vector (Hex)	Name	Type[1]	No. of Parameters	Notes
00	Unused			
04	Machine check	A	6	Parameters reflect machine state; must be serviced on interrupt stack
08	Kernel stack not valid	A	0	Must be serviced on interrupt stack
0C	Unused			
10	Reserved/privileged instruction	F	0	
14	Customer reserved instruction	F	0	XFC instruction
18	Reserved operand	A/F	0	Not always recoverable
1C	Reserved addressing mode	F	0	
20	Access control violation/ vector alignment fault	F	2	Parameters are virtual address, status code
24	Translation not valid	F	2	Parameters are virtual address, status code
28	Trace pending	F	0	
2C	Breakpoint instruction	F	0	
30	Unused			Compatibility mode in other VAX systems
34	Arithmetic	F/T	1	Parameter is type code
38–3C	Unused			
40	CHMK	T	1	Parameter is sign- extended operand word
44	CHME	T	1	Parameter is sign- extended operand word
48	CHMS	T	1	Parameter is sign- extended operand word
4C	CHMU	T	1	Parameter is sign- extended operand word

[1] A = Abort; F = Fault; T = Trap; I = Interrupt.

Table 2- 9 System Control Block Layout (Continued)

Vector (Hex)	Name	Type[1]	No. of Parameters	Notes
50	Unused			
54	Soft error notification	I	0	IPL is 1A (hex)
58	Performance monitoring counter overflow	I		
59–5C	Unused			
60	Hard error notification	I	0	IPL is 1D (hex)
64- 80	Unused			
84	Software level 1	I	0	
88	Software level 2	I	0	Ordinarily used for AST delivery
8C	Software level 3	I	0	Ordinarily used for process scheduling
90–BC	Software levels 4–15	I	0	
C0	Interval timer	I	0	IPL is 16 (hex)
C4	Unused			
C8	Emulation start	F	10	Same mode exception, FPD=0; parameters are opcode, PC, specifiers
CC	Emulation continue	F	0	Same mode exception, FPD=1; no parameters
D0	Device vector	I	0	IPL is 14 (hex)
D4	Device vector	I	0	IPL is 15 (hex), includes console interrupts
D8	Device vector	I	0	IPL is 16 (hex), includes interprocessor interrupts
DC	Device vector	I	0	IPL is 17 (hex)
E0–FFFC	Unused			

[1] A = Abort; F = Fault; T = Trap; I = Interrupt.

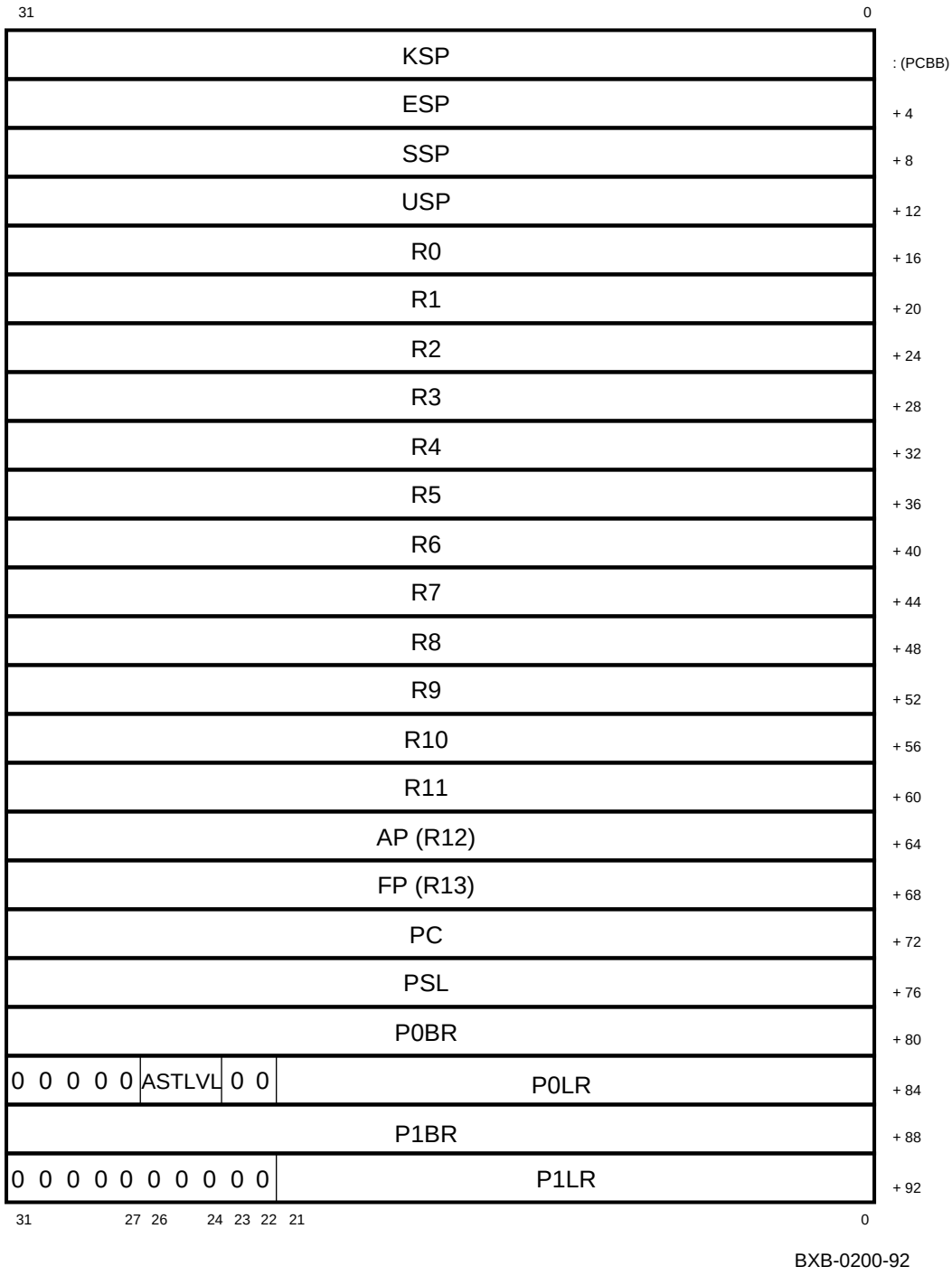
2.8 Process Structure

A process is a single thread of execution. The context of the current process is contained in the process control block (PCB), which can be located anywhere in memory space. The PCB is pointed to by the address contained in the Process Control Block Base (PCBB) register, IPR16. The physical address of the current PCB is changed by writing to the PCBB register. The LDPCTX instruction loads a process context from the PCB as described in the *VAX Architecture Reference Manual*. LDPCTX flushes only the process space entries from the translation buffer; system space entries are preserved. Other process structure functions are implemented as described in the *VAX Architecture Reference Manual*.

NOTE: When the CPU is in 30- bit addressing mode, PCBB<31:30> are ignored.

Figure 2- 18 shows the process control block layout.

Figure 2- 18 Process Control Block



The processor status longword (PSL) is a 32- bit register that contains information about the state of the processor. Figure 2- 19 shows the PSL.

Figure 2- 19 Processor Status Longword

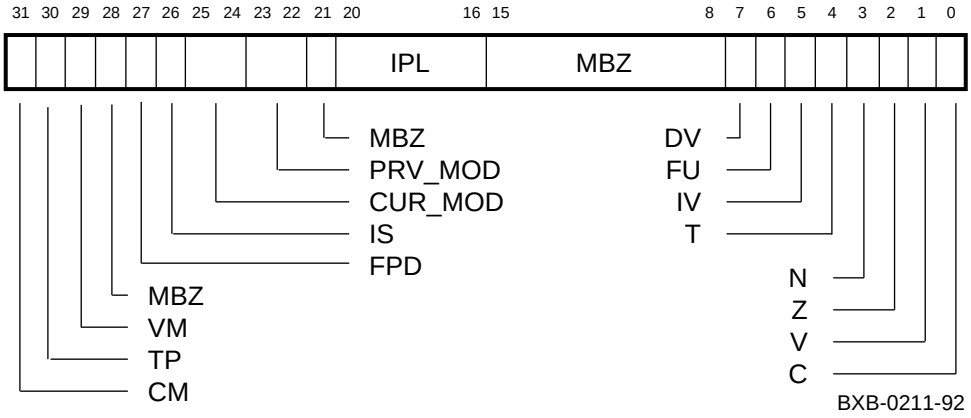


Table 2- 10 identifies the bits in the PSL.

Table 2- 10 Processor Status Longword Bits

Mnemonic	Bit(s)	Name
CM	<31>	Compatibility mode
TP	<30>	Trace pending
VM	<29>	Virtual machine mode[1]
FPD	<27>	First part done
IS	<26>	Interrupt stack
CUR_MOD	<25:24>	Current mode
PRV_MOD	<23:22>	Previous mode
IPL	<20:16>	Interrupt priority level
DV	<7>	Decimal overflow trap enable
FU	<6>	Floating underflow fault enable
IV	<5>	Integer overflow trap enable
T	<4>	Trace trap enable
N	<3>	Negative condition code
Z	<2>	Zero condition code
V	<1>	Overflow condition code
C	<0>	Carry condition code

[1] MBZ unless virtual machine option is implemented.

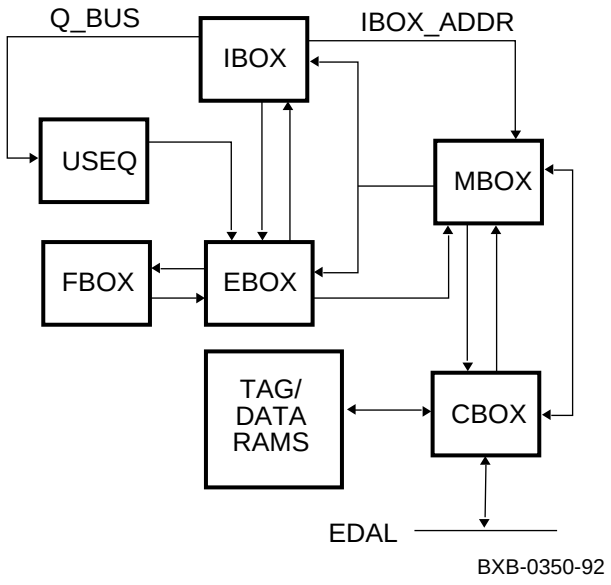
2.9 Functional Partitions

The macropipelined VAX design of the NVAX+ CPU chip is implemented through a group of tightly coupled logic units called boxes. The VAX functions are partitioned among five boxes:

- Ibox
- Ebox and Microsequencer
- Fbox
- Mbox
- Cbox

The NVAX+ chip achieves high performance in large part through the concurrent, and relatively independent, operation of these logic boxes. Figure 2- 20 shows the logic box level block diagram of the CPU module.

Figure 2- 20 NVAX+ Logic Boxes



2.9.1 Ibox

The Ibox (instruction box) decodes VAX instructions and parses operand specifiers. Instruction control, such as the control store dispatch address, is then placed in the instruction queue for later use by the microsequencer and Ebox (see Section 2.9.2). The Ibox processes the operand specifiers at a rate of one specifier per cycle and, as necessary, initiates specifier memory read operations. All the information needed to access the specifiers is placed in the source queue and destination queue in the Ebox.

The Ibox prefetches instruction stream data from a dedicated instruction cache, called the virtual instruction cache (VIC), and places it into the 16-byte prefetch queue (PFQ). The VIC is a 2- Kbyte, direct- mapped cache, with a block and fill size of 32 bytes.

The Ibox has both read and write ports to the general purpose register (GPR) and memory data (MD) portions of the Ebox register file. These ports are used to process the operand specifiers. The Ibox maintains a scoreboard to ensure that reads and writes to the register file are always performed in synchronization with the Ebox. The Ibox stops processing instructions and operands upon issuing certain complex instructions like CALL, RET, or character string instructions, so that proper read/write ordering is maintained while the Ebox alters large amounts of VAX state data. Since the Ibox is often parsing several macroinstructions ahead of the Ebox, the correct value for the PSL condition codes is not known at the time the Ibox executes a conditional branch instruction. Rather than emptying the pipe, the Ibox predicts which direction the branch will take and passes this information on to the Ebox via the branch queue. The Ebox later signals if there was a misprediction, and the hardware backs out of the path. The branch prediction algorithm uses a 512- entry RAM, which caches four bits of branch history per entry.

The instruction fetching logic prefetches several instructions ahead of the instruction parsing logic, identifying and pre- processing each instruction's components. The parsing logic decodes part of the instruction and passes the instruction opcodes and associated information directly into the Ebox instruction queue. Operand specifier information is passed on to the operand specifier processing logic.

Instruction prefetching also provides a buffer, four bytes wide by four elements deep, that isolates the instruction parser from the bursts of data coming in from cache and memory. The result is that the instruction fetching and instruction parsing can be done in parallel.

The operand specifier processing logic locates the operands in registers, in memory, or in the instruction stream. This logic places operand information in the Ebox source and destination queues and makes the required operand memory requests.

The Ibox's branch prediction unit (BPU) monitors each instruction opcode as it is parsed, looking for a branch opcode. Upon identification of a branch opcode, the BPU predicts whether the branch will be taken. If the prediction is that the branch will be taken, the BPU adds the sign-extended branch displacement to the current PC and broadcasts the resulting new PC to the rest of the Ibox. The BPU is controlled by the microcode.

Since branch direction relies on Ebox condition codes, the Ibox has no prior knowledge of branch direction. Branch prediction logic makes a prediction

on which way the branch will go and forces the Ibox to take that path. The PC pointing to the alternate branch path is saved should the prediction prove wrong. If the prediction is wrong, the Ibox is redirected to the correct path.

2.9.2 Ebox and Microsequencer

The Ebox and microsequencer work together to perform the actual "work" of the VAX instruction execution. Together they implement a four- stage micropipelined unit that can both stall and microtrap. The Ebox and microsequencer dequeue instruction and operand information provided by the Ibox through the instruction queue, the source queue, and the destination queue. For literal type operands, the source queue contains the actual operand value. In the case of register, memory, and immediate type operands, the source queue holds a pointer to the data in the Ebox register file. The contents of memory operands are provided by the Mbox (Section 2.9.4) based on earlier requests from the Ibox. GPR results are written directly back to the register file. Memory results are sent to the Mbox, where the data is matched with the appropriate specifier address previously sent by the Ibox. At times, the Ebox initiates its own memory reads and writes.

The Ebox contains a five- port register file, which holds the VAX GPRs, six memory data registers, six microcode working registers, and ten miscellaneous CPU state registers. It also contains an ALU, a shifter, and the VAX processor status longword (PSL). The Ebox uses the retire queue to order the completion of Ebox and Fbox (Section 2.9.3) instructions. Since the Ebox and the Fbox are distinct hardware resources, some execution overlap is allowed between the two units.

The Ebox implements two IPRs: the Patchable Control Store Control Register (PCSCR), used to patch the NVAX+ microcode and select certain Ebox functions, and the Ebox Control Register (ECR), also used to select certain Ebox functions.

The microsequencer determines the next microword to be fetched from the control store. It then provides cycle- by- cycle control of the Ebox.

The control store is an on- chip ROM that contains the microcode used to execute microinstructions and microtraps. It is made up of 1600 microwords. These are arranged as 200 entries, each entry consisting of 8 microwords. Each microword is 61 bits long, with bits <14:0> being used to control the microsequencer. The remainder of the microword, bits <60:15>, is used by the Ebox to control S3 through S5 of the pipeline. The control store is addressed using the output of the current address latch (CAL).

The patchable control store is an on- chip SRAM that contains microcode patches. It consists of up to 20 microwords. It operates in parallel with the control store. The microaddress from the CAL is the input to its content addressable memory (CAM). If the address hits in the CAM, the output of the patchable control store is selected as the new microword, rather than the output of the regular control store.

2.9.3 Fbox

The Fbox is the floating- point unit in the NVAX+ CPU chip. It is a four-stage pipelined floating- point processor, with an additional stage dedicated to assisting division. The Fbox interacts with three segments of the main CPU pipeline: the microsequencer in stage 2, and the Ebox in stages 3 and 4. The Fbox supports the following operations:

- **Floating-Point Instructions and Data Types**
  The Fbox provides instruction and data support for VAX floating- point instructions. VAX F_, D_, and G_floating- point data types are supported.
- **VAX Integer Instructions**
  The Fbox implements longword integer multiply instructions.
- **Pipelined Operation**
  Except for all the divide instructions, DIV{F,D,G}, the Fbox can start a new single- precision floating- point instruction every cycle and a double- precision floating- point or an integer multiply instruction every two cycles. The Ebox can supply two 32- bit operands or one 64- bit operand to the Fbox every cycle on two 32- bit input operand buses. The Fbox drives the result operand to the Ebox on a 32- bit result bus.
- **Conditional "Mini-Round" Operation**
  Result latency is conditionally reduced by one cycle for the most frequently used instructions. Stage 3 can perform a "mini- round" operation on the least significant bits of the fraction for all ADD, SUB, and MUL floating instructions. If the "mini- round" operation does not fail, then stage 3 drives the result directly to the output, bypassing stage 4 and saving a cycle of latency.
- **Fault and Exception Handling**
  The Ebox coordinates the fault and exception handling with the Fbox. Any fault or exception condition received from the Ebox is retried in the proper order. If the Fbox receives or generates any fault or exception condition, it does not change the flow of instructions in progress within the Fbox pipe.

2.9.4 Mbox

The Mbox performs three primary functions:

- **VAX Memory Management**
  The Mbox, in conjunction with the operating system memory management software, manages the allocation and use of physical memory. It performs translations of virtual addresses to physical addresses, access violation checks on all memory references, and initiates the invocation of software memory management code when necessary. The Mbox uses the translation buffer, which contains 96 fully associative entries, to map virtual to physical addresses. In the case of a TB miss, the memory management hardware in the Mbox reads the PTE from the cache or main memory and fills the TB. The Mbox performs all access checks, TNV checks, M- bit checks, and quadword unaligned data processing.
- **Reference Processing**
  Due to the macropipeline structure of the NVAX+ chip, and the coupling between NVAX+ and its memory subsystem, the Mbox can re-

ceive memory references from the Ibox, Ebox, and Cbox simultaneously. The Mbox receives read requests from the Ibox (both in instruction stream and data stream) and from the Ebox (data stream only). It receives write/store requests from the Ebox. Also, the Cbox sends the Mbox fill data and invalidates for the primary cache (P- cache). The Mbox arbitrates between these requesters and queues requests that cannot currently be handled. Once a request is started, the Mbox performs address translation and cache lookup in two cycles, assuming there are no misses or other delays. The two- cycle Mbox operation is pipelined.

- **Primary Cache Control**
  The Mbox contains the primary cache (P- cache), which is an 8- Kbyte, 2- way set associative, write- through cache with a block and fill size of 32 bytes. The Mbox maintains the P- cache state as a subset of the backup cache (B- cache). The Mbox ensures that Ibox specifier reads are ordered correctly with Ebox specifier stores. To prevent the Ibox from using data that the Ebox should have first written, the Mbox "scoreboards" the memory. Scoreboarding is done using the physical address queue, a small list of physical addresses that have a pending Ebox store.

The Mbox has four error registers. Two record TB parity errors, and two record P- cache parity errors. They are as follows:

- **TBADR (TB Parity Address Register)**
  Holds the virtual address associated with a translation buffer parity error.
- **TBSTS (TB Status Register)**
  Holds the error status associated with errors occurring in the Mbox.
- **PCADR (P- Cache Parity Address Register)**
  Holds the physical address associated with a P- cache parity error.
- **PCSTS (P- Cache Status Register)**
  Holds the error status of errors that occur in the P- cache.

When the operating system error handler routine is invoked from a microtrap or interrupt, the handler can read the state of all the error registers through IPR read operations to determine what errors were present when the error handler was invoked. IPR read operations are performed with MFPR instructions.

2.9.5 Cbox

The Cbox provides the interface to the EDAL bus, which is the communication channel between the NVAX+ and the other sections of the KA7AA module (B- cache and the LEVI interface).

The Cbox is tightly coupled to the Mbox. The Mbox sends read requests and writes to the Cbox; the Cbox sends fills and invalidates to the Mbox. The Cbox ensures that the P- cache is a subset of the B- cache through invalidates.

The Cbox communicates with the LEVI interface, and thus with the LSB bus, through the EDAL bus. The Cbox generates reads and receives fills; it receives cache coherence transactions from the EDAL, to which it responds with invalidates and writebacks, as appropriate.

The Cbox implements a programmable interval clock that provides an interrupt at IPL 16 (hex) at programmed intervals. The counter is incremented at 1 microsecond intervals, with at least .01% accuracy. The interval clock consists of three internal processor registers (IPRs) that reside in the privileged register space:

- Interval Count Control and Status Register (ICCS), IPR24
- Next Interval Count Register (NICR), IPR25
- Interval Count Register (ICR), IPR26

To program the interval clock, the negative (2's complement) of the desired interval is loaded into the NICR register. Then, writing 51 (hex) to the ICCS enables interrupts, loads the next interval into the ICR, and sets ICCS<RUN>. Setting this bit causes an interrupt to occur every "interval count" microseconds. The interrupt routine should write C1 (hex) to the ICCS to clear the interrupt. If the Interrupt bit has not been cleared (the interrupt has not been handled) by the time of the next ICR overflow, ICCS<ERR> is set. If the NICR is written while the clock is running, the clock may lose or add a few ticks. If the interval clock interrupt is enabled, this may cause the loss of an interrupt.

2.10 General Purpose Registers

General purpose registers are provided to be used by the operating system in carrying out its normal functions. The NVAX+ processor contains sixteen 32- bit general purpose registers. Some of these registers are assigned to specific functions. The others can be used by the operating system for any required purpose. Table 2- 11 shows the usage of the GPRs.

Table 2- 11 General Purpose Register Usage

GPR	Acronym	Use
R0–R11		General purpose
R12	AP	Argument pointer
R13	FP	Frame pointer
R14	SP	Stack pointer
R15	PC	Program counter

2.11 Internal Processor Registers

Internal processor registers implement functions assigned by hardware. They are used to store data, control, and status information. The internal processor registers are logically divided into three groups:

- **Normal**
  Address individual registers in the NVAX+ chip or system environment.
- **P- Cache Tag**
  The read/write block of IPRs that allow direct access to the P- cache tags.
- **P- Cache Data Parity**
  The read/write block of IPRs that allow direct access to the P- cache data parity bits.

Each group of IPRs is distinguished by a particular pattern of bits in the IPR address, as shown in Figure 2- 21.

Figure 2- 21 IPR Address Formats

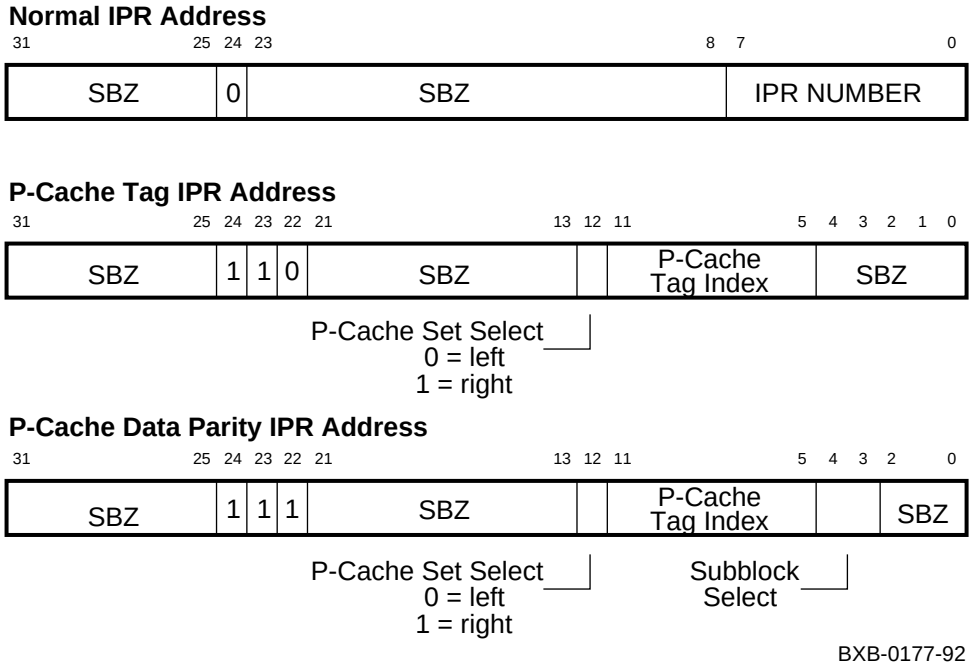


Table 2- 12 shows the numeric range of each group of IPRs.

Table 2- 12 IPR Address Space Decoding

IPR Group	Mnemonic[1]	Address Range (hex)	Contents
Normal		0000 0000 to 0000 00FF[2]	256 individual IPRs.
P- cache tag	PCTAG	0180 0000 to 0180 1FE0[2]	256 P- cache tag IPRs, 128 per P-cache set. Each IPR is separated from the previous one by 20 (hex).
P- cache data parity	PCDAT	01C0 0000 to 01C0 1FF8[2]	1024 P- cache data parity IPRs, 512 per P- cache set. Each IPR is separated from the previous one by 8 (hex).

[1] The mnemonic is for the first IPR in the group.

[2] Unused fields for the IPR addresses for these groups should be zero. Neither hardware nor microcode detects and faults on an address in which these bits are nonzero. Although noncontiguous address ranges are shown for these groups, the entire IPR address space maps into one of these groups. If these fields are nonzero, the operation of the CPU is undefined.

Because of the sparse addressing space used for IPRs in groups other than the normal group, valid IPR addresses in those groups are separated not by 1, as are those of the normal group, but by either 8 or 20 (hex). For example, the IPR address for the first subblock of P- cache data parity is 01C0 0000 (hex), and the IPR address for the second subblock of P- cache data parity is 01C0 0008 (hex).

Table 2- 13 lists the normal IPRs implemented by the NVAX+ chip. Software- accessible registers are described individually in this section.

CAUTION: Many of the IPRs listed in Table 2- 13 are used internally by the microcode during normal operation of the CPU and are not intended to be referenced by software except during test or diagnosis of the system. These registers are flagged with the notation "Testability and diagnostic use only; not for software use in normal operation." References by software to these registers during normal operation can cause undefined behavior of the CPU.

Table 2- 13 KA7AA Internal Processor Registers

Name	Mnemonic	Type	Address Dec (Hex)
Kernel Stack Pointer	KSP	R/W	0 (0)
Executive Stack Pointer	ESP	R/W	1 (1)
Supervisor Stack Pointer	SSP	R/W	2 (2)
User Stack Pointer	USP	R/W	3 (3)
Interrupt Stack Pointer	ISP	R/W	4 (4)
P0 Base Register	P0BR	R/W	8 (8)
P0 Length Register	P0LR	R/W	9 (9)
P1 Base Register	P1BR	R/W	10 (A)
P1 Length Register	P1LR	R/W	11 (B)
System Base Register	SBR	R/W	12 (C)
System Length Register	SLR	R/W	13 (D)
CPU Identification Register[1]	CPUID	R/W	14 (E)
Process Control Block Base Register	PCBB	R/W	16 (10)
System Control Block Base Register	SCBB	R/W	17 (11)
Interrupt Priority Level Register[1]	IPL	R/W	18 (12)
AST Level Register	ASTLVL	R/W	19 (13)
Software Interrupt Request Register	SIRR	WO	20 (14)
Software Interrupt Summary Register[1]	SISR	R/W	21 (15)
Interval Count Control/Status Register[1]	ICCS	R/W	24 (18)
Next Interval Count Register	NICR	WO	25 (19)
Interval Count Register	ICR	RO	26 (1A)
Time- of- Day Register	TODR	R/W	27 (1B)
Machine Check Error Register	MCESR	WO	38 (26)
Console Saved PC Register	SAVPC	RO	42 (2A)
Console Saved PSL Register	SAVPSL	RO	43 (2B)
Memory Management Enable Register[1]	MAPEN	R/W	56 (38)
Translation Buffer Invalidate All Register	TBIA	WO	57 (39)
Translation Buffer Invalidate Single Register	TBIS	WO	58 (3A)
Performance Monitor Enable Register[1]	PME	R/W	61 (3D)
System Identification Register	SID	RO	62 (3E)
Translation Buffer Check Register	TBCHK	WO	63 (3F)
Interrupt System Status Register[2]	INTSYS	R/W	121 (79)
Mailbox Register	LMBOX	R/W	122 (7A)
Performance Monitoring Facility Count	PMFCNT	R/W	123 (7B)
Patchable Control Store Control Register[2]	PCSCR	WO	124 (7C)
Ebox Control Register	ECR	R/W	125 (7D)
Mbox TB Tag Fill Register[2]	MTBTAG	WO	126 (7E)
Mbox TB PTE Fill Register[2]	MTBPTE	WO	127 (7F)
BIU Control Register	BIU_CTL	R/W	160 (A0)

[1] Initialized on reset.
[2] Testability and diagnostic use only; not for software use in normal operation.

Table 2- 18 KA7AA Internal Processor Registers (Continued)

Name	Mnemonic	Type	Address Dec (Hex)
Diagnostic Control Register	DIAG_CTL	WO	161 (A1)
B- Cache Error Tag Register	BC_TAG	RO	162 (A2)
BIU Status Register	BIU_STAT	W1C	164 (A4)
BIU Address Register	BIU_ADDR	RO	166 (A6)
Fill Syndrome Register	FILL_SYND	RO	168 (A8)
Fill Address Register	FILL_ADDR	RO	170 (AA)
STxC Pass Fail/CEFSTS Register	IPR_STR_COND	RO	172 (AC)
Software ECC Register	BEDECC	WO	174 (AE)
Console Halt Register	CHALT	R/W	176 (B0)
Serial I/O Register	SIO	R/W	178 (B2)
SROM_OE_Serial I.E. Register	SOE- IE	WO	180 (B4)
Pack to QW Register	QW_PACK	WO	184 (B8)
Clear I/O Pack Register	CLR_IO_PACK	WO	185 (B9)
VIC Memory Address Register	VMAR	R/W	208 (D0)
VIC Tag Register	VTAG	R/W	209 (D1)
VIC Data Register	VDATA	R/W	210 (D2)
Ibox Control and Status Register	ICSR	R/W	211 (D3)
Ibox Branch Prediction Control Register	BPCR	R/W	212 (D4)
Ibox Backup PC Register	BPC	RO	214 (D6)
Ibox Backup PC with RLOG Unwind Register[3]	BPCUNW	RO	215 (D7)
Mbox P0 Base Register[2]	MP0BR	R/W	224 (E0)
Mbox P0 Length Register[2]	MP0LR	R/W	225 (E1)
Mbox P1 Base Register[2]	MP1BR	R/W	226 (E2)
Mbox P1 Length Register[2]	MP1LR	R/W	227 (E3)
Mbox System Base Register[2]	MSBR	R/W	228 (E4)
Mbox System Length Register[2]	MSLR	R/W	229 (E5)
Mbox Map Enable Register[2]	MMAPEN	R/W	230 (E6)
Physical Address Mode Register	PAMODE	R/W	231 (E7)
MME Address Register	MMEADR	RO	232 (E8)
MME PTE Address Register	MMEPTE	RO	233 (E9)
MME Status Register	MMESTS	RO	234 (EA)
TB Parity Address Register	TBADR	RO	236 (EC)
TB Parity Status Register	TBSTS	R/W	237 (ED)
P- Cache Parity Address Register	PCADR	RO	242 (F2)
P- Cache Parity Status Register	PCSTS	R/W	244 (F4)
P- Cache Control Register	PCCTL	R/W	248 (F8)
P- Cache Tag Registers	PCTAG	R/W	(0180 0000 to 0180 1FE0)
P- Cache Data Parity Registers	PCDAP	R/W	01C0 0000 to 01C0 1FF8)

[2] Testability and diagnostic use only; not for software use in normal operation.
[3] Chip test use only; not for software use.

User- visible normal internal processor registers discussed in the following sections include:

- Identification registers
- Ibox registers
- Ebox registers
- Mbox registers
- Cbox registers

2.11.1 Identification Registers

Table 2- 14 lists the two registers used to identify the operating CPU and the system. Functional descriptions of individual identification registers follow.

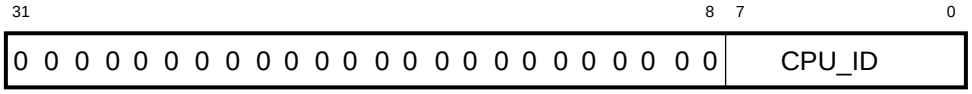
Table 2- 14 Identification Registers

Name	Mnemonic	Type	IPR Address (Hex)
CPU Identification Register	CUPID	RO	0E
System Identification Register	SID	RO	3E

CPUID—CPU Identification Register

Address 000E
Access RO

The CPUID register allows software to determine the currently executing CPU in a multiprocessor system.



BXB-0182-92

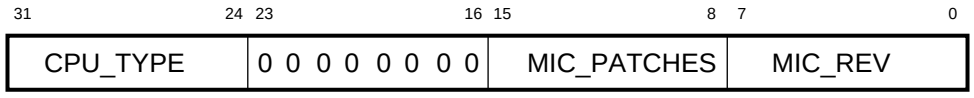
Table 2- 15 CPUID Register Bit Definitions

Name	Bit(s)	Type	Function
CPU_ID	<7:0>	RO	**CPU Identification.** Indicates the currently executing CPU in a multiprocessor system. The KA7AA module defines bits <5:0> only, corresponding to CPU5 to CPU0. This field is loaded by the console firmware at power- up and is readable by software at any time.

SID—System Identification Register

Address 003E
Access RO

The SID register contains information that identifies the processor type and the microcode revision level.



BXB-0180-92

Table 2- 16 SID Register Bit Definitions

Name	Bit(s)	Type	Function
CPU_TYPE	<31:24>	RO	**CPU Type.** Indicates the type of CPU used in the system. The value of this field for the NVAX+ CPU chip is 17 (hex).
MIC_PATCHES	<15:8>	RO	**Microcode Patches.** Indicates the microcode patches if any, included in the CPU chip. Normally, this field has a value of zero.
MIC_REV	<7:0>	RO	**Microcode Revision.** Indicates the microcode revision of the NVAX+ CPU chip. The value of this field is 2 for the first revision of the CPU module or higher for later revisions.

2.11.2 Ibox Registers

Table 2- 17 lists the Ibox registers. Functional descriptions of individual Ibox registers follow.

Table 2- 17 Ibox Registers

Name	Mnemonic	Type	IPR Address (Hex)
VIC Memory Address Register	VMAR	R/W	D0
VIC Tag Register	VTAG	R/W	D1
VIC Data Register	VDATA	R/W	D2
Ibox Control and Status Register	ICSR	R/W	D3
Branch Prediction Control Register	BPCR	R/W	D4

VMAR—VIC Memory Address Register

Address 00D0
Access R/W

The VMAR register provides access to the virtual instruction cache (VIC) arrays. When the VIC is disabled, the VMAR register can be used as an index for direct IPR access to the cache arrays. This register also latches and holds the virtual address on VIC array parity errors.

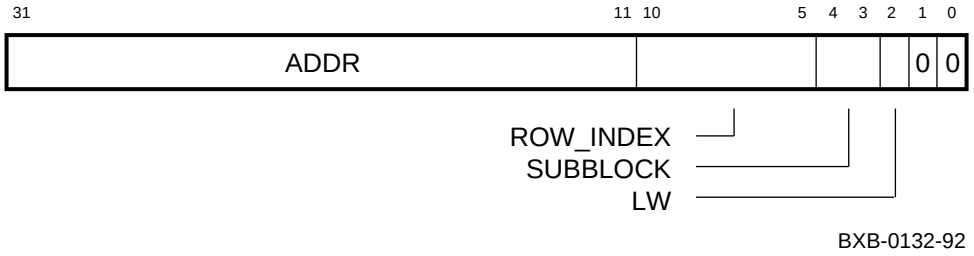


Table 2- 18 VMAR Register Bit Definitions

Name	Bit(s)	Type	Function
ADDR	<31:11>	RO	**Error Address.** Latches tag portion of the virtual address on VIC parity errors.
ROW_INDEX	<10:5>	R/W	**Row Index.** Contains row index for read and write access to cache array; also latches the virtual address.
SUBBLOCK	<4:3>	R/W	**Subblock Select.** Selects data subblock for access to cache array. Also latches bits <4:3> of the virtual address on VIC parity errors.
LW	<2>	WO	**Longword Select.** Selects longword of subblock for access to cache array.

VTAG—VIC Tag Register

Address 00D1
Access R/W

The VTAG register provides diagnostic access to the cache tag array.

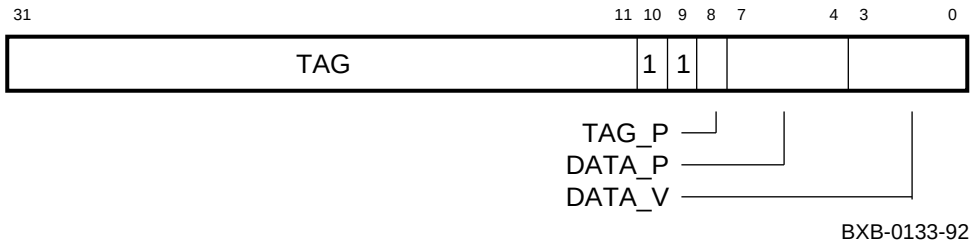


Table 2- 19 VTAG Register Bit Definitions

Name	Bit(s)	Type	Function
TAG	<31:11>	R/W	**Tag.** Supplies tag on tag array read/writes.
TAG_P	<8>	R/W	**Tag Parity.** Supplies tag parity on tag array read/writes.
DATA_P	<7:4>	R/W	**Data Parity**. Supplies four parity bits for four quadwords of data on array read/writes. Each data parity bit has a corresponding data valid bit (VTAG<3:0>). VTAG<4> and VTAG<0> are associated with the quadword of data addressed when VMAR<4:3> = 00; VTAG<5> and VTAG<1> correspond to the quadword of data addressed when VMAR<4:3> = 01, and so on.
DATA_V	<3:0>	R/W	**Data Valid**. Supplies four valid bits for four quadwords of data on array read/writes. Each data valid bit has a corresponding data parity bit (VTAG<7:4>). See description of VTAG<7:4> for quadwords associated with each data valid bit.

VDATA—VIC Data Register

Address 00D2
Access R/W

The VDATA register provides diagnostic access to the cache data array.

BXB-0134-92

Table 2- 20 VDATA Register Bit Definitions

Name	Bit(s)	Type	Function
VIC_DATA	<31:0>	R/W	**VIC Data.** Data for array reads and writes.

VDATA Read/Write

When the VDATA register is written, the cache data array entry indexed by the VMAR register is written with the IPR data. Since the IPR data is a longword, two accesses to the VDATA register are required to read or write a quadword cache subblock.

Writes to the VDATA register with VMAR<2> = 0 accumulate the IPR data destined for the low longword of a subblock in FILL_DATA<31:0>. A subsequent write to the VDATA register with VMAR<2> = 1 directs the IPR data to FILL_DATA<63:32> and triggers a cache write sequence to the subblock indexed by the VMAR register.

Reads to the VDATA register with VMAR<2> = 0 trigger a cache read sequence to the subblock indexed by the VMAR register. The low longword of a subblock is returned as IPR read data. A read of the VDATA register with VMAR<2> = 1 returns the high longword of the subblock as IPR data.

ICSR—Ibox Control and Status Register

Address 00D3
Access R/W

The ICSR register provides control and status functions for the Ibox.

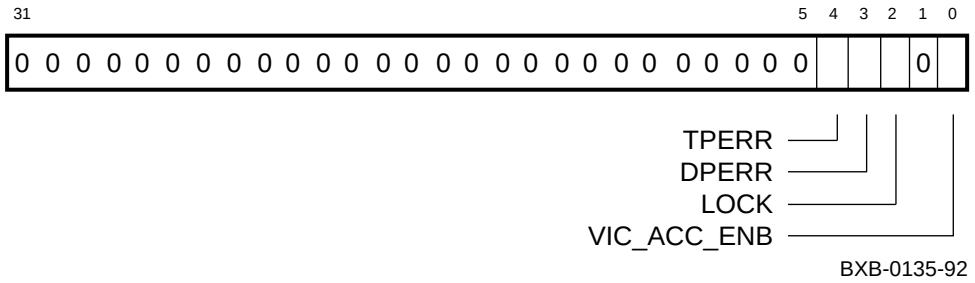


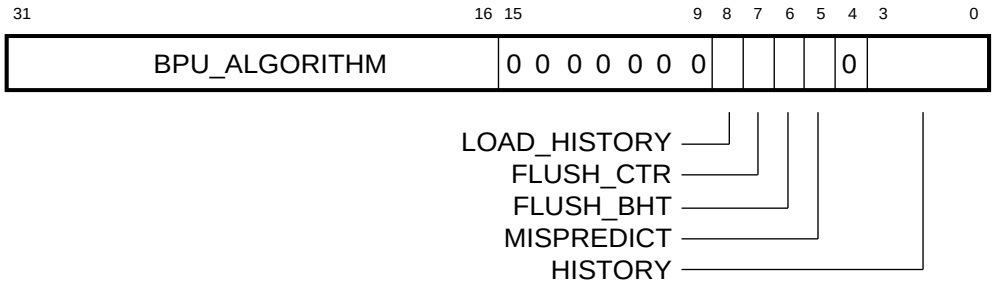
Table 2- 21 ICSR Register Bit Definitions

Name	Bit(s)	Type	Function
TPERR	<4>	RO	**Tag Parity Error**. Sets when a VIC tag parity error occurs in the tag array.
DPERR	<3>	RO	**Data Parity Error**. Sets when a VIC data parity error occurs in the data array.
LOCK	<2>	WC	**Lock.** Sets when either ICSR<TPERR> or ICSR<DPERR> is set. The set state of this bit validates the status of those two error bits and prevents further modification of their status and the error address in the VMAR register. When clear, indicates that no VIC parity error has been recorded and allows updates of the ICSR register and the VMAR register.
VIC_ACC_ENB	<0>	R/W,0	**VIC Access Enable**. When set, allows access to the VIC.

BPCR—Branch Prediction Control Register

Address 00D4
Access R/W

The BPCR register provides control for the branch prediction unit and read/write access to the history array.



As part of the power-up sequence, the microcode will write FECA0000, which is the following bit pattern:

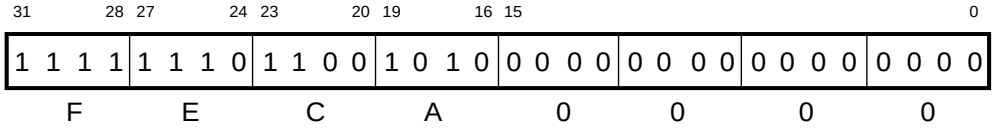


Table 2- 22 BPCR Register Bit Definitions

Name	Bit(s)	Type	Function
BPU_ALGORITHM	<31:16>	R/W	**Branch Prediction Unit Algorithm.** Controls direction of branch for given history.
LOAD_HISTORY	<8>	WO	**Load History.** Write of a one loads the history array into the location addressed by the BPCR address counter.
FLUSH_CTR	<7>	WO	**Flush Center.** Write of a one resets the BPCR address counter to zero. Cleared by hardware.
FLUSH_BHT	<6>	WO	**Flush Branch History.** Write of a one resets all history table entries to a neutral value. Cleared by hardware.
MISPREDICT	<5>	WO	**Mispredict.** Sets if last conditional branch mispredicted.
HISTORY	<3:0>	R/W	**History.** Branch history table entry bits.

BPCR Read/Write

The write- only BPCR<FLUSH_BHT> causes a BPU branch history table flush. The flush is identical to the context switch flush, which resets all branch table entries to a neutral value: HISTORY = 0100. The write- only BPCR<FLUSH_CTR> causes the Branch Table Counter bits <8:0> in the Ibox to be cleared. The Branch Table Counter provides an address into the branch table for IPR read and write accesses. Each IPR read from the BPCR or write to the BPCR with BPCR<LOAD_HISTORY> = 1 increments the counter. This allows IPR branch table reads and writes to step through the branch table array. LOAD_HISTORY enables writes to the branch history table. A write to the HISTORY field with BPCR<8> = 1 causes a BPU branch history table write. The history bits for the entry indexed by the counter are written with the IPR data. The BPCR register reads supply the history bits in BPCR<3:0> for the entry indexed by the counter. BPCR<MISPREDICT> returns a 1 if the last conditional branch mispredicted. BPCR<31:16> contains the branch prediction algorithm. Any IPR write to the BPCR register updates the algorithm. An IPR read returns the value of the current algorithm. For example, a 0 in BPCR<16> means that the next branch encountered will not be taken if the history is 0000. A 1 in BPCR<21> means that the next branch encountered when the prior history is 0101 will be taken.

BPCR bits <8>, <7>, and <6> are defined in Table 2- 23 for IPR writes to the BPCR register.

NOTE: The prediction algorithm is updated on every IPR write to the BPCR.

Table 2- 23 BPCR Write Actions

BPCR <8:6>	Write Action
000	Do nothing except update algorithm.
001	Flush branch table; history not written.
010	Address counter reset to zero; history not written.
011	Flush branch table; reset address counter; history not written.
100	Write history to table; counter automatically increments.
101	UNDEFINED. Branch table flushed; new history written; counter incremented.
110	UNDEFINED. Write history to old counter value; counter reset to zero.
111	UNDEFINED. Branch table flushed; write history to old counter value; counter reset to zero.

2.11.3 Ebox Registers

Table 2- 24 lists the Ebox registers. Functional descriptions of individual Ebox registers follow.

Table 2- 24 Ebox Registers

Name	Mnemonic	Type	IPR Address (Hex)
Patchable Control Store Control Register	PCSCR	WO	7C
Ebox Control Register	ECR	R/W	7D

PCSCR—Patchable Control Store Control Register

Address 007C
Access WO

The PCSCR register is used to load control store patches. This register is not used in normal operation.

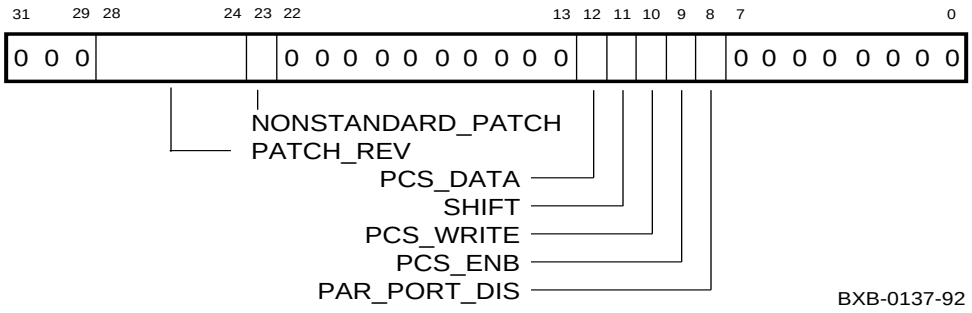


Table 2- 25 PCSCR Register Bit Definitions

Name	Bit(s)	Type	Function
PATCH_REV	<28:24>	R/W	**Patch Revision.** Updated by software after loading a microcode patch. Indicates the revision of the standard microcode patch that has been loaded.
NONSTANDARD_PATCH	<23>	WO	**Nonstandard Patch.** When set, indicates a nonstandard microcode patch has been loaded.
PCS_DATA	<12>	WO, 0	**Patchable Control Store Data.** Loaded with the data bit to be shifted into PCS read/write chain for writing to PCS RAM or CAM. The shift occurs whenever PCSCR<11> is written with one. By repeatedly loading this bit with one or zero, and setting PCSCR<11> after each load, any data pattern can be written into the PCS read/write chain.
SHIFT	<11>		**Shift. A** write of one loads the PCS_DATA bit into the PCS R/W chain and shifts the PCS read/write chain by one.

Table 2- 25 PCSCR Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
PCS_WRITE	<10>	WO, 0	**Patchable Control Store Write.** A write of one loads the contents of the PCS read/write chain to the PCS RAM and CAM.
PCS_ENB	<9>	WO, 0	**Patchable Control Store Enable**. A write of one enables PCS outputs, so that patches supersede the control store ROM.
PAR_PORT_DIS	<8>	WO, 0	**Parallel Port Disable.** A write of one disables control of load/shift operations to the PCS read/write chain.

ECR—Ebox Control Register

Address 007D
Access R/W

The ECR register is used to configure certain Ebox functions.

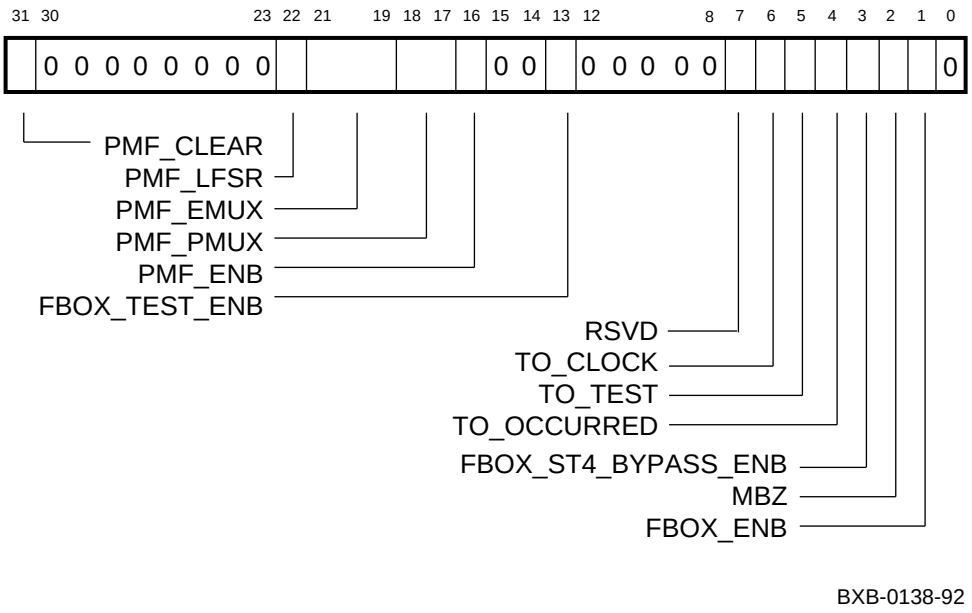


Table 2- 26 Ebox Control Register Bit Definitions

Name	Bit(s)	Type	Function
PMF_CLEAR	<31>	WO, 0	**Performance Monitor Facility Clear.** Writing one to this position clears the performance monitor facility counters. This function is handled by microcode.
PMF_LFSR	<22>	R/W, 0	**PMF Linear Feedback Shift Register.** When set, enables the LFSR accumulator. This is a testability feature.
PMF_EMUX	<21:19>	R/W, 0	**PMF EMUX.** Selects the Ebox events counted by the performance monitor facility, when the PMF is configured to count Ebox events.

Table 2- 26 Ebox Control Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
PMF_PMUX	<18:17>	R/W, 0	**PMF PMUX.** Selects the source of events counted by the performance monitor facility to be the Ibox, Ebox, Mbox, or the Cbox.
PMF_ENB	<16>	R/W, 0	**PMF Enable.** This bit is the internal implementation of the PME processor register.
FBOX_TEST_ENB	<13>	R/W, 0	**Fbox Testability Enable.** When set, places the Fbox in test mode. in which data is passed from stage to stage unaltered.
TO_CLOCK	<6>	R/W, 0	**Timeout Clock.** This is the most significant bit of the timeout base counter. When timeout is functioning, this bit should be one half of the time and zero the other half of the time.
TO_TEST	<5>	R/W, 0	**Timeout Test.** When set, the S3 stall timeout time is roughly 50 microseconds instead of roughly 3 seconds.
TO_OCCURRED	<4>	W1C, 0	**Timeout Occurred.** Set when a S3 stall timeout occurs. Cleared by a write of one.
FBOX_ST4_ BYPASS_ENB	<3>	R/W, 0	**Fbox Stage 4 Bypass Enable.** Set by configuration code to enable Fbox stage 4 bypass.
MBZ	<2>	R/W, 0	**Must Be Zero.** Must always be written as zero. Writing one disables S3 timeouts.
FBOX_ENB	<1>	R/W, 0	**Fbox Enable.** Set by configuration code to enable the Fbox.

2.11.4 Mbox Registers

Table 2- 27 lists the Mbox registers. Functional descriptions of individual Mbox registers follow.

Table 2- 27 Mbox Registers

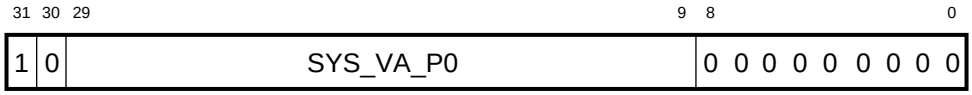
Name	Mnemonic	Type	IPR Address (Hex)
Mbox P0 Base Register [1]	MP0BR	R/W	E0
Mbox P0 Length Register[1]	MP0LR	R/W	E1
Mbox P1 Base Register[1]	MP1BR	R/W	E2
Mbox P1 Length Register[1]	MP1LR	R/W	E3
Mbox System Base Register[1]	MSBR	R/W	E4
Mbox System Length Register[1]	MSLR	R/W	E5
Mbox Map Enable Register[1]	MMAPEN	R/W	E6
Address Mode Register	PAMODE	R/W	E7
MME[2] Address Register[1]	MMEADR	RO	E8
MME PTE Address Register[1]	MMEPTE	RO	E9
MME Status Register[1]	MMESTS	RO	EA
TB Parity Address Register	TBADR	RO	EC
TB Parity Status Register	TBSTS	R/W	ED
P- Cache Parity Address Register	PCADR	RO	F2
P- Cache Parity Status Register	PCSTS	R/W	F4
P- Cache Control Register	PCCTL	R/W	F8
P- Cache Tag Registers	PCTAG	R/W	0180 0000 to 0180 1FE0
P- Cache Data Parity Registers	PCDAP	R/W	01C0 0000 to 01C0 1FF8

[1] Testability and diagnostic use only; not for software use in normal operation.
[2] Memory management exception.

MP0BR—Mbox P0 Base Register

Address 00E0
Access R/W

The MP0BR register contains the base address of the P0 region of the process space.



BXB-0139-92

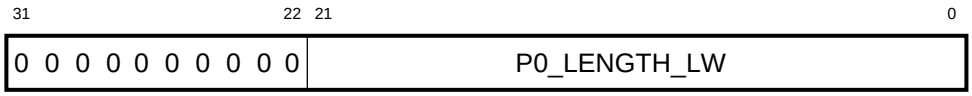
Table 2- 28 MP0BR Register Bit Definitions

Name	Bit(s)	Type	Function
SYS_VA_P0	<29:9>	R/W	**System Virtual Address of P0.** Contains the system virtual address of the P0 page table.

MP0LR—Mbox P0 Length Register

Address 00E1
Access R/W

The MP0LR register contains the size of the P0 region of the process space.



BXB-0140-92

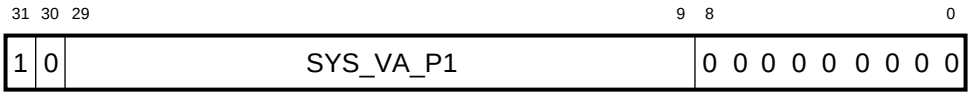
Table 2- 29 MP0LR Register Bit Definitions

Name	Bit(s)	Type	Function
P0_LENGTH_LW	<21:0>	R/W	**P0 Length Longwords.** Contains the length of the P0 page table in longwords.

MP1BR—Mbox P1 Base Register

Address 00E2
Access R/W

The MP1BR register contains the base address of the P1 region of the process space.



BXB-0141-92

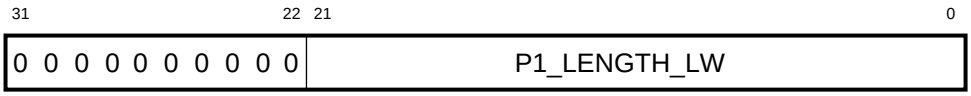
Table 2- 30 MP1BR Register Bit Definitions

Name	Bit(s)	Type	Function
SYS_VA_P1	<29:9>	R/W	**System Virtual Address of P1.** Contains the system virtual address of the P1 page table.

MP1LR—Mbox P1 Length Register

Address 00E3
Access R/W

The MP1LR register contains the size of the P1 region of the process space.



BXB-0142-92

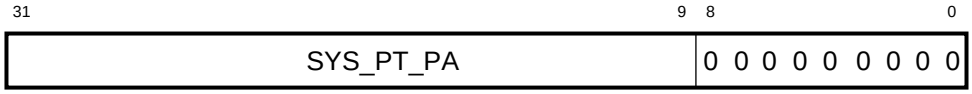
Table 2- 31 MP1LR Register Bit Definitions

Name	Bit(s)	Type	Function
P1_LENGTH_LW	<21:0>	R/W	**P1 Length Longwords.** Contains the length of the P1 page table in longwords.

MSBR—Mbox System Base Register

Address 00E4
Access R/W

The MSBR register contains the physical address of the system page table.



BXB-0143-92

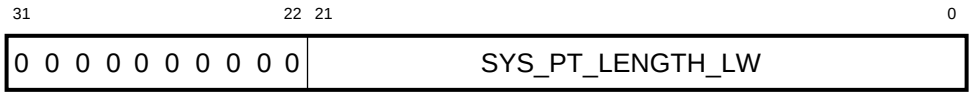
Table 2- 32 MSBR Register Bit Definitions

Name	Bit(s)	Type	Function
SYS_PT_PA	<31:9>	R/W	**System Page Table Physical Address.** Contains the physical address of the system page table in longwords.

MSLR—Mbox System Length Register

Address 00E5
Access R/W

The MSLR register contains the length of the system page table in longwords.



BXB-0144-92

Table 2- 33 MSLR Register Bit Definitions

Name	Bit(s)	Type	Function
SYS_PT_LENGTH_LW	<21:0>	R/W	**System Length Longwords.** Contains the length of the system page table in longwords.

MMAPEN—Mbox Map Enable Register

Address 00E6
Access R/W

The MMAPEN register contains the bit that enables/disables memory management.

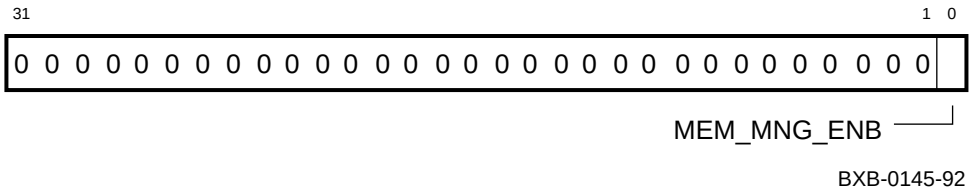


Table 2- 34 MMAPEN Register Bit Definition

Name	Bit	Type	Function
MEM_MNG_ENB	<0>	R/W	**Memory Management Enable.** When set, enables Mbox memory management. When clear, disables memory management.

PAMODE—Physical Address Mode Register

Address 00E7
Access R/W

The PAMODE register selects the size of the physical address space to be used in mapping virtual addresses to physical addresses.

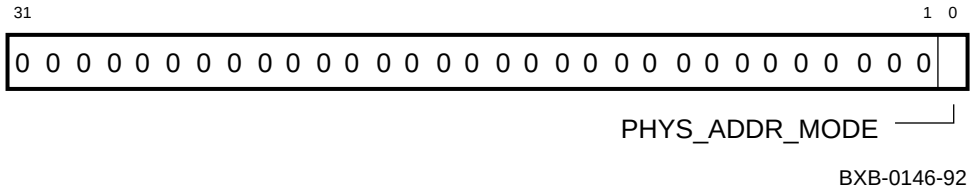


Table 2- 35 PAMODE Register Bit Definition

Name	Bit	Type	Function
PHYS_ADDR_MODE	<0>	R/W	**Physical Address Mode.** When set, NVAX+ maps addresses from a 32- bit physical address space. When clear, NVAX+ maps addresses from a 30- bit physical address space.

MMEADR—MME Address Register

Address 00E8
Access RO

The MMEADR register contains the address associated with the MME fault.

BXB-0147-92

Table 2- 36 MMEADR Register Bit Definitions

Name	Bit(s)	Type	Function
MME_FAULT_ADDR	<31:0>	RO	**MME Fault Address.** Contains the address associated with the MME fault.

MMEPTE—MME PTE Address Register

Address 00E9
Access RO

The MMEPTE register contains the PTE address associated with an address corresponding to a modify fault.

BXB-0148-92

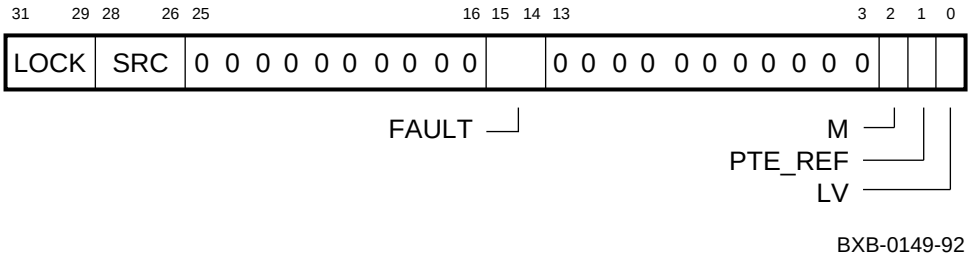
Table 2- 37 MMEPTE Address Register Bit Definitions

Name	Bit(s)	Type	Function
MOD_FAULT_PTE_ ADDR	<31:0>	RO	**Modify Fault PTE Address.** Contains the PTE address associated with an address corresponding to a modify fault.

MMESTS—MME Status Register

Address 00EA
Access R/W

The MMESTS register reports faults or violations in memory management operations.



BXB-0149-92

Table 2- 38 MMESTS Register Bit Definitions

Name	Bit(s)	Type	Function
LOCK	<31:29>	RO	**Lock.** Indicates the lock status of the MMESTS register. This field is shadowed by MMESTS<SRC> bits.
SRC	<28:26>	RO	**Source.** Complemented shadow copy of LOCK bits. However, the SRC bits do not get cleared when the LOCK bits are cleared.

MMESTS <31:29>	Definition
000	MMESTS, MMEADR, MMEPTE are unlocked.
001	Valid IREAD error is stored.
011	Valid Ibox specifier error is stored.
111	Valid Ebox reference error is stored.

Table 2- 38 MMESTS Register Bit Definitions (Continued)

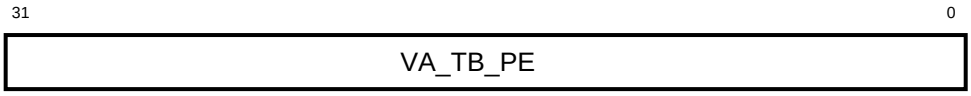
Name	Bit(s)	Type	Function
FAULT	<15:14>	RO	**Fault.** Indicates type of memory management fault.
M	<2>	RO	**Modify.** Indicates corresponding reference had write or modify intent.
PTE_REF	<1>	RO	**Page Table Entry Reference.** Indicates ACV/TNV fault occurrence on PTE reference corresponding to the MMEADR register.
LV	<0>	RO, 0	**Length Violation.** Indicates ACV fault occurrence due to length violation.

MMESTS <15:14>	Fault
01	ACV fault. Highest priority fault in the presence of multiple simultaneous faults.
10	TNV fault. Next priority fault to ACV.
11	M=0 fault. Lowest priority fault.

TBADR—Translation Buffer Parity Address Register

Address 00EC
Access R/W

The TBADR register contains the address of the reference that caused the TB parity error.



BXB-0150-92

Table 2- 39 TBADR Register Bit Definitions

Name	Bit(s)	Type	Function
VA_TB_PE	<31:0>	RO	**Virtual Address TB Parity Error.** Contains the virtual address of the reference that caused the translation buffer parity error.

TBSTS—Translation Buffer Parity Status Register

Address 00ED
Access R/W

The TBSTS register reports the status of the TB parity error.

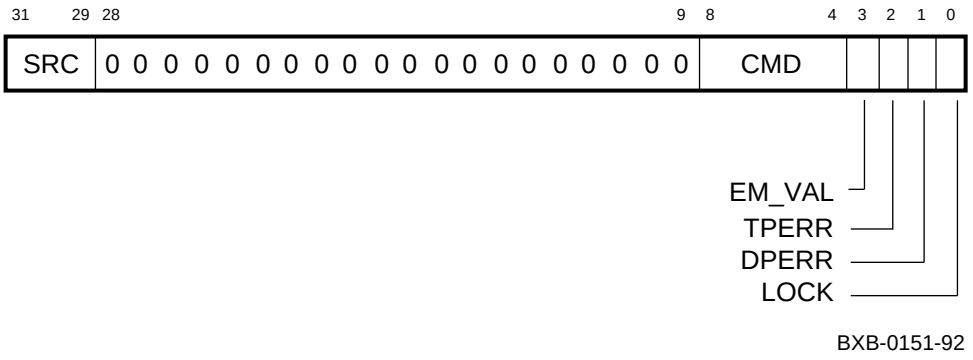


Table 2- 40 TBSTS Register Bit Definitions

Name	Bit(s)	Type	Function
SRC	<31:29>	RO	**Source.** Indicates the original source of the reference causing the TB parity error.
CMD	<8:4>	RO	**Command.** S5 command corresponding to TB parity error.

TBSTS <31:29>	Definition
111	Valid Mbox reference error is stored.
110	Valid IREAD error is stored.
100	Valid Ibox specifier error is stored.
000	Valid Ebox reference error is stored.

Table 2- 40 TBSTS Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
EM_VAL	<3>	RO	**EM Latch Valid.** Indicates if EM_LATCH was valid at the time of the TB parity error detection. This helps the software error handler determine if a write operation may have been lost due to the TB parity error.
TPERR	<2>	RO	**Tag Parity Error.** Set when a TB tag parity error occurs.
DPERR	<1>	RO	**Data Parity Error.** Set when a TB data parity error occurs.
LOCK	<0>	W1C,0	**Lock**. When set, validates TBSTS contents and prevents any other field from further modification. When clear, indicates that no TB parity error or PTE error has been recorded, and allows updates of the TBSTS and TBADR registers.

PCADR—P- Cache Parity Error Address Register

Address 00F2
Access RO

The PCADR register contains the quadword physical address associated with the recorded P- cache parity error.

BXB-0152-92

Table 2- 41 PCADR Register Bit Definitions

Name	Bit(s)	Type	Function
PC_PE_PA	<31:0>	RO	**P- Cache Parity Error Physical Address.** Contains the quadword physical address associated with the recorded P- cache parity error.

PCSTS—P- Cache Parity Status Register

Address 00F4
Access R/W

The PCSTS register reports information about the P- cache parity error.

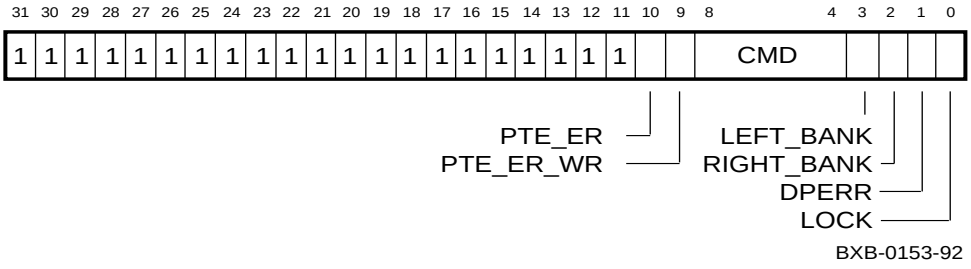


Table 2- 42 PCSTS Register Bit Definitions

Name	Bit(s)	Type	Function
PTE_ER	<10>	W1C, 0	**PTE Error.** Set when a hard error occurs on a PTE DREAD.
PTE_ER_WR	<9>	W1C, 0	**PTE Error Write.** Set when a hard error occurs on a PTE DREAD which resulted from a TB miss on a Write or Write_Unlock.
CMD	<8:4>	RO	**Command.** S6 command corresponding to P- cache parity error.
LEFT_BANK	<3>	RO	**Left Bank Tag Error.** Set when a P- cache tag parity error occurs on the left bank.
RIGHT_BANK	<2>	RO	**Right Bank Tag Error**. Set when a P- cache tag parity error occurs on the right bank.
DPERR	<1>	RO	**Data Parity Error**. Set when a P- cache data parity error occurs.
LOCK	<0>	W1C, 0	**Lock**. When set, validates PCSTS<8:1> contents and prevents modification of these fields. When clear, invalidates PCSTS<8:1> and allows these fields and PCADR to be updated.

PCCTL—P- Cache Control Register

Address 00F8
Access R/W

The PCCTL register controls the state of P- cache operations.

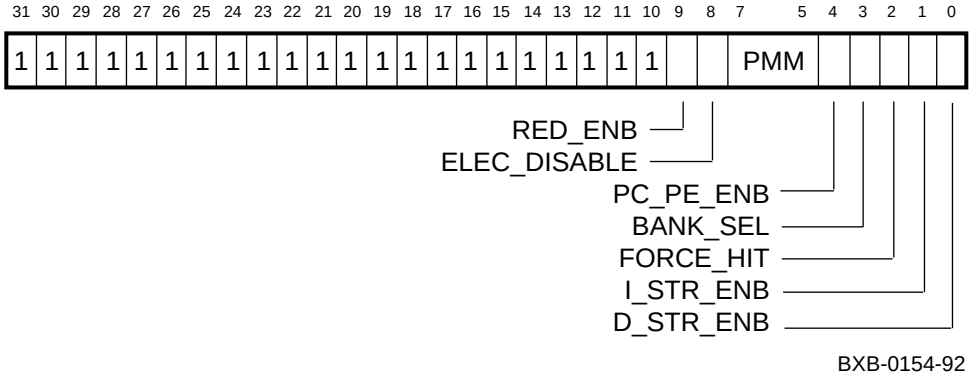


Table 2- 43 PCCTL Register Bit Definitions

Name	Bit(s)	Type	Function
RED_ENB	<9>	RO	**Redundancy Enable.** When set, indicates that one or more P- cache redundancy elements are enabled.
ELEC_DISABLE	<8>	R/W, 0	**Electrically Disable.** When set, the P- cache is disabled electrically to reduce power dissipation. This bit should only be set when the P- cache is functionally turned off by clearing of both I_STR_ENB and D_STR_ENB. UNPREDICTABLE operation results if this bit is set when I_STR_ENB or D_STR_ENB is also set. Note that P- cache tag parity IPRs do not function properly when ELEC_DISABLE is unconditionally set.
PMM	<7:5>	R/W,0	**Performance Monitor Mode.** Specifies Mbox performance monitor mode. This field does not control or affect the operation of the P- cache in any way.

Table 2- 43 PCCTL Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
PC_PE_ENB	<4>	R/W,0	**P- Cache Parity Error Enable.** When set, enables detection of P- cache tag and data parity errors. When clear, disables P- cache parity error detection.
BANK_SEL	<3>	R/W,0	**Bank Select.** When set with PCCTL<2> = 1, selects the "right bank" (Set 1) of the addressed P- cache index. When clear with PCCTL<2> = 1, selects the "left bank" (Set 0) of the addressed P- cache index. This bit is a don't care when PCCTL<2> = 0. Note that BANK_SEL does not affect bank selection during IPR reads and IPR writes to the P- cache tags or P- cache data parity bits. Bank selection for these operations is determined by the specified IPR address.
FORCE_HIT	<2>	R/W,0	**Force P- Cache Hit.** If set, forces a P- cache hit on all reads and writes when P- cache is enabled for I- or D- stream operation.
I_STR_ENB	<1>	R/W,0	**I- Stream Enable.** When set, enables P- cache processing of INVAL, IREAD, and I_CF commands. When clear, forces a P- cache miss on IREAD operations and prevents state modification due to an I_CF operation.
D_STR_ENB	<0>	R/W,0	**D- Stream Enable.** When set, enables P- cache for all INVAL operations and for all D- stream read/write/fill operations, qualified by other control bits. When clear, forces a P- cache miss on all P- cache D- stream read/write/fill operations. Note, however, that an ACV/TNV/M = 0 condition overrides a deasserted D_STR_ENB by forcing a P- cache hit condition with D_STR_ENB = 0.

PCTAG—P- Cache Tag Registers

Address 0180 0000 to 0180 1FE0
Access R/W

The PCTAG registers provide diagnostic access to the tag and the associated state bits of the P- cache.

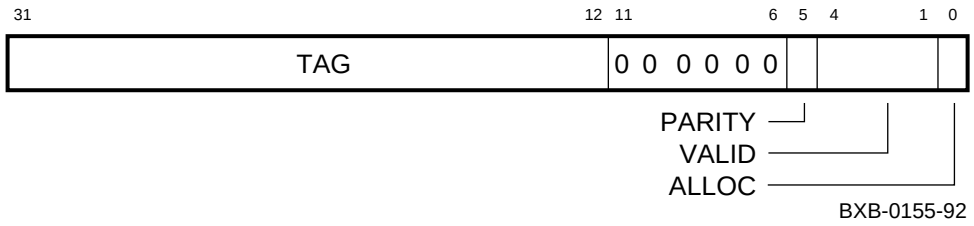


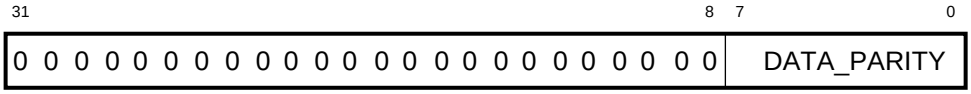
Table 2- 44 PCTAG Register Bit Definitions

Name	Bit(s)	Type	Function
TAG	<31:12>	R/W	**Tag.** Tag data.
PARITY	<5>	R/W	**Parity.** Even tag parity.
VALID	<4:1>	R/W	**Valid.** Valid bits corresponding to the four data subblocks. PCTAG<4> corresponds to uppermost quadword in the block; PCTAG<1> corresponds to lower-most quadword in the block.
ALLOC	<0>	R/W	**Allocation.** Allocation bit corresponding to index of this block.

PCDAP—P- Cache Data Parity Registers

Address 01C0 0000 to 01C0 1FF8
Access R/W

The PCDAP registers provide diagnostic access to the even byte parity corresponding to the addressed quadword of data.



BXB-0156-92

Table 2- 45 PCDAP Register Bit Definitions

Name	Bit(s)	Type	Function
DATA_PARITY	<7:0>	R/W	**Data Parity.** Even byte parity corresponding to addressed quadword of data.

2.11.5 Cbox Registers

The internal processor registers (IPRs) in the system as well as in the Cbox are accessed through IPR read commands and IPR write commands from the Mbox to the Cbox.

All IPR reads and writes to the Cbox flush the write queue before they are completed. Any IPR read sets conflict bits in all valid entries in the write queue so that all preceding writes of any kind complete before the IPR read. An IPR write is placed in the write queue after the preceding writes so that the ordering takes place naturally.

If a read arrives before an IPR write has been processed, the write queue conflict bits are set so that the write queue takes priority over the read. If the IPR read addresses one of the Cbox registers, the Cbox returns just one quadword or less of data, rather than the usual four quadwords.

If a write- only Cbox register is read, the Cbox returns UNPREDICTABLE data. Reading an unimplemented Cbox register also returns UNPREDICTABLE data. If a write is performed to an unimplemented register, the write is discarded by the Cbox and normal operation continues.

If the Cbox receives an IPR write to an address that is not within its block of IPR addresses, the Cbox discards the write and normal operation continues. If the Cbox receives an IPR read to an address that is not within its block of IPR addresses, the Cbox returns UNPREDICTABLE data.

Table 2- 46 lists the Cbox registers. Functional descriptions of individual Cbox registers follow.

Table 2- 46 Cbox Registers

Register	Mnemonic	Type	IPR Address (Hex)
Interval Count Control/Status Register	ICCS	R/W	18
Next Interval Count Register	NICR	WO	19
Interval Count Register	ICR	RO	1A
Time- of- Day Register	TODR	R/W	1B
BIU Control Register	BIU_CTL	R/W	A0
Diagnostic Control Register	DIAG_CTL	WO	A1
B- Cache Error Tag Register	BC_TAG	RO	A2
BIU Status Register	BIU_STAT	W1C	A4
BIU Address Register	BIU_ADDR	RO	A6
Fill Syndrome Register	FILL_SYND	RO	A8
Fill Address Register	FILL_ADDR	RO	AA
Software ECC Register	BEDECC	WO	AE
Console Halt Register	CHALT	R/W	B0

ICCS—Interval Count Control and Status Register

Address 0018
Access R/W

The ICCS register contains control and status information on the interval clock.

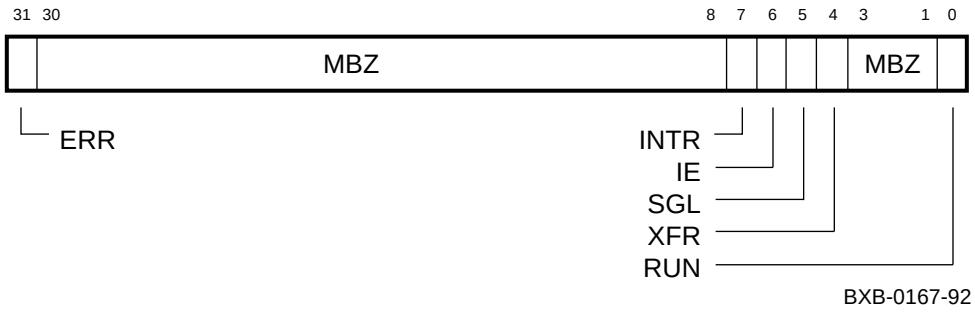


Table 2- 47 ICCS Register Bit Definitions

Name	Bit(s)	Type	Function
ERR	<31>	W1C, 0	**Error.** Set whenever the Interval Count Register overflows and INTR (ICCS<7>) is already set. Thus, ERR indicates a missed overflow.
MBZ	<30:8	R/W	**Must be zero.** Must always be written as zero.
INTR	<7>	W1C, 0	**Interrupt.** Set whenever the Interval Count Register overflows. If IE (ICCS<6>) is set when INTR is set, an interrupt is posted. Whenever the Interval Count Register overflows and INTR is already set, ERR (ICCS<31>) is set. Reset clears ICCS<6> and ICCS<0>, but leaves the rest of ICCS UNPREDICTABLE.
IE	<6>	R/W, 0	**Interrupt Enable.** When set, an interrupt request is generated on ICR overflows, that is, every time INTR (ICCS<7>) is set. When clear, no interrupt is requested. Similarly, if INTR is already set and the software sets IE, an interrupt is generated. This bit is cleared by reset.

Table 2- 47 ICCS Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
SGL	<5>	WO	**Single Step.** When RUN (ICCS<0>) is clear, writing one to SGL generates a pulse that causes the Interval Count Register to be incremented by one. When RUN is set, or if SGL and XFR (ICCS<4>) are written at the same time, the write to SGL is ignored. Multiple SGLs produce multiple increments. SGL does not require clearing. It always reads as zero.
XFR	<4>	WO	**Transfer.** Writing one to XFR generates a pulse that causes the NICR register to be copied to the ICR register. Multiple XFRs produce multiple transfers. XFR does not require clearing. It always reads as zero.
MBZ	<3:1>	R/W	**Must be zero.** Must always be written as zero.
RUN	<0>	R/W	**Run.** When set, the ICR register is incremented once per microsecond. When clear, the ICR register does not increment automatically. RUN is cleared during reset.

NICR—Next Interval Count Register

Address 0019
Access R/W

The NICR register contains the value to be loaded into the ICR register on count overflow or in response to a write of one to XFR (ICCS<4>).

BXB-0169-92

Table 2- 48 NICR Register Bit Definitions

Name	Bit(s)	Type	Function
NINT_COUNT	<31:0>	R/W	**Next Interval Count.** Contains the count that is loaded into the ICR register on overflow of ICR<31>. The value of NINT_COUNT is retained after the ICR register is loaded. NINT_COUNT is cleared by reset.

ICR—Interval Count Register

Address 001A
Access RO

The ICR register contains the interval count.

BXB-0168-92

Table 2- 49 ICR Register Bit Definitions

Name	Bit(s)	Type	Function
INT_COUNT	<31:0>	R/W	**Interval Count.** Contains the interval count. This field is incremented by a write of one to SGL (ICCS<5>) when RUN (ICCS<0>) is clear. When RUN is set, INT_COUNT is incremented by one every microsecond. Upon a carry- out (overflow) from ICR<31>, INT_COUNT is automatically loaded from the NICR register and continues incrementing. That is, the value of the ICR register on successive microseconds will be FFFF FFFD, FFFF FFFE, FFFF FFFF, value of NICR<31:0>. The counter overflow sets INTR (ICCS<7>), which generates an interrupt, if the interrupt is enabled (that is, ICCS<6> is set). INT_COUNT can specify a maximum delay of approximately 1.2 hours. It is cleared by reset.

TODR—Time- of- Day Register

Address 001B
Access R/W

The TODR register forms an unsigned 32- bit binary counter that is driven from a 100 Hz oscillator, so that the least significant bit of the clock represents a resolution of 10 milliseconds. The TODR register counts only when it contains a non- zero value.

BXB-0166-92

Table 2- 50 TODR Register Bit Definitions

Name	Bit(s)	Type	Function
TOD	<31:0>	RO	**Time- of- Day.** Unsigned counter that indicates the time of day with a resolution of 10 ms.

BIU_CTL—BIU Control Register

Address 00A0
Access R/W

The BIU_CTL register controls certain operations and parameters related to the P- cache, B- cache, and I/O mapping. This register reads the complement of its contents.

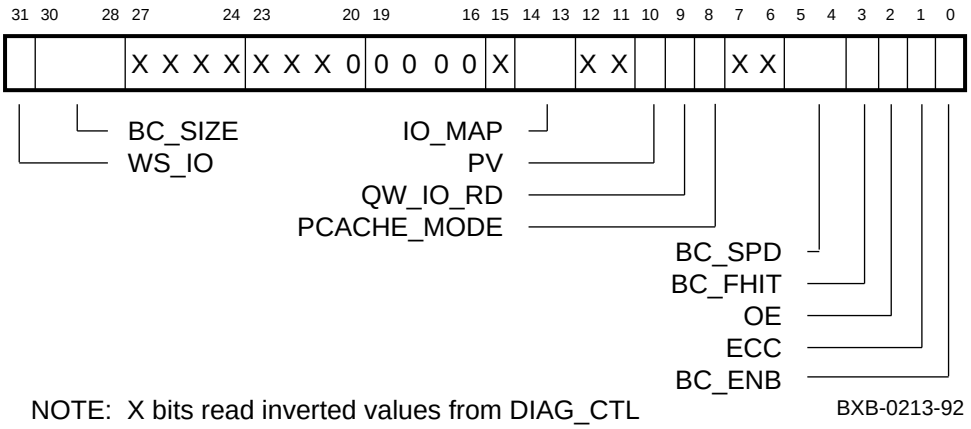


Table 2- 51 BIU_CTL Register Bit Definitions

Name	Bit(s)	Type	Function
WS_IO	<31>	R/W	**Workstation I/O.** When set, indicates that I/O space is mapped for workstations.

Table 2- 51 BIU_CTL Register Bit Definitions (Continued)

<table>
<tr><th>Name</th><th>Bit(s)</th><th>Type</th><th>Function</th></tr>
<tr><td>BC_SIZE</td><td><30:28></td><td>R/W</td><td>B- Cache Size. Specifies the size of the B- cache as follows:

<table><tr><th>BIU_CTL <30:28></th><th>B- Cache Size</th></tr><tr><td>000</td><td>128 Kbytes</td></tr><tr><td>001</td><td>256 Kbytes</td></tr><tr><td>010</td><td>512 Kbytes</td></tr><tr><td>011</td><td>1 Mbyte</td></tr><tr><td>100</td><td>2 Mbytes</td></tr><tr><td>101</td><td>4 Mbytes</td></tr><tr><td>110</td><td>8 Mbytes</td></tr></table>
BC_SIZE is not initialized on reset. Therefore, it must be explicitly written before the B- cache is enabled.</td></tr>
<tr><td>IO_MAP</td><td><14:13></td><td>R/W</td><td>I/O Map. Used on I/O references to allow selection of the range for I/O mapping by different systems.</td></tr>
<tr><td>PV</td><td><10></td><td>R/W</td><td>PV System Mode. Affects write probes to the B-cache. When set, writes execute as if the B- cache were disabled; when clear, enables B- cache probes on writes. Read probes are not affected by the state of this bit. PV is set by hardware at initialization or on reset.</td></tr>
<tr><td>QW_IO_RD</td><td><9></td><td>R/W</td><td>Quadword I/O Read. When set, IO_SPACE DREADs that are not quadword- aligned return data from an internal register containing bits <63:32> of the previous quadword- aligned read.</td></tr>
<tr><td>PCACHE_MODE</td><td><8></td><td>R/W</td><td>P- Cache Mode. Controls P- cache mapping mode. When set, P- cache functions as 2- way set associative; when clear, P- cache functions as direct-mapped.</td></tr>
</table>

Table 2- 51 BIU_CTL Register Bit Definitions (Continued)

<table>
<tr><th>Name</th><th>Bit(s)</th><th>Type</th><th>Function</th></tr>
<tr><td>BC_SPD</td><td><5:4></td><td>R/W, 0</td><td>B- Cache Speed. Indicates to the BIU the read and write access time, in CPU cycles, of the RAMs used to implement the B- cache. NVAX+ uses the BC_SPD field to program the read and write cache access time.

<table><tr><th>BIU_CTL <5:4></th><th>B- Cache Speed</th></tr><tr><td>00</td><td>2 x CPU cycle</td></tr><tr><td>01</td><td>3 x CPU cycle</td></tr><tr><td>10</td><td>4 x CPU cycle</td></tr></table>
BC_SPD is initialized on reset to 2 x CPU cycles (00 bin).</td></tr>
<tr><td>BC_FHIT</td><td><3></td><td>R/W</td><td>B- Cache Force Hit. When this bit is set, and BC_ENB is also set, all READ_BLOCK and WRITE_BLOCK transactions are forced to hit in the B- cache. Tag and tag control parity are ignored when the BIU operates in this mode. BC_ENB takes precedence over BC_FHIT. When BC_ENB is clear and BC_FHIT is set, no tag probes occur.</td></tr>
<tr><td>OE</td><td><2></td><td>R/W</td><td>Output Enable. When set, NVAX+ does not assert its chip enable lines during RAM write cycles, thus allowing these lines to be connected to the output enable lines of the cache RAMs.</td></tr>
<tr><td>ECC</td><td><1></td><td>R/W</td><td>Error Correction and Control. When set, NVAX+ generates/expects ECC report. When clear, NVAX+ generates/expects parity report.</td></tr>
<tr><td>BC_ENB</td><td><0></td><td>R/W</td><td>B- Cache Enable. When clear, disables the B- cache. When the B- cache is disabled, the BIU does not probe the B- cache tag store for read and write references; it launches a cycle request immediately.</td></tr>
</table>

BIU_CTL Read/Write

A read of the BIU_CTL register provides the complement of its contents (inverted values of all bits). Bit positions not used for BIU_CTL functions reflect inverted values of the DIAG_CTL bits of the corresponding positions. Since the DIAG_CTL register is write only, its state can be determined by reading the BIU_CTL register.

The following examples show how to read and write to the BIU_CTL register through the console and how to decode BIU_CTL reads.

Example 2- 1 shows how to read the BIU_CTL register and decode its contents. The read provides bit states for both the BIU_CTL register and the DIAG_CTL register. Only BIU_CTL register bits are commented on.

Example 2- 1 BIU_CTL Read

```
>>>  e biu_ctl
ipr: 000000A0  (BIU_CTL)  AFE09FF8
>>>  !
>>>  ! The complement of BIU_CTL is 501F 6007
>>>  ! This decodes to the following state:
>>>  !
>>>  ! BC_ENB = B-cache enabled
>>>  ! ECC = ECC mode
>>>  ! OE = OE mode
>>>  ! BC_FHIT = Not B-cache force hit
>>>  ! BC_SPD = 2x CPU cycle B-cache speed
>>>  ! PCACHE_MODE = P-cache in 2-way set associative mode
>>>  ! QW_IO_RD = Not enabled
>>>  ! IO_MAP = I/O map is 11
>>>  ! BC_SIZE = B-cache size is 4 Mbytes
>>>  ! WS_IO = Not enabled
```

Example 2- 2 shows how to change B- cache speed from 2x CPU cycle time to 4x CPU cycle time.

Example 2- 2 BIU_CTL Write

```
>>>  e biu_ctl
ipr: 000000A0  (BIU_CTL)  AFE09FF8
>>>  !
>>>  ! The complement of BIU_CTL is 501F 6007
>>>  ! Change BC_SPD from 00 (bin) to 10 (bin) gives 501F 6027
>>>  ! Write this value to BIU_CTL
>>>  !
>>>  dep biu_ctl 501F6027
>>>  !
>>>  e biu_ctl
ipr: 000000A0  (BIU_CTL)  AFE0 9FD8
>>>  !
```

See the description of the DIAG_CTL register for examples on how to read and write to the DIAG_CTL register.

DIAG_CTL—Diagnostic Control Register

Address 00A1
Access WO

The DIAG_CTL register is used for diagnostics.

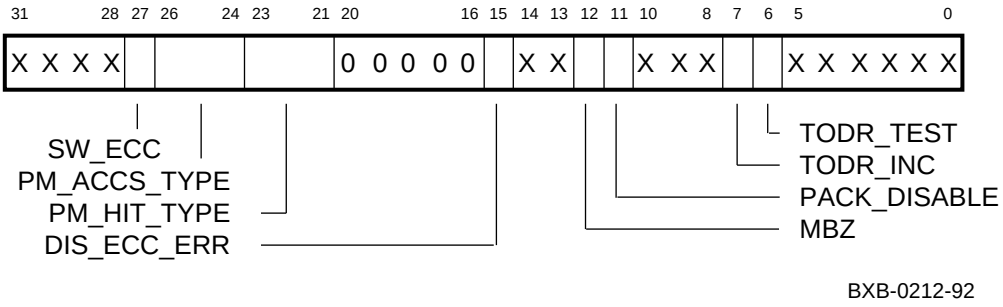


Table 2- 52 DIAG_CTL Register Bit Definitions

Name	Bit(s)	Type	Function
SW_ECC	<27>	WO, 0	**Software ECC.** When set, enables the use of ECC check bits from the BEDECC internal processor register as given by software for write data. If SW_ECC is set when parity mode is selected (BIU_CTL<1> = 0), then BEDECC<0> is the parity bit generated for data <31:0> and BEDECC<7> is the parity bit for data <64:32>.
PM_ACCS_TYPE	<26:24>	WO	**Performance Monitor Access Type.** Selects B- cache tag compare type for performance monitor access.

Table 2- 52 DIAG_CTL Register Bit Definition (Continued)

Name	Bit(s)	Type	Function
PM_HIT_TYPE	<23:21>	WO	**Performance Monitor Hit Type.** Selects B- cache tag compare type for performance monitor access.
DIS_ECC_ERR	<15>	WO, 1	**Disable ECC Error.** When set, disables ECC/data parity error reporting.
MBZ	<12>	WO, 0	**Must Be Zero.** Must always be written as zero.
PACK_DISABLE	<11>	WO	**Pack Disable.** When set, disables write packing, except for quadword packing, which is directed by microcode.
TODR_INC	<7>	WO	**Time- of- Day Register Increment.** When set, the contents of the TODR register are incremented for testing, provided DIAG_CTL<TODR_TEST> is also set.
TODR_TEST	<6>	WO	**Time- of- Day Register Test.** When set, places the TODR register in test mode.

DIAG_CTL Read/Write

Since the DIAG_CTL register is write only, it cannot be read directly. However, the contents of the DIAG_CTL register can be obtained indirectly by reading the BIU_CTL register. Note that the state of a DIAG_CTL bit appears inverted when the BIU_CTL register is read (see description of the BIU_CTL register).

The following examples show how to read and write to the DIAG_CTL register through the console, and how to decode DIAG_CTL reads (through BIU_CTL reads).

Example 2- 3 shows how to read the contents of the DIAG_CTL register. The read provides bit states for both the DIAG_CTL register and the BIU_CTL register. Only DIAG_CTL register bits are commented on.

Example 2- 3 DIAG_CTL Read

```
>>>  e biu_ctl
ipr: 000000A0  (BIU_CTL)  AFE09FF8
>>>  !
>>>  ! The complement of BIU_CTL is 501F 6007
>>>  ! TODR_TEST = Not enabled
>>>  ! TODR_INC = Not enabled
>>>  ! PCACHE_MODE = P-cache in 2-way set associative mode
>>>  ! QW_IO_RD = Not enabled
>>>  ! PACK_DISABLE = Write packing enabled
>>>  ! MBZ = Written as zero
>>>  ! DIS_ECC_ERR = ECC reporting enabled
>>>  ! PM_HIT_TYPE = PMCNTR1 enabled for B-cache hit counting
>>>  ! PM_ACCS_TYPE = PMCNTR0 enabled for B-cache access counting
>>>  ! SW_ECC = Not enabled
```

Example 2- 4 shows how to change reporting of ECC errors from enabled to disabled.

Example 2- 4 DIAG_CTL Write

```
>>>  e biu_ctl
ipr: 000000A0  (BIU_CTL)  AFE0 9FF8
>>>  !
>>>  ! The complement of BIU_CTL is 501F 6007
>>>  ! Set DIS_ECC_ERR gives 501F E007
>>>  ! Write this value to DIAG_CTL
>>>  !
>>>  dep diag_ctl 501FE007
>>>  !
>>>  e biu_ctl
ipr: 000000A0  (BIU_CTL)  AFE0 1FF8
>>>  !
```

See the description of the BIU_CTL register for examples on how to read and write to the BIU_CTL register.

BC_TAG—B- Cache Error Tag Register

Address 00A2
Access RO

The BC_TAG register is loaded with the results of every B- cache tag probe. When a tag or tag control parity error or primary fill data error (parity or ECC) occurs, this register is locked against further updates. The BC_TAG register is unlocked when the BIU_STAT<3:2> field is cleared. Software can read this register by using the MFPR instruction.

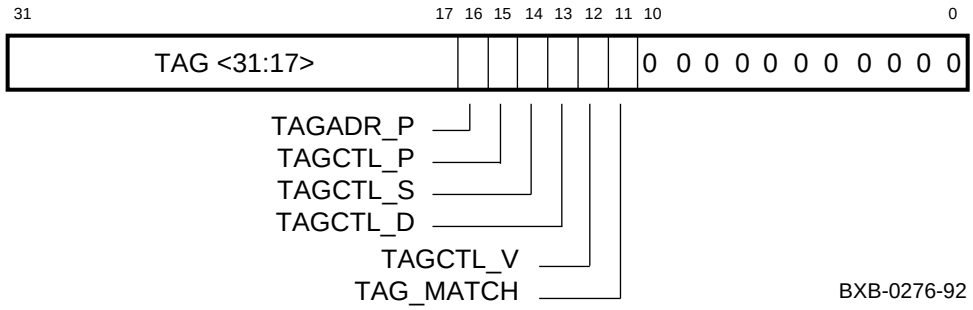


Table 2- 53 BC_TAG Register Bit Definitions

9	

Name	Bit(s)	Type	Function
TAG	<31:17>	RO	**Tag.** Contains the tag that is being probed currently.
TAGADR_P	<16>	RO	**Tag Address Parity.** Reflects the state of the TAGP_H signal of the NVAX+ chip when a tag, tag control, or data parity error occurs.
TAGCTL_P	<15>	RO	**Tag Control Parity.** Reflects the state of the TAGCTLP_H signal of the NVAX+ chip when a tag, tag control, or data parity error occurs.
TAGCTL_S	<14>	RO	**Tag Control Shared.** Reflects the state of the TAGCTLS_H signal of the NVAX+ chip when a tag, tag control, or data parity error occurs.
TAGCTL_D	<13>	RO	**Tag Control Dirty.** Reflects the state of the TAGCTLD_H signal of the NVAX+ chip when a tag, tag control, or data parity error occurs.
TAGCTL_V	<12>	RO	**Tag Control Valid.** Reflects the state of the TAGCTLD_H signal of the NVAX+ chip when a tag, tag control, or data parity error occurs.
TAG_MATCH	<11>	RO	**Tag Match.** When set, indicates that there was a tag match when a tag, tag control, or data parity error occurred.

BIU_STAT—BIU Status Register

Address 00A4
Access W1C

The BIU_STAT register reports error conditions that occur at the bus interface unit. This register is not unlocked or cleared by reset. It must be cleared explicitly by microcode.

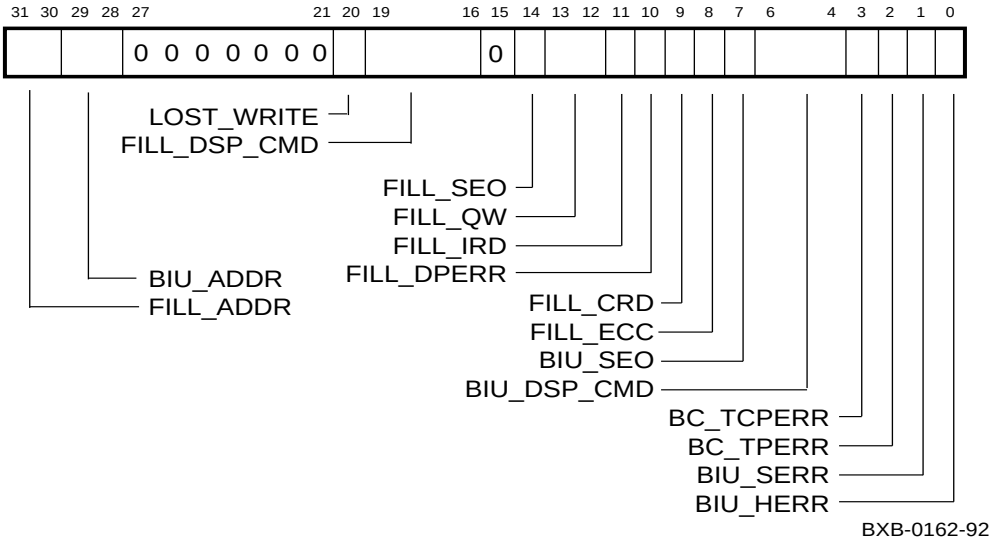


Table 2- 54 BIU_STAT Register Bit Definitions

Name	Bit(s)	Type	Function
FILL_ADDR<33:32>	<31:30>	RO	**Fill Address <33:32>.** Bits <33:32> of the FILL_ADDR register. Should be set only for I/O space address. This field is locked against further updates when FILL_ADDR<31:5> is locked.
BIU_ADDR<33:32>	<29:28>	RO	**BIU Address <33:32>.** Bits <33:32> of the BIU_ADDR register. Should be set only for I/O space address. The field is locked against further updates when BIU_ADDR<31:5> is locked.
LOST_WRITE	<20>	W1C, 0	**Lost Write.** A second error, and command is write.

Table 2- 54 BIU_STAT Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
FILL_DSP_CMD	<19:16>	RO	**Fill Dispatch Command.** This field latches the DSP_CMD (dispatch command) that resulted in the BIU error and locks till BIU_STAT<14, 10:8> are cleared.

BIU_STAT <19:16>	Dispatch Command
100X	DREAD
1010	DREAD_IO
1100	DREAD_LOCK
1101	DREAD_LOCK_IO
0010	IREAD
0011	IREAD_IO
0111	WRITE_UNLOCK
0110	WRITE
0101	IO_WRITE
0001	WRITE_UNLOCK_IO

FILL_SEO	<14>	W1C, 0	**Fill SEO.** When set, indicates that a P- cache fill operation resulted in either an uncorrectable ECC error or in a parity error while FILL_ECC or FILL_DPERR was already set.
FILL_QW	<13:12>	RO	**Fill Quadword.** Identifies the quadword within the hexword P- cache fill block that caused the error. It can be used together with FILL_ADDR<33:5> to get the complete physical address of the quadword in error. This field is decoded as follows:

BIU_STAT <13:12>	Quadword in Error
00	1
01	2
10	3
11	4

FILL_QW is only meaningful when either FILL_ECC or FILL_DPERR is set.

Table 2- 54 BIU_STAT Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
FILL_IRD	<11>	RO	**Fill I- Cache Read.** When set, indicates that the error that caused FILL_ECC or FILL_DPERR to set occurred during an I- cache fill. When clear, it indicates that the error occurred during a D- cache fill. FILL_IRD is only relevant when either FILL_ECC or FILL_DPERR is set.
FILL_DPERR	<10>	W1C, 0	**BIU Parity Error**. When set, indicates that the BIU received data with a parity error from outside the CPU chip while performing either a D- cache or an I- cache fill. FILL_DPERR is only relevant when the CPU chip is in parity mode, as opposed to ECC mode.
FILL_CRD	<9>	W1C, 0	**Corrected Read.** When set, indicates that the ECC error that caused FILL_ECC to set was correctable. When clear, it indicates that the error was not correctable. FILL_CRD is only relevant when FILL_ECC is also set.
FILL_ECC	<8>	W1C, 0	**ECC Error.** When set, indicates that the P- cache fill data received from outside the CPU chip contained an ECC error.
BIU_SEO	<7>	W1C, 0	**BIU SEO.** When set, indicates that an external cycle was terminated with a hard error or that a B- cache probe encountered bad parity in the tag address or tag control RAMs while BIU_HERR, BIU_SERR, BC_TPERR, or BC_TCPERR was already set.

Table 2- 54 BIU_STAT Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
BIU_DIS_CMD	<6:4>	RO	**BIU Dispatch Command.** This field latches dispatch command bits <3:1>, inverting bit <1> if the command is WRITE_UNLOCK, when a BIU_HERR, BIU_SERR, BC_TPERR, or BC_TCPERR error occurs, and locks till BIU_STAT<7, 3:0> are cleared.
BC_TCPERR	<3>	W1C, 0	**B- Cache Tag Control Parity Error.** When set, indicates that a B- cache tag probe encountered bad parity in the tag control RAM.
BC_TPERR	<2>	W1C, 0	**B- Cache Tag Parity Error.** When set, indicates that a B- cache tag probe encountered bad parity in the tag address RAM.
BIU_SERR	<1>	W1C, 0	**BIU Soft Error.** When set, indicates that an external cycle was terminated with a soft (recoverable) error.
BIU_HERR	<0>	W1C, 0	**BIU Hard Error.** When set, indicates that an external cycle was terminated with a hard (nonrecoverable) error.

BIU_STAT <6:4>	Dispatch Command
100	DREAD
101	DREAD_IO
110	DREAD_LOCK
110	DREAD_LOCK_IO
001	IREAD
001	IREAD_IO
011	WRITE_UNLOCK
010	WRITE
010	IO_WRITE
000	WRITE_UNLOCK_IO

BIU_STAT Updates

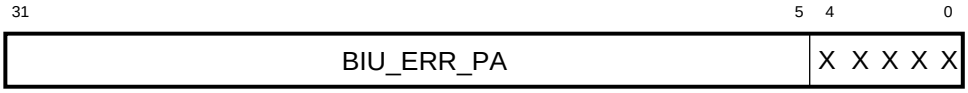
When any of bits BIU_HERR, BC_SERR, BC_TPERR, or BC_TCPERR is set, BIU_STAT<6:0> are locked against further updates. The address associated with the error is latched and locked in the BIU_ADDR register. BIU_STAT<7:0> and the BIU_ADDR register are unlocked when BIU_STAT<7, 3:0> are written with ones.

When FILL_ECC or FILL_DPERR is set, BIU_STAT<13:8> are locked against further updates. The address associated with the error is latched and locked in the FILL_ADDR register. BIU_STAT<14:0> and the FILL_ADDR register are unlocked when BIU_STAT<14, 11:8> are written with ones.

BIU_ADDR—BIU Address Register

Address 00A6
Access RO

The BIU_ADDR register contains the physical address associated with any errors reported in the BIU_STAT register. This register is locked against further updates upon detection of an error and is unlocked when it is read by the processor.



BXB-0186-92

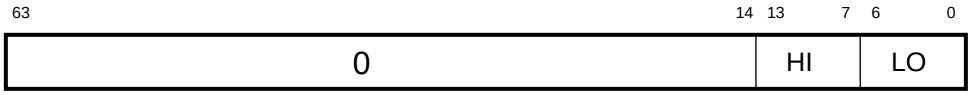
Table 2- 55 BIU_ADDR Register Bit Definitions

Name	Bit(s)	Type	Function
BIU_ERR_PA	<31:5>	RO	**BIU Error Physical Address.** Contains the physical address of the reference that caused the error reported in the BIU_STAT register. BIU_ERR_PA is locked against further updates upon detection of an error and is unlocked when it is read by the processor.

FILL_SYND—Fill Syndrome Register

Address 00A8
Access RO

The FILL_SYND register contains the syndrome bits associated with the quadword in error if the NVAX+ chip is in ECC mode and an ECC error is recognized during a P- cache fill operation. The contents of the FILL_SYND register remain locked with the quadword in error until the processor reads the FILL_ADDR register. The FILL_SYND register is 14 bits wide.



BXB-0163-92

Table 2- 56 FILL_SYND Register Bit Definitions

Name	Bit(s)	Type	Function
HI	<13:7>	RO	**High.** Latches the ECC syndrome bits for the high longword.
LO	<6:0>	RO	**Low.** Latches the ECC syndrome bits for the low longword.

A syndrome of 0 indicates no ECC error for the associated longword. Table 2- 57 outlines syndromes for all single- bit errors. Any non- zero syndrome not listed in the table indicates a double- bit error.

Table 2- 57 Syndromes for Single- Bit Errors

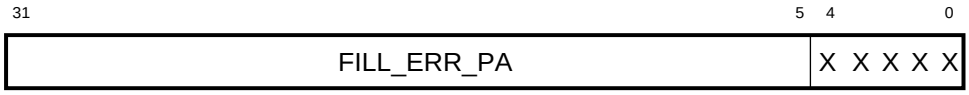
Bit	Syndrome (Hex) [1]	Bit	Syndrome (Hex)
Data<0>	4F	Data<20>	16
Data<1>	4A	Data<21>	19
Data<2>	52	Data<22>	1A
Data<3>	54	Data<23>	1C
Data<4>	57	Data<24>	62
Data<5>	58	Data<25>	64
Data<6>	5B	Data<26>	67
Data<7>	5D	Data<27>	68
Data<8>	23	Data<28>	6B
Data<9>	25	Data<29>	6D
Data<10>	26	Data<30>	70
Data<11>	29	Data<31>	75
Data<12>	2A	ECC<0>	01
Data<13>	2C	ECC<1>	02
Data<14>	31	ECC<2>	04
Data<15>	34	ECC<3>	08
Data<16>	0E	ECC<4>	10
Data<17>	0B	ECC<5>	20
Data<18>	13	ECC<6>	40
Data<19>	15		

[1] Any non- zero syndrome of 0 not listed in this table indicates a double- bit error.

FILL_ADDR—Fill Address Register

Address 00AA
Access RO

The FILL_ADDR register contains the physical address associated with any errors reported in the BIU_STAT<14:8>. This register is locked against further updates upon detection of an error and is unlocked when it is read by the processor.



BXB-0187-92

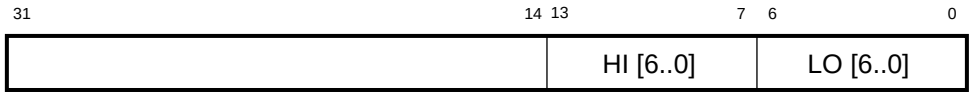
Table 2- 58 FILL_ADDR Register Bit Definitions

Name	Bit(s)	Type	Function
FILL_ERR_PA	<31:5>	RO	**Fill Error Physical Address.** Contains the physical address of the reference that caused the error reported in the BIU_STAT<14:8>. FILL_ERR_PA is locked against further updates upon detection of an error and is unlocked when it is read by the processor.

BEDECC—Software ECC Register

Address 00AE
Access WO

The BEDECC register is a 14- bit write- only register. If BIU_CTL<SW_ECC> = 1, the check bits for write data are sourced from the BEDECC register instead of the normal check bit generation logic.



BXB-0262-92

Table 2- 59 BEDECC Register Bit Definitions

Name	Bit(s)	Type	Function
HI	<13:7>	WO	**High.** Check bits of data<63:32>.
LO	<6:0>	WO	**Low.** Check bits of data<31:0>.

CHALT—Console Halt Register

Address 00B0
Access R/W

The CHALT register contains the base address of the console program. Microcode uses this register to determine where to start executing the console program whenever a console halt occurs as a result of an error condition or operator intervention (Ctrl/P typed at the terminal keyboard).

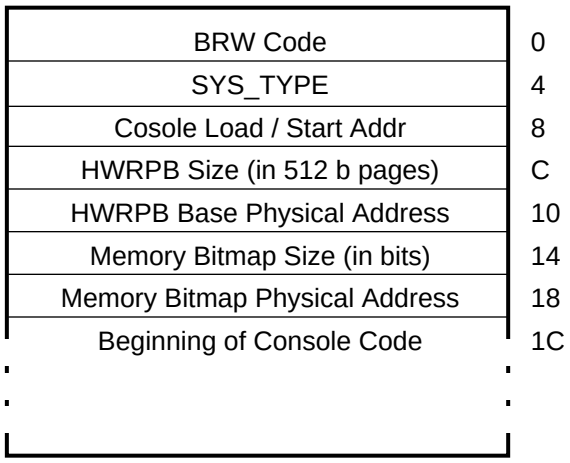
BXB-0170-92

Table 2- 60 CHALT Register Bit Definitions

Name	Bit(s)	Type	Function
CON_BASE_ADDR	<31:0>	R/W	**Console Base Address.** Contains the console base address that is used by microcode to determine the starting address of the console program on a console halt. CON_BASE_ADDR is loaded at power- up by the console code.

Figure 2- 22 shows the console dispatch data structure.

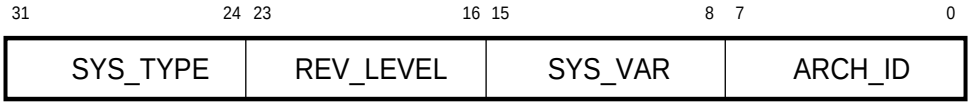
Figure 2- 22 Console Dispatch Data Structure



BXB-0263-92

The SYS_TYPE line contains information that distinguishes different NVAX+ based module implementations. Figure 2- 23 shows the parameters of the SYS_TYPE line.

Figure 2- 23 SYS_TYPE Parameters



BXB-0181-92

Table 2- 61 gives the encoding of the SYS_TYPE parameters in the console dispatch data structure.

Table 2- 61 SYS_TYPE Parameter Definitions

Name	Bit(s)	Type	Function
SYS_TYPE	<31:24>	RO	**System Type.** Indicates the type of system: 0 for VAX 7000 and 1 for VAX 10000.
REV_LEVEL	<23:16>	RO	**Revision Level.** Contains the revision number of the CPU module's console firmware. For the KA7AA CPU module, this field contains 01 (hex).
SYS_VAR	<15:8>	RO	**System Variant.** Distinguishes variants of similar systems. For the KA7AA CPU module, this field contains 01 (hex).
ARCH_ID	<7:0>	RO	**Architectural ID.** Licensing bits that distinguish timesharing systems from workstations. For the KA7AA CPU module, this field contains 01 (hex).

Chapter 3

Cache Subsystem

The cache subsystem of the KA7AA CPU module is organized as a three-level instruction cache and a two- level data cache. It consists of the following structures:

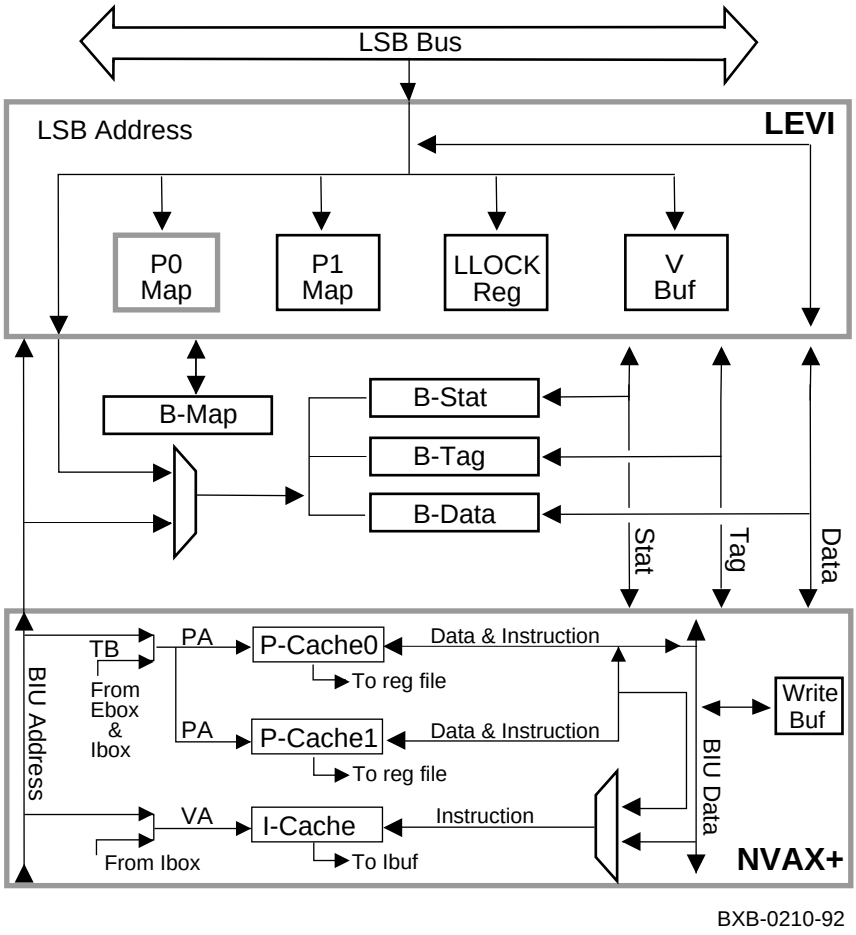
- **Virtual Instruction Cache (VIC)**
  On- chip, 2 Kbytes, direct mapped, virtually addressed, instruction stream only; has 32- byte block size and a valid bit per quadword subblock (fill) size.
- **Primary Cache (P- cache)**
  On- chip, 8 Kbytes, 2- way set associative, physically addressed, read allocate, no- write allocate, write through, mixed instruction and data; has 32- byte block size and a valid bit per quadword subblock (fill) size.
- **Backup Cache (B- cache)**
  Off- chip, 4 Mbytes, direct mapped, physically addressed, write back, mixed instruction and data; has 64- byte block size to match the LSB bus block size, and 64- byte fill size.

The KA7AA CPU caches are not strictly hierarchical. Since the VIC is virtually addressed and holds I- stream data only, it is not necessarily a subset of the P- cache, which is always a subset of the B- cache. Maintaining coherency in this type of cache organization requires strict adherence to the proper sequence of turning caches on and off.

Although the instruction caching is not necessarily hierarchical, fetching instructions operates as if it were. If an instruction is not found in the VIC, it is looked for first in the P- cache, then in the B- cache, and finally in memory. Data is sought first from the P- cache, then the B- cache, and finally from memory.

Figure 3- 1 shows the KA7AA CPU module cache organization.

Figure 3- 1 KA7AA CPU Module Cache Organization.



3.1 Virtual Instruction Cache

The virtual instruction cache (VIC) acts as the primary source of instruction stream for the Ibox. It is direct- mapped with a 2- Kbyte cache size. Table 3- 1 summarizes the VIC attributes.

Table 3- 1 Virtual Instruction Cache Attributes

Attribute	Implementation
Cache size	2 Kbytes
Access type	Direct- mapped
Block size	32 bytes
Subblock size	8 bytes
Valid bits	4 per block; 1 per subblock
Data parity bits	4 even parity bits per cache block; 1 per subblock
Number of tags	64
Tag parity bit	1 even parity bit per tag
Fill algorithm	Fill forward, random cycle allocate if no tag hit or data subblock valid
Access size	8 bytes
Bus size	8 bytes
Prefetching	None
Data stored	I- stream only
Virtual/physical	Virtual

The VIC contains four internal processor registers (IPRs) that provide control and read/write access to the cache arrays. They are:

- VIC Memory Address Register (VMAR), IPR208
- VIC Tag Register (VTAG), IPR209
- VIC Data Register (VDATA), IPR210
- Ibox Control and Status Register (ICSR), IPR211

The VIC registers are described in Section 2.11.2.

3.2 Primary Cache

The primary cache (P- cache) represents the first level of D- stream memory hierarchy and the second level of I- stream memory hierarchy. The P- cache is 2- way set associative. It stores 8 Kbytes of data and 256 tags corresponding to 256 hexword blocks (1 hexword = 32 bytes). Each tag is 20 bits wide corresponding to bits <31:12> of the physical address. There are four quadword subblocks per block with a valid bit associated with each subblock. The access size for P- cache reads and writes is one quadword. Parity is maintained for each byte of data (32 bits per block). One bit of parity is maintained for every tag. The P- cache has a one cycle access and a one cycle repetition rate for both reads and writes.

Memory updates affect the P- cache as follows. If the block is in the P- cache, then external logic invalidates the P- cache entry and updates the B- cache. If the block is in the B- cache but NOT in the P- cache, external logic invalidates the B- cache entry and does nothing to the P- cache.

Table 3- 2 summarizes the P- cache attributes.

Table 3- 2 Primary Cache Attributes

Attribute	Implementation
Cache size	8 Kbytes
Access type	2- way set associative
Block size	32 bytes
Subblock size	8 bytes
Valid bits	4 per block; 1 per subblock
Data parity bits	4 even parity bits per cache block; 1 per subblock
Number of tags	256
Tag parity bit	1 even parity bit per tag
Fill algorithm	Fill forward, random cycle allocate if no tag hit or data subblock valid
Access size	8 bytes
Bus size	8 bytes
Prefetching	None
Data stored	Both I- and D- stream
Virtual/physical	Physical

Five different internal processor registers (IPRs) control the P- cache and provide read/write access to the cache arrays. They are:

- P- Cache Tag Registers (PCTAG) —256 IPRs
- P- Cache Data Parity Registers (PCDAP) —1024 IPRs, 512 for each P- cache set
- P- Cache Parity Error Address Register (PCADR), IPR242
- P- Cache Control Register (PCCTL), IPR248
- P- Cache Parity Error Status Register (PCSTS), IPR244

The P- cache registers are described in Section 2.11.4.

3.3 Backup Cache

The backup cache (B- cache) is implemented in RAMs and represents the second level of D- stream memory hierarchy and the third level of I- stream memory hierarchy. It consists of three RAM structures: B- tag, B- data, and B- stat, located between the NVAX+ chip and the LEVI interface, and is controlled by the Cbox and the LEVI interface.

The B- cache stores 4 Mbytes of data. It is organized as direct- mapped, with a block (line) size of 64 bytes to match the LSB bus. For each block, the following information is stored:

- Tag: Consists of bits <33:22> of the physical address
- Tag parity: Reflects even parity over the tag field
- Valid bit (V): Indicates whether the line can be considered
- Shared bit (S): Indicates whether this line might be resident in another cache in the system.
- Dirty bit (D): Indicates whether the line has been written to by this CPU and has more recent data than memory.
- Status parity bit: Reflects even parity over the V, S, and D bits.

The B- cache organization groups the status bits in a single 64K X 4 RAM (B- stat) and allows these bits to be updated without changing the value of the tag. This in turn allows the CPU to set the Dirty bit on write hits to nonshared blocks. In general, the tag field is only loaded by the LEVI interface, and the status and data stores are loaded by both the processor and the LEVI interface.

3.3.1 B- Cache States

The B- cache state is defined by the three status bits: Valid, Shared, and Dirty. Table 3- 3 shows the legal combinations of the status bits.

From the perspective of the NVAX+ chip, a tag probe for a read is successful if the tag matches the address and the V bit is set. A tag probe for a write is successful if the tag matches the address, the V bit is set, and the S bit is clear.

Table 3- 3 B- Cache States

B- Stat			
V	S	D	State of Cache Line Assuming Tag Match
0	X	X	Not valid miss.
1	0	0	Valid for read or write. This cache line contains the only cached copy of the block. The copy in memory is identical to this block.
1	0	1	Valid for read or write. This cache line contains the only cached copy of the block. The contents of the block have been modified more recently than the copy in memory.
1	1	0	Valid for read or write but writes must be broadcast on the bus. This cache line may also be present in the cache of another CPU. The copy in memory is identical to this block.
1	1	1	Valid for read or write but writes must be broadcast on the bus. This cache line may also be present in the cache of another CPU. The contents of the block have been modified more recently than the copy in memory.

3.3.2 B- Cache State Changes

The state of any given cache line in the B- cache is affected by either processor actions or actions of other nodes on the LSB bus.

Table 3- 4 shows what effect processor actions have on the state of a given B- cache line and the resulting/required bus traffic. Table 3- 5 shows what effect bus actions have on the state of a given B- cache line, and the resulting/required module action.

LSB writes always clean (make non- dirty) the cache line in both the initiating node and all nodes that choose to take the update. They also update the appropriate location in main memory.

The KA7AA CPU module decides whether to take an update or not as a function of the state of the P- cache backmap (P- map, Section 3.4.1). If the LEVI interface determines that the block referenced by an LSB write command is resident in the P- cache, the relevant block is updated in the B- cache with the LSB write data and also invalidated in the P- cache, since the P- cache must be a subset of the B- cache at all times. If the LEVI interface determines that the block referenced by an LSB write command is not resident in the P- cache, but is resident in the B- cache, it invalidates the relevant block in the B- cache.

The KA7AA CPU module also compares incoming LSB addresses to those found in the LLOCK register, LVICT register, and LWPEND register (see Chapter 7). The behavior of the KA7AA CPU module in these cases is shown in Table 3- 6.

Table 3- 4 Effect of Processor Action on B- Cache Line

Processor Request	Tag Probe Result[1]	Action on LSB	LSB Response	Next Cache State
Read	*Invalid*	Read	$\overline{\text{Shared}}$	$\overline{\text{Shared}}$, $\overline{\text{Dirty}}$
Read	*Invalid*	Read	*Shared*	*Shared*, $\overline{\text{Dirty}}$
Write	*Invalid*	Read	$\overline{\text{Shared}}$	$\overline{\text{Shared}}$, *Dirty*
Write	*Invalid*	Read, Write	*Shared*	*Shared*, $\overline{\text{Dirty}}$
Read	$\overline{\text{Match}}$ AND $\overline{\text{Dirty}}$	Read	$\overline{\text{Shared}}$	$\overline{\text{Shared}}$, $\overline{\text{Dirty}}$
Read	$\overline{\text{Match}}$ AND *Dirty*	Read	*Shared*	*Shared*, $\overline{\text{Dirty}}$
Write	$\overline{\text{Match}}$ AND *Dirty*	Read	$\overline{\text{Shared}}$	$\overline{\text{Shared}}$, *Dirty*
Write	$\overline{\text{Match}}$ AND $\overline{\text{Dirty}}$	Read, Write	*Shared*	*Shared*, $\overline{\text{Dirty}}$
Read	$\overline{\text{Match}}$ AND *Dirty*	Read, Wr- Victim	$\overline{\text{Shared}}$	$\overline{\text{Shared}}$, $\overline{\text{Dirty}}$
Read	$\overline{\text{Match}}$ AND *Dirty*	Read, Wr- Victim	*Shared*	*Shared*, $\overline{\text{Dirty}}$
Write	$\overline{\text{Match}}$ AND *Dirty*	Read, Wr- Victim	$\overline{\text{Shared}}$	$\overline{\text{Shared}}$, *Dirty*
Write	$\overline{\text{Match}}$ AND *Dirty*	Read, Write, Wr- Victim	*Shared*	*Shared*, $\overline{\text{Dirty}}$
Read	*Match*	None	None	No change
Write	*Match* AND $\overline{\text{Shared}}$	None	None	$\overline{\text{Shared}}$, *Dirty*
Write	*Match* AND *Shared*	Write	$\overline{\text{Shared}}$	$\overline{\text{Shared}}$, $\overline{\text{Dirty}}$
Write	*Match* AND *Shared*	Write	*Shared*	*Shared*, $\overline{\text{Dirty}}$

[1] An overscore on a cache block status bit indicates the complement of the state. For example, $\overline{\text{Shared}}$ = Not Shared.

Table 3- 5 Effect of LSB Bus Action on B- Cache Line

LSB Operation	Tag Probe Result[1]	Module Response	Next Cache State	Comment
Read	$\overline{Match}$ OR *Invalid*	$\overline{Shared}$, $\overline{Dirty}$	No change	
Write	$\overline{Match}$ OR *Invalid*	$\overline{Shared}$, $\overline{Dirty}$	No change	
Read	*Match* AND $\overline{Dirty}$	*Shared*, $\overline{Dirty}$	*Shared*, $\overline{Dirty}$	
Read	*Match* AND *Dirty*	*Shared, Dirty*	*Shared, Dirty*	This module must supply the data.
Write	*Match* and line is interesting	*Shared*, $\overline{Dirty}$	*Shared*, $\overline{Dirty}$	This module takes the update.
Write	*Match* and line is uninteresting	$\overline{Shared}$, $\overline{Dirty}$	*Invalid*	This module takes the invalidate.

[1] An overscore on a cache block status bit indicates the complement of the state. For example, $\overline{Shared}$ = Not Shared.

Table 3- 6 KA7AA CPU Module Response to Incoming Addresses

LSB Operation	Address Register Matched	Module Response	Action
Read	LLOCK register	*Shared*	No action
Write	LLOCK register	$\overline{Dirty}$	Clear LLOCK<31>
Read	LVICT register	*Shared, Dirty*	Supply data from victim buffer
Write	LVICT register	$\overline{Shared}$, $\overline{Dirty}$	Invalidate victim buffer; remove bus request
Read	LWPEND register	*Shared*	No action
Write	LWPEND register	*Shared*, $\overline{Dirty}$	Accept update to B- cache

[1] An overscore on a cache block status bit indicates negation.

3.4 Cache Backmaps

The KA7AA CPU module implements two backmaps (or duplicate tag stores) that keep track of the contents of the P- cache and the B- cache. They are referred to as P- map and B- map. The backmaps are cycled with every bus transaction to allow the KA7AA CPU module to properly respond to a given bus command/address.

NOTE: No backmap is necessary for the VIC, because the VIC is virtually addressed and is flushed on REI (Return from Exception or Interrupt) instructions.

3.4.1 P- Map

The P- map is located in the LEVI gate arrays and consists of two structures, each 128 entries deep. Each entry contains a value that is equal to the difference between the B- cache tag (address bits <33:22>) and the P- cache tag (address bits <31:12>), valid bit, and parity. Thus, the P- map is 12 bits wide: Address bits <21:12>, V, and P. The P- map is loaded by the NVAX+ chip during B- cache D- stream read hits and by LEVI during B- cache D- stream read misses. LEVI control can read the P- map whenever an LSB write hits in the B- map.

The KA7AA CPU module enforces inclusion, which ensures that the valid contents of the P- cache are always a subset of the valid contents of the B- cache. Therefore, the KA7AA CPU module must invalidate P- cache lines whenever the given block becomes invalid in the B- cache. This occurs on refills (either a dirty victim or a nonshared victim) and on updates.

When an update occurs on the LSB bus, and the given address yields a tag match and the entry is valid in the P- map, the B- cache takes the update and the CPU module invalidates the corresponding entry in the P- cache.

3.4.2 B- Map

The B- map is located on the module and is a structure 64K entries deep. Each entry consists of the B- cache tag (address bits <33:22>), valid bit, and parity. The B- map is written by LEVI at the same time that the B- cache tag is written (within the context of B- cache manipulation, due to either processor action or bus action). The B- map is read on every LSB bus command/address cycle. The contents of the B- map inform the KA7AA CPU control logic when to request the B- cache to form an appropriate bus response.

3.5 Victim Buffer

The KA7AA CPU module implements a victim buffer to hold the contents of a victimized block in the B- cache. A victim block is a B- cache line that is valid and dirty but has a tag mismatch for a processor request. The processor tag probe yields a miss and the appropriate block is fetched from memory. However, the block in the B- cache at this index must be written back to memory since it is dirty. The KA7AA CPU module posts the miss refill to the bus before actually performing the victim write.

A single victim block and victim address pair is stored in the LEVI chips for later transmission on the bus. While the victim buffer contains a valid victim, the KA7AA CPU module treats this block like a second set in the B- cache, compares all bus addresses to the victim address, and responds to bus reads and writes as required by the bus protocol (see Table 3- 6).

The KA7AA CPU module implements only a single victim buffer. It therefore does not process a second B- cache miss before writing the victim block to memory.

3.6 Write Policy

The NVAX+ chip contains an 8- byte write packer and an 8- entry write queue where each entry is a quadword of data. The function of the write packer is to accumulate memory- space writes that arrive sequentially to the same quadword, so that only one write needs to be done to the cache. Only normal write commands to the same quadword are packed together. All other writes (that is, Write_Unlock) pass immediately from the write packer into the write queue. Data in the write queue is assembled into an octaword (when possible) and written to the B- cache when the Cbox arbiter does not see a request of higher priority, or when one of the following conditions exists:

- A DREAD_LOCK or WRITE_UNLOCK command.
- An IPR_READ or IPR_WRITE command (including Clear Write Buffer).
- An I/O space read or I/O space write.
- A DREAD or IREAD conflict with data in the write packer or write queue. (Addresses are checked to hexword granularity on address bits <31:5>.)

Depending on how it is able to pack and assemble data in the write queue, the Cbox is able to write between one and four longwords of data at a time to the B- cache. For the case of a byte/word write, the Cbox does a Read-Modify- Write operation, which is implemented in hardware.

When the Cbox detects a byte/word write condition that incurred a B- cache miss, it does the following:

1. Issues an LDxL command and drives the EDAL address lines with the write address.
2. Receives a quadword of data, checks ECC, and merges the read data with the byte/word write data.
3. Issues an STxC command along with the correct LW_MASK<7:0> and write address. If the STxC command fails (LLOCK<31> not set), microcode loops back to 1 (first element in the current list) and issues an LDxL.

Otherwise, the Cbox does a B- cache read hit, merge, and B- cache write hit.

The KA7AA CPU module performs LSB bus write operations as follows:

- **Victims**
  If a given cache line is valid and dirty and the tag does not match the address for the given processor request, the line must be written back to memory. To enhance performance, this victim is written back to memory after the refill. The victim data must be removed from the B-cache data store and held in a victim buffer for later transmission on the LSB bus. While a block is in a victim buffer, the KA7AA CPU must respond to all reads and writes that reference the block (see Table 3- 6).
- **Shared Blocks**
  If the response to a tag probe for a processor write is shared, the write must be broadcast on the LSB bus.

3.7 B- Cache Operating Modes

The backup cache has two modes of operation:

- B- cache on
- B- cache force hit

The operating modes are controlled by two bits in the BIU_CTL register: BC_ENB (bit <0>) and BC_FHIT (bit <3>).

Table 3- 7 shows how the operating mode of the B- cache is selected.

Table 3- 7 Selection of the B- Cache Operating Mode

BIU_CTL<3>	BIU_CTL<0>	Operating Mode
0	1	B- cache on
1	1	Force hit

The On state is the normal operating mode of the B- cache. It is selected by setting BIU_CTL<0> and clearing BIU_CTL<3>.

The B- cache force hit mode is selected by setting FORCE_HIT (BIU_CTL<3>) when the B- cache is enabled. When FORCE_HIT is set, all memory space reads and writes to the B- cache, both I- stream and D- stream, are forced to hit. The tag store state is not changed. The data RAMs are accessed as if the tag store access produced a dirty- valid hit. In a multiprocessor environment, the B- cache must be flushed of all dirty blocks before force hit mode is selected.

Force hit mode is intended to be used only for testing and initialization. Tag store parity and data RAM ECC errors are detected in this mode.

3.8 Cache Initialization

On power- up or following a reset, the processor microcode and the console firmware initialize the P- cache and the B- cache. In the initialized state, the P- cache is enabled for I- stream and D- stream operations, and the B- cache is on.

CAUTION: The cache subsystem is initialized to a determined state. Software must never turn the B- cache off once the system is up and running. Turning the B- cache off during normal operation places the system in an UNDETERMINED state.

Chapter 4

LSB Bus Interface

The CPU module connects to the LSB bus through LEVI, the LSB interface, which is implemented in two gate array chips, LEVI- A and LEVI- B. LEVI controls all the tags, maps, and data RAMs on the CPU module. It contains the P- map, which maps the processor P- cache.

LEVI performs the following major tasks:

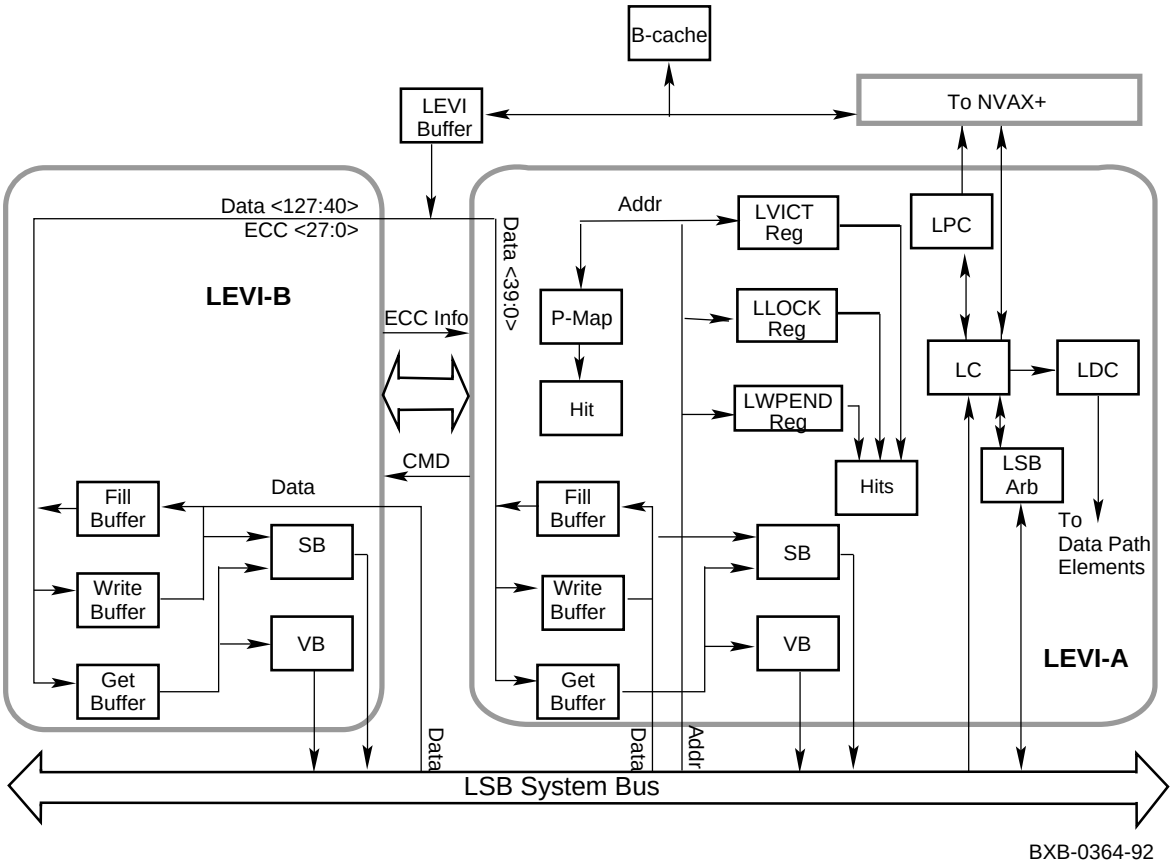
- Translates CPU memory and I/O space references to the appropriate LSB transactions.
- Supports control of writebacks to memory and cache fills from memory in reponse to processor actions.
- Supports control of cache invalidates, cache updates, and cache block transfers to the LSB bus in response to LSB actions.
- Initiates reads and writes to the CPU node private address space (the Gbus on the CPU module).
- Supports LSB required interrupt logic.
- Implements all LSB required registers.

This chapter discusses the role of LEVI in transactions between the CPU module and other modules on the LSB bus. Sections include:

- LEVI Address Path
- LEVI Data Path
- LEVI Controllers
- Interfacing Rules
- Address Space Mapping
- LEVI Transactions

Figure 4- 1 shows a block diagram of the LEVI chips.

Figure 4- 1 LEVI Block Diagram



4.1 LEVI Address Path

The LEVI address path (see Figure 4- 1) is implemented in the LEVI- A chip. It consists of the following major elements:

- **P- Map**
  The P- map consists of four 64 X 16 dual- ported RAMs maintained exclusively by LEVI. Each entry in the P- map represents a P- cache block in the processor. LEVI writes to the P- map during processor read hits to the B- cache. One use of the P- map is deciding whether to update or invalidate a B- cache block during LSB writes from another node. If the LSB write hits in the P- map, the update is taken; otherwise LEVI invalidates the B- cache. Another use is to invalidate P- cache blocks that are being displaced by B- cache fills.
- **LVICT Register**
  LEVI keeps the address of the last victimized B- cache block and a valid bit in the LVICT register. Once the LSB Victim Write takes place, the Valid bit is cleared. Should another LSB (non- Victim) write match the address in the LVICT register, LEVI invalidates its own LVICT register (see Table 3- 6).
- **LLOCK Register**
  The address is latched and LLOCK<31> is set when the processor issues an LDxL instruction. LLOCK<31> is cleared after a successful

STxC instruction. LSB reads that hit in the LLOCK register cause LEVI to respond "Shared" so that all subsequent writes to the address are visible on the LSB bus. LSB writes that hit in the LLOCK register cause LEVI to clear the LLOCK<31> (see Table 3- 6).

- **LWPEND Register**
  This register contains the address of the pending write and a valid bit. If an LSB Write hits the LWPEND register, LEVI- A takes the update even if the write missed in the P- map to assure that the write about to be issued has the latest data (see Table 3- 6).

4.2 LEVI Data Path

The LEVI data path, like the LSB bus and CPU module data paths, is 156 bits wide: 128 bits of data and 28 bits of ECC (7 bits for each longword). Note that LEVI treats the data and ECC bits identically, since there is no ECC correction between the B- cache and the LSB bus.

An array of buffers in the LEVI data path serve to store data and synchronize data movement within LEVI. The buffers are implemented in both LEVI chips, as shown in Figure 4- 1. The main buffer elements on the LEVI data path are the following:

- **Fill Buffer**
  The fill buffer works with the LEVI buffer on the module to receive, and possibly hold, four octawords of LSB data headed for the B- cache. The data pipeline shifts from the gate array time domain to the clock-forwarded module time domain in the fill buffer. The fill buffer also merges write buffer data with LSB data following B- cache write misses.
- **Get Buffer**
  The get buffer also works with the LEVI buffer; it captures B- cache blocks headed for the LSB bus. The data pipeline shifts back from the module to the gate array time domain in the get buffer.
- **Write Buffer**
  The write buffer captures two octawords of write data from the processor in three situations:

  B- cache write misses
  B- cache write hits to shared blocks
  CSR writes

  The write buffer also receives a write data mask (LEVI- A gets four of eight bits; LEVI- B gets all eight bits) and ADDR<5> from the processor. The mask and address bits indicate which longwords are to be merged with B- cache data.
- **Stall Buffer**
  The stall buffer holds four octawords of B- cache data for broadcast onto the LSB bus. It also merges write buffer data with B- cache data during writes to shared blocks and processor CSR writes.
- **Victim Buffer**
  The victim buffer holds B- cache blocks victimized by cache fills. The buffer holds only a single cache block so transactions that cause other victims are held off until the current victim reaches the LSB bus.

4.3 LEVI Controllers

The control functions on the LEVI transactions are implemented in three controllers in the LEVI chips:

- LEVI processor controller (LPC)
- LEVI data controller (LDC)
- LSB controller (LC)

4.3.1 LEVI Processor Controller

The LPC provides the control interface between the processor and LEVI. It fields requests from the processor and initiates LEVI responses. Major functions include:

- **LoadLock/StoreCond**
  The LPC first probes the cache; misses generate requests to the LSB controller for LSB transactions.
- **Gbus Read**
  The LPC asks the LSB controller for a CSR read and does extra handshaking on the Gbus. It appears on the LSB as a private command (not a CSR read).
- **Gbus Write**
  The LPC controls the write to the LEVI write buffer, then handshakes with the Gbus and acknowledges the processor. No LSB transaction is requested.
- **Processor Read Fill**
  After a processor read miss and after LEVI has received the missed data from the LSB bus, the LPC loads the processor with the two octawords it has waited for. The LEVI data controller (LDC) briefly interrupts the B- cache fill after the first octaword write to allow the processor to load two octawords from the LEVI buffer. The LDC then completes the final three octaword writes.
- **Processor Write Data**
  The LPC controls the processor writes to the write buffer on LEVI when the processor cannot write directly to the B- cache.
- **P- Cache Invalidates**
  Whenever LEVI invalidates P- map entries, the LPC invalidates the corresponding P- cache entries in the processor.

The LPC runs in the processor time domain.

4.3.2 LEVI Data Controller

The LDC directs data traffic moving between the LSB and the B- cache based on requests from the LC. Each transaction described below moves one B- cache block. The LDC is involved in the following transactions:

- **GetRAM**
  GetRAM moves one B- cache block to the LSB by way of the stall buffer. The LSB controller (LC) requests this transfer when another node has issued a read to a block and the local (and only valid) copy is dirty.

- **GetWBRAM**
  GetWBRAM is used on processor writes to shared B- cache blocks and processor CSR writes. The fetched B- cache block is conditionally merged with the contents of the write buffer (based on the values of the write data mask and ADDR<5>) before being driven onto the LSB by way of the stall buffer.
- **GetVic**
  GetVic is used to route B- cache blocks that are victimized by B- cache fills to the victim buffer. Note that blocks in the victim buffer must await an LSB slot; blocks in the stall buffer have already had their LSB slots allocated (by the LC).
- **FillRAM**
  FillRAM moves one block directly from the LSB to the B- cache. The LC requests this to take an update to a shared block or to complete the first read of a processor write miss to a shared B- cache block.
- **FillProcRAM**
  FillProcRAM is requested following processor read misses. The LDC moves data from the LSB to the fill buffer and the LEVI buffer on the module. The data pipeline is frozen briefly after the first octaword write to the B- cache to allow the LPC to load the first two octawords into the processor (in its own time domain) by way of the LEVI buffer. The LPC then releases the processor but retains control of the B- cache so that the LDC can write the remaining three octawords to the B- cache.
- **FillProc**
  FillProc services processor CSR reads. This transaction is identical to FillProcRAM discussed above except that writes to the B- cache are suppressed. (CSR data is not cached.)
- **FillWBRAM**
  FillWBRAM merges processor write data in the Write buffer with the incoming LSB data and writes the result into the B- cache. Merging is based on the values of the write data mask and ADDR<5>. The LC requests this transaction following processor write misses to blocks that are not shared.

The LDC uses clock forwarding on the CPU module for data transfers between LEVI and the B- cache.

4.3.3 LSB Controller

The LC is the central controller of the LEVI chipset. It receives requests from the LPC and issues requests to the LPC, LDC, and the LSB arbiter. The LC responds to both processor- initiated and LSB- initiated transactions. Specifically, the LC performs the following functions:

- Controls the address path and LEVI access to the B- map, B- stat, and B- tag RAMs on the CPU module.
- Schedules all LEVI and CPU module operations except B- cache hits and Gbus transactions. Requests from the LC to the LPC and LDC move data around the module, the gate arrays, and the LSB bus.
- Asserts the LSB address and control signals (CNF, ERR) according to LSB protocol. LSB SHARED and DIRTY are asserted based on the re-

sults of B- tag and B- stat lookup. LSB STALL is asserted when required by internal conflicts.

- The LC also controls B- cache access from the processor or LEVI with the LSYNC signal (Section 4.4.1).

The LC schedules transfers of data between the LEVI and the CPU module during dedicated LSB cycles.

4.4 Interfacing Rules

Logic on the CPU module synchronizes dual- ported accesses to the B- cache and the P- map, since these components are accessed by both the processor and LEVI. On the other hand, arbitration rules govern node accesses to the LSB bus.

All cache data is longword ECC protected (seven bits per longword). LEVI does look- aside ECC error detection but no ECC error correction.

The LEVI chips calculate ECC for each longword and compare it against the received ECC. Any difference between calculated and received ECC indicates an error, which is signaled to the system. The ECC for longword 0 and a partial ECC syndrome for longword 1 are passed each cycle from LEVI- B to LEVI- A.

4.4.1 Dual- Ported Access Synchronization

Dual- ported B- cache and P- map accesses are synchronized with the LSYNC semaphore. LSYNC is also used to synchronize access to the CPU module data path during Gbus references.

Whenever LSYNC is deasserted (default state), the processor can read or write the B- stat, B- tag, and B- data RAMs directly.

NOTE: The B- map RAMs are never accessed by the processor.

During LoadLock and StoreCond requests, whenever LSYNC is deasserted, the LPC can read or write the B- stat, B- tag, and B- data RAMs directly.

During RBlock and WBlock requests to Gbus addresses, whenever LSYNC is deasserted, the LPC can transfer data between the processor and the Gbus buffer on the CPU module. LEVI has priority to assert LSYNC and access the B- cache to service LSB transactions, since the LSB is non-pended. LEVI also accesses the B- cache to complete processor transactions that miss the B- cache. When the LC asserts LSYNC, the processor and the LPC suspend B- stat/B- tag references (tag probes), P- map updates (PMapWE), and B- data references within a fixed number of LSB cycles. The LEVI is then free to access any resource within the CPU module until it deasserts LSYNC at the completion of the LSB- related access.

4.4.2 LSB Arbitration

LEVI watches all LSB traffic to adhere to the arbitration rules. Specifically, read, write, or victim transactions from any node that reference a common memory bank cannot occur more frequently than once every three transactions (or once every 15 LSB cycles). CSR transactions are also limited in the same manner.

4.5 Address Space Mapping

The LEVI chips define which portion of the address space is cacheable or noncacheable. Cacheable address space is memory space and noncacheable address space is I/O space. The LEVI chips further separate I/O space into LSB bus CSR space and local Gbus space.

The LEVI interface ensures that processor references to memory result in an LSB bus read or write command, while references to I/O space result in an LSB read CSR, write CSR command, or private command.

Table 4- 1 gives the encodings of commands that LEVI can send to the LSB bus.

Table 4- 1 LSB Command Field Encodings

LSB D<37:35>	Command
000	Read
001	Write
010	Reserved
011	Write Victim
100	Read CSR
101	Write CSR
110	Reserved
111	Private

4.6 LEVI Transactions

As the CPU module's interface to the LSB bus, LEVI responds to transactions initiated from two sources:

- Processor (CPU chip)
- LSB bus (other nodes)

These transactions require that both the processor and LEVI have access to the B- cache on the CPU module and the P- map in LEVI. The dual-ported accesses to these components are synchronized with the LSYNC semaphore (Section 4.4.1). The two LEVI chips operate in both the processor and the LSB bus time domains.

4.6.1 Processor- Initiated Transactions

LEVI responds to the following processor requests:

- **Read/Write Hit**
  During a D- stream read hit, LEVI updates its P- map. It takes no other action.
- **Block Read/Write**
  LEVI captures B- stat and B- tag data, arbitrates for the LSB bus, issues the read/write command code on the bus, receives/drives data on the bus, and updates all tags, maps, and B- stat bits.

- **LoadLock/StoreCond**
  LEVI waits for LSYNC to deassert, if necessary, then probes the B- tag. On an LDxL (LoadLock) command that hits in the B- cache, LEVI completes the read request and sets LLOCK<31>. If the LDxL is a B- cache miss, LEVI issues an LSB bus read command and sets LLOCK<31>. On an STxC (StoreCond) request from the processor, LEVI checks LLOCK<31>. If this bit is set, (success) and the B- cache tag lookup results in a hit, LEVI immediately completes the write and clears LLOCK<31>. If the tag probe results in a miss, and LLOCK<31> is set, LEVI issues an LSB bus write command. On an STxC, if LLOCK<31> is clear, LEVI returns failed status to the processor.
- **Gbus Read/Write**
  LEVI waits for LSYNC to deassert, if necessary. For Gbus reads, LEVI- A arbitrates for the LSB bus, issues a private command, forwards data from the Gbus to the processor by way of the LSB bus. Gbus writes slip through LEVI to the Gbus without an LSB transaction.

The processor can be engaged in only one external operation at a time. This means that once the processor makes a transaction request to LEVI, it remains idle until released by LEVI.

4.6.2 LSB- Initiated Transactions

LEVI responds to transactions initiated by other nodes on the LSB. These transactions include:

- **Read**
  LEVI checks each read address against the B- map. If there is a match, LEVI then checks the B- stat RAMs. It returns B- cache data if the Dirty bit is set. LEVI returns a victimized block if the block's address matches the read address.
- **Write**
  When the write address matches that of a valid block in the B- map, LEVI reacts as follows. If the address also hits in the P- map, LEVI takes the update and invalidates the P- cache block in the processor. Otherwise, the B- cache block is simply invalidated. Note that this behavior can be altered with the LMODE register.
- **Victim Write**
  LEVI ignores victim writes from other nodes. Victimized blocks do not hit in the local B- cache.
- **CSR Read/Write**
  Only registers in the LSB node space can be read or written from the LSB. Gbus registers cannot be accessed from the LSB. Note that LEVI can also respond to its own processor- generated CSR transactions on the bus.
- **Private**
  Private transactions are used to return Gbus data to the processor, to allow access to the B- tag, B- stat, B- map, and P- map structures directly by the processor, and to resolve STxC boundary conditions. LEVI does not respond to private commands from other modules.

LEVI is pipelined to track up to three interleaved LSB transactions.

4.6.3 Transaction Ordering

The processor controller (LEVI PC, Section 4.3.1) and the LSB controller (LEVI LC, Section 4.3.3) work together to guarantee strict ordering of transactions issued on the LSB. Processor and LEVI actions proceed in stages as shown in Table 4- 2.

Table 4- 2 Processor- LEVI Actions During Transactions

Processor Action	LEVI Action
P1. The processor issues a request with address A1.	In response to **P1**, LEVI performs the following actions: **L1.** LEVI initiates an LSB transaction with address A1.
P2. The processor can issue a new request with address A2 any time after **L1** completes.	**L2**. If **P1** was a WBlock and **L1** was an LSB Read that received a shared response, LEVI issues an LSB Write with address A1. **L3**. If **L1** was an LSB Read and the B- cache block being displaced had the Dirty bit set, LEVI issues an LSB Write Victim command. In response to **P2**, LEVI performs the following action: **L4**. LEVI initiates an LSB transaction with address A2.

Chapter 5

Console Overview

The KA7AA CPU module supports the LSB system console with combined hardware/software elements that control the system at power- up, on reset, or on CPU halts. This chapter describes the console hardware that resides on the CPU module. Sections include:

- CPU Console Hardware
- Console Program Invocation
- Console Registers

The console user interface and commands are discussed in the *Console Reference Manual.*

5.1 CPU Console Hardware

The KA7AA CPU module provides hardware to support the console functions. This hardware includes:

- A serial ROM (read- only memory) for first- level console program storage
- A set of FEPROMs (flash programmable ROMs) for second- level console program storage
- An EEPROM (electrically erasable/programmable ROM) for miscellaneous parameter/log storage
- A set of UARTs (universal asynchronous receivers/transmitters) that allow the console program to communicate serially with one console terminal and the system power supplies
- A watch chip that provides a battery- backed- up time- of- year (TOY) clock for use by operating system software
- A set of parallel I/O ports for functions such as LED status indicators and node identification
- A serial I/O port for manufacturing diagnostic use

The CPU module provides access to ROM, EEPROM, console UARTs, the watch chip, and other functions through the 8- bit Gbus.

All Gbus component registers and memory stores are located in node private space, which means that their addresses are constant and are independent of slot identification. Table 5- 1 gives the address ranges allocated to the Gbus components.

Every Gbus memory store byte or register byte is located on a 64- byte, naturally aligned boundary. For example, the first byte of FEPROM storage is located at byte address F000 0000; the second byte is at F000 0040. Also note that a single 128- Kbyte FEPROM consumes 8 Mbytes of address space. This addressing restriction implies that processor code cannot be executed from this address space.

Table 5- 1 Gbus Components

Component	Address
Console ROM	F000 0000 to 0000
Console EEPROM	F380 0000 to FFFF
UART registers	F400 0000 to FFFF
WATCH registers	F600 0000 to 0FC0
Gbus$WHAMI	F700 0000
Gbus$LEDs	F700 0040
Gbus$PMask	F700 0080
Gbus$Intr	F700 00C0
Gbus$Halt	F700 0100
Gbus$LSBRST	F700 0140
Gbus$Misc	F700 0180
Gbus$RMode	F780 0000
Gbus$LTagRW	F780 0100

5.1.1 Serial ROM

After power- up, node reset, or system reset, but before any instructions are executed, the NVAX+ chip automatically loads its internal P- cache through the serial I/O port from an external, 8- Kbyte serial ROM.

The serial ROM contains the first level of console/diagnostic/bootstrap code (serial ROM code). This code initializes all programmable features of the NVAX+ chip, diagnosing any faults detected along the bootstrap path and bootstrapping code execution out to the second level of console/diagnostic/ bootstrap code (the main console program). The first level bootstrap copies the main console program code from FEPROM storage to the P- cache and transfers control flow to the P- cache. Once the serial ROM is loaded into the P- cache, the same serial I/O port becomes available for use by software as a diagnostic interface.

5.1.2 Serial Port

The NVAX+ chip provides an initialization and diagnostic interface in the form of a serial I/O port. The serial I/O port is a full duplex connection between the CPU chip and a module connector. The port is accessed and controlled through internal processor registers.

The serial I/O port drives a LED indicator, which may flash as data is transmitted over the serial port, but is otherwise available to diagnostic code as a status indicator.

5.1.3 FEPROMs

The console program is stored in a set of 128K X 8 FEPROM chips. This code does not appear in a structure of contiguous locations in the processor's address space. Specifically, each byte of FEPROM storage appears on a 64- byte naturally aligned boundary. This implies that the console program cannot execute directly out of FEPROM, but instead, must be copied into a more compact contiguous space in cacheable memory and executed from there. This process of copying the code store and transferring control flow is known as the first- level bootstrap and is performed by the serial ROM code, as explained in Section 5.1.1.

The FEPROMs can be programmed online without assistance from an external programming device. The FEPROMs cannot be patched; they can only be erased and programmed as a whole.

5.1.4 EEPROM

A single 8K X 8 EEPROM is used for miscellaneous parameter and log storage. This store does not appear in a contiguous address space. Specifically, each byte of EEPROM storage appears on a 64- byte boundary.

The EEPROM can be written to byte- by- byte online, without assistance from an external programming device.

5.1.5 UARTs

The CPU module has six serial communication lines but uses only three. The communication lines are named and assigned as follows:

- UART0A is connected to the LSB local console terminal line LOC_RX/ LOC_TX (computer room terminal for field service).
- UART1B is connected to the LSB power supply status lines PS_RX and PS_TX.
- UART2A is dedicated to Ctrl/P character detection. Its receive line can tap receive characters off LOC_RX, OP_RX, or RD_RX as selected by the Gbus$PMask register. Its transmit line is unused.
- UART0B, UART1A, and UART2B are unused.

The LSB console serial lines are connected to all CPU slots. After power-up or system initialization, the CPU modules arbitrate for use of the common console lines; the winner is allowed to drive them. The default configuration of the serial lines at power- up is as follows:

Baud rate set to 9600
No parity
One stop bit
8- bit characters

One physical component (DUART) implements two UARTs, hence the naming of the UARTs as UART0A, UART0B, and so on, where the number indicates the physical component and the letter indicates the individual UART within the component. Control of these UARTs is accomplished through a set of registers in each UART. These registers are listed in Table 5- 2.

5.1.5.1 Ctrl/P Character Detection and Halt Protection

UART2A is dedicated to detecting Ctrl/P characters received from the console terminal.

UART2A intercepts a copy of all UART receive characters from the console terminal line and compares for Ctrl/P. Characters other than Ctrl/P result in an IPL15 interrupt posted to the processor (reflected in the Gbus$Intr register). Ctrl/P characters result in an IPL1F interrupt (halt) posted to the processor (reflected in the Gbus$Halt register). Note that the IPL1F interrupt is in addition to the IPL15 interrupt. If no serial lines are selected for console operation (the processor is halt- protected), then all receive characters result in an IPL15 interrupt. For UART2A to detect Ctrl/P characters, all control settings must be programmed to match the console terminal UART.

5.1.5.2 UART Register Addressing

Each UART in a DUART component is controlled independently through its own set of registers (some registers are shared between two UARTs within a DUART). All UART registers are either read only (for status and data receive) or write only (for control and data transmit). Read registers and write registers share common addresses, that is, reading and writing a single address accesses two separate registers.

For each UART there are two read registers and two write registers that are directly accessible in the processor's address space: RR0, WR0, RR8, and WR8. RR0 and WR0 are the main status and control registers for the UART. RR8 and WR8 are the data receive and transmit registers.

For each UART there are a number of other control and status registers that are indirectly accessible through RR0 and WR0. These registers are accessed by writing the correct index value into WR0 and then reading RR0 or writing WR0. After the second read/write operation occurs, the index value is automatically reset back to zero.

5.1.6 Watch Chip

A watch chip resides on the Gbus and provides a battery- backed- up time-of- year clock and 50 bytes of battery- backed- up RAM. The chip contains a built- in crystal oscillator and a 10- year lithium battery.

5.2 Console Program Invocation

The NVAX+ chip operates in console mode when the CPU module encounters one of the following conditions:

- System reset through power- up, control panel reset, or reset through the Gbus$LSBRST register
- Module reset performed by setting NRST (LCNR<30>)
- Module halted by setting NHALT (LCNR<29>)
- Ctrl/P character received from the console terminal

When the NVAX+ chip enters console mode, it transfers control to the location addressed by the contents of the CHALT register.

5.3 Console Registers

Table 5- 2 lists the console registers with their addresses and indicates the components in which they are implemented.

Table 5- 2 Console Registers

Register	Address	Implementation
UARTxx$WR0[1]	UARTxx_BASE[1]	DUART chip
UARTxx$WR1	Index 0001	DUART chip
UARTxx$WR2	Index 0010	DUART chip
UARTxx$WR3	Index 0011	DUART chip
UARTxx$WR4	Index 0100	DUART chip
UARTxx$WR5	Index 0101	DUART chip
UARTxx$WR6	Index 0110	DUART chip
UARTxx$WR7	Index 0111	DUART chip

[1] **UART Base Addresses:**

xx = 0B; BASE = F400 0000
xx = 0A; BASE = F400 0080
xx = 1B; BASE = F480 0000
xx = 1A; BASE = F480 0080
xx = 2B; BASE = F500 0000
xx = 2A; BASE = F500 0080

Table 5- 2 Console Registers (Continued)

Register	Address	Implementation
UARTxx$WR8	UARTxx_BASE+40H	DUART chip
UARTxx$WR9	Index 1001	DUART chip
UARTxx$WR10	Index 1010	DUART chip
UARTxx$WR11	Index 1011	DUART chip
UARTxx$WR12	Index 1100	DUART chip
UARTxx$WR13	Index 1101	DUART chip
UARTxx$WR14	Index 1110	DUART chip
UARTxx$WR15	Index 1111	DUART chip
UARTxx$RR0	UARTxx_BASE	DUART chip
UARTxx$RR1	Index 0001	DUART chip
UARTxx$RR2	Index 0010	DUART chip
UARTxx$RR3	Index 0011	DUART chip
UARTxx$RR8	UARTxx_BASE+40H	DUART chip
UARTxx$RR10	Index 1010	DUART chip
UARTxx$RR13	Index 1101	DUART chip
UARTxx$RR15	Index 1111	DUART chip
Watch$Seconds	F600 0000	Watch chip
Watch$Minutes	F600 0080	Watch chip
Watch$Hours	F600 0100	Watch chip
Watch$Day_of_Month	F600 01C0	Watch chip
Watch$Month	F600 0200	Watch chip
Watch$Year	F600 0240	Watch chip
Watch$CSRA	F600 0280	Watch chip
Watch$CSRB	F600 02C0	Watch chip
Watch$CSRC	F600 0300	Watch chip
Watch$CSRD	F600 0340	Watch chip
BBackup RAM (50 bytes)	F600 0380 to F600 FC00	Watch chip
Gbus$WHAMI	F700 0000	CPU module
Gbus$LEDs	F700 0040	CPU module
Gbus$PMask	F700 0080	CPU module
Gbus$Intr	F700 00C0	CPU module
Gbus$Halt	F700 0100	CPU module
Gbus$LSBRST	F700 0140	CPU module
Gbus$Misc	F700 0180	CPU module
Gbus$RMode	F780 0000	CPU module
Gbus$LTagRW	F780 0100	LEVI

[1] **UART Base Addresses:**

xx = 0B; BASE = F400 0000
xx = 0A; BASE = F400 0080
xx = 1B; BASE = F480 0000
xx = 1A; BASE = F480 0080
xx = 2B; BASE = F500 0000
xx = 2A; BASE = F500 0080

A number of console/diagnostic/interrupt related registers listed in Table 5- 2 are referred to with a prefix of Gbus$. These registers provide the following control and status functions:

- Node identification
- LED status indicators

- Interrupt status summaries
- Console terminal selection
- Halt protection
- System reset

This section provides descriptions of individual Gbus registers. The remaining console registers are listed in Table 5- 2 for reference only. All Gbus registers are eight bits wide.

Gbus$WHAMI

Address BB + 0000 0000
Access RO

The Gbus$WHAMI register provides information on system configuration and reflects the status of certain backplane signals.

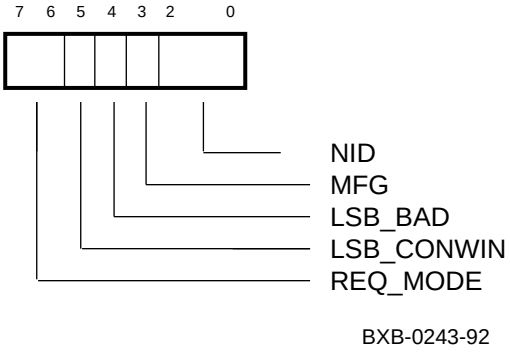


Table 5- 3 Gbus$WHAMI Register Bit Definitions

Name	Bit(s)	Type	Function
REQ_MODE	<7:6>	RO	**Request Mode.** Indicates the maximum number of CPU modules that this CPU module supports in a system.

Gbus$WHAMI <7:6>	CPUs Allowed in LSB Slots
00	1 to 6
01	Reserved
10	Reserved
11	Reserved

Table 5- 3 Gbus$WHAMI Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
LSB_CONWIN	<5>	RO	**LSB CONWIN.** Reflects the inverted state of the LSB_CONWIN L backplane signal. When set, indicates that Gbus$LEDs<1> is clear (asserted) in one or more CPU modules.
LSB_BAD	<4>	RO	**LSB Bad.** Reflects the inverted state of the LSB_BAD L backplane signal. When set, indicates that LSB_BAD L is driven by one or more CPU modules.
MFG	<3>	RO	**Manufacturing Status.** Used by manufacturing.
NID	<2:0>	RO	**Node ID.** Identifies the CPU module by the slot (0- 7) where it resides.

Gbus$LEDs

Address F700 0000
Access R/W

The Gbus$LEDs register is used for lighting a series of LEDs on the module to aid in debug and to indicate self- test status. Writing a zero to a bit in this register lights the corresponding LED.

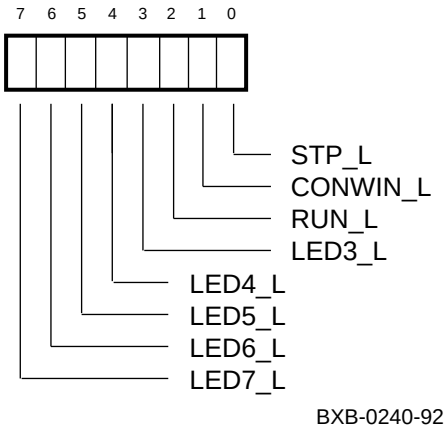


Table 5- 4 Gbus$LEDs Register Bit Definitions

Name	Bit(s)	Type	Function
LEDs_L	<7:3>	R/W	**LEDs Low.** When a bit in this field is set, the associated LED signal is asserted low.
RUN_L	<2>	R/W	**RUN Low**. When set, the associated LED signal is asserted low. The state of this bit also indicates whether the currently running software is the operating system (and not the diagnostic/console program).
CONWIN_L	<1>	R/W	**CONWIN Low.** When set, the associated LED signal is asserted low. Also drives the backplane signal LSB_CONWIN L. The state of this signal can be read through the Gbus$WHAMI register.
STP_L	<0>	R/W	**Self- Test Passed Low.** When set, the associated LED signal is asserted low.

Gbus$PMask

Address F700 0040
Access R/W

The Gbus$PMask register controls halts to the processor.

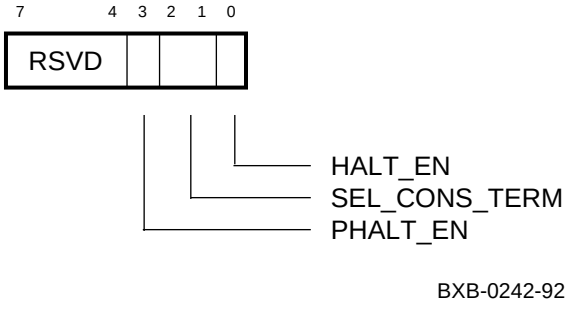


Table 5- 5 Gbus$PMask Register Bit Definitions

Name	Bit(s)	Type	Function
RSVD	<7:4>	R/W, 1	**Reserved.** Initialized to ones.
PHALT_EN	<3>	R/W, 1	**Ctrl/P Halt Enable.** When set, enables Ctrl/P characters received by the UART selected in the Select Console Terminal field of this register to halt the processor. The Halt Enable bit of this register must also be set for a Ctrl/P character to generate a halt.

Table 5- 5 Gbus$PMask Register Bit Definitions (Continued)

<table>
<tr><th>Name</th><th>Bit(s)</th><th>Type</th><th>Function</th></tr>
<tr><td>SEL_CONS_TERM</td><td><2:1></td><td>R/W, 1</td><td>Select Console Terminal. Selects one of three console terminals for Ctrl/P character detection.
<table>
<tr><th>Gbus$PMask <2:1></th><th>Console Terminal Selected</th></tr>
<tr><td>00</td><td>UART0A (local terminal)</td></tr>
<tr><td>01</td><td>UART0B (Reserved)</td></tr>
<tr><td>10</td><td>UART1A (remote diagnostic control)</td></tr>
<tr><td>11</td><td>UART2A placed into module-level loopback mode. In this mode, the UART2A receive line is driven by the UART2A transmit line. PHALT_EN (bit <3> of this register) must be zero (Ctrl/P halts disabled) while modifying SEL_CONS_TERM to avoid erroneous halts.</td></tr>
</table></td></tr>
<tr><td>HALT_EN</td><td><0></td><td>R/W, 1</td><td>Halt Enable. When set, enables halts to the processor, including halts generated by LCNR<NHALT> or by detection of a Ctrl/P character received by a UART selected in the Select Console Terminal field of this register. When clear, all halts to the processor are disabled. PHALT_EN must also be set for Ctrl/P characters to generate a halt.</td></tr>
</table>

Gbus$Intr

Address F700 0080
Access R/W

The Gbus$Intr register stores interrupt summary information. Specifically, it provides a means to determine the source of IPL14, IPL15, and IPL16 interrupts to the processor.

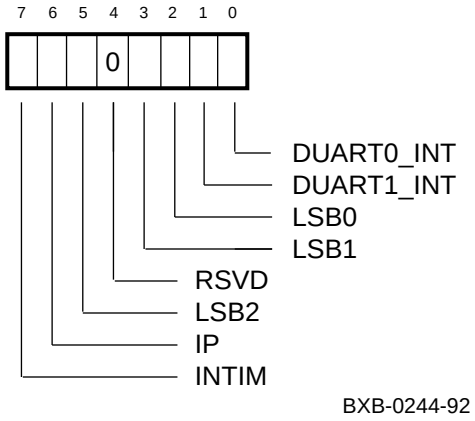


Table 5- 6 Gbus$Intr Register Bit Definitions

Name	Bit(s)	Type	Function
INTIM	<7>	RO, 0	**Interval Timer.** When set, indicates that the watch chip is asserting its interval timer output.
IP	<6>	W1C, 0	**Interprocessor.** When set, indicates that the LEVI-A chip has detected a write to the LIPINTR register with data selecting this node.
LSB2	<5>	RO, 0	**LSB 2**. When set, indicates that the LEVI- A chip has an LSB level 2 interrupt pending.
RSVD	<4>	R0	**Reserved.** Reads as zero.

Table 5- 6 Gbus$Intr Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
LSB1	<3>	RO, 0	**LSB 1**. When set, indicates that the LEVI- A chip has an LSB level 1 interrupt pending.
LSB0	<2>	RO, 0	**LSB 0**. When set, indicates that the LEVI- A chip has an LSB level 0 interrupt pending.
DUART1_INT	<1>	RO, 0	**DUART1 Interrupt.** When set, indicates that either UART1A or UART1B is requesting an interrupt for the processor. This bit is cleared when all possible DUART1 interrupt sources are cleared.
DUART0_INT	<0>	RO, 0	**DUART0 Interrupt.** When set, indicates that either UART0A or UART0B is requesting an interrupt for the processor. This bit is cleared when all possible DUART0 interrupt sources are cleared.

Gbus$Halt

Address F700 0100
Access R/W

The Gbus$Halt register summarizes halt and power conditions.

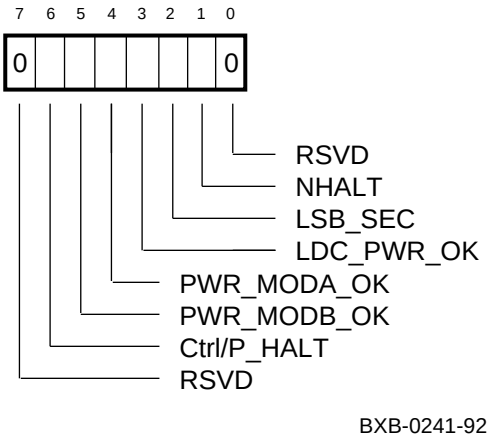


Table 5- 7 Gbus$Halt Register Bit Definitions

Name	Bit(s)	Type	Function
RSVD	<7>	R0	**Reserved.** Reads as zero.
Ctrl/P_HALT	<6>	W1C, 0	**Ctrl/P Halt.** Set when a Ctrl/P character is received by the UART selected in the Gbus$PMask register.
PWR_MODB_OK	<5>	RO	**Power Module B Okay.** Set when Power Module B of the I/O PIUs (plug- in unit) is working properly. Cleared when Module B fails.
PWR_MODA_OK	<4>	RO	**Power Module A Okay.** Set when Power Module A of the I/O PIUs (plug- in unit) is working properly. Cleared when Module A fails.

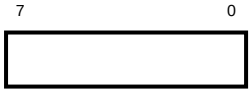
Table 5- 7 Gbus$Halt Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
LDC_PWR_OK	<3>	RO	**LDC Power Okay.** Is set when all local disk converters (LDC) in the platform are working properly. Cleared when no LDCs are installed or when one or more of the LDCs fails.
LSB_SEC	<2>	RO	**LSB Secure.** Reflects the inverted state of the backplane signal LSB_SECURE L. When set, indicates that the control panel keyswitch is in the Secure position and that Ctrl/P halts to the processor are disabled by hardware.
NHALT	<1>	RO	**Node Halt.** Reflects the state of LCNR<NHALT>.
RSVD	<0>	R0	**Reserved.** Reads as zero.

Gbus$LSBRST

Address F700 0140
Access R/W

The Gbus$LSBRST register is used for initiating a system reset sequence. When the CPU chip writes any value to this register, the LSB RESET signal is asserted for 512 LSB cycles.

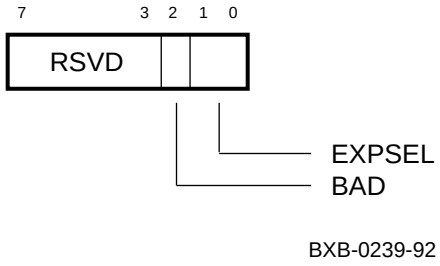


BXB-0264-92

Gbus$Misc

Address F700 0180
Access R/W

The Gbus$Misc register controls various system functions.



BXB-0239-92

Table 5- 8 Gbus$Misc Register Bit Definitions

Name	Bit(s)	Type	Function
RSVD	<7:3>	RO, 1	**Reserved.** Initialized to ones.
BAD	<2>	R/W, 1	**Bad.** When set, causes the module to drive LSB BAD which, in turn, lights the control panel fault LED. The state of this bit does not affect the Self- Test- Passed LED on the module or the STP bits in the Gbus$LEDs and LCNR registers. This bit allows software to assert LSB BAD on behalf of another system component. To determine if any module is driving LSB BAD, software should read Gbus$WHAMI<LSB_BAD>, not Gbus$Misc<BAD>.

Table 5- 8 Gbus$Misc Register Bit Definitions (Continued)

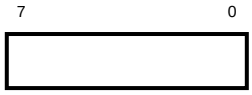
Name	Bit(s)	Type	Function
EXPSEL	<1:0>	R/W, 1	**Expander Select.** Selects which cabinet the power supply UART lines are logically connected to, and therefore, which of three 48V regulators are connected to the power supply lines.

Gbus$Misc <1:0>	Power Supply Connection
00	PS lines logically connected to main CPU cabinet.
01	PS lines logically connected to right expander cabinet.
10	PS lines logically connected to left expander cabinet.
11	PS transmit line is looped back to PS receive line.

Gbus$RMode

Address F780 0000
Access R/W

The Gbus$RMode register is a write- only register. A write to it sets LDIAG<FRIGN> and logically disconnects the CPU module from the LSB bus. This register is intended for use as a backup system should there be a problem with the LSB interface and writes to the LDIAG register be unsuccessful (writes to the LDIAG register require a successful LSB transaction while writes to Gbus space are completed without any LSB access). Note that software should write to the LDIAG register as a first choice and use the Gbus$RMode register only if the write to the LDIAG register fails.

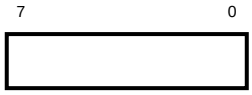


BXB-0264-92

Gbus$LTagRW

Address F780 0100
Access R/W

The Gbus$LTagRW register, when used with LTAGA, LTAGW, and LDIAG registers, allows software to read and write the B- cache, B-map, and P- map tags. See descriptions of the LTAGA, LTAGW, and LDIAG registers in Chapter 7.



BXB-0264-92

Chapter 6

I/O Operations

I/O operations handled by the KA7AA CPU module include I/O reads, I/O writes, and device interrupts. The NVAX+ CPU chip uses four hardcoded SCB vectors for all device interrupts. Interrupt service routines at the four SCB vectors are required to determine the source of the interrupt and invoke the appropriate service routine.

From the perspective of I/O operations, registers are divided into two groups: local registers and remote registers. Registers that reside on the KA7AA CPU module and the LSB bus are local registers. Those that reside on I/O buses are remote registers. Local registers are directly accessible to software; remote registers are not. Access to remote registers is achieved by means of the mailbox protocol. The LMBOX register is provided to assist software in the mailbox protocol.

6.1 Mailbox Data Structure

Remote control and status registers (CSRs) are accessed through 64- byte naturally aligned mailbox data structures located in main memory. Read requests are posted in mailboxes. Data is returned in memory with status in the following quadword. Mailboxes are allocated and managed by the operating system software. Figure 6- 1 shows a mailbox data structure.

Figure 6- 1 Mailbox Data Structure

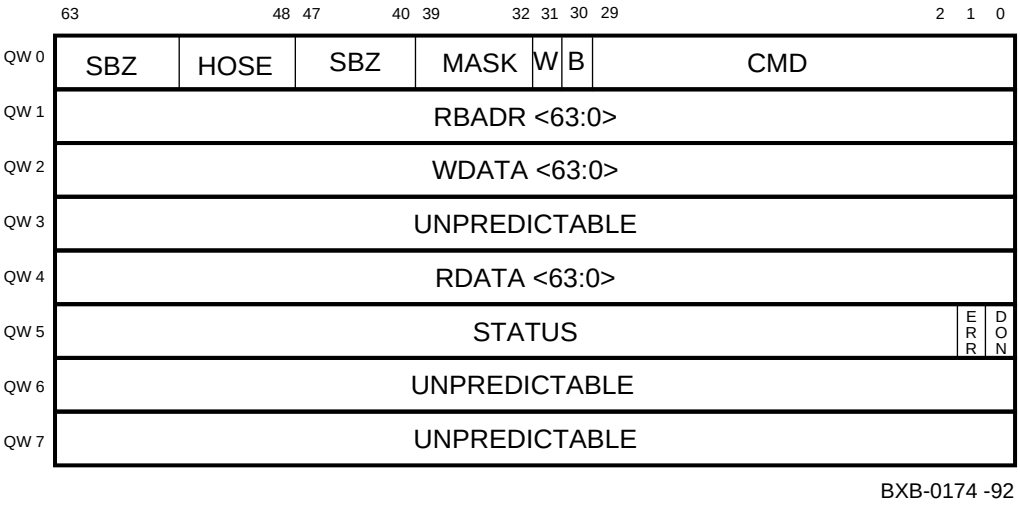


Table 6- 1 describes the mailbox data structure. Refer to the *DEC 7000 AXP System/VAX 7000 I/O System Technical Manual* for a detailed description of the mailbox protocol.

Table 6- 1 Mailbox Data Structure

Field	Bit(s)	Type	Quad-word	Function
HOSE	<63:48>	R/W	0	**Hose.** Used to determine which remote bus the command is meant for.
MASK	<39:32>	R/W	0	**Mask.** Contains the byte mask. The I/O module does not use this field.
CMD	<29:0>	R/W	0	**Command.** Contains the command. A value of one is a read; a value of seven is a write.
RBADR	<63:0>	R/W	1	**Remote Broadcast Address.** Contains the address to be broadcast on the remote bus.
WDATA	<63:0>	R/W	2	**Write Data.** Contains the write data to be broadcast on the remote bus.
RDATA	<31:0>	R/W	4	**Read Data.** Contains read data returned from the remote bus.
STATUS	<63:2>	R/W	5	**Status.** Contains status information provided by the remote bus.
ERR	<1>	R/W	5	**Error.** When set, indicates that a mailbox operation failed.
DON	<0>	R/W	5	**Done.** Status bit set by the I/O module when a mailbox operation is complete.

6.2 Mailbox Operation

The I/O module services mailbox requests by means of four mailbox pointer CSRs (LMBPR registers; see Section 6.4) located in the I/O module's node space. There is one LMBPR register for each CPU node. Software sees only one LMBPR register address, but the CPU module replaces the least significant two bits of the address (that is, D<2:1>) with the least significant two bits of the node ID (that is, NIOD<1:0>). If a given LMBPR register is in use when it is written to, the I/O module does not acknowledge it and CNF is not asserted. Processors use the lack of CNF assertion on writes to the LMBPR register to indicate a busy status. The write is retried later under software control.

To perform a write to the LMBPR register, microcode must know the address of the LMBPR register and the address of the mailbox data structure to be loaded into the LMBPR register. Another memory structure needs to be created to pass this information to microcode. This structure is called the Mailbox Pointer and consists of two longwords. Figure 6- 2 shows the mailbox pointer structure. Table 6- 2 gives the bit definitions of the mailbox pointer structure.

Figure 6- 2 Mailbox Pointer Structure

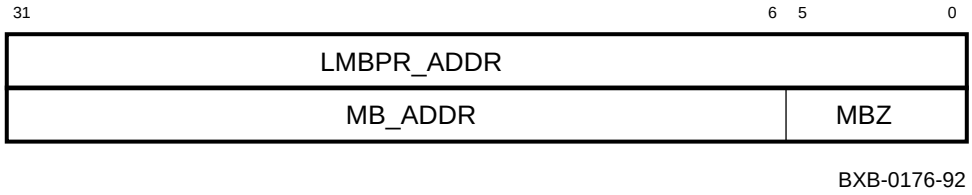


Table 6- 2 Mailbox Pointer Structure

Name	Bit(s)	Type	Function
MB_ADDR	<31:6>	WO	**Mailbox Address.** Contains the physical address of the mailbox data structure. Since this structure is aligned on a 64- byte boundary, bits <5:0> of the address must be zero.
LMBPR_ADDR	<31:0>	WO	**LMBPR Address.** Contains the virtual address of the LMBPR register.

When software has created the mailbox data structure and the mailbox pointer structure, it can start the I/O operation. An MTPR to the LMBOX register (Section 6.4) initiates the I/O operation. Microcode reads the MB_ADDR field out of the mailbox pointer structure and then writes the value to the LMBPR register using the address provided in the mailbox pointer structure. An EDAL store conditional command is used to perform the write. Microcode then checks the Zero Condition Code bit (PSL<2>) in the BIU_STAT register to determine if the write passed or failed. If the write passed, PSL<2> is set; otherwise, PSL<2> is cleared.

Software can loop on the MTPR to the LMBOX register until the write passes.

After the I/O module has accepted the write to LMBPR, it performs the I/O operation. Software can now poll the status bit in the mailbox data structure until the I/O operation is complete. When the I/O operation has completed, DON in the mailbox data structure (see Table 6- 1) is set. If an error occurred during the transaction, LBER<E> (see Chapter 7) is also set. If the operation was an I/O write, no further action is required. If the operation was an I/O read, software can now fetch the returned data from the RDATA field in the mailbox data structure.

6.3 Device Interrupt Handling

The KA7AA CPU module uses the device interrupts as shown in Table 6- 3. Interrupts from the LSB and the UARTs (device interrupts) are handled by both hardware and software. After an interrupt has been posted to the CPU chip through one of the four IRQ lines, the CPU chip passes control to the operating system through four dedicated SCB entry points. Table 6- 3 shows the device interrupt sources and their matching SCB entry points.

Table 6- 3 KA7AA CPU Interrupts

Interrupt Level (Hex)	Interrupt Condition	NVAX+ IRQ Pin	SCB Vector
17	LSB level 3 interrupts	3	DC
16	Interprocessor interrupt	2	D8
16	LSB level 2 interrupts	2	D8
15	Console UARTs	1	D4
15	LSB level 1 interrupts	1	D4
14	LSB level 0 interrupts	0	D0

For IPL16 and IPL17 interrupts, software reads the Gbus$Intr register to determine if the interrupt is posted by an LSB I/O device, another processor in the system, or a UART. If an interprocessor or a UART interrupt has been received, software can directly pass control to the appropriate service routine. For LSB I/O interrupts, software must get the device interrupt vector from the I/O module.

6.4 I/O Operation Registers

Two registers are used for I/O operations:

- Mailbox Pointer CSR (LMBPR)
- Mailbox Register (LMBOX)

The LMBPR register resides on the IOP module and is described in the *DEC 7000 AXP System/VAX 7000 I/O System Technical Manual.* The description of the LMBOX register follows.

LMBOX—LSB Mailbox Register

Address BB + 00
Access R/W

The LMBOX register contains the physical address of the mailbox pointer structure.

Table 6- 4 LMBOX Register Bit Definitions

Name	Bit(s)	Type	Function
MBXREG	<31:0>	WO	**Mailbox Register.** Contains the physical address of the mailbox pointer structure.

Chapter 7

CPU Module Registers

The KA7AA CPU module, like the memory and I/O modules on the LSB bus, contains two groups of registers:

- LSB required registers
- CPU- specific registers

LSB required registers are used for internode communication over the LSB bus. CPU- specific registers are used to perform functions specific to the CPU module.

7.1 Register Mapping

All CPU module registers reside in node space. The only exceptions to this rule are the two interrupt registers, LIOINTR and LIPINTR, which reside in LSB broadcast space.

CPU module registers are mapped to the node space as offsets to a base address (BB). The base address is implemented in hardware and depends on the node ID, which is determined by the LSB backplane slot occupied by the module. Table 7- 1 gives the physical base addresses of nodes on the LSB bus.

Table 7- 1 LSB Node Space Base Addresses

Node ID	Module	Physical Base Address (BB) (Byte)
0	CPU/Memory	F800 0000
1	CPU/Memory	F840 0000
2	CPU/Memory	F880 0000
3	CPU/Memory	F8C0 0000
4	CPU/Memory	F900 0000
5	CPU/Memory	F940 0000
6	CPU/Memory	F980 0000
7	CPU/Memory	F9C0 0000
8	I/O	FA00 0000

Table 7- 2 lists the CPU module registers and gives the address of each register as an offset from a selected node space base address.

NOTE: Two CPU registers listed in Table 7- 2, LIOINTR and LIPINTR, are located in LSB broadcast space, the base address of which is FE00 0000.

Table 7- 2 CPU Module Registers

Register Name	Mnemonic	Address (Byte Offset)
LSB Required		
Device Register	LDEV	BB[1] + 0000
Bus Error Register	LBER	BB + 0040
Configuration Register	LCNR	BB + 0080
Memory Mapping Register 0	LMMR0	BB + 0200
Memory Mapping Register 1	LMMR1	BB + 0240
Memory Mapping Register 2	LMMR2	BB + 0280
Memory Mapping Register 3	LMMR3	BB + 02C0
Memory Mapping Register 4	LMMR4	BB + 0300
Memory Mapping Register 5	LMMR5	BB + 0340
Memory Mapping Register 6	LMMR6	BB + 0380
Memory Mapping Register 7	LMMR7	BB + 03C0
Bus Error Syndrome Register 0	LBESR0	BB + 0600
Bus Error Syndrome Register 1	LBESR1	BB + 0640
Bus Error Syndrome Register 2	LBESR2	BB + 0680
Bus Error Syndrome Register 3	LBESR3	BB + 06C0
Bus Error Command Register 0	LBECR0	BB + 0700
Bus Error Command Register 1	LBECR1	BB + 0740
I/O Interrupt Register	LIOINTR	BSB[2] + 0000
Interprocessor Interrupt Register	LIPINTR	BSB + 0040
CPU- Specific		
Mode Register	LMODE	BB + 0C00
Module Error Register	LMERR	BB + 0C40
Lock Address Register	LLOCK	BB + 0C80
LSB Diagnostic Control Register	LDIAG	BB + 0D00
Tag Address Register	LTAGA	BB + 0D40
Tag Write Data Register	LTAGW	BB + 0D80
Console Communication Register 0	LCON0	BB + 0E00
Console Communication Register 1	LCON1	BB + 0E40
Performance Counter Control Register	LPERF	BB + 0F00
Performance Counter 0 Register	LCNTR0	BB +0F40
Performance Counter 1 Register	LCNTR1	BB + 0F80
Last Miss Address Register	LMISSADDR	BB + 0FC0

[1] BB is the node space base address of the CPU module in hex.
[2] BSB is the broadcast space base address, which is FE00 0000.

7.2 Register Descriptions

LSB required registers have the following characteristics:

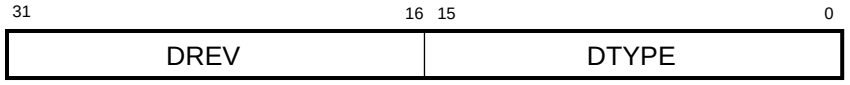
- All writes are 32 bits wide. Byte or word operations are not supported.
- Writes directed to a read- only register may be accepted and acknowledged, but no action is taken, and the content of the register is not affected.

CPU- specific registers appear in the LSB CSR space.

LDEV—Device Register

Address BB + 0000
Access R/W

The LDEV register is loaded during initialization with information that identifies a node.



BXB-0100-92

Table 7- 3 LDEV Register Bit Definitions

Name	Bit(s)	Type	Function
DREV	<31:16>	R/W, 0	**Device Revision.** Identifies the revision level of an LSB node. For the KA7AA CPU module, the value of this field is zero.
DTYPE	<15:0>	R/W, 0	**Device Type.** Identifies the type of node. For the KA7AA CPU module, the value of this field is set to 8002 hex.

LBER—Bus Error Register

Address BB + 0040
Access R/W

The LBER register stores the error bits that are flagged when an LSB node detects errors in the LSB operating environment and logs the failing commander ID. The status of this register remains locked until software resets the error bit(s).

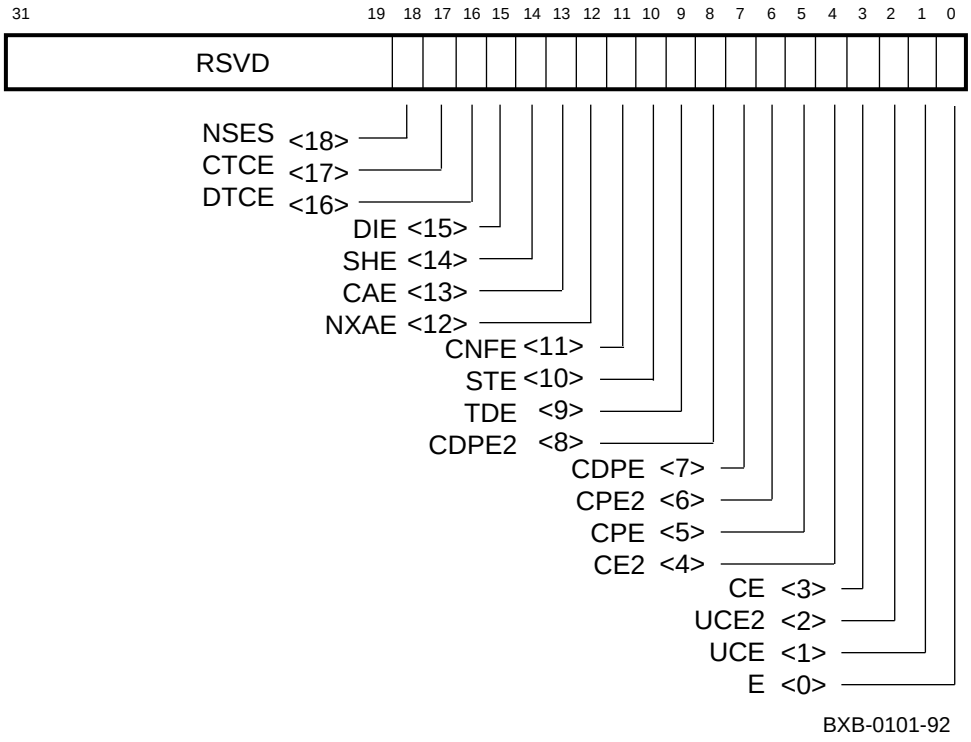


Table 7- 4 LBER Register Bit Definitions

Name	Bit(s)	Type	Function
RSVD	<31:19>	R0	**Reserved.** Read as zero.
NSES	<18>	RO, 0	**Node- Specific Error Summary**. Set when an error condition is reported in the LMERR register.
CTCE	<17>	W1C, 0	**Control Transmit Check Error**. Set when an LSB control line is driven incorrectly by the CPU module. When CTCE is set, ERR is asserted by the CPU module.
DTCE	<16>	W1C, 0	**Data Transmit Check Error.** Set when the CPU module detects an error while driving the the D<127> and ECC<27:0> lines during a data or command cycle. When DTCE is set, ERR is asserted by the CPU module.
DIE	<15>	W1C, 0	**Dirty Error.** Set if the CPU module receives an asserted DIRTY signal during a cycle when DIRTY signals are not allowed. When DIE is set, ERR is asserted by the CPU module.
SHE	<14>	W1C, 0	**Shared Error.** Set if the CPU module receives an asserted SHARED signal during a cycle when SHARED signals are not allowed. When SHE is set, ERR is asserted by the CPU module.
CAE	<13>	W1C, 0	**Command/Address Error.** Set if the CPU module receives an asserted CA signal during a cycle when CA signals are not allowed. When CAE is set, ERR is asserted by the CPU module.
NXAE	<12>	W1C, 0	**Nonexistent Address Error.** Set when the CPU module does not receive confirmation for a command it sent on the LSB. When NXAE is set, ERR is asserted by the CPU module.
CNFE	<11>	W1C, 0	**CNF Error.** Set if the CPU module receives a confirmation signal during a cycle that does not permit confirmation. When CNFE is set, ERR is asserted by the CPU module.
STE	<10>	W1C, 0	**STALL Error.** Set when the CPU module receives a STALL signal during a cycle that does not permit stalls. When STE is set, ERR is asserted by the CPU module.
TDE	<9>	W1C, 0	**Transmitter During Error.** Set if a CE, UCE, CPE, or CDPE error occurs during a cycle when the CPU module was driving D<127:0>. When TDE is set, ERR is asserted by the CPU module.

Table 7- 4 LBER Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
CDPE2	<8>	W1C, 0	**Second CSR Data Parity Error.** Set when a second parity error occurs while CDPE is set on a CSR data cycle.
CDPE	<7	W1C, 0	**CSR Data Parity Error.** If a parity error occurs during a CSR data cycle, the CPU module sets CDPE, asserts ERR, and locks D<38:0> in the LBERCR registers.
CPE2	<6>	W1C, 0	**Second Command Parity Error.** Set when a second parity error occurs on a command cycle while CPE is set.
CPE	<5>	W1C, 0	**Command Parity Error.** If a parity error occurs on a command cycle, the CPU module sets CPE, asserts ERR, and locks D<38:0> in the LBECR registers.
CE2	<4>	W1C, 0	**Second Correctable Data Error.** Set when a second correctable ECC error occurs on a data cycle while CE is set.
CE	<3>	W1C, 0	**Correctable Data Error.** If the CPU module detects an ECC error on the LSB, it sets CE, asserts ERR, and locks D<38:0> of the command cycle in the LBECR registers.
UCE2	<2>	W1C, 0	**Second Uncorrectable Data Error.** Set when the CPU module detects a second uncorrectable data error while UCE is set.
UCE	<1>	W1C, 0	**Uncorrectable Data Error.** If the CPU module detects an uncorrectable ECC error on the LSB during a data cycle, it sets UCE, asserts ERR, and locks D<38:0> of the command cycle in the LBECR registers.
E	<0>	W1C, 0	**Error.** Set whenever the CPU module detects assertion of ERR on the LSB.

LCNR—Configuration Register

Address BB + 0080
Access R/W

The LCNR register contains LSB configuration setup and status information.

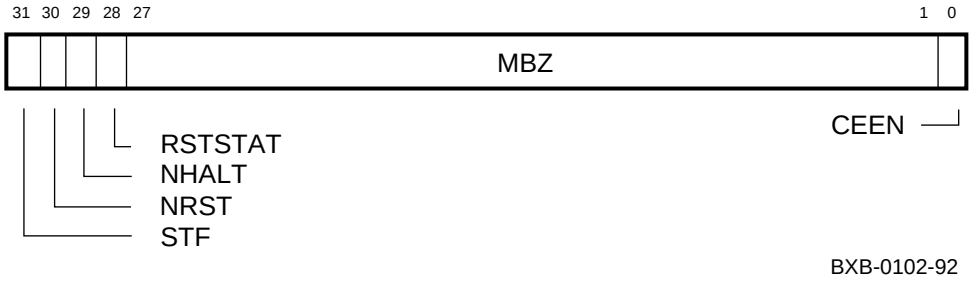


Table 7- 5 LCNR Register Bit Definitions

Name	Bit(s)	Type	Function
STF	<31>	W1C, 1	**Self- Test Fail.** When set, indicates that this node has not yet completed self- test.
NRST	<30>	WO, 0	**Node Reset.** When set, the node enters console mode and undergoes a reset sequence.
NHALT	<29>	R/W, 0	**Node Halt.** When set, a CPU node enters console mode.
RSTSTAT	<28>	W1C, 0	**Reset Status.** When set, provides an indication to console software that a given CPU node should not attempt to become the boot processor, but should rather join an already running system. This bit is set when NRST (LCNR<30>) is set. It is cleared with a write of one, at system power- up, or with an LSB RESET command. This bit is not cleared in a reset sequence caused by setting NRST.
MBZ	<27:1>	R/W, 0	**Must Be Zero.** Must always be written as zero.
CEEN	<0>	R/W, 0	**Correctable Error Detection Enable**. When set, enables detection of correctable errors.

LMMR0–7—Memory Mapping Registers

Address BB + 0200 to BB + 03C0
Access R/W

Eight LMMR registers define the memory configuration for all memory modules installed in the system. They are copies of the equivalent AMR registers in memory modules installed in the system. Each LMMR register is associated with the LSB module whose node ID matches the three lower bits of the LMMR address. Thus, LMMR0 is associated with node 0, LMMR1 is associated with node 1, and so on. LMMR registers are loaded during system initialization when the memory modules are initialized and configured.

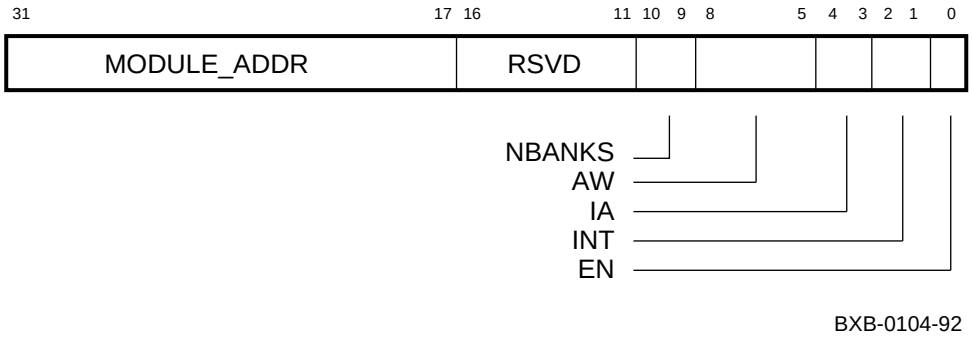


Table 7- 6 LMMR Register Bit Definitions

Name	Bit(s)	Type	Function
MODULE_ADDR	<31:17>	R/W	**Module Address.** Specifies the most significant bits of the base address of the memory region spanned by the memory module associated with this register (LMMR0–LMMR7). These bits correspond to bits <39:25> of the byte address or D<34:20> of the command cycle.
RSVD	<16:11>	R0	**Reserved.** Read as zero. Writes ignored.

Table 7- 6 LMMR Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
NBANKS	<10:9>	R/W	**Number of Banks.** Specifies the number of individual memory banks (1, 2, 4, or 8) contained on the memory module associated with this register (LMMR0–7). The value of this field determines how many bits of the memory address (0, 1, 2, or 3) are inserted into the bank number.

LMMR <10:9>	Banks per Module	Bits in Bank Number
00	1	0
01	2	1
10	4	2
11	8	3

AW	<8:5>	R/W	**Address Width.** Specifies the number of valid bits in MODULE_ADDR (LMMR<31:17>), starting from the MSB. The remaining bits of MODULE_ADDR are ignored.
IA	<4:3>	R/W	**Interleave Address.** Specifies which interleave, within a group of interleaved modules, is served by the module associated with this register (LMMR0–7).
INT	<2:1>	R/W	**Interleave.** Specifies the number of memory modules interleaved with this module (1, 2, or 4). This value determines the number of bits in the INT field (0, 1, or 2, starting from the LSB) that are compared to the LSBs of the memory address.

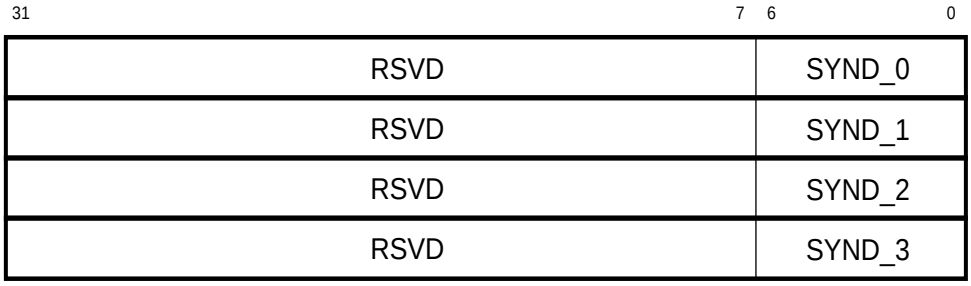
LMMR <2:1>	Modules Interleaved	Address Bits Compared
00	1	0
01	2	1
10	4	2
11	Reserved	Reserved

EN	<0>	R/W	**Enable.** When set, indicates that the module associated with this register (LMMR0–7) is installed, and it is a memory module.

LBESR0- 3—Bus Error Syndrome Registers

Address BB + 0600 06C0
Access RO

The LBESR registers contain the syndrome computed from the LSB Data and ECC fields received during the cycle when an error was detected. The syndrome is the bit- by- bit difference between the ECC check code generated from the received data and the ECC field received over the bus. The LBESR registers lock only on the first occurrence of an ECC error (LBER<CE> or LBER<UCE>). Subsequent ECC errors set LBER<CE2> or LBER<UCE2> until software clears those error bits.



BXB-0105-92

Table 7- 7 LBESR Register Bit Definitions

Name	Bit(s)	Type	Function
RSVD	<31:7>	R0	**Reserved.** Read as zero.
SYND_0	<6:0>	RO	**Syndrome 0**. Syndrome computed from D<31:0> and ECC<6:0> during error cycle.
SYND_1	<6:0>	RO	**Syndrome 1**. Syndrome computed from D<63:32> and ECC<13:7> during error cycle.
SYND_2	<6:0>	RO	**Syndrome 2**. Syndrome computed from D<95:33> and ECC<20:14> during error cycle.
SYND_3	<6:0>	RO	**Syndrome 3**. Syndrome computed from D<127:96> and ECC<27:21> during error cycle.

Syndrome Values

A syndrome of zero indicates no ECC error for the given longword. Table 7- 8 gives the syndromes for all single- bit errors. Any non- zero syndrome not listed in Table 7- 8 indicates a double- bit error.

Table 7- 8 Syndromes for Single- Bit Errors

Bit	Syndrome (Hex)	Bit	Syndrome (Hex)
Data<0>	4F	Data<20>	16
Data<1>	4A	Data<21>	19
Data<2>	52	Data<22>	1A
Data<3>	54	Data<23>	1C
Data<4>	57	Data<24>	62
Data<5>	58	Data<25>	64
Data<6>	5D	Data<26>	67
Data<7>	23	Data<27>	68
Data<8>	25	Data<28>	6B
Data<9>	26	Data<29>	6D
Data<10>	29	Data<30>	70
Data<11>	2A	Data<31>	75
Data<12>	2C	ECC<0>	01
Data<13>	31	ECC<1>	02
Data<14>	34	ECC<2>	04
Data<15>	0E	ECC<3>	08
Data<16>	0B	ECC<4>	10
Data<17>	13	ECC<5>	20
Data<18>	15	ECC<6>	40
Data<19>			

LBECR0,1—Bus Error Command Registers

Address BB + 0700 and BB + 0740
Access RO

The LBECR registers save the contents of the LSB command and address fields during the command cycle when an error is detected. The following errors detected by the CPU module lock the LBECR registers:

LSB uncorrectable ECC error (LBER<1>)
LSB correctable ECC error (LBER<3>)
LSB command parity error (LBER<5>)
LSB CSR data parity error (LBER<7>)
LSB nonexistent address error (LBER<12>)
LSB arbitration drop error (LMERR<10>
LEVI P- map parity error (LMERR<3:0>)
LEVI B- cache tag parity error (LMERR<4>)
LEVI B- cache status parity error (LMERR<5>)
LEVI B- map parity error (LMERR<6>)

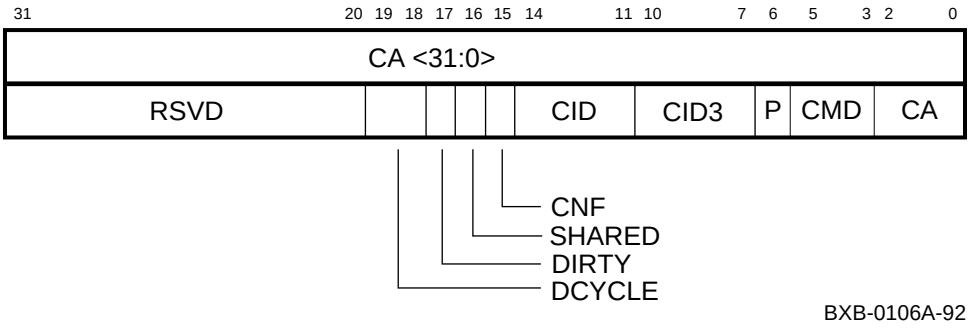


Table 7- 9 LBECR Register Bit Definitions

Name	Bit(s)	Type	Function
CA	<31:0>	RO	**Command/Address.** Contents of D<31:0> during the command cycle.
RSVD	<31:20>	R0	**Reserved.** Read as zero.

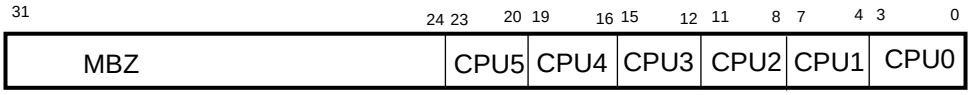
Table 7- 9 LBECR Register Bit Definitions (Continued)

<table>
<tr><th>Name</th><th>Bit(s)</th><th>Type</th><th>Function</th></tr>
<tr><td>DCYCLE</td><td><19:18></td><td>RO</td><td>Data Cycle. Indicates which data cycle had data error.
<table>
<tr><th>LBECR <19:18></th><th>Data Cycle in Error</th></tr>
<tr><td>00</td><td>0</td></tr>
<tr><td>01</td><td>1</td></tr>
<tr><td>10</td><td>2</td></tr>
<tr><td>11</td><td>3</td></tr>
</table></td></tr>
<tr><td>DIRTY</td><td><17></td><td>RO</td><td>Dirty. Set when DIRTY is asserted for the current command.</td></tr>
<tr><td>SHARED</td><td><16></td><td>RO</td><td>Shared. Set when SHARED is asserted for the current command.</td></tr>
<tr><td>CNF</td><td><15></td><td>RO</td><td>Confirmation. Set when CNF is asserted for the current command.</td></tr>
<tr><td>CID</td><td><14:11></td><td>RO</td><td>Commander ID. Contents of REQ<3:0> during command cycle.</td></tr>
<tr><td>CID3</td><td><10:7></td><td>RO</td><td>Commander ID 3. This field is the duplicate of CID (bits <14:11>). It reads the same as CID. In some early versions of the KA7AA module, CID3 reads as zero.</td></tr>
<tr><td>P</td><td><6></td><td>RO</td><td>Parity. Contents of D<38> during command cycle.</td></tr>
<tr><td>CMD</td><td><5:3></td><td>RO</td><td>Command. Contents of D<37:35> during command cycle. CMD is decoded as follows:
<table>
<tr><th>Command</th><th>Function</th><th>DAS Value[1]</th></tr>
<tr><td>000</td><td>Read</td><td>00</td></tr>
<tr><td>001</td><td>Write</td><td>08</td></tr>
<tr><td>010</td><td>Reserved</td><td></td></tr>
<tr><td>011</td><td>Write Victim</td><td>18</td></tr>
<tr><td>100</td><td>Read CSR</td><td>20</td></tr>
<tr><td>101</td><td>Write CSR</td><td>28</td></tr>
<tr><td>110</td><td>Reserved</td><td></td></tr>
<tr><td>111</td><td>Private</td><td>38</td></tr>
</table>
[1] The hex value commonly found in the low byte of this register when less than 8 Gbytes of memory are present in the system.</td></tr>
<tr><td>CA</td><td><2:0></td><td>RO</td><td>Command/Address. Contents of D<34:32> during command cycle.</td></tr>
</table>

LIOINTR—I/O Interrupt Register

Address BSB + 0000
Access R/W

The LIOINTR register is used by the LSB I/O module to signal interrupts from the LSB I/O system to processors.



BXB-0109-92

Table 7- 10 LIOINTR Register Bit Definitions

Name	Bit(s)	Type	Function
MBZ	<31:16>	R/W, 0	**Must Be Zero.** Must always be written as zero.
CPU5	<23:20>	W1S	**CPU5 I/O Interrupt**. When a bit is set in this field, an interrupt is posted to CPU5.
CPU4	<19:16>	W1S	**CPU4 I/O Interrupt**. When a bit is set in this field, an interrupt is posted to CPU3.
CPU3	<15:12>	W1S	**CPU3 I/O Interrupt**. When a bit is set in this field, an interrupt is posted to CPU3.
CPU2	<11:8>	W1S	**CPU2 I/O Interrupt**. When a bit is set in this field, an interrupt is posted to CPU2.
CPU1	<7:4>	W1S	**CPU1 I/O Interrupt**. When a bit is set in this field, an interrupt is posted to CPU1.
CPU0	<3:0>	W1S	**CPU0 I/O Interrupt**. When a bit is set in this field, an interrupt is posted to CPU0.

Interrupt Mapping

Each interrupt target is assigned four bits of interrupt in the LIOINTR register corresponding to the four VAX I/O interrupt levels. A given CPU only looks at the four bits that correspond to its target assignment. This allows interrupts to be targeted to a single CPU or up to six CPUs, depending on the data supplied in the bus CSR write transaction from the I/O module.

This register appears in LSB broadcast space. Writes that address this location are accepted without regard to node ID. Thus, all CPUs accept writes to the register. The register bits are write one to set (W1S). Multiple writes with a value of one to a given bit in this register post an equal number of interrupts to the targeted CPU. Reads to this location are undefined. Any given CPU implements only four bits of this register.

Table 7- 11 shows the mapping of LSB interrupt levels to NVAX+ interrupt levels.

Table 7- 11 LSB Interrupt Mapping

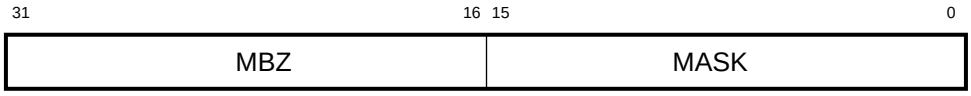
LSB Interrupt Level	NVAX+ IPL (Dec)
3	IPL 23
2	IPL 22
1	IPL 21
0	IPL 20

When any of the four interrupt- pending bits is set, the LEVI gate array correspondingly asserts the IOINTR<3:0> signals. The CPU module then uses these signals to assert the appropriate interrupt request to the NVAX+ chip. The LEVI- A gate array also watches for LSB CSR reads to the LILID0–3 registers in the IOP module. When an LSB CSR read for LILID0 is asserted on the LSB bus, the LEVI- A gate array correspondingly deasserts IOINTR<0>. The LEVI- A gate array performs the same function on LILID3, LILID2, and LILID1.

LIPINTR—Interprocessor Interrupt Register

Address BSB + 0040
Access WO

The LIPINTR register is used by the CPU modules to signal interprocessor interrupts.



BXB-0120-92

Table 7- 12 LIPINTR Register Bit Definitions

Name	Bit(s)	Type	Function
MBZ	<31:16>	R/W, 0	**Must Be Zero.** Must always be written as zero.
MASK	<15:0>	W1S, 0	**Interprocessor Interrupt Mask.** When a given bit is set, an interprocessor interrupt is posted to a specific processor. Bits are mapped to specific CPUs within a multiprocessor system as follows:

LIPINTR Bits	NVAX+ CPU
<15:6>	Not used.
<5>	CPU5
<4>	CPU4
<3>	CPU3
<2>	CPU2
<1>	CPU1
<0>	CPU0

Interprocessor Interrupt

When a processor wishes to post an interrupt to another processor, it simply writes to the LIPINTR register to set the relevant bit. The bits in LIPINTR<3:0> are write one to set (W1S).

This register appears in LSB broadcast space. Writes that address this location are accepted without regard to node ID. Thus, all CPUs accept writes to the register. Reads to this location are undefined.

The contents of LIPINTR<3:0> are qualified by the node ID. If a given CPU node is selected, the LEVI- A gate array asserts the IPINTR signal for one processor external clock. The CPU module ORs this signal and issues the appropriate interrupt request to the NVAX+ chip.

LMODE—Mode Register

Address BB + 0C00
Access R/W

The LMODE register contains mode setup for an operational CPU module (as opposed to the LDIAG register which provides mode setup for a CPU module while running diagnostics).

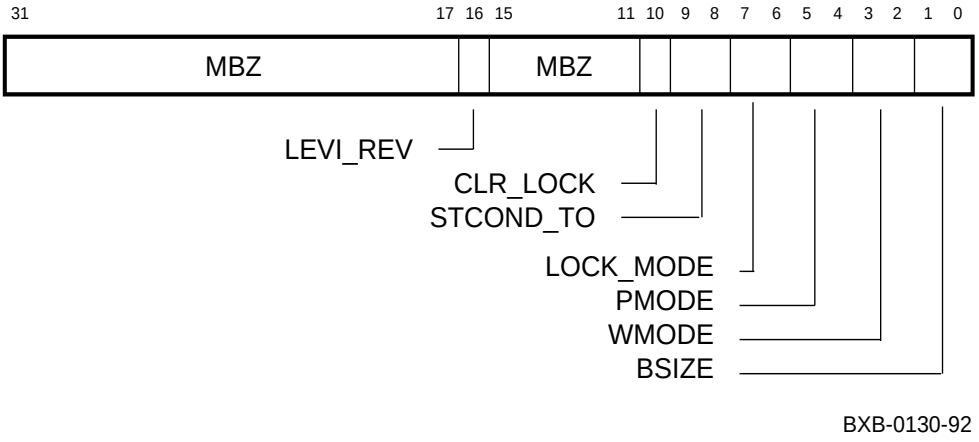


Table 7- 13 LMODE Register Bit Definitions

Name	Bit(s)	Type	Function
MBZ	<31:17>	R/W, 0	**Must Be Zero.** Must always be written as zero.
LEVI_REV	<16>	RO, X	**LEVI Revision.** When clear, indicates pass 1 LEVI-A. When set, indicates pass 2 LEVI- A.
MBZ	<15:11>	R/W, 0	**Must Be Zero.** Must always be written as zero.
CLR_LOCK	<10>	WO, 0	**Clear Lock.** When set, forces LEVI to deassert LSB_LOCKOUT and clear any relevant saved state irrespective of the state of LOCK_TIME, and so on.

Table 7- 13 LMODE Register Bit Definitions (Continued)

<table>
<tr><th>Name</th><th>Bit(s)</th><th>Type</th><th>Function</th></tr>
<tr><td>STCOND_TO</td><td><9:8></td><td>R/W, 0</td><td>Store Conditional Timeout. Specifies the timeout period for lockout assertions. This field is only relevant when LOCK_MODE is non- zero.
<table>
<tr><th>LMODE <9:8></th><th>StoreCond Lockout Time</th></tr>
<tr><td>00</td><td>Count 1 STx_C failure before asserting LSB_LOCKOUT</td></tr>
<tr><td>01</td><td>Count 8 STx_C failures before asserting LSB_LOCKOUT</td></tr>
<tr><td>10</td><td>Count 16 STx_C failures before asserting LSB_LOCKOUT</td></tr>
<tr><td>11</td><td>Count 32 STx_C failures before asserting LSB_LOCKOUT</td></tr>
</table></td></tr>
<tr><td>LOCK_MODE</td><td><7:6></td><td>R/W, 0</td><td>Lock Mode. Specifies how LEVI will drive and respond to LSB_LOCKOUT.
<table>
<tr><th>LMODE <7:6></th><th>LEVI Lockout Behavior</th></tr>
<tr><td>00</td><td>LOCKOUT is off</td></tr>
<tr><td>01</td><td>All writes prevented with LOCKOUT asserted</td></tr>
<tr><td>10</td><td>Only write due to STx_C prevented with LOCKOUT asserted</td></tr>
<tr><td>11</td><td>LEVI behavior undefined</td></tr>
</table></td></tr>
<tr><td>PMODE</td><td><5:4></td><td>R/W, 0</td><td>P- Cache Mode. Allows LEVI to work with CPU chips with varying internal cache organizations. The value of this field for the NVAX+ CPU chip is 10 (bin), which denotes an 8K P- cache and 2K virtual instruction cache.</td></tr>
</table>

Table 7- 13 LMODE Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
WMODE	<3:2>	R/W, 0	**Write Mode.** Selects the behavior of LEVI in response to LSB writes.
BSIZE	<1:0>	R/W, 0	**B- Cache Size.** Tells LEVI about the size of the B- cache.

LMODE <3:2>	LEVI Behavior
00	Use results of P- map lookup to determine invalidate/update.
01	Invalidate the B- cache.
10	Update the B- cache.
11	LEVI behavior undefined.

LMODE <1:0>	B- Cache Size
00	4 Mbytes
01	1 Mbyte
10	16 Mbytes
11	LEVI behavior undefined.

LMERR—Module Error Register

Address BB + 0C40
Access R/W

The LMERR register provides module- specific error information. If any bits are set in this register, NSES (LBER<18>) is also set.

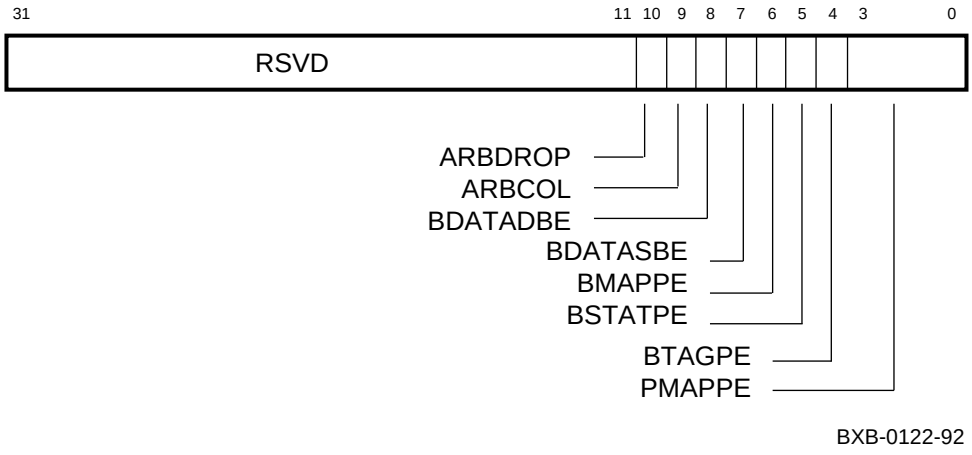


Table 7- 14 LMERR Register Bit Definitions

Name	Bit(s)	Type	Function
MBZ	<31:11>	R/W, 0	**Must Be Zero.** Must always be written as zero.
ARBDROP	<10>	W1C, 0	**Arbitration Drop.** Set when the LEVI arbitration logic detects an LSB cycle in which a node has failed to assert a command after having gained access to the LSB bus. When ARBDROP is set, the LSB command and address are latched in the LBECR register.
ARBCOL	<9>	W1C, 0	**Arbitration Collision.** Set when the LEVI arbitration logic detects an attempt to arbitrate for the LSB bus in an illegal time slot.

Table 7- 14 LMERR Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
BDATADBE	<8>	W1C, 0	**B- Cache Data Double- Bit Error.** When set, indicates that the LEVI chips have detected a double- bit ECC error when unloading the B- cache RAMs. This bit is set when data being transmitted on the LSB bus incurs a double- bit error. The address and the associated ECC syndrome are latched in the LBECR and LBESR registers, respectively.
BDATASBE	<7>	W1C, 0	**B- Cache Data Single- Bit Error.** When set, indicates that the LEVI chips have detected a single- bit ECC error when unloading the B- data RAMs. This bit is set when data being transmitted on the LSB bus incurs a single- bit error. The address and the associated ECC syndrome are latched in the LBECR and LBESR registers, respectively.
BMAPPE	<6>	W1C, 0	**B- Map Parity Error.** When set, indicates that the LEVI- A chip has detected bad parity when reading the B- map RAMs. The associated address is latched in the LBECR register.
BSTATPE	<5>	W1C, 0	**B- Cache Status Store Parity Error.** When set, indicates that the LEVI- A chip has detected bad parity when reading the B- stat RAM. The associated address is latched in the LBECR register for LSB probes. For LEVI probes due to processor misses (LDx_L and so on), BIU_STAT<0> is set and the address is latched in the FILL_ADDR register.
BTAGPE	<4>	W1C, 0	**B- Cache Tag Store Parity Error.** When set, indicates that the LEVI chips have detected bad parity when reading the B- tag RAMs. The associated address is latched in the LBECR register for LSB probes. For LEVI probes due to processor misses (LDx_L and so on), BIU_STAT<0> is set and the address is latched in the FILL_ADDR register.
PMAPPE	<3:0>	W1C, 0	**P- Map Parity Error.** A bit is set in this field if the LEVI- A chip detects bad parity when reading one of the four internal P- map RAM structures. The associated address is latched in the LBECR register.

LLOCK—Lock Address Register

Address BB + 0C80
Access RO

The LLOCK register contains the physical address and lock bit of the most recently executed LDxL instruction that referenced memory space.

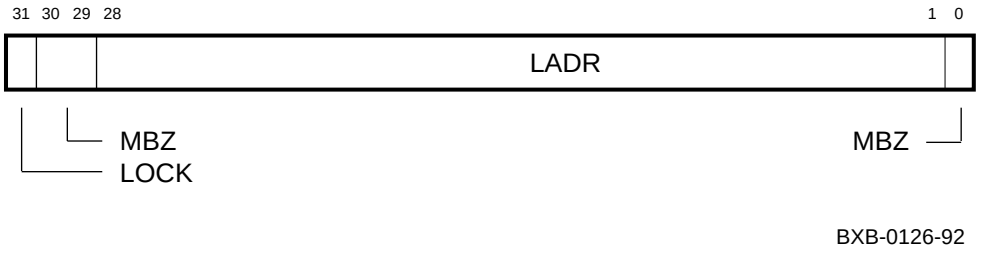


Table 7- 15 LLock Register Bit Definitions

Name	Bit(s)	Type	Function
LOCK	<31>	W1C, 0	**Lock.** When set, indicates that the LLOCK register contains a valid address used in the most recent memory space LDx_L instruction executed by the processor and that no LSB writes that reference the same 64- byte LSB block have occurred. This bit is used to determine the response to a subsequent STx_C instruction. Software can clear this bit explicitly with an LSB write to the 64- byte block referenced in LLOCK<28:1>.
MBZ	<30:29>	R/W, 0	**Must Be Zero.** Must always be written as zero.
LADR	<28:1>	RO, 0	**Lock Address**. Lock address bits <33:6>.
MBZ	<0>	R/W, 0	**Must Be Zero.** Must always be written as zero.

LDIAG—LSB Diagnostic Control Register

Address BB + 0D00
Access R/W

The LDIAG register allows a diagnostic program to manipulate various sections of the CPU module for complete testing.

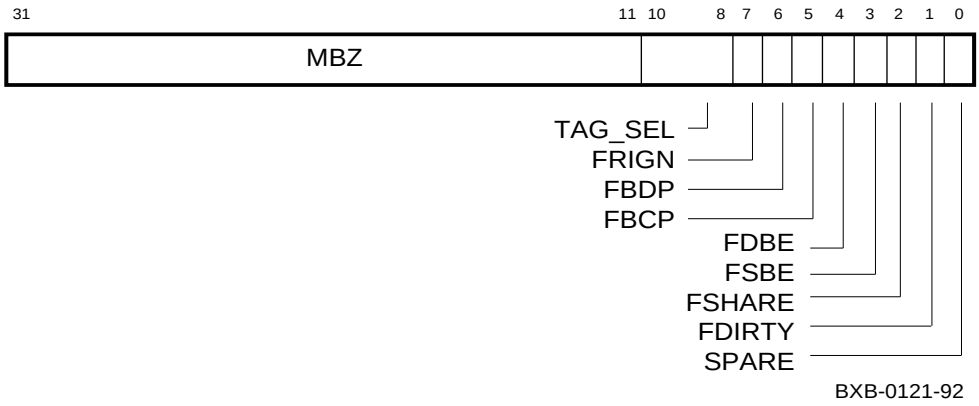


Table 7- 16 LDIAG Register Bit Definitions

Name	Bit(s)	Type	Function
MBZ	<31:11>	R/W, 0	**Must Be Zero.** Must always be written as zero.
TAG_SEL	<10:8>	R/W, 0	**Tag Select.** Specifies which tag store is to be read/written when Gbus$LtagRW is accessed.

LDIAG<10:8>	Tag Store Selected
100	B- cache
010	B- map
001	P- map

When LDIAG is being used to read a tag, only one bit in TAG_SEL is allowed to be set. If more than one bit is set in TAG_SEL when Gbus$LtagRW is written, all specified tags are written.

Table 7- 16 LDIAG Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
FRIGN	<7>	R/W, 1	**Force LSB Ignore.** When set, forces the LEVI gate arrays to ignore all LSB bus traffic except transactions initiated by this node. When FRIGN transitions from set to clear, the LEVI gate array sets LSB ERR to allow all LSB arbitration to resync.
FBDP	<6>	R/W, 0	**Force Bad Data Parity.** When set, forces the LEVI- A gate array to assert bad parity on the LSB during CSR data cycles.
FBCP	<5>	R/W, 0	**Force Bad Command Parity.** When set, forces the LEVI- A gate array to assert bad parity on the LSB during CSR command cycles.
FDBE	<4>	R/W, 0	**Force Double- Bit Error.** Allows a diagnostic program to force the LEVI gate arrays to load data into the B- cache with double- bit ECC errors. When set, LEVI inverts every ECC bit for each longword loaded into the B- cache from the LSB bus. This bit is only relevant when the LEVI gate arrays are loading the B- cache data store (fills).
FSBE	<3>	R/W, 0	**Force Single- Bit Error.** Allows a diagnostic program to force the LEVI gate arrays to load data into the B- cache with single- bit ECC errors. When set, LEVI inverts every ECC bit for each longword loaded into the B- cache from the LSB bus. This bit is only relevant when the LEVI gate arrays are loading the B- cache data store (fills).
FSHARE	<2>	R/W, 0	**Force Share.** When set, the LEVI- A chip responds to all LSB memory transactions with assertion of SHARED.
FDIRTY	<1>	R/W, 0	**Force Dirty.** When set, the LEVI- A chip responds to all LSB memory space read transactions with assertion of DIRTY and supplies the data from the B- cache to the LSB.
SPARE	<0>	R/W, 0	**Spare.** Has no effect on the CPU module.

Diagnostic Notes

The following notes offer additional information for performing diagnostics on the CPU module.

- **How to Make the B- Cache Emulate Main Memory**
  The CPU module can be made to present its cache as main memory to the LSB environment by setting BIU_CTL<FHIT>, LDIAG<FDIRTY>, and LMODE<WMODE>=10 (bin). The selection of this mode is possible under the following two conditions: (1) Only a single CPU module is placed in this mode; (2) No memory module is present in the system.

- **How to Read/Write Tags**
  The combinations of LDIAG, LTAGA, LTAGW, and Gbus$LtagRW registers allow diagnostic programs or error recovery programs to read or write any tag store on the CPU module. LDIAG<TAG_SEL> selects the tag store of interest; LTAGA selects the location in the tag store; LTAGW supplies the value to be written into the tag; and Gbus$LtagRW provides the mechanism. The use of the Gbus register allows LEVI to perform the tag access without the need for any special setup (that is, FRIGN). Even though the Gbus registers are specified to be a byte in length, reads from Gbus$LtagRW return a full longword of data, since no physical Gbus location is actually being read. Gbus address space is used for convenience only.

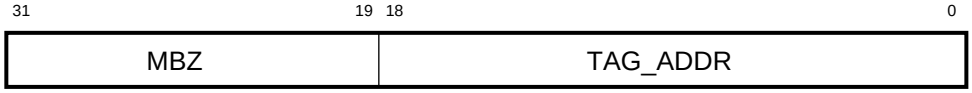
  Writing Tags
  1. Write LDIAG<TAG_SEL> to select the tag store.
  2. Write LTAGA to select the location.
  3. Write LTAGW to specify the value to be written.
  4. Write Gbus$LtagRW with any random data. This action triggers LEVI to perform the tag write as set up.

  Reading Tags
  1. Write LDIAG<TAG_SEL> to select the tag store.
  2. Write LTAGA to select the location.
  3. Read Gbus$LtagRW. This action triggers LEVI to perform the tag read as set up. The data returned is the value from the selected tag location in the format specified by the LTAGW register.

LTAGA—Tag Address Register

Address BB + 0D40
Access R/W

The LTAGA register provides a means by which a diagnostic program can specify the location to be accessed in the CPU cache data and tag RAM structures.



BXB-0123-92

Table 7- 17 LTAGA Register Bit Definitions

Name	Bit(s)	Type	Function
MBZ	<31:19>	R/W, 0	**Must Be Zero.** Must always be written as zero.
TAG_ADDR	<18:0>	R/W, 0	**Tag Address.** Specifies the location (tag address bits <23:5>) to be accessed in the tag store selected by LDIAG<TAG_SEL>.

LTAGW—Tag Write Data Register

Address BB + 0D80
Access R/W

The LTAGW register provides a means by which a diagnostic program can specify the value to be loaded into the CPU caches and tag RAM structures.

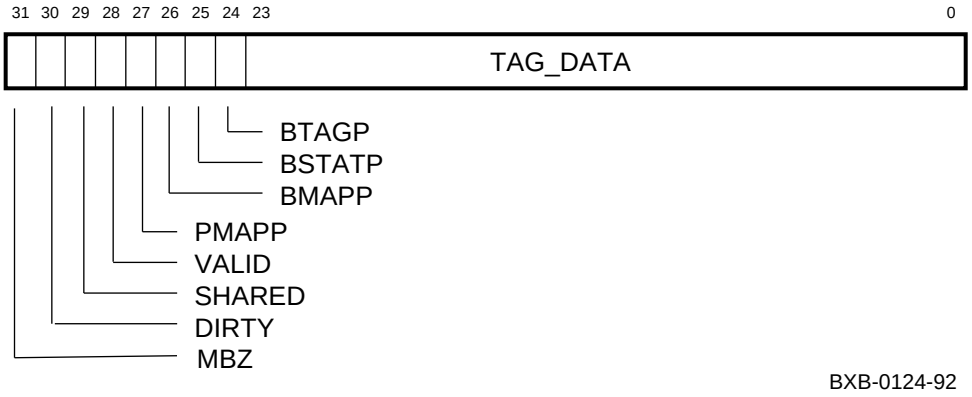


Table 7- 18 LTAGW Register Bit Definitions

Name	Bit(s)	Type	Function
MBZ	<31>	R/W, 0	**Must Be Zero.** Must always be written as zero.
DIRTY	<30>	R/W, 0	**Dirty.** Loaded into the Dirty field (if any) of the B- stat store specified in LDIAG<TAG_SEL> when WTAG (LDIAG<4>) is set.
SHARED	<29>	R/W, 0	**Shared.** Loaded into the Shared field (if any) of the B- stat store specified in LDIAG<TAG_SEL> when WTAG (LDIAG<4>) is set.
VALID	<28>	R/W, 0	**Valid.** Loaded into the Valid field (if any) of the B- stat store specified in LDIAG<TAG_SEL> when WTAG (LDIAG<4>) is set.
PMAPP	<27>	R/W, 0	**P- Map Parity.** Specifies the value to be loaded in the B- map parity location when the Gbus$LtagRW register is written. PMAPP covers tag data bits <23:10> and the valid bit (even parity).

Table 7- 18 LTAGW Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
BMAPP	<26>	R/W, 0	**B- Map Parity.** Specifies the value to be loaded in the B- map parity location when the Gbus$LtagRW register is written. BMAPP covers tag data bits <33:22> and the valid bit (even parity).
BSTATP	<25>	R/W, 0	**B- Stat Parity.** Specifies the value to be loaded in the B-stat parity location when the Gbus$LtagRW register is written. BSTATP covers the Shared, Dirty, and Valid bits.
BTAGP	<24>	R/W, 0	**B- Tag Parity.** Specifies the value to be loaded in the B-tag parity location when the Gbus$LtagRW register is written. BTAGP covers tag data bits <33:22> (even parity).
TAG_DATA	<23:0>	R/W, 0	**Tag Data.** Loaded into the tag store specified in LDIAG<TAG_SEL> when the Gbus$LtagRW register is written. Mapping is performed as follows:

LTAGW<23:0>	Tag	RAM Structure
<23:12>	<33:22>	B- cache tag
<23:12>	<33:22>	B- map tag
<13:0>	<23:10>	P- map tag

LCON0,1—Console Communication Registers

Address BB + 0E00 and BB + 0E40
Access R/W

The LCON register provides a nonmemory communication location for the KA7AA console firmware. The value contained in this register has no direct effect on any CPU module hardware.

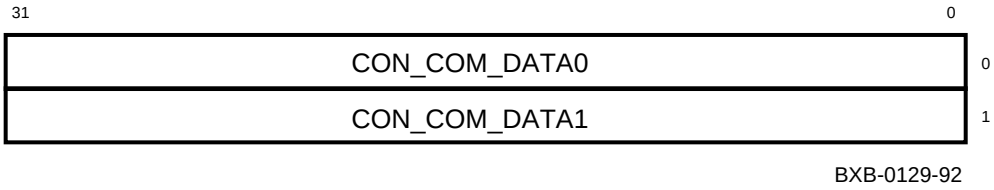


Table 7- 19 LCON Register Bit Definitions

Name	Bit(s)	Type	Function
CON_COM_DATA0	<31:0>	R/W, 0	**Console Communication Data 0.** Data stored in the LCON0 register.
CON_COM_DATA1	<31:0>	R/W, 0	**Console Communication Data 1.** Data stored in the LCON1 register.

LPERF—Performance Counter Control Register

Address BB + 0F00
Access R/W

The LPERF register defines how the LEVI performance registers (LCNTR0, LCNTR1, and LMISSADDR) behave. Each counter register has an event select field, control bits, and an overflow bit. Some of the events to be counted are subject to the node ID mask. The LMISSADDR register loads miss addresses based on the value of the Miss Address Frequency field.

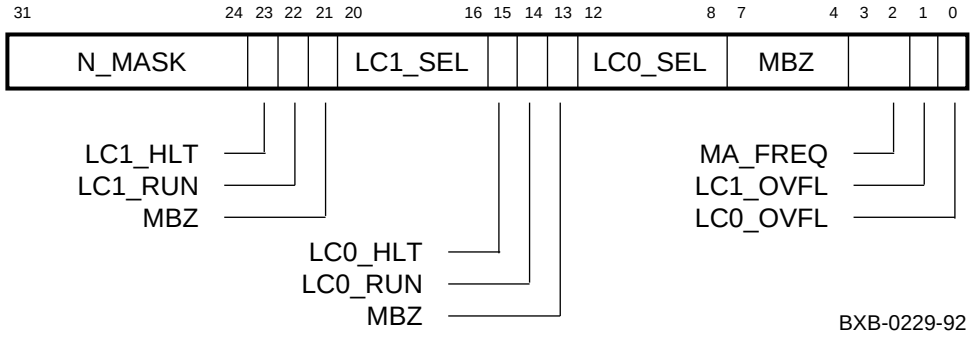


Table 7- 20 LPERF Register Bit Definitions

Name	Bit(s)	Type	Function
N_MASK	<31:24>	R/W, 0	**Node Mask.** When a bit is set in this field, the LSB reads, LSB writes, or victim writes are counted for the associated node. The bits are in one- to- one correspondence, so that bit <31> is associated with node 7, bit <30> with node 6, and so on, except for bit <24>, which is associated with node 0 and the IOP. More than one bit may be set, allowing transactions from multiple nodes to be counted. This field applies to both LCNTR registers.
LC1_HLT	<23>	WO, 0	**LCNTR1 Halt.** Write one to disable LCNTR1 counting.
LC1_RUN	<22>	WO, 0	**LCNTR1 Run.** Write one to enable LCNTR1 counting.
MBZ	<21>	R/W, 0	**Must Be Zero.** Must always be written as zero.

Table 7- 20 LPERF Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
LC1_SEL	<20:16>	R/W, 0	**LCNTR1 Select.** Selects event for the LCNTR1 register.

LPERF <20:16>[1]	Event to Count in LCNTR1
00000	Misses due to reads
00001	Misses due to writes.
00010	Misses due to shared blocks
00011	LDL_X instructions
00100	STC_X failures
00101	LSB B- map hits
00110	Bank conflict delays —LSB cycles
00111	Arbitration losses
01000	Victim buffer hits —wrapped only
01001	CSR reads
01010	CSR writes
01011	LMBPR writes
01100	LSB interrupts
01101	Gbus reads
01110	Gbus writes
01111	LSB reads —subject to node mask
10000	LSB writes —subject to node mask
10001	Victim writes —subject to node mask
10010	Stall cycles
10011	Total memory latency
10100	Carry out from LCNTR0
Other	Reserved

[1] When counting victim buffer hits, LC1_SEL = 01000 counts only hits for which LEVI has to swap (or wrap) the first and second hexwords to satisfy the read request in the proper order (see LVICT register in this section). LC0_SEL = 01000 counts all (wrapped or unwrapped) hits.

LC0_HLT	<15>	WO, 0	**LCNTR0 Halt.** Writing one disables LCNTR0 counting.
LC0_RUN	<14>	WO, 0	**LCNTR0 Run.** Writing one enables LCNTR0 counting.
MBZ	<13>	R/W, 0	**Must Be Zero.** Must always be written as zero.

Table 7- 20 LPERF Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
LC0_SEL	<12:8>	R/W, 0	**LCNTR0 Select.** Selects event for the LCNTR0 register.
MBZ	<7:4>	R/W, 0	**Must Be Zero.** Must always be written as zero.

LPERF <12:8>[1]	Event to Count in LCNTR0
00000	Misses due to reads
00001	Misses due to writes.
00010	Misses due to shared blocks
00011	LDL_X instructions
00100	STC_X failures
00101	LSB B- map hits
00110	Bank conflict delays —LSB cycles
00111	Arbitration losses
01000	Victim buffer hits —wrapped or unwrapped
01001	CSR reads
01010	CSR writes
01011	LMBPR writes
01100	LSB interrupts
01101	Gbus reads
01110	Gbus writes
01111	LSB reads —subject to node mask
10000	LSB writes —subject to node mask
10001	Victim writes —subject to node mask
10010	Stall cycles
10011	Total memory latency
10100	Carry out from LCNTR1
Other	Reserved

[1] When counting victim buffer hits, LC1_SEL = 01000 counts only hits for which LEVI has to swap (or wrap) the first and second hexwords to satisfy the read request in the proper order (see LVICT register in this section). LC0_SEL = 01000 counts all (wrapped or unwrapped) hits.

Table 7- 20 LPERF Register Bit Definitions (Continued)

Name	Bit(s)	Type	Function
MA_FREQ	<3:2>	R/W, 0	**Miss Address Frequency.** Determines how often the LMISSADDR register loads the last miss address.

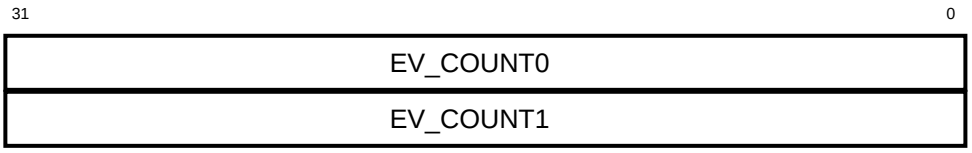
LPERF <3:2>	Miss Frequency
00	Load every 32nd miss address
01	Load every 64th miss address
10	Load every 128th miss address
11	Load every 256th miss address

LC1_OVFL	<1>	RO, 0	**LCNTR1 Overflow.** Records overflow from the LCNTR1 register. Clears when the LCNTR1 register is reset.
LC0_OVFL	<0>	RO, 0	**LCNTR0 Overflow.** Records overflow from the LCNTR0 register. Clears when the LCNTR0 register is reset.

LCNTR0,1—Performance Counter Registers

Address BB + 0F40 and BB + 0F80
Access R/W

The LCNTR registers count events selected by the Select (LC0_SEL and LC1_SEL) and N_MASK fields of the LPERF register. Each register is a 32- bit counter with an associated overflow bit (LPERF<0> for LCNTR0 and LPERF<1> for LCNTR1). With the correct values of the Select fields, the LCNTR registers can be cascaded to form a single 64- bit register. The two counters are enabled and disabled independently through their associated LPERF control bits. The LCNTR registers can be read while the counters are stopped or running. The registers can also be stopped and restarted without resetting to zero. A write to an LCNTR register resets the counter and the associated overflow bit in the LPERF register, and sets the counter to the stopped state. Writes to these registers are ignored.



BXB-0228-92

Table 7- 21 LCNTR Register Bit Definitions

Name	Bit(s)	Type	Function
EV_COUNT0	<31:0>	R/W, 0	**Event Count 0.** Number of events (selected through LPERF<LC0_SEL>) that occurred while LCNTR0 was enabled. Writes ignored.
EV_COUNT1	<31:0>	R/W, 0	**Event Count 1.** Number of events (selected through LPERF<LC1_SEL>) that occurred while LCNTR1 was enabled. Writes ignored.

LMISSADDR—Last Miss Address Register

Address BB + 0FC0
Access RO

The LMISSADDR register captures every *n*th B- cache miss address (determined by LPERF<MA_FREQ>. The miss may be due to a read, a write, or a shared block.

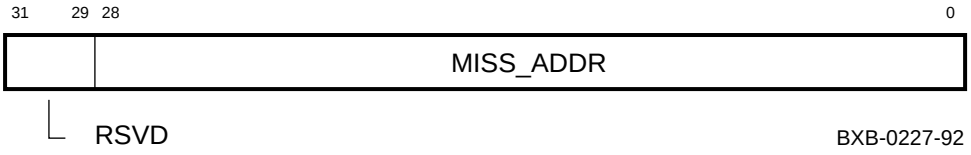


Table 7- 22 LMISSADDR Register Bit Definitions

Name	Bit(s)	Type	Function
RSVD	<31:29>	R0	**Reserved.** Read as zero.
MISS_ADDR	<28:0>	R/W, 0	**Missed Address.** Block address of one of the last *n* B- cache misses. LPERF<MA_FREQ> determines the value of *n*.

Chapter 8

Initialization

The KA7AA CPU module can be initialized in three ways:

- **Power- Up Sequence.** When the LSB system is powered up, the CPU module generates a local reset signal.
- **System Reset.** Whenever the LSB RESET signal is asserted, the CPU module is initialized. LSB RESET can be asserted by any node or from the control panel keyswitch through CCL_RESET.
- **Node Reset.** A single CPU module can be reset by setting LCNR<NRST>.

The processor chip (NVAX+) can be reset independently of the other components on the CPU module through the serial I/O port (see Section 5.1.2).

8.1 Initialization Overview

A CPU reset causes the console code to first invoke the on- board self- test sequence. Self- test begins by testing a very small portion of the CPU logic and gradually expands the scope of testing until all hardware functions of the module have been verified.

Action subsequent to the completion of the CPU module self- test depends on the state of LCNR<RSTSTAT>. If the state of this bit indicates a power- up or system reset, CPU module self- test is followed by CPU-based testing of other system components.

Once all appropriate testing has been completed, the KA7AA console program determines a primary processor. The primary processor is then responsible for displaying the results of all testing, configuring the LSB system (memory, registers, and so on), and creating the software data structures necessary to communicate between processors and the operating system. The console program then enters its input loop.

If the CPU module reset was caused by a node reset, no additional system components are tested. CPU registers are set to their firmware-initialized state, but no other LSB system configuration is performed. There is no change in the primary processor, and the console program enters its input loop.

8.2 Self- Test

CPU self- test is a layered process that is called as the first part of the console entry sequence. It starts with a simple load of code from the SROM into the P- cache and augments itself through additional ROM- resident code that is copied to the B- cache and runs from the B- cache. The process completes by returning a GO/NOGO status to the console entry sequence. The following subsections summarize the various stages in the CPU self- test. A complete description and flowchart of self- test sequences can be found in the *Advanced Troubleshooting* manuals.

8.2.1 SROM Operation

Following the deassertion of reset to the NVAX+ chip, the contents of the SROM are loaded into the internal cache (see Section 5.1.2 the PC is pointed to location zero and instruction execution is started. This code performs the following:

- A quick internal test of the processor chip
- Tests the external B- cache tag, status, and data store RAMs and associated control
- Determines that access to the Gbus resources is operational
- Copies the balance of CPU self- test and the CPU console program from the Gbus ROMs to the B- cache and, following a checksum verification, transfers control to it.

At any point in this process, the SROM code can signal failures through the Gbus$LEDs register.

8.2.2 CPU Module Self- Test

Following the transfer of control from the SROM code to the main body of CPU self- test in the B- cache, the CPU module is tested thoroughly. Highlights include:

- Test of all Gbus resources including the UARTs and the watch chip
- LEVI tests including LSB transfers
- Tests of all RAM structures

8.2.3 Additional Power- Up Testing

If the CPU module self- test completes successfully, additional power- up testing is next performed to verify untested system components. Additional testing performed by all processors includes:

- Tests of the processor/memory LSB interface
- Tests of CPU multiprocessor logic
- Tests of the LSB I/O port module

The boot processor then performs tests on interfaces to the I/O port.

8.3 Console Entry

When the power- up test sequence is complete, the console code entry sequence continues. This section briefly describes the system- level initialization functions that are required to start the operating system, performed by the console code following self- test.

8.3.1 Boot Processor Arbitration

The console program determines the boot processor. A boot processor must also be determined on an interim basis, between phases of power- up test sequence, for the purpose of printing out test results.

The boot processor is selected dynamically. Any processor in a multiprocessor system can become the boot processor. By default, the CPU with the lowest LSB node number that has passed all of its power- up tests thus far, and is eligible, is selected as the boot processor.

If all processors fail self- test, or if all processors have been disabled through console commands from becoming boot processors, then no boot processor is assigned. In this case, a unique code is placed on the LEDs, and all processors monitor the console terminal lines waiting for a sequence to be typed by the operator, which would force one of the processors to become the boot processor.

The console **set cpu** command can be used to change the boot processor once the console is running. Refer to the *Console Reference Manual* for further information on the **set cpu** command.

8.3.2 Boot Processor System Setup

Following configuration of memory, the console creates data structures in memory that are required to communicate between processors and with the operating system. These data structures include:

- Hardware restart parameter block (HWRPB)
- A physical memory descriptor
- A bitmap of good and bad pages of physical memory
- Console routines block (CRB)
- Console terminal block (CTB)

8.3.3 Operating System Startup

The KA7AA console program's primary role in operating system startup is to load and transfer control to the primary bootstrap program. The method the console generally uses to load the primary bootstrap program is called bootblock booting. To begin the boot, the console reads the first logical block (LBN 0) on a disk. This is the bootblock, which contains information that points to the location of the primary bootstrap program on the disk. Using the same routines that loaded the bootblock, the console then uses this information to load the primary bootstrap program.

When booting from a network, the console must request the bootstrap image from an external server.

Once control is passed to the primary bootstrap, the console program's only remaining role is to allow access to console terminal routines and I/O routines. The console terminal routines allow the operating system software to send/receive characters to/from the console terminal. The I/O routines allow the primary bootstrap program to utilize the console boot drivers to load the secondary bootstrap or operating system software.

Chapter 9

Error Handling

Errors detected by or reported to the KA7AA processor can occur anywhere in the system. If errors occur during data movement within the NVAX+ chip or between the NVAX+ and its environment, they are detected by the NVAX+ chip. Errors that occur during data movement within KA7AA hardware external to the NVAX+ (B- cache and LEVI) or during interactions with the LSB bus (other nodes) are detected by the LEVI interface and reported to the NVAX+ chip.

This chapter does not offer an exhaustive discussion of error handling on the KA7AA CPU module. Although it provides detailed parse trees for error isolation, it limits specific error discussions to the necessary. The parse trees and accompanying texts should provide enough information for the experienced system programmer to determine the source of an error on the CPU module and apply a recommended recovery method.

Topics discussed in this chapter include:

- Software Error Handling
- Error Reports
- Console Halt and Halt Interrupt
- Machine Checks
- Hard Error Interrupts
- Soft Error Interrupts
- Kernel Stack Not Valid Exception
- System Environment Errors

9.1 Software Error Handling

Software error handling (by operating system routines) can be logically divided into the following steps:

- State collection
- Analysis
- Recovery
- Retry

The following terminology is used in subsequent discussions:

Fill —Any quadword or data returned to the NVAX+ chip in response to read- type operations. The quadword containing the requested data is a fill.

Flush —Causes victim writebacks to memory of all dirty blocks in the B-cache.

9.1.1 Error State Collection

All relevant state must be collected before error analysis can begin. The stack frame provides the PC/PSL pair for all exceptions and interrupts. For machine checks, the stack frame also provides details about the error.

Besides the stack frame, machine checks and hard and soft error interrupts usually require analysis of other registers. The state of some registers should be saved prior to analysis so that analysis is not complicated by changes in state in the registers as the analysis progresses. Errors incurred during analysis and recovery can be processed within that context.

The state of the following registers should be read and saved:

Ibox

ICSR: Ibox Control and Status Register
VMAR: VIC Memory Address Register

Ebox

ECR: Ebox Control Register

Mbox

TBSTS: TB Parity Status Register
TBADR: TB Parity Address Register
PCSTS: P- Cache Status Register
PCADR: P- Cache Parity Address Register

Cbox

BIU_STAT: Bus or Fill error status
BC_TAG: Contains tag of tag parity, control parity, or fill error
BIU_ADDR: Address associated with cache probe or bus error (BIU_HERR, BIU_SERR, BC_TPERR, BC_TCPERR)
FILL_ADDR: Address associated with fill error, FILL_ECC, or FILL_DPERR
FILL_SYNDROME: Syndrome bits associated with FILL_ADDR

NOTE: In discussions and examples in this chapter, it is assumed that each of these register states is saved in a variable whose name is constructed by prepending "S_" to the register name. For example, the ICSR register would be saved in the variable S_ICSR.

Example 9- 1 shows allocation of memory storage for the error state.

Example 9- 1 Memory Storage Allocation to the Error State

```
;ERROR STATE COLLECTION DATA STORAGE
                              ;IBOX
S_ICSR            .LONG  0    ;IBOX VIC CONTROL AND STATUS REGISTER
S_VMAR            .LONG  0    ;IBOX VIC ERROR ADDRESS REGISTER

                              ;EBOX
S_ESR             .LONG  0    ;EBOX CONTROL AND STATUS REGISTER
                              ;MBOX
S_TBSTS           .LONG  0    ;TB STATUS REGISTER
S_TBADR           .LONG  0    ;TB ADDRESS REGISTER
S_PCSTS           .LONG  0    ;P-CACHE STATUS REGISTER
S_PCADR           .LONG  0    ;P-CACHE ERROR ADDRESS REGISTER
                              ;CBOX
S_BIU_STAT        .LONG  0    ;Bus or fill error status
S_BCTAG           .LONG  0    ;Contains tag of tag_parity, control_parity,
                              ;or fill error
S_BIU_ADDR        .LONG  0    ;Address associated with BIU_HERR, BIU_SERR,
                              ;BC_TPERR, BC_TCPERR
S_FILL_ADDR       .LONG  0    ;Address associated with fill error, FILL_ECC,
                              ;or FILL_DPERR
S_FILL_SYNDROME   .LONG  0    ;Syndrome bits associated with FILL_ADDR
```

Example 9- 2 shows collection error state that would normally be done in the error handling routine. If a second bus or fill error is detected, the SEO second error bit is set, but the error address and status are lost.

Example 9- 2 Collection Error State

```
SAVE_STATE:                             ;Save all error state upon entry
                                        ;to error routine
                                        ;CBOX
                MFPR    #PR19S_BIU_STAT,S_BIU_STAT
                MFPR    #PR19S_BIU_ADDR,S_BIU_ADDR
                MFPR    #PR19S_FILL_ADDR,S_FILL_ADDR
                MFPR    #PR19S_FILL_SYNDROME,S_FILL_SYNDROME
                MFPR    #PR19S_BC_TAG,S_BC_TAG
                                        ;IBOX
                MFPR    #PR19S_ICSR,S_ICSR
                MFPR    #PR19S_VMAR,S_VMAR
                                        ;EBOX
                MFPR    #PR19S_ECR,S_ECR
                                        ;MBOX
                MFPR    #PR19S_TBSTS,S_TBSTS
                MFPR    #PR19S_TBADR,S_TBADR
                MFPR    #PR19S_PCSTS,S_PCSTS
                MFPR    #PR19S_PCADR,S_PCADR

                                        ;System environment
        ;Collection of system environment error registers goes here
```

Additional state collection is recommended while/after flushing the B-cache because certain errors can occur as a result of the flush operation.

9.1.2 Error Analysis

The error condition is analyzed with the error state obtained during the collection process. The purpose is to determine, if possible, what error event caused the error notification, and what other errors may also have occurred. Analysis of machine checks and hard and soft error interrupts should be guided by the parse trees given in the appropriate sections.

Errors detected in or by one of the caches usually result in the cache being automatically disabled. To minimize the possibility of nested errors, analysis and recovery for memory or cache- related errors should be performed with both the P- cache and B- cache disabled (BIU_CTL<0>=0).

In some cases a notification for a single error occurs in two ways. For example, an uncorrectable error in the B- cache data RAMs will cause a soft error interrupt and may also cause a machine check. Software needs to handle cases where a machine check handler clears error bits and then the soft error handler is entered with no error bits set.

In general, an error reporting register can report events that lead to machine checks, soft errors, or hard errors. A given error can result in either a machine check or a soft error interrupt, or both. Events that lead to hard error interrupts generally cannot also cause a machine check or soft error interrupt. However, if a hard error occurs from a write operation, a subsequent read error can result in a machine check with SEO set.

Each category of error—machine check exception, soft error interrupt, and hard error interrupt—is analyzed by tracing an error cause determination parse tree. The IPLs assigned to these errors establish a hierarchy going in descending order from the machine check exception to the soft error interrupt. This hierarchy is important because knowledge of which notification event occurred is used to distinguish between certain error events. For example, an error on the initial fill quadword for a read- lock is distinguished from a fill error on a subsequent quadword by a machine check notification issued on the former error.

9.1.3 Error Recovery

Error recovery consists of clearing any latched error state and restoring the system to normal operation. Analysis and recovery from cache and memory errors require special care and are discussed separately.

Multiple simultaneous errors may make useful recovery impossible. However, in cases where no conflict exists in the reporting of multiple errors, and recovery from each error is possible, then recovery from the set of errors is accomplished by recovering from all. For example, recovery from a P- cache tag parity error and a B- cache correctable data error which are reported together, is possible by following the recovery procedures for each error in sequence.

All recovery procedures in this section assume that only one error is present. None of the procedures are valid in multiple error cases without further analysis.

In some instances, it may be desirable to stop using the hardware that is the source of a large number of errors. For example, if a cache reports an

excessive number of errors, it may be preferable to bypass (disable) it. Software should maintain error counts that can be compared against error thresholds on every error report. If the count (per unit time) exceeds the threshold, the hardware should be disabled.

Cache and memory error recovery requires special considerations:

- Cache and memory error recovery should always be done with the P-cache and VIC off.
- B- cache flush should always be done one block at a time, recapturing the relevant error registers after each block flush.
- Cache coherence requires a specific procedure for reenabling the caches.
- Error recovery should be performed starting with the most distant component and working toward the CPU and Ebox. This rule requires that system environment memory errors be processed first, then B- tag and B- data, P- cache, TB, and, finally, VIC errors.
- BIU and FILL errors are cleared by writing the W1C (write 1 to clear) bits in the BIU_STAT register.
- P- cache tag and data errors are cleared by writing the W1C bits in the PCSTS register. The suggested way to do this is to write a one to the specific error bit. P- cache flush is necessary after P- tag parity errors.
- Translation buffer errors are cleared by writing the W1C bits in the TBSTS register. The suggested way to do this is to write a one to the specific error bit.
- PTE read errors are cleared by writing the W1C bits in the PCSTS register. The suggested way to do this is to write a one to the specific error bit.
- VIC errors are cleared by writing the W1C bits in the ICSR register. The suggested way to do this is to write a one to the specific error bit. VIC flush and reenable is necessary after VIC tag store parity errors.

9.1.3.1 Cache Coherence in Error Handling

Certain procedures must be followed to maintain cache coherence while enabling NVAX+ caches. Since many errors cause caches to be disabled, and since cache and memory error recovery is normally done with the P-cache and VIC off, the complete cache enable procedure is done as part of recovery from all cache and memory errors.

The VIC is not automatically kept coherent with memory. It is flushed as a side effect of the REI instruction (as required by the VAX architecture). Normally in error recovery, there is no need to flush the VIC. For consistency and for the sake of beginning error retry in a known state, flushing the VIC during error recovery is recommended. However, in the event of VIC tag parity errors, the complete VIC flush procedure must be done.

The translation buffer is not automatically kept coherent with memory. Software uses the TBIS and TBIA functions to maintain coherence, and the LDPCTX instruction clears the process PTEs in the TB. Normally in error recovery, there is no need to flush the TB. For consistency and for the sake of beginning error retry in a known state, flushing the TB during error recovery is recommended. When a TB parity error occurs, The

Mbox hardware flushes the TB by itself (through an internally generated TBIA), but it would be appropriate for software to test the TB after a parity error.

9.1.3.2 Cache Enable, Disable, and Flush Procedures

Before NVAX+ caches are enabled, they must be flushed. The caches then must be enabled in a specific order. This is necessary for coherence between the B- cache, P- cache, and memory. For simplicity, one procedure is given here for enabling the NVAX+ caches, even though variations on the procedure could also produce correct results.

NOTE: In error handling, the VIC and the P- cache are disabled.

To disable NVAX+ caches for error handling, proceed as follows:

1. Disable the VIC (MTPR to ICSR)
2. Disable the P- cache (MTPR to PCCTL)
3. Disable the B- cache (MTPR to BIU_CTL)

The procedure to enable the NVAX+ caches after an error is the same as the one used to initialize the caches after power- up. This procedure ensures that error retry/restart occurs with the caches in a known state. Proceed as follows:

1. Disable all NVAX+ caches and the B- cache.
2. Flush the B- cache.
3. Enable the B- cache (MTPR to BIU_CTL).
4. Flush the P- cache (Loop on MTPR to PCTAG IPRs).
5. Enable the P- cache (MTPR to PCCTL).
6. Flush the TB.

```
MTPR   #0,   #PR19S_TBIA
```

7. Flush the VIC (loop on MTPRs to VMAR and VTAG, writing different initial values into the left and right banks).
8. Enable the VIC (MTPR to ICSR).

9.1.3.3 Extracting Data from the B- Cache

To extract data from the B- cache, place the B- cache in force hit mode with an MTPR to BIU_CTL.

After the B- cache is flushed, set the B- cache in force hit mode and extract the data. Note that the code that executes this procedure and its local data must be in I/O space. The TB entries (PTEs) that map this code and local data must be fixed in the TB. This is most easily done by flushing the TB through an MTPR to TBIA and then accessing all the relevant pages in sequence.

With the B- cache in force hit mode, a read in memory space of any address whose index portion matches the index of the cache data will return the data (provided there is no uncorrectable data RAM error). This is most easily accomplished by reading from the true address of the data.

NOTE: In force hit mode, fill ECC errors are detected. Software should prepare for an ECC error (BIU_STAT<FILL_ECC>).

9.1.3.4 Cache and TB Test Procedures

Testing is generally done using the force hit mode of a cache. The code and data of the test procedure must reside in I/O space. Assuming memory management is enabled during the test procedure, the needed PTEs must be in the TB before entering force hit mode in the P- cache or B- cache. For the B- cache, testing should be done with errors disabled (DIAG_CTL<DISABLE_ERRORS>=1). The ECC logic should be tested thoroughly on one location by forcing various check bit patterns and examining the syndrome latched on the read (FILL_SYNDROME is loaded on every read in B- cache disable errors mode). Presently FILL_SYNDROME is valid if an error occurs and the syndrome bits for the last fill cannot be recovered with an IPR_RD of this register otherwise. P- cache and VIC parity checking should be tested by writing bad parity into the arrays. TB testing can be accomplished by writing to MTBTAG and MTBPTE (being careful not to change any TB entry necessary for the test code and data, and not to cause two TB entries to exist for one address). Probe read and probe write (setting PSL<PRV_MOD>) are then used to verify the protection bits. Testing the Modify bit in the memory management stack frame would be difficult, though approaches exist.

9.1.4 Error Retry

Error retry is a function of the error notification (machine check or error interrupt), error type, and error state. The sections below specify the conditions under which the instruction stream may be restarted.

If retry is to be attempted, the stack must be trimmed of all parameters except the PC/PSL pair. An REI will then restart the instruction stream and retry the error. Some form of software loop control should be provided to limit the possibility of an error loop. Note that pending error interrupts may be taken before the retry occurs, depending on the IPL of the interrupted or machine checked code.

Strictly speaking, an REI from a hard or soft error interrupt handler is not a retry since these interrupts are recognized between macroinstructions. A machine check exception is an instruction abort, and an REI from the handler will cause the failing instruction to be retried. What these cases all have in common is that the interrupted instruction stream is restarted. This is only done when the result of error analysis and recovery is such that all damaged state has been repaired and there is no reason to suspect that incorrect results will be produced if the image is restarted and another error does not occur.

If complete recovery from one or more errors is not possible, software must determine if the error is fatal to the current process, to the processor, or to the entire system, and take the appropriate action.

It is expected that software handles machine checks, soft error interrupts, and hard error interrupts independently. For example, after handling a machine check from which retry is to occur, software does not check for errors that might cause a pending hard or soft error interrupt.

NOTE: In the case of HARD_ERROR interrupt, the machine check code must not clear the BIU_STAT register if the interrupt is to be taken.

The machine check handler is exited by REI (after trimming the machine check information off the stack). If the IPL of the machine checked instruction stream is low enough, any pending hard or soft error interrupt is taken before the retry occurs. However, if the interrupted instruction stream was running at a high IPL, the system will continue without dealing with the remaining errors.

Multiple errors can be reported at the same time. In some cases the NVAX+ pipeline will contain multiple operand prefetches to the same memory block. This can cause multiple errors from a single nontransient failure. Two separate errors could also occur at nearly the same time and be reported simultaneously.

Multiple error scenarios can be grouped into the following classes:

- Class 1 errors are multiple distinct errors for which no error report interferes with the analysis of any other (that is, no lost error bit is set).
- Class 2 errors are multiple errors that could have been caused by the NVAX+ pipeline issuing more than one reference to a given block before the error interrupt or the machine check forced a pipeline flush.
- Class 3 errors are multiple errors for which analysis is complicated because the reports interfere with each other.

This chapter treats class 1 errors as separate errors, each with its own recovery. Retry or restart evaluation is based on the cumulative result of the recovery and repair procedures for each error.

The chapter identifies and deals with specific cases of class 2 errors in which lost errors are tolerated. These cases are selected because the NVAX+ pipeline can easily cause them (given one error), and because sufficient safeguards exist to ensure that correct operation is maintained.

Class 3 errors are generally not considered recoverable and the system is crashed.

Lost correctable errors are not considered serious problems, since hardware recovers from these errors automatically.

9.2 Error Reports

The KA7AA processor reports errors to the operating system software through machine checks and interrupts. Error notification occurs through one of the following events listed in order of decreasing severity:

- **Console error halt**
  A halt to console mode is caused by one of several errors such as interrupt stack not valid. For certain halt conditions, the console prompts for a command and waits for operator input. For other halt conditions, the console may attempt a system restart or system bootstrap.
- **Machine check**
  A hardware error occurred synchronous to the execution of instructions. Instruction- level recovery and retry may be possible.
- **Kernel stack not valid**
  During exception processing, a memory management exception occurred while trying to push information on the kernel stack.
- **Hard error interrupts**
  A hardware error occurred asynchronous to the execution of instructions. Usually, data is lost or state is corrupted, and instruction- level recovery may not be possible.
- **Soft error interrupts**
  A hardware error occurred asynchronous to the execution of instructions. The error is not fatal to the execution of instructions, and instruction- level recovery is usually possible.

All errors (except those leading to a console halt) go through SCB (system control block) vector entry points and are handled by service routines provided by the operating system. A console halt, on the other hand, transfers control to a hardware- prescribed I/O space address. Software- driven recovery or retry is not recommended for errors resulting in console halt.

Table 9- 1 gives a summary of errors and notification entry points for the various error categories. For each SCB entry point, discussions provide the following information:

- Parameters pushed on the stack
- Defined failure codes
- What additional information exists and should be collected for analysis
- How to restore the state of the machine and what level of recovery is possible

Error categories are discussed in the next sections.

Table 9- 1 Error Categories by SCB Entry Points

SCB Index (Hex)	Error Category	Summary of Errors
None	Console halt[1]	Interrupt stack not valid Kernel mode halt Double error halt Illegal SCB vector
04	Machine check	Memory management Interrupt Microcode- detected CPU errors CPU stall timeout TB parity errors VIC tag or data parity errors Uncorrectable data read errors CACK_H error on reads
08	Kernel stack not valid	Error during exception processing
54	Soft error interrupt	VIC tag or data parity errors P- cache tag or data parity errors B- cache tag parity error on reads Uncorrectable data read errors Correctable data read errors
60	Hard error interrupt	Uncorrectable data errors on write operations B- cache tag parity error on writes CACK_H error on writes

[1] Does not go through an SCB entry point.

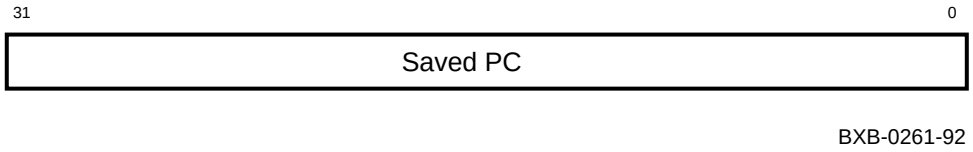
9.3 Console Halt and Halt Interrupt

A console halt is a transfer of control by the NVAX+ CPU microcode directly into console macrocode at the address stored in the CHALT register. Console halts are initiated:

- At power- up
- When the microcode detects certain double- error conditions, specifically when a second error occurs during error processing
- When LBER<NHALT> is set
- When Ctrl/P is typed on the console terminal
- When the system is reset
- When a kernel- mode HALT instruction is executed
- When the external signal HALT_H is asserted

No exception stack frame is associated with a console halt. Instead, the state is saved in the SAVPC (IPR42) and SAVPSL (IPR43) processor registers. Figure 9- 1 shows the SAVPC IPR.

Figure 9- 1 Console Saved PC



The PSL halt code, MAPEN<0>, and a validity bit are saved in SAVPSL. Figure 9- 2 shows the SAVPSL IPR.

Figure 9- 2 Console Saved PSL

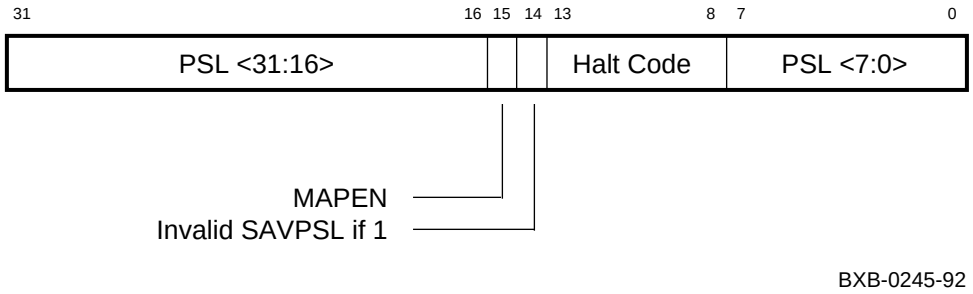


Table 9- 2 lists the possible halt codes that can appear in SAVPSL<13:8>.

Table 9- 2 Console Halt Codes

Code (Hex)	Mnemonic	Description:
02	ERR_HLTPIN	HALT_H asserted
03	ERR_PWRUP	Initial power- up
04	ERR_INTSTK	Interrupt stack not valid
05	ERR_DOUBLE	Machine check during exception processing
06	ERR_HLTINS	HALT instruction in kernel mode
07	ERR_ILLVEC	Illegal SCB vector (bits <1:0> = 11)
08	ERR_WCSVEC	WCS SCB vector (bits <1:0> = 10)
0A	ERR_CHMFI	CHMx on interrupt stack
10	ERR_IE0	ACV/TNV during machine check processing
11	ERR_IE1	ACV/TNV during kernel- stack- not- valid processing
12	ERR_IE2	Machine check during machine check processing
13	ERR_IE3	Machine check during kernel- stack- not- valid processing
19	ERR_IE_PSL_26_24_101	PSL<26:24> = 101 during interrupt or exception
1A	ERR_IE_PSL_26_24_110	PSL<26:24> = 110 during interrupt or exception
1B	ERR_IE_PSL_26_24_111	PSL<26:24> = 111 during interrupt or exception
1D	ERR_REI_PSL_26_24_101	PSL<26:24> = 101 during REI
1E	ERR_REI_PSL_26_24_110	PSL<26:24> = 110 during REI
1F	ERR_REI_PSL_26_24_111	PSL<26:24> = 111 during REI
3F	ERR_SELFTEST_FAILED	Microcoded power- up self- test failed

At the time of the halt, the current stack pointer is saved in the appropriate IPR (0 to 4), and SAVPSL<31:16,7:0> are loaded from PSL<31:16,7:0>. SAVPSL<15> is set to MAPEN<0>. SAVPSL<14> is clear if the PSL is valid, and set if it is not (SAVPSL<14> is undefined after a halt due to a system reset). SAVPSL<13:8> is set to the console halt code.

To complete the hardware restart sequence and thereby pass control to the console macrocode, the CPU is initialized as shown in Table 9- 3.

Table 9- 3 CPU State Initialized on Console Halt

CPU State	Initialized Value
SP	IPR 4 (IS)
PSL	041F 0000
PC	from CHALT register
MAPEN	0
ICCS	0 (after reset, code = 3, only)
SISR	0 (after reset, code = 3, only)
ASTLVL	4 (after reset, code = 3, only)
PAMODE	0 (after reset, code = 3, only)
BPCR<31:16>	FECA (after reset, code = 3, only)
CPUID	0 (after reset, code = 3, only)
All else	Undefined

9.4 Machine Checks

A machine check exception indicates a serious system error. Under certain conditions, the error may be recoverable by restarting the instruction. The recoverability is a function of the following parameters:

- Machine check code
- VAX Restart bit (VR) in the machine check stack frame
- Opcode
- State of PSL<FPD>
- State of certain second error bits in internal error registers
- External error state

A machine check results from an internally detected consistency error (for instance, the microcode reaches an "impossible" state), or hardware-detected error (for instance, an uncorrectable FILL_ECC error on a data read).

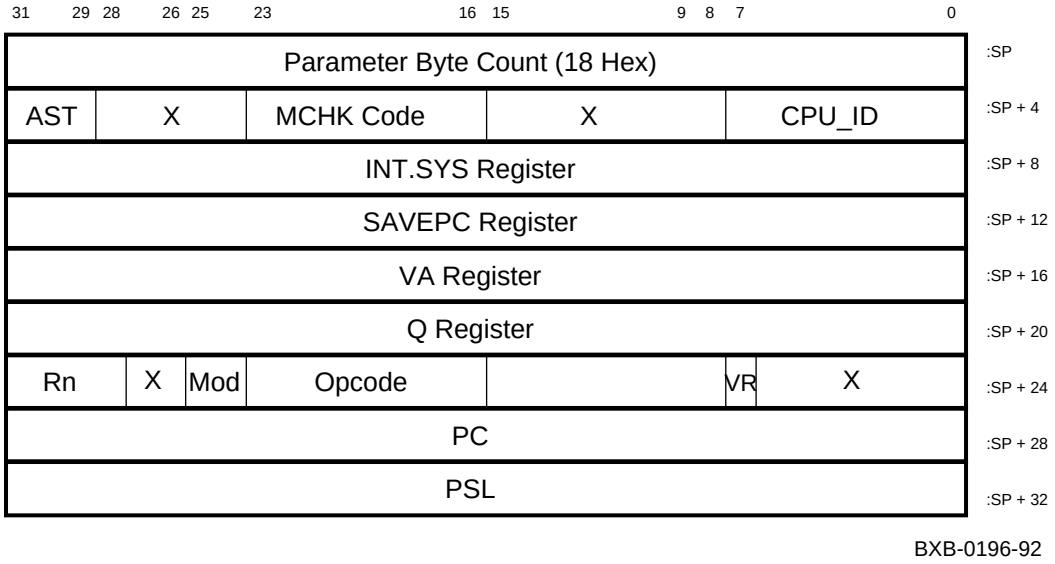
Technically, a machine check is an aborted macro instruction. The NVAX+ microcode attempts to convert the condition to a fault by unwinding the current instruction, with no guarantee that the instruction can be properly restarted. As much information as possible is pushed on the machine check stack frame and provided in other error registers. The rest of the error parsing is left to the operating system.

When the software machine check handler routine receives control, it must explicitly acknowledge receipt of the machine check early in the routine to clear the internal machine- check- in- progress flag with the following instruction:

```
MTPR     #0, #PR19S_MCESR
```

Figure 9- 3 shows the machine check stack frame.

Figure 9- 3 Machine Check Exception Stack Frame



BXB-0196-92

Table 9- 4 describes the contents of the fields in the machine check stack frame. Fields not explicitly defined in Figure 9- 3 are UNDEFINED.

Table 9- 4 Machine Check Stack Frame Fields

Longword Bits	Contents
SP+0 <31:0>	**Byte count.** The size of the stack frame in bytes, not including the PC, PSL, or the byte count longword. Stack frame PC and PSL values should always be referenced using this count as an offset from the stack pointer.
SP+4 <31:29>	**AST LVL.** The current value of the register.
<23:16>	**Machine check code**. The reason for the machine check, as listed in Table 9- 5.
<7:0>	**CPUID**. Contains the current value of the CPUID register.
SP+8 <31:0>	**INT.SYS register.** Contains the value of the INT.SYS register. This value is read onto the A- bus by the microcode.
SP+12 <31:0>	**SAVEPC register**. The SAVEPC register which is loaded by microcode with the PC value in certain circumstances. It is used in error handling for PTE read errors with PSL<FPD> set in this stack frame.
SP+16 <31:0>	**VA register.** The contents of the Ebox VA register, which may be loaded from the output of the ALU.
SP+20 <31:0>	**Q register**. The contents of the Ebox Q register, which may be loaded from the output of the shifter.
SP+24<31:28>	**Rn**. The value of the Rn register, which is used to obtain the register number for the CVTPL and EDIV instructions. In general, the value of this field is UNPREDICTABLE.
<25:24>	**Mode**. A copy of the current mode field, PSL<CUR_MOD>.
<23:16>	**Opcode.** Bits <7:0> of the instruction opcode. The FD bit is not included.
<7>	**VR.** The VAX Restart bit, which is used to communicate restart information between the microcode and the operating system. When set, this bit indicates that no architectural state has been changed by the instruction that was executing when the error was detected. When clear, it indicates that architectural state was modified by the instruction.
SP+28 <31:0>	**PC.** The value of the program counter at the time of the fault.
SP+32 <31:0>	**PSL.** The value of the processor status longword at the time of the fault.

Table 9- 5 describes the machine check codes.

Table 9- 5 Machine Check Codes in the Stack Frame

Code (Hex)	Mnemonic	Description:
01	MCHK_UNKNOWN_MSTATUS	Unknown memory management status error
02	MCHK_INT.ID_VALUE	Illegal interrupt ID error
03	MCHK_CANT_GET_HERE	Illegal microcode dispatch occurred
04	MCHK_MOVC.STATUS	Illegal combination of state bits detected during string instruction
05	MCHK_ASYNC_ERROR	Asynchronous hardware error occurred
06	MCHK_SYNC_ERROR	Synchronous hardware error occurred

Figure 9- 4 is a parse tree that can be used to analyze the cause of a machine check exception. The parse trees include error isolation at the processor level and module level. Discussions that follow provide detailed descriptions of the errors, give the recovery procedures and, when appropriate, indicate the conditions for retry.

NOTE: In the error descriptions it is assumed that the state being analyzed is the saved state. Otherwise the state could change during the analysis procedure, possibly leading to incorrect conclusions.

Figure 9- 4 Machine Check Exception Parse Tree

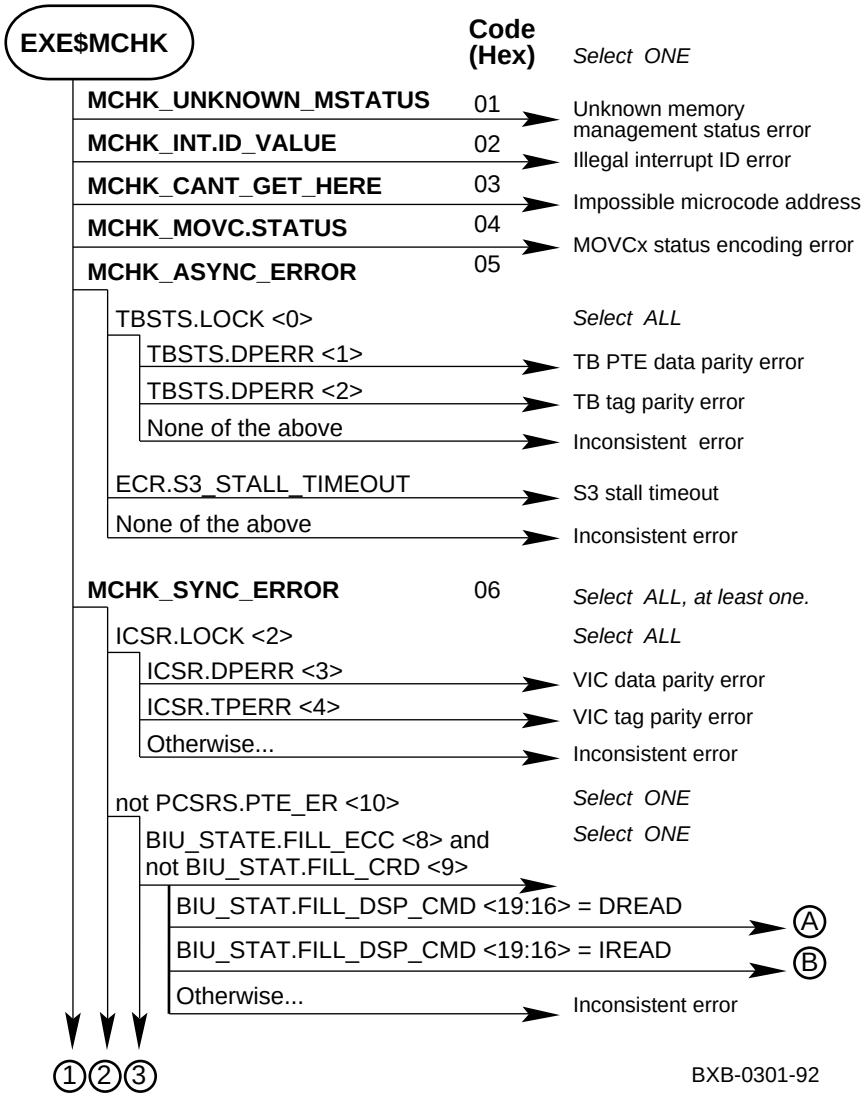


Figure 9- 5 Machine Check Exception Parse Tree (Continued)

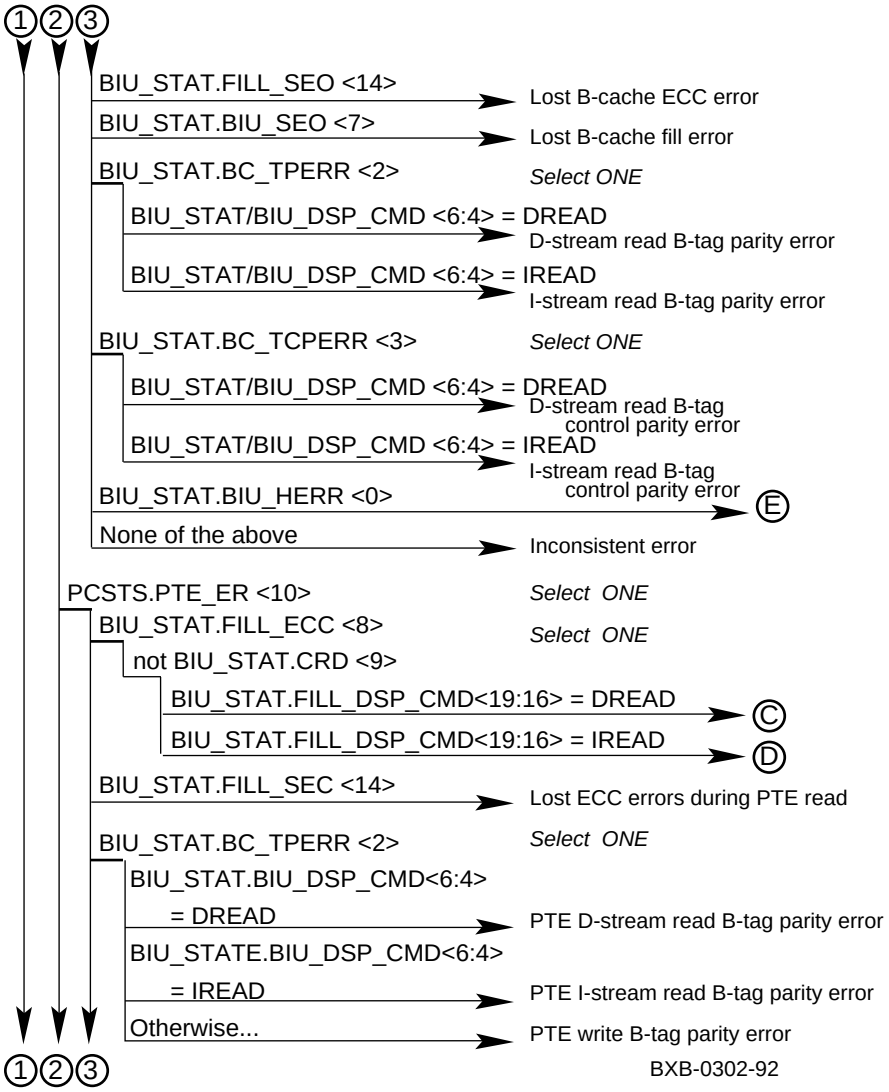


Figure 9- 5 Machine Check Exception Parse Tree (Continued)

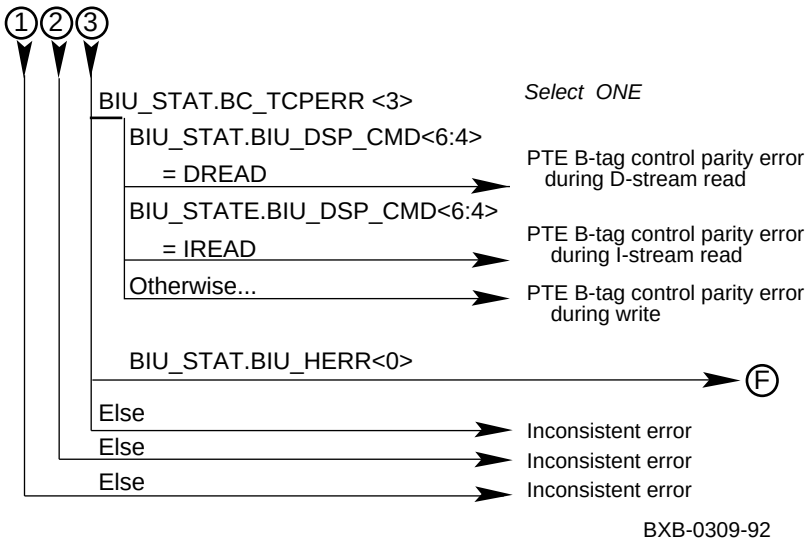


Figure 9- 5 Machine Check Exception Parse Tree (Continued)

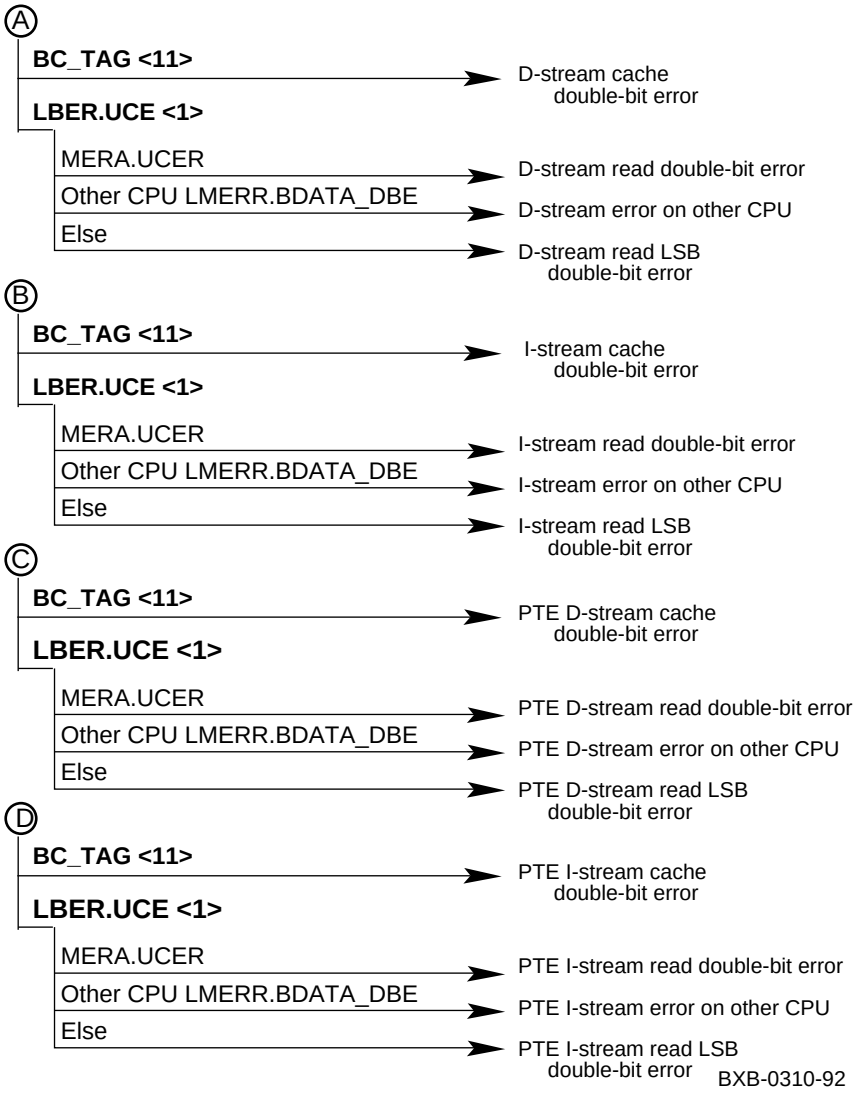


Figure 9- 5 Machine Check Exception Parse Tree (Continued)

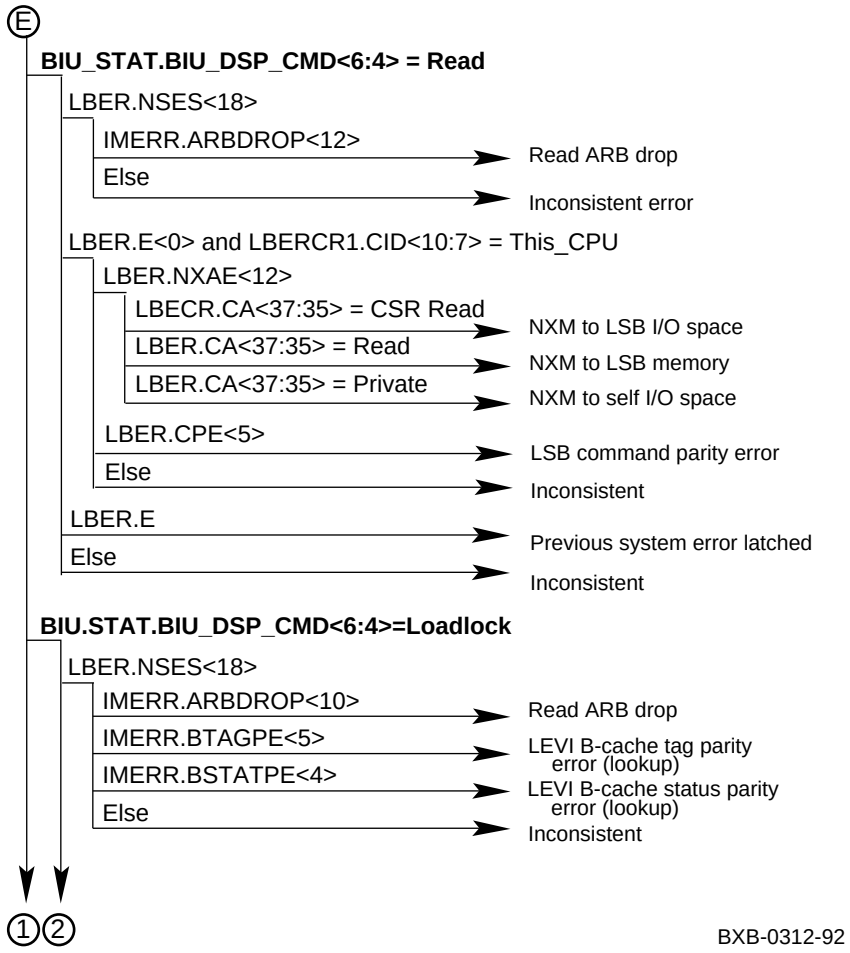


Figure 9- 5 Machine Check Exception Parse Tree (Continued)

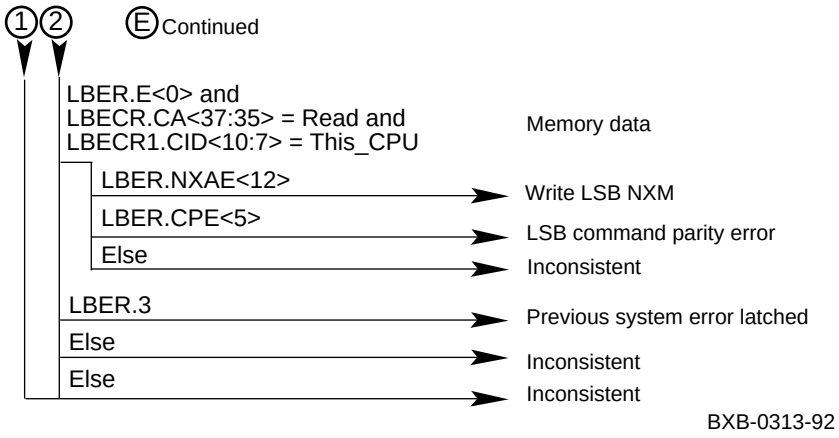
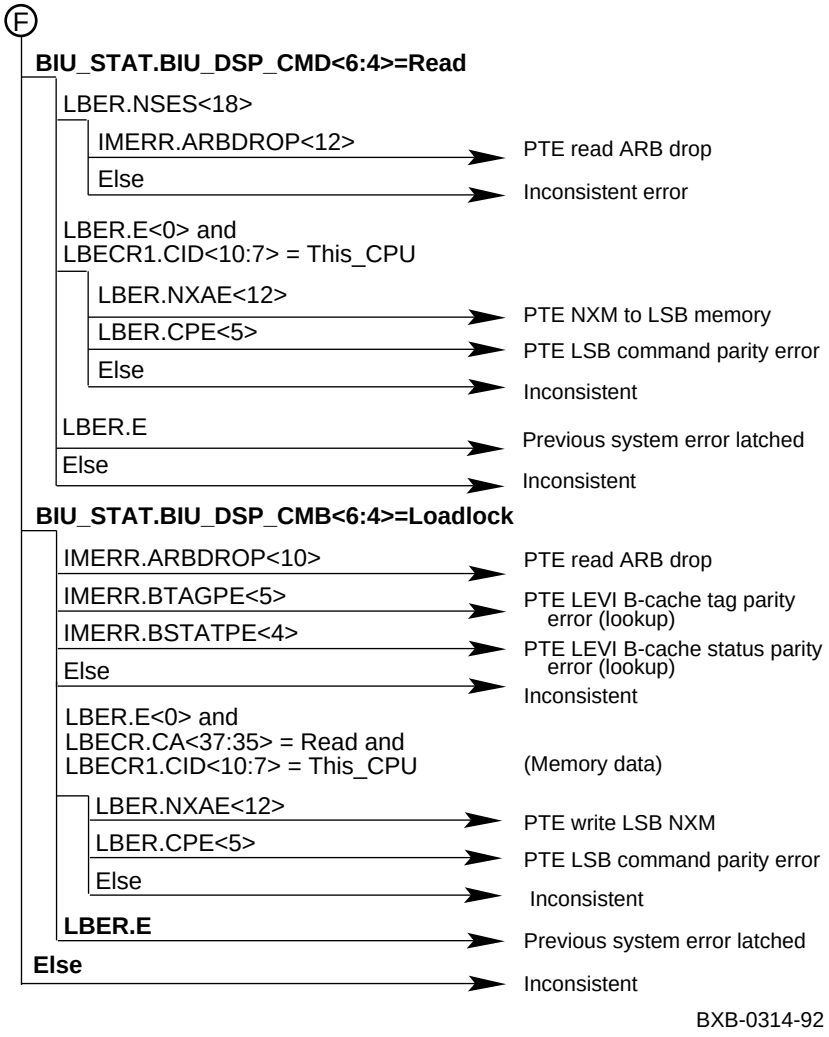


Figure 9- 5 Machine Check Exception Parse Tree (Continued)



9.4.1 MCHK_UNKNOWN_MSTATUS

Description: An unknown memory management status was returned from the Mbox in response to a microcode memory management probe. This error is probably caused by an internal error in the Mbox, Ebox, or microsequencer.

Recovery procedure: No explicit error recovery is required.

Retry condition: This error only happens in microcode processing of memory management faults for a virtual memory reference. Retry if:

(VR = 1) OR (PSL<FPD> = 1)

9.4.2 MCHK_INT.ID_VALUE

Description: An illegal interrupt ID was returned in INT.SYS during interrupt processing in microcode. This error is probably caused by an internal error in the interrupt hardware, Ebox, or microsequencer.

Recovery procedure: No explicit error recovery is required.

Retry condition: This error only happens in microcode processing of interrupts that occur between instructions or the middle of interruptible instructions. Retry if:

(VR = 1) OR (PSL<FPD> = 1)

9.4.3 MCHK_CANT_GET_HERE

Description: Microcode execution reached a presumably impossible address. This error is probably caused by an internal error in the Ebox or microsequencer.

Recovery procedure: No explicit error recovery is required.

Retry condition: Retry if:

(VR = 1) OR (PSL<FPD> = 1)

9.4.4 MCHK_MOVC_STATUS

Description: During execution of MOVCx, the two state bits that encode the state of the move (forward, backward, fill) were set to the fourth (illegal) combination. This condition is probably caused by an internal error in the Ebox or microsequencer.

Recovery procedure: No explicit error recovery is required.

Retry condition: Because the state bits encode the operation, the instruction cannot be restarted in the middle of the MOVCx. If software can determine that no specifiers have been overwritten (MOVCx destroys R0–R5 and memory due to string writes), the instruction can be restarted from the beginning by clearing PSL<FPD>. Retry should be done only if the source and destination strings do not overlap and if:

PSL<FPD > = 1

9.4.5 MCHK_ASYNC_ERROR

This machine check code reports serious errors that interrupt the microcode at an arbitrary point. Many internal machine states (that is, bits in PSL, PC, and SP) are questionable and recovery is typically not possible.

TB Parity Error

Description: Parity errors in tags and PTE data in the TB that hit cause an asynchronous machine check by directly forcing a microtrap in the microsequencer.

Since the Ibox is nearly always able to issue instruction prefetches, TB parity errors could occur at any time, making it impossible to determine what machine state is incorrect. There is no guarantee that all writes with

a different PSL<CUR_MOD> completed successfully. Therefore, even the stack frame PSL<CUR_MOD> cannot be used to determine whether system data is uncorrupted.

Recovery procedure: Clear TBSTS<LOCK>.

Retry condition: Retry is not possible. Crash the system.

Ebox Stage 3 Stall Timeout Error

Description: Stage 3 stall timeout errors occur when the Ebox microcode is stalled waiting for some result or action that will probably never happen. S4 stalls in the Ebox cause S3 stalls and, therefore, can lead to an S3 stall timeout. Additionally, field queue stall and instruction queue stall can cause this timeout. The timeout can occur in any microflow for a number of reasons. Machine state may be corrupted because of an internal error in NVAX+ such that one box would be waiting for another to do something which the second box will not do. An example would be the Ebox microcode expecting one more source specifier than the Ibox delivered and timeout occurs while the Ebox is waiting. S3 timeout errors can be caused by failures of various pipeline control circuits in the Ebox. Also, this error can be caused by a deadlock within a box or across multiple boxes.

Recovery procedure: Clear ECR<TO_OCCURRED>.

Retry condition: It is not possible to determine what machine state is incorrect. Retry is not possible. Crash the system.

9.4.6 MCHK_SYNC_ERROR

This machine check code reports errors that occur in memory or I/O space instruction fetches or data reads. Except in the case of PTE read errors, NVAX+ state should be consistent since microcode has to explicitly access an operand or instruction in order to incur this error. Microcode does not access memory results or dispatch for a new instruction execution with the NVAX+ in an inconsistent state.

PTE read errors on write transactions can cause a microtrap at an arbitrary time, and so the NVAX+ state may be inconsistent.

Many of the error events described below for synchronous machine checks are possible causes. If more than one is present, there is no way to determine which actually caused the machine check. If only one cause is discovered, then the machine check can be attributed to that cause. The reason multiple causes may be present is that the NVAX+ CPU prefetches instructions and data. If the CPU branches or takes an exception before using data it has requested, then the pending machine check is taken as a soft error interrupt (although it might not be recoverable).

If multiple errors occur, recovery and retry may be possible. It is recommended that retry from multiple errors be done only if one error report does not interfere with analysis of, and recovery from, another error.

If two errors are entirely separate, neither interfering with the analysis and recovery of the other, then it is acceptable to retry from these errors provided all the error analysis and recovery procedures result in a retry indication.

In several cases, lost errors are tolerated. The strong tendency to prefetch data exhibited by the NVAX+ pipeline makes the particular lost error likely, given that one error of that kind occurred. Also, in each case, if data

is lost in the lost error, a hard error interrupt is posted. So these errors are tolerated as long as they do not cause a hard error interrupt. BIU_STAT<WRITE_LOST> is maintained to report unrecorded errors on write operations. If BIU_STAT<WRITE_LOST> is set, the H_ERR interrupt is asserted.

Errors in opcode or operand specifier fetching are always detected before architecturally visible state within the CPU is modified. This means the VR bit from the machine check stack frame should be one. This error handling analysis attempts to recover from multiple errors, so the retry condition for each error is made as general as possible. If the machine check handler finds only errors of the kind listed here, then VR should be one and it is an inconsistent report if it is not.

- VIC parity errors
- B- data uncorrectable ECC FILL errors in I- stream reads
- CACK_H error in I- stream reads

VIC Parity Errors

Description: A parity error was detected in the VIC tag or data store in the Ibox.

- VIC data parity error
  A parity error occurred in the data portion of the VIC.
- VIC tag parity error
  A parity error occurred in the tag portion of the VIC.

In all cases, the quadword virtual address of the error is in VMAR.

NOTE: ICSR<LOCK> must be set for ICSR<DPERR> or ICSR<TPERR> to be valid. The Lock bit also latched the information in the VMAR register. If this bit is clear, then no parity errors have occurred and the registers can be updated.

Pending interrupts: A soft error interrupt should be pending.

Recovery procedure: Disable and flush the VIC by rewriting all the tags. Clear ICSR<LOCK>.

Retry condition: Retry if:

(VR = 1) OR (PSL<FPD> = 1)

FILL Uncorrectable ECC Errors

Description: (uncorrectable ECC errors): An uncorrectable data error was detected by the Cbox in an I- stream or D- stream read fill. Fill uncorrectable data errors can result from reads of the B- cache, another CPU cache, memory, or LSB (double error). S_FILL_ADDR contains the address of the error, and S_FILL_SYNDROME contains the syndrome calculated by the ECC logic.

Pending interrupts: A soft or hard error interrupt should be pending.

Recovery procedure: The machine check handling software should defer recovery until the expected hard or soft interrupt occurs. If one of these interrupts is taken, the B- cache should be flushed and BIU_STAT<FILL_ECC> should be cleared.

Retry condition: If no writeback error occurs in the B- cache flush, retry if:

(VR = 1) OR (PSL<FPD)>= 1)

If a writeback error occurs in the B- cache flush, then the data is presumed to be unrecoverable. Given that the address is available (no error in the tag store), software should determine if the error is fatal to one process or the whole system and take appropriate action. Otherwise, crash the system.

Lost Fill Error

Description: Some fill errors were not latched because a previous fill error was reported in the BIU_STAT register. If the reported error is not a read, a fill error while merging write data from a write has been logged. The logged error is not the cause of the machine check, but FILL_SEO (BIU_STAT<14>) might be. H_ERR should be pending if the reported error was not correctable. If the reported error is a read or a correctable fill error and LOST_WRITE (BIU_STAT<20>) is not set, the error causing FILL_SEO to set may be the cause of the machine check. The operation can be retried unless the aborted instruction has altered essential state. If S_PCSTS<PTE_ER> is set, refer to PTE errors, discussed below in this section.

Lost fill errors may be caused by more than one operand prefetch to the same cache block.

Recovery from lost fill errors depends on whether the pending interrupt is hard or soft. The machine check error handling software should defer recovery until the expected hard or soft error interrupt occurs. Once the interrupt is taken, the error recovery and restart instructions found in the hard error interrupt and soft error interrupt sections should be referenced (Sections 9.5 and 9.6, respectively).

Software should employ some mechanism to record that an interrupt for a lost fill error is pending. This mechanism should allow detection of a case in which an expected interrupt does not occur (once the IPL is lowered). If the expected interrupt does not occur when the IPL is lowered, then a serious inconsistency exists and the system should be crashed.

Pending interrupts: A hard or soft error interrupt, possibly both, should be pending.

Recovery procedure: No specific recovery action is required. BIU_STAT<FILL_SEO> should be cleared by the hard or soft error interrupt handler.

Retry condition: Retry only if:

(VR = 1) OR (PSL<FPD> = 1)

BIU_HERR

Description: An I- stream or D- stream read returned CACK_H error or did not complete because of a tag (BIU_STAT<TPERR>=1) or tag control parity error (BIU_STAT<TCPERR>=1) .

I- stream errors cause a machine check when the Ebox microcode requests dispatch to a new instruction execution microflow or attempts to access an operand within an instruction execution microflow.

D- stream errors cause a machine check when the Ebox microcode accesses prefetched operand data or when the Mbox returns data tagged with an error indication to the Ebox register file.

D- stream ownership read errors cause a machine check when the Ebox microcode accesses prefetched operand data. If the address is in I/O space, the status is inconsistent.

Pending interrupts: A soft error interrupt should be pending.

Recovery procedure: Clear BIU_STAT<BIU_HERR>.

Retry condition: Retry if:

(VR = 1) OR (PSL<FPD> = 1)

Lost BIU Error

Description: Some fill errors were not latched because a previous BIU error was reported in the BIU_STAT register. If the reported error is not a read, a fill error while merging write data from a write has been logged. The logged error is not the cause of the machine check, but BIU_SEO (BIU_STAT<7>) might be. H_ERR should be pending. If the reported error is a read or a correctable fill error and LOST_WRITE (BIU_STAT<20>) is not set, the error causing BIU_SEO to set may be the cause of the machine check, and the operation can be retried unless the aborted instruction has altered essential state. If S_PCSTS<PTE_ER> is set, refer to PTE errors, discussed below in this section.

Lost BIU errors may be caused by more than one operand prefetch to the same cache block.

Recovery from lost BIU errors depends on whether the pending interrupt is hard or soft. The machine check error handling software should defer recovery until the expected hard or soft error interrupt occurs. Once the interrupt is taken, the error recovery and restart instructions found in the hard error interrupt and soft error interrupt sections should be referenced (Sections 9.5 and 9.6, respectively).

Software should employ some mechanism to record that an interrupt for a lost fill error is pending. This mechanism should allow detection of a case in which an expected interrupt does not occur (once the IPL is lowered). If the expected interrupt does not occur when the IPL is lowered, then a serious inconsistency exists and the system should be crashed.

Pending interrupts: A hard or soft error interrupt, possibly both, should be pending.

Recovery procedure: No specific recovery action is required. BIU_STAT<FILL_SEO> should be cleared by the hard or soft error interrupt handler.

Retry condition: Retry only if:

(VR = 1) OR (PSL<FPD> = 1)

PTE Read Errors

PTE read errors happen in reads issued by the Mbox in handling a TB miss. Handling of these errors differs from handling the same underlying error (BIU_HERR, BC_TPERR, BC_TCPERR, FILL_ECC) when PTE read is not the cause.

If S_PCSTS<PTE_ER> is set, then a PTE read issued by the Mbox in processing a TB miss had an unrecoverable error. The TB miss sequence was aborted because of the error. The original reference can be any I- stream or D- stream read or write. If the original reference was issued by the Ebox, then the PTE read that incurred the error will have been retried once because of a special hardware/microcode mechanism for handling PTE read errors on Ebox references.

PTE read errors are difficult to analyze, partly because the read error report in the Cbox does not directly indicate that the failing read was a PTE read. Because of this and because PTE read errors should be rare (a very small percentage of the reads issued by the Mbox are PTE reads), multiple errors that interfere with the analysis of the PTE error are not considered recoverable.

The mechanism for reporting PTE read errors on Ebox references involves the Mbox forcing the Ebox into the microcode routine which normally handles memory management faults. This routine probes the address of the original reference, effectively retrying the failing PTE read. Assuming the error is not transient, the probe by microcode will cause a machine check. If the error does not occur on the probe, microcode restarts the current instruction stream. So machine checks caused by PTE read errors can easily occur with the particular PTE read error having occurred twice (with a lost error bit set in the relevant Cbox error register).

The analysis here tolerates these particular multiple error reports and allows retry in those cases, provided the remainder of the error analysis indicates retry is appropriate.

If the reference that incurs the PTE read error is a write, S_PCSTS<PTE_ER_WR> will be set. In this case the original write is lost. No retry is possible partly because the instruction that took the machine check may be subsequent to the one that issued the failing write. Also, PTE read errors on write transactions can cause a machine check at an arbitrary time in a microcode flow, and core machine state may not be consistent.

PTE Read Errors in Interruptible Instructions

Another special case associated with PTE read errors exists for interruptible instructions (specifically CMPC3, CMPC5, LOCC, MOVC3, MOVC5, SCANC, SKPC, and SPANC). For these instructions, if the PTE read error occurred for an Ebox reference, the PC in the machine check stack frame points to the instruction following the interrupted instruction. In this case, the SAVEPC element in the machine check stack frame is the PC of the interrupted instruction.

However, in all other cases SAVEPC is UNPREDICTABLE. This case is not considered recoverable because analysis of the error information cannot unambiguously conclude that this case is present. To see if this case is present, the error handler examines PSL<FPD> in the machine check stack frame. If FPD is set in the stack frame in the case of a PTE read error, then one of the following is true:

- One of the interruptible instructions listed above incurred the PTE read error. In this case, SAVEPC from the machine check stack frame points to the interrupted instruction, and the PC in the stack frame points to the next instruction.

- An REI instruction loaded a PSL with FPD set and a certain PC. The Ibox incurred the PTE read error in fetching the opcode pointed to by that PC. In this case, the PC in the stack frame points to the instruction that was the target of the REI, and SAVEPC from the stack frame is UNPREDICTABLE.

It is not possible to determine which of the two above cases is the cause of a machine check with S_PCSTS<PTE_ER> set and stack frame PSL<FPD> set. Retry is not possible since software cannot tell which PC to restart with.

Uncorrectable ECC Fill Errors and PTE Errors on Reads

Description (uncorrectable ECC errors): A fill uncorrectable data error was detected by the Cbox in a PTE read. Uncorrectable data errors are the result of a multiple- bit error in the data read from the B- cache on a fill from the system during a READ_BLOCK or LDxL.

Description (all cases): S_FILL_ADDR contains the cache address of the error, and FILL_SYNDROME contains the syndrome calculated by the ECC logic. If the address is in I/O space, this is an inconsistent status.

S_BIU_STAT<FILL_SEO> may be set. This error is probably caused by the same PTE error occurring more than once. This is an acceptable assumption unless a hard error interrupt occurs after handling this error.

Pending interrupts: A soft error interrupt should be pending.

Recovery procedure (uncorrectable ECC errors): Clear BIU_STAT<FILL_ECC>.

Recovery procedure (both cases): Flush the B- cache. Clear PCSTS<PTE_ER>.

Retry condition: If no writeback error occurs in the B- cache flush, retry if:

(VR = 1) AND (PSL<FPD> = 0) AND
(S_PCSTS<PTE_ER_WR> = 0)

Crash the system if:

(PSL<FPD> = 1) OR (S_PCSTS<PTE_ER_WR> = 1)

If a writeback error occurs in the B- cache flush, then the data is presumed to be unrecoverable. Software must determine if the error is fatal to one process or the whole system and take appropriate action.

CACK_H Error on PTE Read

Description: A PTE read returned CACK_H error. S_BIU_STAT<BIU_SEO> may be set. This error is probably caused by the same PTE error occurring more than once. This is an acceptable assumption unless a hard error interrupt occurs after handling the error.

Pending interrupt: A soft error interrupt should be pending.

Recovery procedure: Clear BIU_STAT<BIU_ HERR> and PCSTS<PTE_ER>.

Retry condition: Retry if:

(VR = 1) AND (PSL<FPD> = 0) AND
(S_PCSTS<PTE_ER_WR> = 0)

Otherwise, crash the system.

Post retry recovery: If the same fill error occurs on retry, then the block is probably "lost." In this case the more general sense of "lost" is implied. Software must determine if the error is fatal to one process or the whole system and take appropriate action.

NOTE: In a block "lost" case, it would be appropriate to first cause each CPU in the system to flush its B- cache, then retry once more.

9.4.7 Inconsistent Status in Machine Checks

Description: A presumed impossible error report was found in the error registers. This could be due to a hardware failure.

Pending interrupts: Either a hard or soft error interrupt, or both should be pending.

Recovery procedure: No specific recovery action is called for.

Retry condition: No retry is possible. Crash the system.

9.5 Hard Error Interrupts

A hard error interrupt reports an error that was detected asynchronous with respect to the instruction execution.

A hard error interrupt results in an interrupt at IPL 1D (hex) being dispatched through SCB vector 60 (hex). Typically, these errors indicate that machine state has been corrupted and that a retry is not possible. The stack frame for a hard error interrupt consists of two parameters, PC and PSL.

Figure 9- 5 contains the hard error interrupt parse tree, which isolates errors at the processor level and module level. The sections following the parse tree provide a description of the hard error, the procedure to recover, and the conditions for restarting the operation.

Figure 9- 5 Hard Error Interrupt Parse Tree

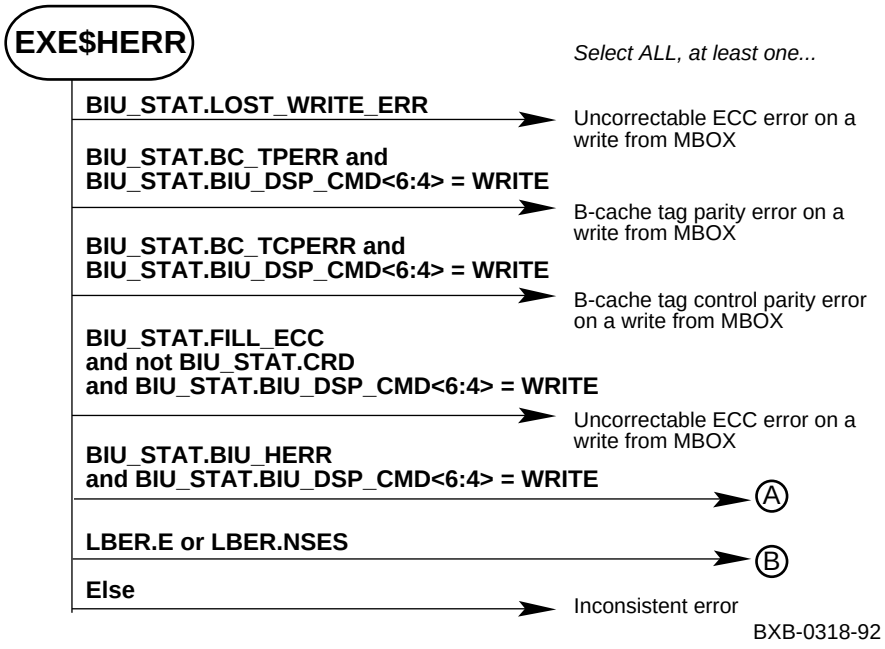


Figure 9- 6 Hard Error Interrupt Parse Tree (Continued)

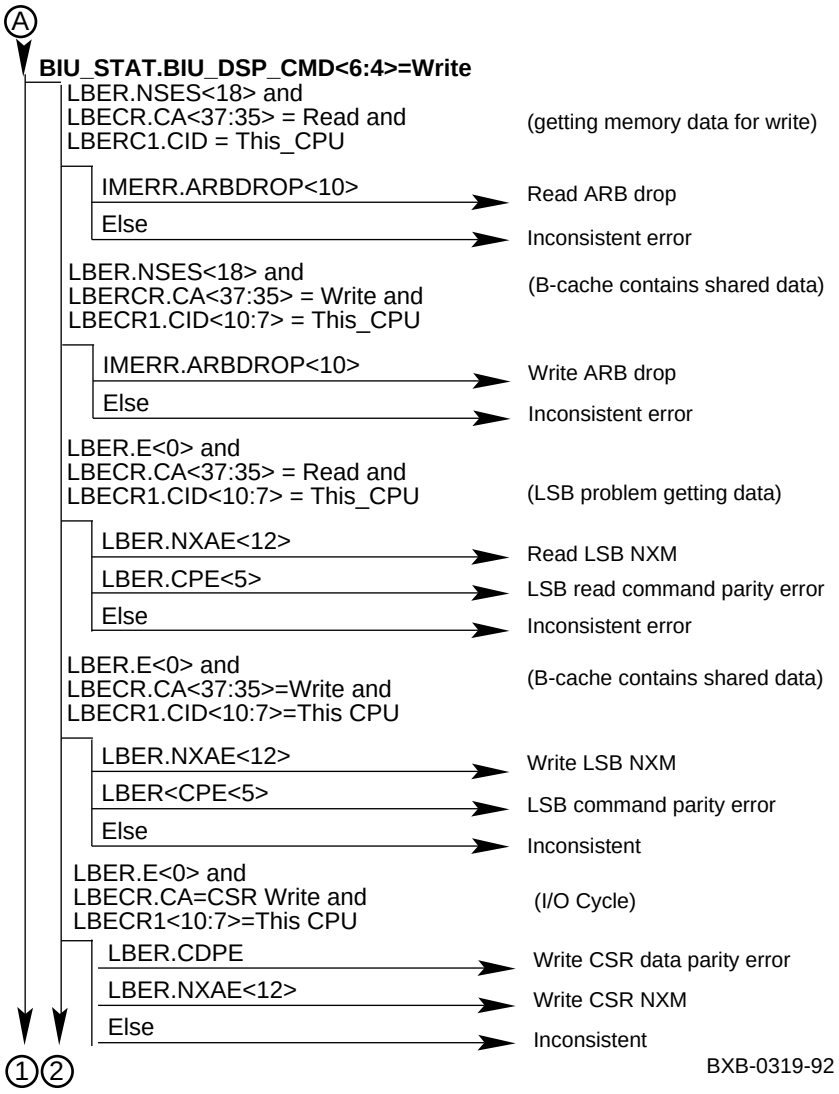


Figure 9- 6 Hard Error Interrupt Parse Tree (Continued)

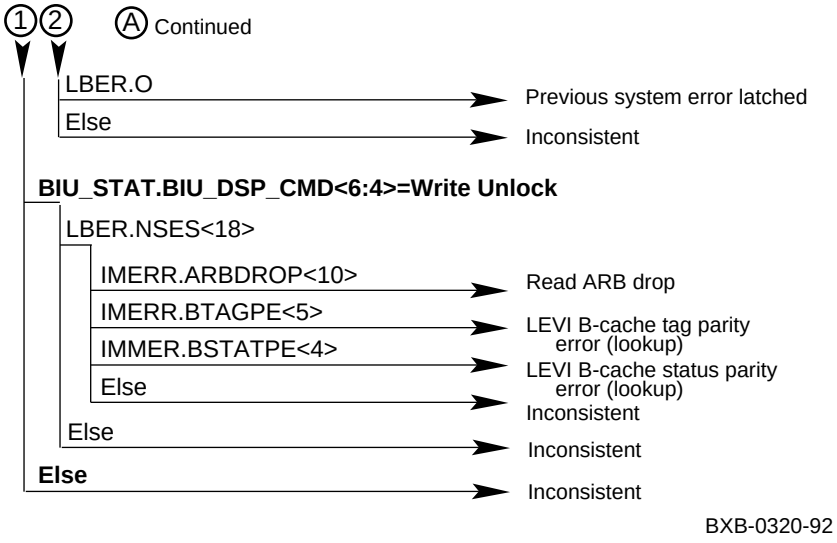


Figure 9- 6 Hard Error Interrupt Parse Tree (Continued)

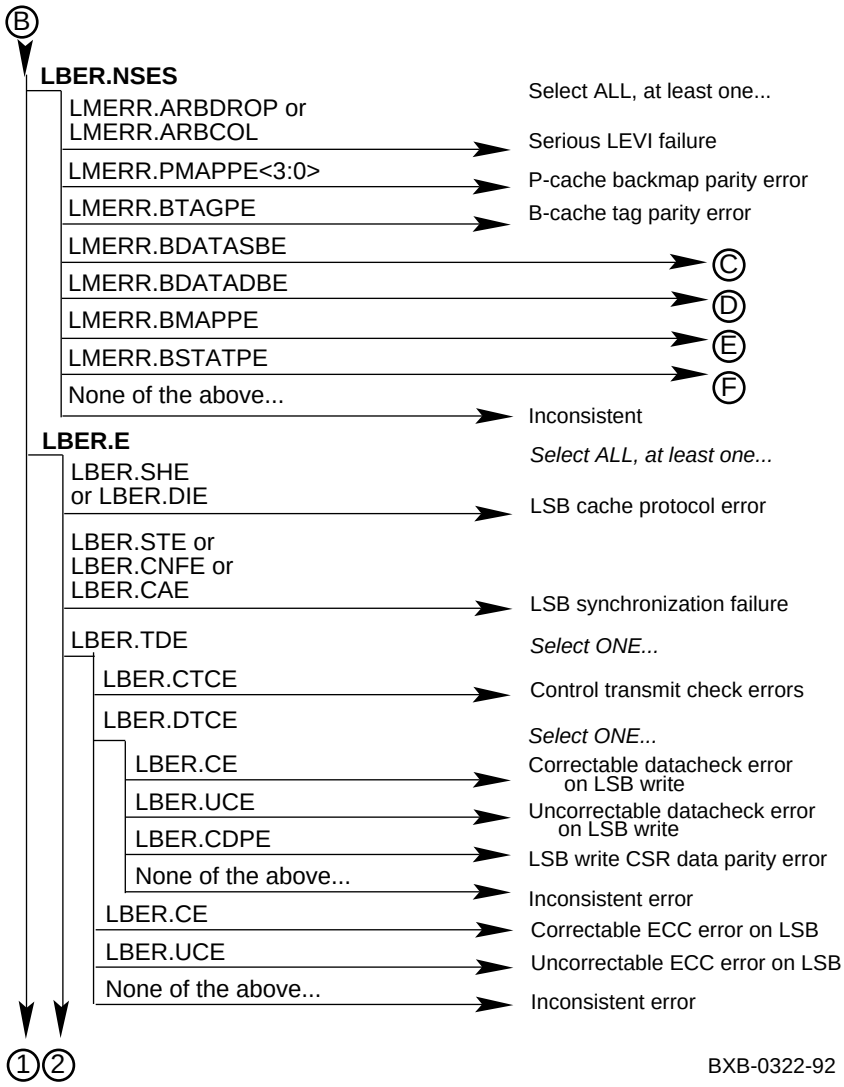


Figure 9- 6 Hard Error Interrupt Parse Tree (Continued)

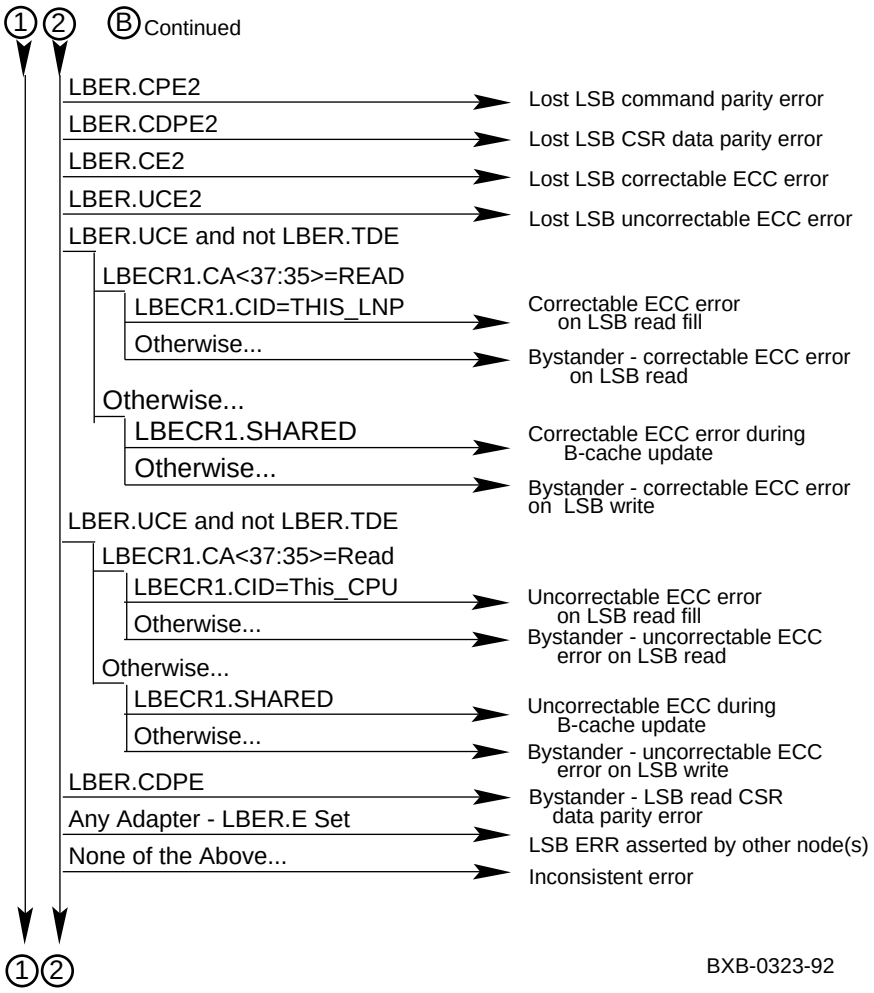


Figure 9- 6 Hard Error Interrupt Parse Tree (Continued)

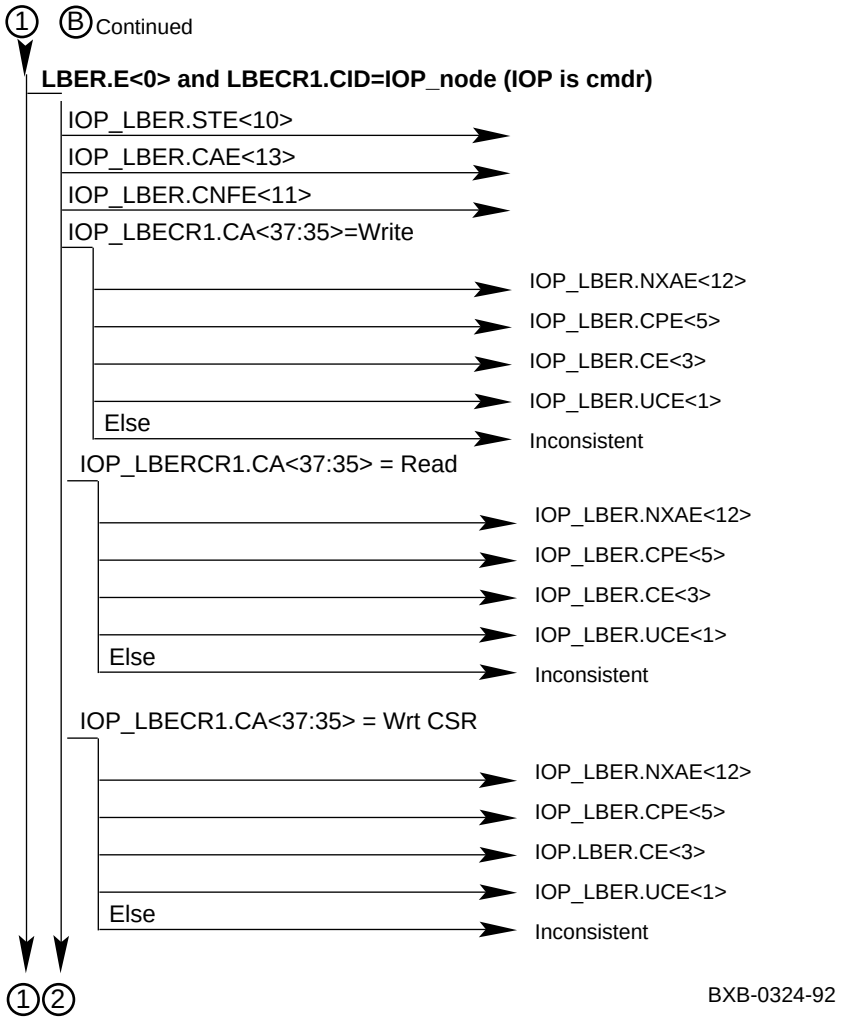


Figure 9- 6 Hard Error Interrupt Parse Tree (Continued)

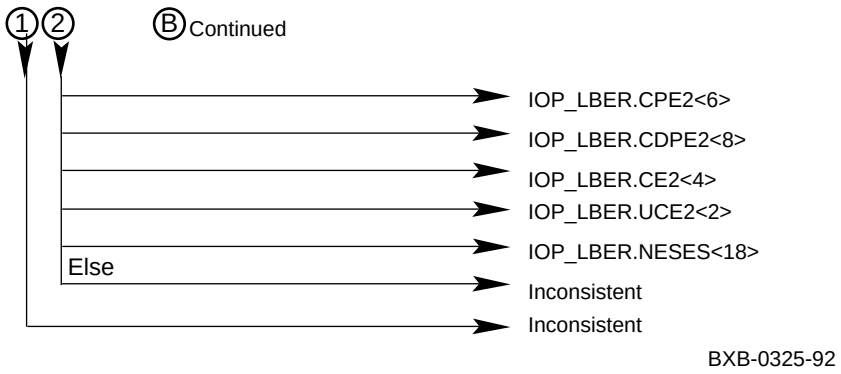
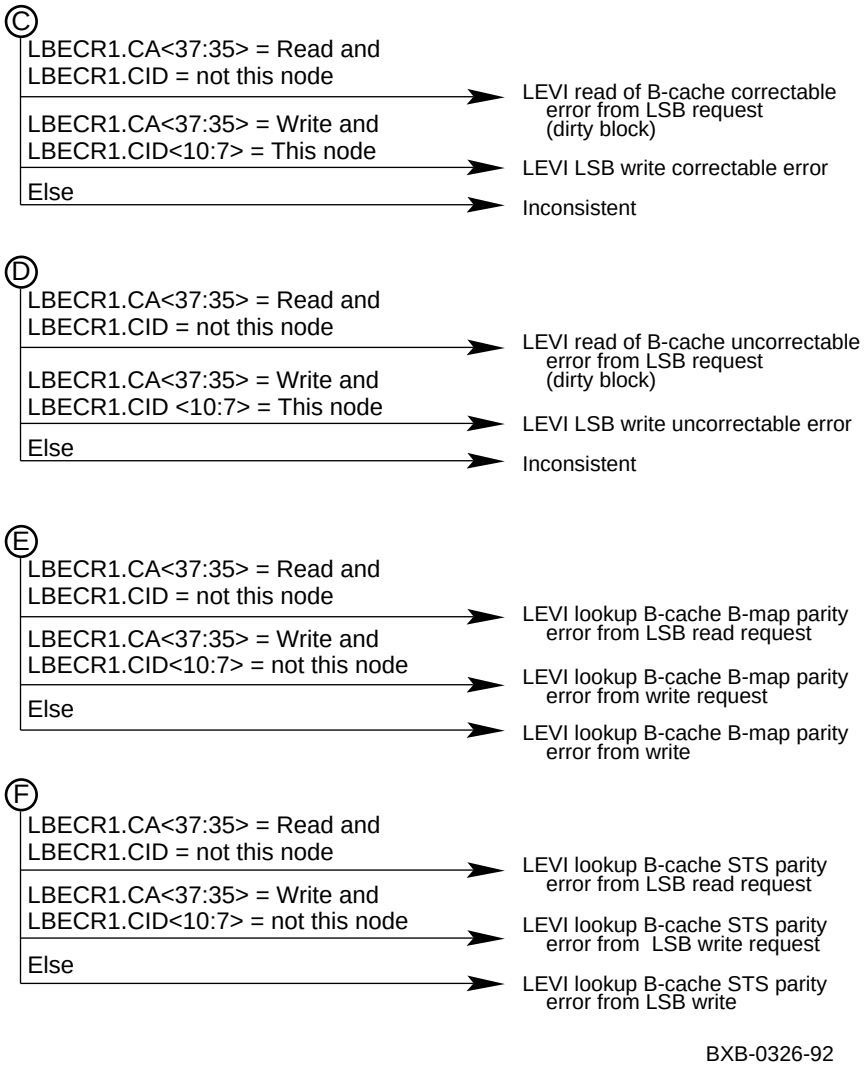


Figure 9- 6 Hard Error Interrupt Parse Tree (Continued)



Uncorrectable Errors During Write or Write Unlock Processing

Description: In processing a write or write unlock, the Cbox detected a CACK_H error from the system, a tag parity error, a control parity error, or an uncorrectable ECC error on the data read, which is to be merged data from the write, is lost.

Uncorrectable ECC errors indicate that two or more bits of the stored data quadword have changed and the error correcting code cannot correct the data. The write merge sequence is aborted.

Recovery procedure: The data in this block is lost.

Restart condition: If the address of the data is available and no unexpected writeback errors occurred during the B- cache flush, software must determine if the lost data is fatal to one process or the whole system, and take appropriate action.

System Environment Hard Error Interrupts

See Section 9.8.

Inconsistent Status in Hard Error Interrupt Cause Analysis

Description: A presumed impossible error report was found in the error registers. This could be due to a hardware failure or bug.

Recovery procedure: No specific error action is called for.

Restart condition: No retry is possible. Crash the system.

9.6 Soft Error Interrupts

A soft error interrupt is requested to report an error that was detected but which did not affect instruction execution. This results in an interrupt at IPL 1A (hex) to be dispatched through SCB vector 54 (hex). The stack frame for a soft error interrupt consists of two parameters, PC and PSL.

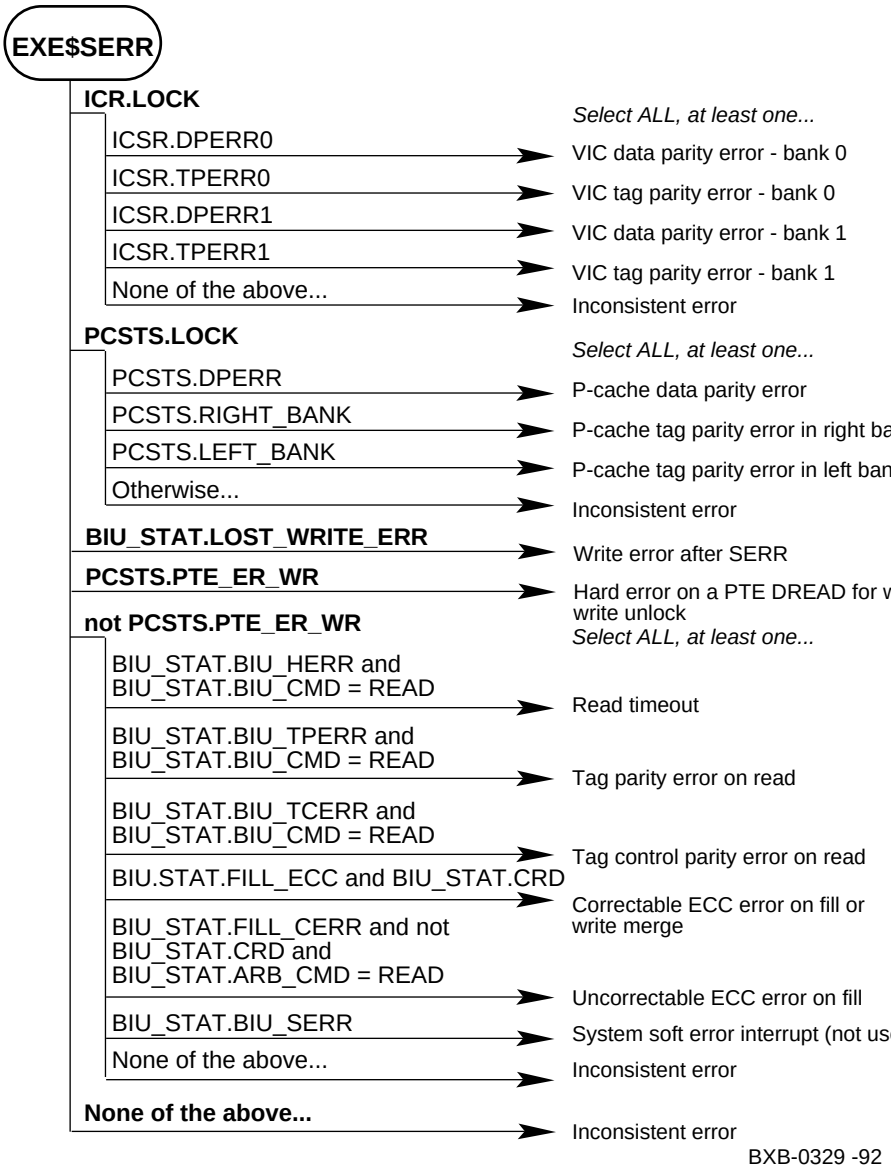
Many errors that cause a soft error interrupt may also lead to a machine check exception. For this reason, a soft error interrupt with no apparent cause is not an inconsistent state unless the CPU has executed an instruction while IPL was lower that 1A (hex) since the most recent machine check exception.

When a soft error interrupt is the only notification for a memory read error which could cause a machine check, the error did not cause a machine check for one of the following reasons:

- The error did not occur on the quadword the Ebox or the Ibox requested. (P- cache fill error).
- The Ebox took an interrupt before accessing an instruction or operand which was prefetched by the Ibox. (It could be this software interrupt.)
- A prefetched instruction or operand belonged to an instruction following the mispredicted branch, so the Ebox never executed the instruction and it was flushed from the pipeline when the branch mispredict was recognized.
- The Ebox took an exception for a different reason before attempting to use an instruction execution dispatch or access operand prefetched by the Ibox. (The pipeline was flushed because of the exception.)

Figure 9- 6 contains the soft error interrupt parse tree. The sections following the parse tree provide a description of the soft error, the procedure to recover, and the conditions for restarting the operation.

Figure 9- 6 Soft Error Interrupt Parse Tree



VIC Parity Errors

Description: A parity error was detected in the VIC tag or data store in the Ibox. The data parity error may occur in bank 0 (DPERR0) or bank 1 (DPERR1) of the VIC. The tag parity error may occur in bank 0 (TPERR0) or bank 1 (TPERR1) of the VIC. The quadword virtual address of the error is in S_VMAR.

Recovery procedure: Disable and flush the VIC by rewriting all the tags and clear ICSR<LOCK>.

P- Cache Parity Errors

Description: A parity error was detected in the P- cache. Either a tag parity error or a data parity error is reported, although tag parity errors in both the left and right banks may be reported simultaneously. The reference, whether it was a read or write, was passed to the Cbox as if the P-cache had missed. No data is lost. The P- cache is disabled because PCSTS<LOCK> is set.

S_PCADR contains the physical address of the operation incurring the error. The address should not be in I/O space. If it is, it is an inconsistent status.

Recovery procedure: Clear PCSTS<LOCK>. Flush the P- cache and initialize the P- cache tag store.

FILL Uncorrectable ECC Errors on I- Stream or D- Stream Reads

Description: An uncorrectable ECC error was detected by the Cbox in an I-stream or D- stream read. Uncorrectable data errors are the result of multiple bit errors in the data read. S_FILL_ADDRESS contains the address of the error, and S_FILL_SYNDROME contains the syndrome calculated by the ECC logic. If the physical address is found to be in I/O space, it is an inconsistent status.

Recovery procedure: Clear BIU_STAT<FILL_ECC>. Flush the B- cache. (BC_TAG can be used to determine if the fill is from the B- cache.) If the data is dirty in the B- cache and the error repeats itself (it is not transient), then a writeback error will result from the flush procedure.

Restart conditions: If a writeback error occurs in the B- cache flush, then the data is presumed to be unrecoverable. Software must determine if the error is fatal to one process or the whole system and take appropriate action.

If the address of the error in the flush is not the same as that of the original error, this is a multiple error case in the data RAMs and is a serious failure. Crash the system.

Multiple Errors Associated with PTE Errors

PTE errors are difficult to analyze, partly because the read error report in the Cbox does not directly indicate the failing read was a PTE read. Because of this and because PTE read errors should be rare (a very small percentage of the reads issued by the Mbox are PTE reads), multiple errors that interfere with the analysis of the PTE error are considered recoverable.

If the reference that incurs the PTE read error is a write, S_PCSTS<PTE_ER_WR> will be set. In this case the original write is lost. No retry is possible partly because the instruction that took the machine check may be subsequent to the one that issued the failing write. Also, PTE read errors on write transactions can cause a machine check at an arbitrary time in a microcode flow, and core machine state may not be consistent.

Restart condition: If no writeback error occurs in the B- cache flush, restart if:

S_PCSTS<PTE_ER_WR> = 0

If:

S_PCSTS<PTE_ER_WR> = 1

crash the system.

If a writeback error occurs in the B- cache flush, then the data is presumed to be unrecoverable (software must determine if the error is fatal to one process or the whole system and take appropriate action). Clear PCSTS<PTE_ER_WR>.

Restart condition: Restart if:

S_PCSTS<PTE_ER_WR> = 0

Otherwise, crash the system.

Multiple Errors That Interfere with Analysis of PTE Read Errors

Because analysis of PTE read errors leads to several unusual causes, restart is not recommended in the event that other errors cloud the analysis of the PTE error.

Pending interrupts: A hard or software interrupt should be pending, or possibly both.

Recovery procedure: No specific recovery action is called for.

Restart condition: Not possible. Crash the system.

9.7 Kernel Stack Not Valid Exception

A kernel stack not valid exception occurs when a memory management exception is detected while attempting to push information on the kernel stack during microcode processing of another exception. Note that a console halt with an error code of ERR_INTSTK is taken if a memory management exception is encountered while attempting to push data on the interrupt stack. The kernel stack not valid exception is dispatched through SCB vector 08 (hex). The kernel stack not valid stack frame consists of two parameters, PC and PSL.

9.8 System Environment Errors

System environment errors are detected by the LEVI gate arrays, which signal them to the NVAX+ processor by two mechanisms corresponding to the two sets of pins in the processor used for error reporting: CACK_H signals and IRQ_H signals.

Parse trees given in Figures 9- 4 and 9- 5 isolate errors to the system environment as well as to the processor.

9.8.1 Error Categories

Module and system level errors can be divided into two groups: synchronous errors and asynchronous errors (not to be confused with NVAX+ synchronous and asynchronous errors, which are internal to the processor).

9.8.1.1 Synchronous Errors

Synchronous errors are those detected on the CPU module or in the LSB system while the processor was waiting (idle) for service from LEVI. These errors imply that something drastic went wrong while LEVI was trying to complete the processor's request, and that LEVI is unable to complete the processor's transaction. Specifically, only the following errors induce synchronous errors if the processor is waiting for an acknowledgment from LEVI:

- B- cache tag parity error
- B- cache status parity error
- LSB arbitration drop
- LSB command parity error
- LSB nonexistent address error

Other errors detected while the processor is waiting for an acknowledgment are signaled as asynchronous errors because they do not prevent completion of the processor's request.

LEVI reports synchronous errors to the processor by returning H_ERR status instead of the normal acknowledgment. (The CACK_H signals are driven with H_ERR instead of a normal cycle acknowledge. This in turn sets BIU_STAT<HERR>.)

9.8.1.2 Asynchronous Errors

Asynchronous errors include all errors detected on the CPU module or in the LSB system while the processor was not waiting for service from LEVI. Module and LSB errors not included in the synchronous error list above are categorized as asynchronous errors even if the processor is waiting for LEVI. The distinction is that only certain errors prevent LEVI from completing the processor's request.

LEVI reports asynchronous errors to the processor through the interrupt signals (IRQ_H). LEVI asserts a module signal (LEVI_ERROR) which is dispatched to the appropriate IRQ_H pin on the processor.

9.8.2 Environment Error Sources

The environment errors originate either from the LEVI operations within the CPU module or from interactions of the LEVI with other nodes through the LSB bus.

9.8.2.1 LEVI Errors

LEVI B- Cache Status Parity Error

Description: LEVI detected a B- cache status parity error while examining the status bits on behalf of the processor or the LSB. If LEVI detects the error while evaluating status in response to a processor request, it returns an H_ERR acknowledgment.

If LEVI detects the error while evaluating status to determine a response to an LSB request, it signals the error to the processor even if the processor is waiting for an acknowledgment.

Processor response: If H_ERR is returned in response to a read request, NVAX+ takes a machine check at IPL 31 through SCB offset 04 (hex). If H_ERR is returned in response to a write, or IRQ_H<4> is asserted, NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). In either case, LEVI asserts ERROR on the LSB.

State captured: LMERR<BSTATPE> and LBER<NSES> are set. Address and LSB command are captured in the LBECR register.

Recovery: None possible. Crash the system. The CPU module responds by asserting ERROR on the LSB, while SHARED and DIRTY remain unasserted.

LEVI B- Cache Tag Parity Error

Description: LEVI detected a B- cache tag parity error while examining the tag bits on behalf of the processor. (Note that LEVI never examines the B-tag RAMs on behalf of the LSB. It uses the B- map to perform this function.) It returns H_ERR acknowledgment.

Processor response: If H_ERR is returned in response to a read request, NVAX+ takes a machine check through SCB offset 04 (hex). If H_ERR is returned in response to a write, NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). In either case, LEVI asserts ERROR on the LSB.

State captured: LMERR<BTAGPE> and LBER<NSES> are set. The address and LSB command are captured in the LBECR register.

Recovery: None possible.

LEVI P- Map Parity Error

Description: LEVI detected a parity error while examining its internal P-map to determine how to respond (invalidate or update) to an LSB write. (Note that this is only done if the write address hits in the B- map.) It notifies the processor by asserting the IRQ_H <5> signal.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LMERR<PMAPPEn> and LBER<NSES> are set. The address and LSB command are captured in the LBECR register.

Recovery: Flush the relevant internal processor cache and invalidate the index in the map RAM structure. LEVI responds by asserting DIRTY from the results of the B- map only and assumes a P- map hit (asserts SHARED).

LEVI B- Map Parity Error

Description: LEVI detected a parity error while examining the B- map to check for an LSB hit in this cache. LEVI asserts the IRQ_H<5> signal to notify the processor.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LMERR<BMAPPE> and LBER<NSES> are set. The address and LSB command are captured in the LBECR register.

Recovery: No recovery is possible since this error is detected in the LSB time domain in response to an LSB command. Crash the system. The CPU module responds by asserting ERROR on the LSB, while SHARED and DIRTY remain unasserted.

9.8.2.2 LSB Errors

LSB Arbitration Drop

Description: LEVI signals an arbitration drop error when a node, having requested and been granted the LSB, fails to initiate a command/address cycle (by asserting CA on the LSB). If LEVI is the commander that failed to initiate a command in response to a processor request, it acknowledges the processor with a H_ERR status. If LEVI caused the error while the processor was not waiting for a response (while writing a victim for example) or if any other node caused the error, LEVI notifies the processor by asserting the IRQ_H<4> signal.

Processor response: If H_ERR is returned in response to a read request, NVAX+ takes a machine check through SCB offset 04 (hex). If H_ERR is returned in response to a write, or IRQ_H<4> is asserted, NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). In either case, LEVI asserts ERROR on the LSB.

State captured: LMERR<ARBDROP> and LBER<NSES> are set. The address and LSB command are captured in the LBECR register.

Recovery: All nodes resynchronize the arbitration when ERROR is asserted on the LSB. Crash the system.

LSB Arbitration Collision

Description: LEVI signals an arbitration collision error by asserting the IRQ_H<5> signal when a node tries to request the LSB (that is, drives the REQ lines) during an inappropriate cycle.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LMERR<ARBCOL> and LBER<NSES> are set.

Recovery: All nodes resynchronize the arbitration when ERROR is asserted on the LSB. Crash the system.

LSB Transmit Check Error

Description: LEVI signals a transmit check error when it detects a discrepancy between the command (or data) it drives onto the LSB and the command (or data) it receives from the LSB. It notifies the processor by asserting the IRQ_H<5> signal.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LBER<CTCE> or LBER<DTCE> is set.

Recovery: All nodes will resynchronize the arbitration when ERROR is asserted on the LSB. Software should probably crash.

LSB Dirty Error

Description: LEVI signals a Dirty error when LSB DIRTY is asserted during an inappropriate cycle. It notifies the processor by asserting the IRQ_H<5> signal.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LBER<DIE> is set.

Recovery: LSB ERROR is asserted and all nodes resynchronize the arbitration. Crash the system.

LSB Shared Error

Description: LEVI signals a Shared error when LSB SHARED is asserted during an inappropriate cycle. It notifies the processor by asserting the IRQ_H<5> signal.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LBER<SHE> is set.

Recovery: LSB ERROR is asserted and all nodes resynchronize the arbitration. Crash the system.

LSB CA Error

Description: LEVI signals a CA error when LSB CA is asserted during an inappropriate cycle. It notifies the processor by asserting the IRQ_H<5> signal.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LBER<CAE> is set.

Recovery: LSB ERROR is asserted and all nodes will resynchronize the arbitration. Software should probably crash.

LSB Nonexistent Address Error

Description: LEVI signals a nonexistent address error when there is no CNF response to a command/address cycle on the LSB. If the missing CNF was needed to complete a processor request, LEVI returns an H_ERR acknowledgment. If the error occurred while the processor was not waiting for a response or if the error was in response to a request from another node, LEVI signals the processor by asserting the IRQ_H<4> signal.

Processor response: If H_ERR is returned in response to a read request, NVAX+ takes a machine check through SCB offset 04 (hex). If H_ERR is returned in response to a write, or IRQ_H<4> is asserted, NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). In either case, LEVI asserts ERROR on the LSB.

State captured: LBER<NXAE> is set. The address and LSB command are captured in the LBECR register. Note that although there is no CNF signal on the LSB, the transaction continues and data parity or ECC errors will likely follow when the requesting node reads an undriven LSB during the corresponding data cycles. Any of the following LBER bits may be set along with NXAE: UCE, UCE2, CE, CE2, CDPE, CDPE2.

Recovery: LSB ERROR is asserted and all nodes resynchronize the arbitration. Software should probably crash if this was not expected (like not during INIADAPT).

LSB CNF Error

Description: LEVI signals a CNF error when CNF is asserted during an inappropriate cycle. It notifies the processor by asserting the IRQ_H<5> signal.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LBER<CNFE> is set.

Recovery: LSB ERR is asserted and all nodes will resynchronize the arbitration. Software should probably crash.

LSB Stall Error

Description: LEVI signals a Stall error when LSB STALL is asserted during an inappropriate cycle. It notifies the processor by asserting the IRQ_H<5> signal.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LBER<STE> is set.

Recovery: LSB ERR is asserted and all nodes will resynchronize the arbitration. Software should probably crash.

LSB Transmitter During Error

Description: This bit is set in the LBER of the node driving the LSB when an error is detected. Only one node should have this bit set for any given error. This bit is set only in response to LSB ERROR; it does not cause LSB ERROR to be asserted. LEVI notifies the processor when it detects the original error.

Processor response: None.

State captured: LBER<TDE> is set.

Recovery: All nodes resynchronize the arbitration once ERROR is asserted. Software should probably crash.

LSB CSR Data Parity Error

Description: LEVI detected a parity error on the LSB during a CSR data cycle. It notifies the processor by asserting the IRQ_H<5> signal regardless of whether the processor is waiting for this data.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LBER<CDPE> or LBER<CDPE2> is set. Address and LSB command are captured in the LBECR register (only on the first CDPE; not on CDPE2).

Recovery: LSB ERROR is asserted and all nodes resynchronize the arbitration.

LSB Command Parity Error

Description: LEVI detected bad parity on an LSB command cycle. If the command was issued to satisfy a processor request LEVI returns an H_ERR acknowledgment. If LEVI caused the error while the processor was not waiting for a response (while writing a victim for example) or if any other node caused the error, LEVI notifies the processor by asserting one of the IRQ_H signals.

Processor response: If H_ERR is returned in response to a read request, NVAX+ takes a machine check through SCB offset 04 (hex). If H_ERR is returned in response to a write, or IRQ_H<4> is asserted, NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). In either case, LEVI asserts ERROR on the LSB.

State captured: LBER<CPE> or LBER<CPE2> is set. The address and LSB command are captured in the LBECR register (only on the first command parity error; not on CPE2).

Recovery: LSB ERROR is asserted and all nodes resynchronize the arbitration.

LSB Correctable ECC Error

Description: LEVI detected a correctable data error on the LSB. It notifies the processor by asserting one of the IRQ_H signals only if LCNR<CEEN> is set.

Processor response: NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: If LCNR<CEEN> is clear, nothing is captured. Otherwise, LBER<CE> or CE2 is set, the LSB command and address are captured in the LBECR register, and the ECC syndromes are captured in the LBESR register (only on the first correctable ECC error; not on CE2). If the data was driven by LEVI onto the LSB, LMERR<BDATASBE> is also set.

Recovery: If LCNR<CEEN> is set, LSB ERROR is asserted and all nodes will resynchronize the arbitration. The eventual consumer of this data (if any) will also report a single- bit error and correct it. The LEVI gate arrays will not correct data; only set- aside checking is performed for fault isolation purposes.

LSB Uncorrectable ECC Error

Description: LEVI detected an uncorrectable data error on the LSB. It notifies the processor by asserting the IRQ_H<5> signal.

*Processor respo*nse: If the IRQ_H<5> pin is asserted, NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LBER<UCE> or UCE2 is set. The address and LSB command are captured in the LBECR register and ECC syndromes are captured in LBESR (only on the first Uncorrectable ECC error, not on UCE2). If the data was driven by LEVI onto the LSB, LMERR<BDATADBE> is also set.

Recovery: LSB ERROR is asserted and all nodes resynchronize the arbitration. The eventual consumer of this data (if any) also reports a double-bit error and takes appropriate action. LEVI does not correct data; only set- aside checking is performed for fault isolation purposes.

LSB Error Line Asserted

Description: Some node on the LSB asserted ERROR. (All nodes set the LBER<ERR> bit whenever LSB ERROR is asserted.) LEVI notifies the processor through the IRQ_H<5> signal.

Processor response: If the IRQ_H<5> signal is asserted, NVAX+ takes a hard error interrupt at IPL 29 through SCB offset 60 (hex). LEVI asserts ERROR on the LSB.

State captured: LBER<E> is set.

Recovery: LSB ERROR is asserted and all nodes resynchronize the arbitration. Some other node on the LSB has noticed an error condition that was not noticed by this node. Software should query the LBER registers of other nodes and take appropriate action.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X