VAX 6200 System Technical User's Guide

Order Number: EK-620AA-TM-001

This manual serves as a reference on how to write software to this machine and covers the information needed to do field-level repair or programming customized to the CPU. It includes information on interrupts, error handling, and detailed theory of operation.

digital equipment corporation maynard, massachusetts

First Printing, May 1988

Contents

INDEX

FIGURES

TABLES

Preface

Intended Audience

This manual is written for the DIGITAL field service engineers installing and repairing in the field and for OEMs who are writing specialized applications, such as their own operating systems.

Document Structure

This manual has six chapters.

- **Chapter 1** gives you a basic introduction to the VAX 6200 system and its parts.
- **Chapter 2** tells you about the XMI bus and protocol.
- **Chapter 3** explains the KA62A CPU module.
- **Chapter 4** explains the MS62A memory module.
- **Chapter 5** tells you about the DWMBA and its DWMBA/A module and DWMBA/B module.
- **Chapter 6** explains the components of the power system and the cooling system.
- The **Index** provides additional reference support.

Associated Documents

Documents in the VAX 6200 documentation set include:

Title	Order Number
VAX 6200 Installation Guide	EK-620AA-IN
VAX 6200 Owner's Manual	EK-620AA-OM
VAX 6200 Mini-Reference	EK-620AA-HR
VAX 6200 Options and Maintenance	EK-620AA-MG
VAX 6200 System Technical User's Guide	EK-620AA-TM

Other documents that you may find useful include:

Title	Order Number
The VAX Architecture Reference Manual	EY-3459E-DP
VAX Hardware Handbook	EB-25949-46
VAX Software Handbook Set	EJ-28250-DP
VAXBI Options Handbook	EB-29228-46
TK50 Tape Drive Subsystem User's Guide	EK-OTK50-UG

1 The VAX 6200 System Overview

This chapter describes the system packages and system components and notes the location of components in the cabinet.

This chapter includes the following sections:

- VAX 6200 Introduction
- VAX 6200 Configurations
- VAX 6200 System Architecture
- Typical System
- VAX 6200 System (Front View)
- VAX 6200 System (Rear View)
- Supported VAXBI Adapters and Options
- XMI Backplane and Card Cage
- VAXBI Backplane and Card Cage
- VAXBI Expander Cabinet
- TK Tape Drive
- Standard I/O Connections
- I/O Bulkhead Connections
- Power System
- Cooling System

1.1 VAX 6200 Introduction

The VAX 6200, a general purpose computer system designed for growth, is configured for many different applications. Like other VAX systems, the VAX 6200 can support many users in a time-sharing environment.

The VAX 6200 does the following:

- Supports a full set of VAX applications
- Functions as a stand-alone system, a member of a VAXcluster, or as a boot node of a local area VAXcluster
- Allows for expansion of processors, memory, and I/O
- Implements multiprocessing where all processors have equal access to memory
- Uses the VAXBI bus (VAX Bus Interconnect) as the I/O interconnect
- Uses a high-bandwidth internal system bus designed for multiprocessing
- Interleaves memory bank accesses in a user-definable sequence
- Performs automatic self-test on power-up, reset, reboot, or system initialization

1.2 VAX 6200 Configurations

The VAX 6200 system family has configuration packages that differ in the number of processors and amount of memory.

Refer to the *VAX Systems and Options Catalog* for the available configurations.

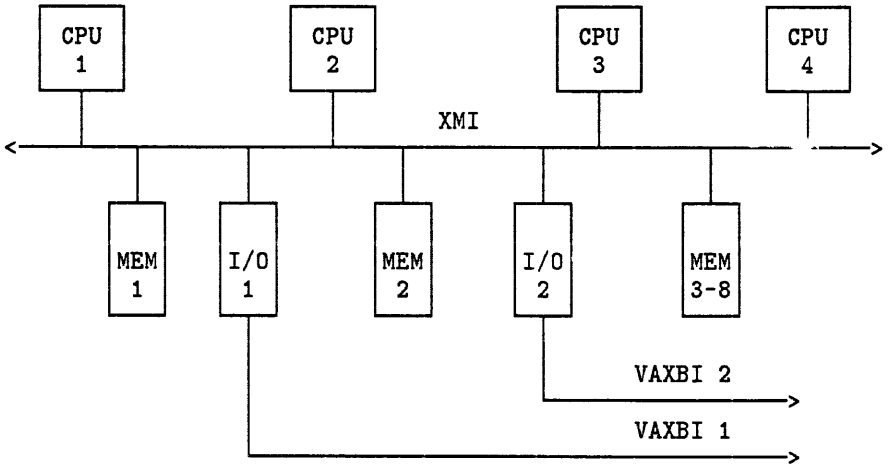
Each configuration has a 60-inch system cabinet that includes one 14-slot high-bandwidth internal system bus backplane (XMI) and two 6-slot VAXBI backplanes.

1.3 VAX 6200 System Architecture

The VAX 6200 uses a high-speed system bus, called the XMI bus, to interconnect its KA62A CPU module(s) and its MS62A memory module(s). All I/O devices connect to the VAXBI bus. The VAX 6200 supports multiprocessing with multiple KA62A CPU modules.

Figure 1-1 shows a four-processor VAX 6200.

Figure 1-1 VAX 6200 System Architecture



The XMI is the VAX 6200 system bus; the VAXBI bus supports the I/O subsystem. The XMI is a 64-bit system bus with a 64 nanosecond bus cycle, and a maximum throughput of 100 megabytes per second. The DWMBA interconnects the I/O adapters.

The VAXBI and XMI share similar but incompatible connector and module technology. Both the VAXBI and XMI buses have the concept of a node. A node is a single functional unit that consists of one or more modules. On the VAXBI bus, a node may be more than one VAXBI module that operates as a single functional unit. On the XMI bus, a node is a single module that occupies one of the 14 logical and physical slots on the XMI bus.

The XMI has three types of nodes: processor nodes (KA62A CPU module), memory nodes (MS62A memory module), and I/O adapters (DWMBA).

A processor node, called a KA62A CPU module, is a single-board processor that contains a central processor unit (CPU) that executes instructions and manipulates data contained in memory; a floating-point processor; 256 Kbytes of onboard cache; and 1 Kbyte of on-chip cache.

The KA62A CPU module communicates with main memory over the XMI bus. The VAX 6200 supports multiprocessing of KA62A CPU modules. A KA62A CPU module becomes the boot processor during power-up and that boot processor handles all system communication. The other CPUs become secondary processors and receive system information from the boot processor.

A memory node is an MS62A memory module. Memory is a global resource equally accessible to all processors on the XMI. Each MS62A memory module uses MOS 1-megabit dynamic RAMs, ECC logic, and control logic. The memories are automatically interleaved for maximum performance. An optional battery backup unit protects memory in case of power failure.

The DWMBA supports bidirectional communication between the XMI and the VAXBI. That is, from CPUs on the XMI to I/O options on the VAXBI and from I/O options on the VAXBI to memory modules on the XMI.

The DWMBA is a 2-board adapter. The DWMBA/A module is installed on the XMI bus, and it communicates with the DWMBA/B module on the VAXBI.

1.4 Typical System

A typical VAX 6200 system has a main cabinet, a console terminal, a disk drive cabinet, an accessories kit, and a set of documentation. It may have additional tape or disk drives and may be a member of a VAXcluster.

Figure 1–2 Typical VAX 6200 System

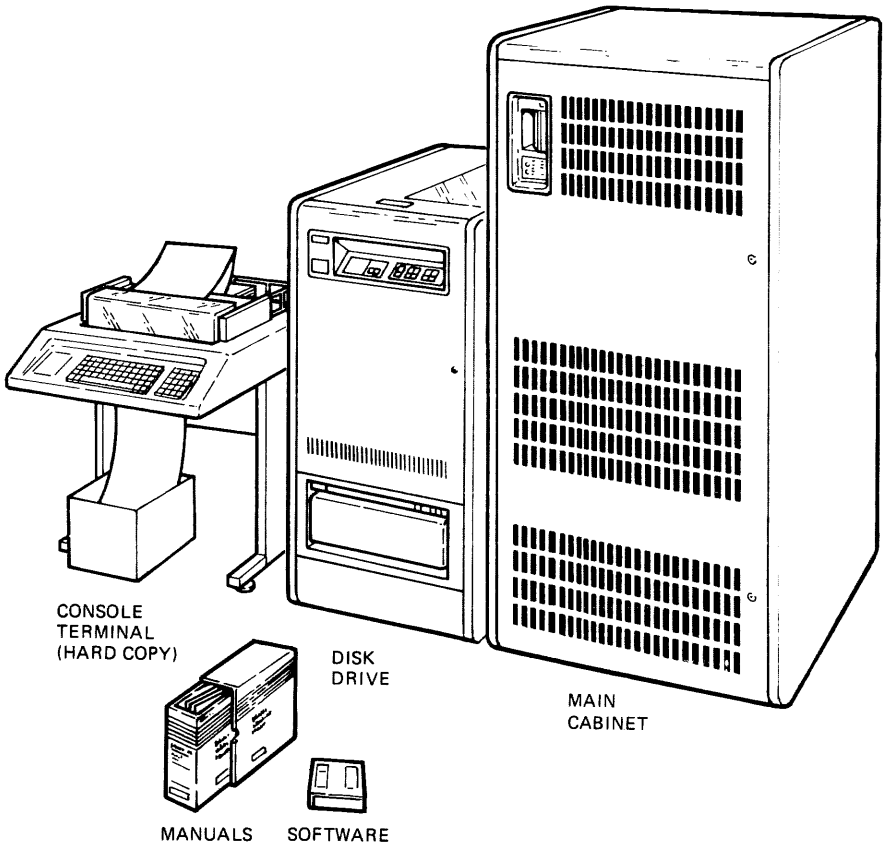


Table 1-1 Typical VAX 6200 System

Component	Function
Main cabinet	Houses system components
TK tape drive	Software distribution; stores and transfers data
Console terminal	Manages system and its resources
System documentation	See *Preface* for full list of documentation related to the VAX 6200
Disk expansion cabinet	Provides storage capacity

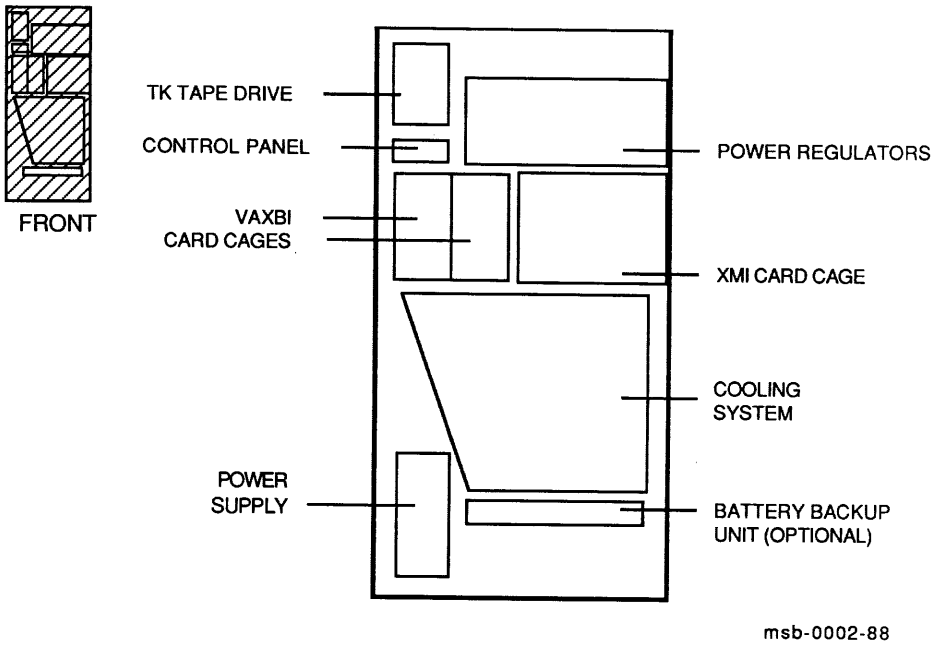
The VAX 6200 components include:

- The main cabinet which houses a TK tape drive, the XMI card cage, two VAXBI card cages, the control panel switches, status indicators, and restart controls.
- The TK tape drive, in the main cabinet, is used for installing operating systems, software, and diagnostics and may be used for data interchange.
- The disk drive cabinet is used for local storage.
- The console terminal is used for booting and for system management operations.
- VAX 6200 documentation includes:
  - *VAX 6200 Installation Guide*
  - *VAX 6200 Owner's Manual*
  - *VAX 6200 Mini-Reference*
  - *VAX 6200 Options and Maintenance*
  - *VAX 6200 System Technical User's Guide*

1.5 VAX 6200 System (Front View)

The TK tape drive and control panel are on the front of the system cabinet. With the front door open, there is access to the system control panel, VAXBI and XMI card cages, the cooling system, battery backup unit (if present), and power regulators.

Figure 1–3 VAX 6200 System (Front View)



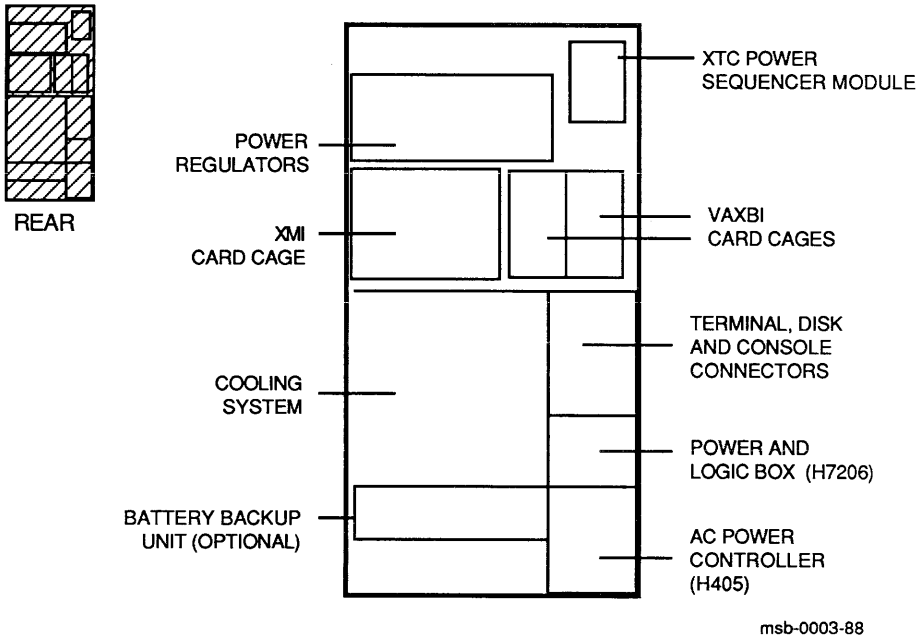
These components are visible from the inside front of the cabinet (see Figure 1–3 for their location):

- TK tape drive
- Control panel
- Power regulators
- Battery backup (if installed)
- XMI card cage
- Two VAXBI card cages
- Cooling system

1.6 VAX 6200 System (Rear View)

With the rear door open, there is access to the following: power regulators; the TK tape drive and control panel connections; cooling system; power and logic box; battery backup unit (if present); AC power controller; console, terminal, and disk connectors; and the I/O bulkhead space.

Figure 1–4 VAX 6200 System (Rear View)



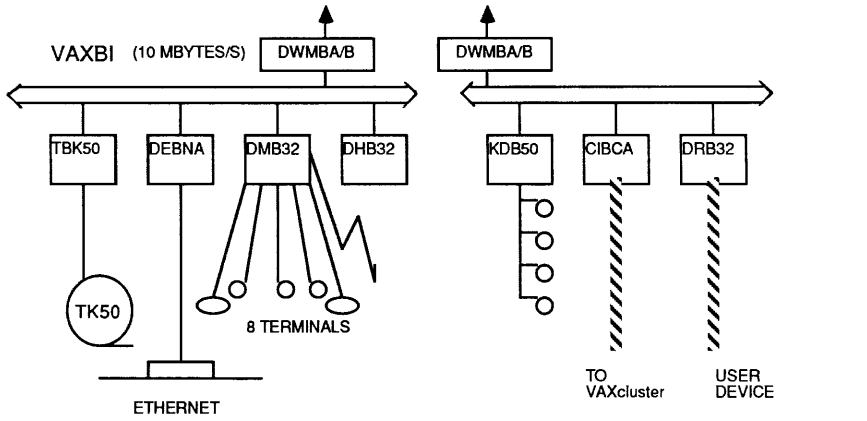
These components are visible from the rear of the cabinet (see Figure 1–4):

- Five replaceable power regulators
- TK tape drive and XTC power sequencer connections
- Cooling system, with open grid over a blower
- Power and logic unit (H7206)
- Battery backup unit (optional)
- AC power controller (H405)
- DWMBA cables
- Terminal, disk, and console connectors
- I/O bulkhead space (The I/O bulkhead panel covers the XMI and VAXBI areas.)

1.7 Supported VAXBI Adapters and Options

The VAX 6200 system supports the use of the many different VAXBI adapters including CIBCA, DEBNA, TBK50, DHB32, DMB32, DRB32, KDB50, and TU81E.

Figure 1–5 VAXBI Adapters

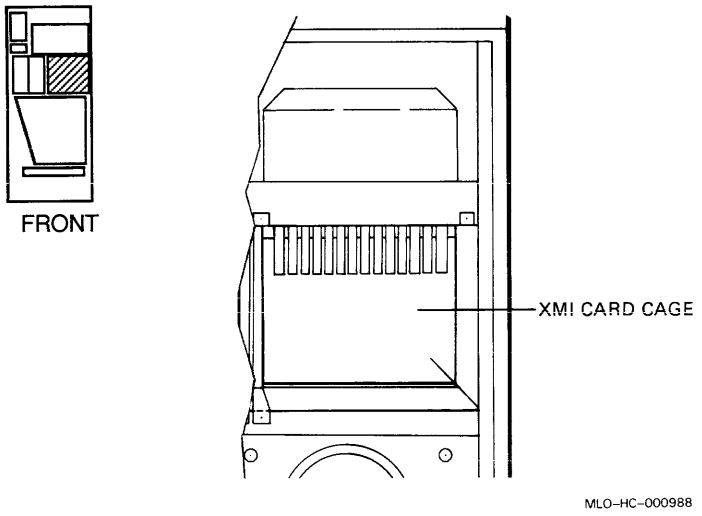


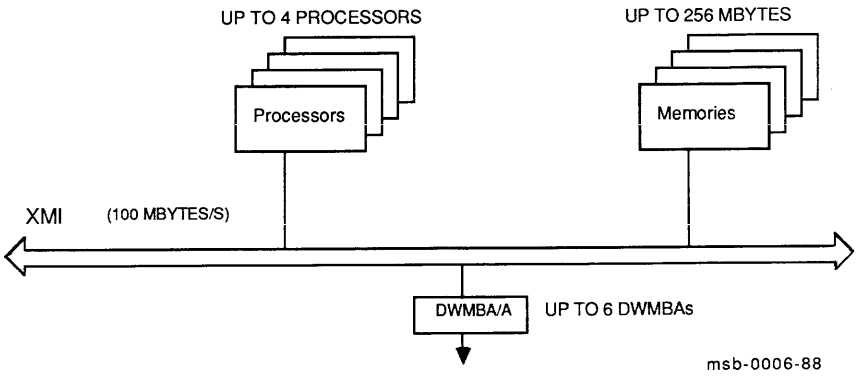
See the *VAX Systems and Options Catalog* for a complete list of VAXBI adapters available for the VAX 6200 and the *VAXBI Options Handbook* for detailed information on each VAXBI adapter.

1.8 XMI Backplane and Card Cage

The XMI high-speed system bus interconnects KA62A CPU module(s) and MS62A memory module(s). The XMI card cage has 14 slots and a maximum bandwidth of 100 megabytes per second.

Figure 1–6 VAX 6200's XMI





The XMI is a limited-length, pended synchronous bus with centralized arbitration. The XMI bus can process several transactions simultaneously, making efficient use of the bus bandwidth. The bus includes the XMI backplane, the electrical environment of the bus, the protocol that nodes use on the bus, and the logic to implement this protocol.

The XMI backplane and 14-slot (nodes 1 through E) card cage are located in the upper third of the cabinet on the right side, as viewed from the front of the cabinet. A clear latched door protects the components housed in the XMI card cage and helps to direct the airflow over the modules. Indicator lights on the XMI modules can be viewed through this clear front door.

Each slot of the XMI card cage is hard-wired to a 4-bit node ID code that corresponds to the physical slot number in the card cage. The node ID number of the module is its slot position. The nodes are numbered 1 through E (hex) from right to left, as you view the card cage from the front of the cabinet.

For information on installing modules in the XMI card cage, see the *VAX 6200 Options and Maintenance* manual. For in-depth technical information, see the appropriate chapter of this manual.

Table 1–2 XMI Slots

Slot	Node	Permissible Modules[1]
1	1	CPU, I/O
2	2	CPU, Mem, I/O
3	3	CPU, Mem, I/O
4	4	CPU, Mem, I/O
5	5	CPU, Mem
6	6	CPU, Mem
7	7	CPU, Mem
8	8	CPU, Mem
9	9	CPU, Mem
10	A	CPU, Mem
11	B	CPU, Mem, I/O
12	C	CPU, Mem, I/O
13	D	CPU, Mem, I/O
14	E	CPU, I/O

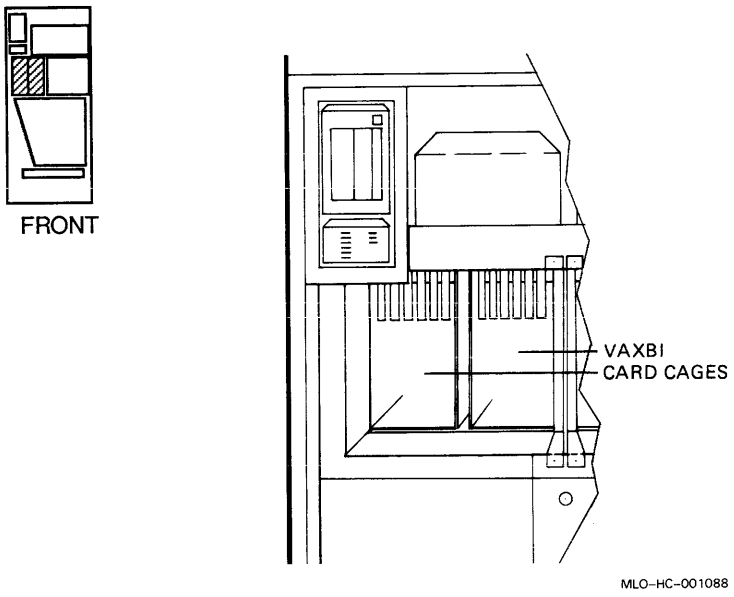
[1]Key to permissible modules:

CPU = KA62A CPU module
Mem = MS62A memory module
I/O = DWMBA

1.9 VAXBI Backplane and Card Cage

The VAXBI is the I/O interface. The VAXBI card cages house modules that connect the system to the Ethernet, VAXclusters, multiple terminals, and other peripherals.

Figure 1–7 VAX 6200's VAXBI



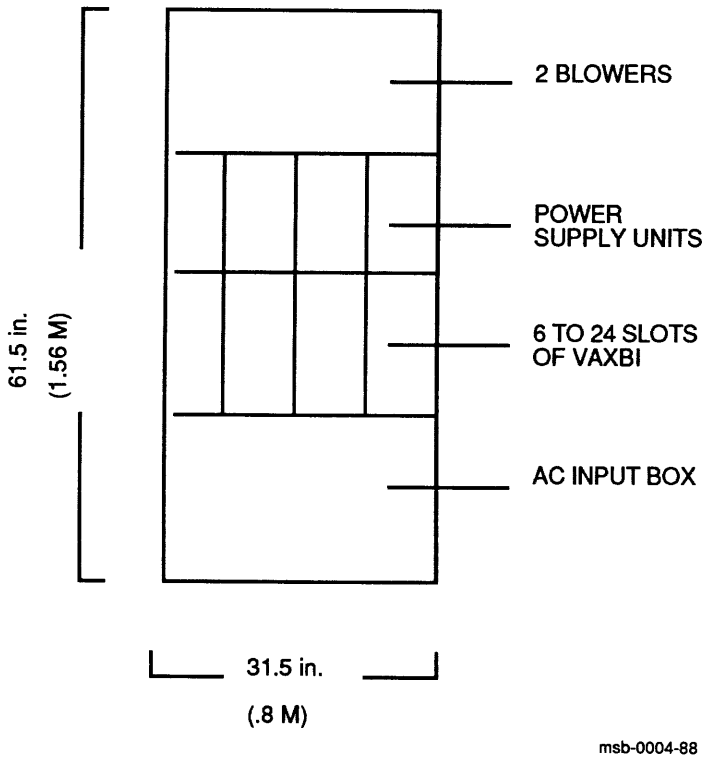
The VAXBI card cages are located in the upper third of the cabinet on the left side, as viewed from the front of the cabinet. A clear latched door protects the components housed in the VAXBI card cages and helps to direct the airflow over the modules. Indicator lights on the VAXBI modules can be viewed through this clear front door.

The VAXBI is available in this system only as a 6-slot fixed-length, nonexpandable card cage. The VAX 6200 system has two 6-slot VAXBI card cages. A VAXBI expander cabinet can also be added to the system (see Section 1.10).

1.10 VAXBI Expander Cabinet

The VAXBI expander cabinet enlarges the system's VAXBI. The VAXBI expander cabinet can have one 6-slot VAXBI card cage through as many as four 6-slot VAXBI card cages.

Figure 1-8 VAXBI Expander Cabinet



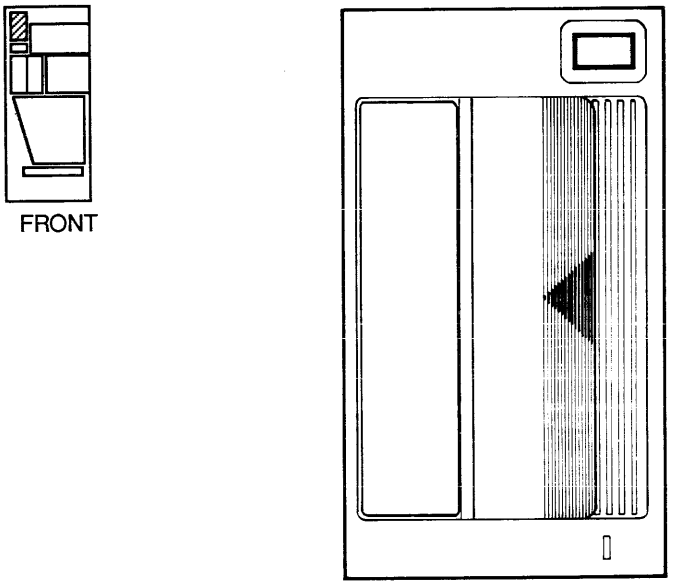
A VAXBI expander cabinet (see Figure 1-8) allows you to attach additional VAXBI channels to provide up to 24 additional slots of VAXBI. The maximum number of nodes for each VAXBI is 6, including the one node required for its DWMBA/B module. The system accepts multiple VAXBI expander cabinets.

The cabinet is 61.5 inches high by 31.5 inches wide. Four power supply units provide power to the VAXBI backplanes. Two blowers cool the cabinet, and an AC power controller completes the power system.

1.11 TK Tape Drive

The TK tape drive is mounted at the front of the system cabinet in the upper left corner.

Figure 1-9 TK Tape Drive



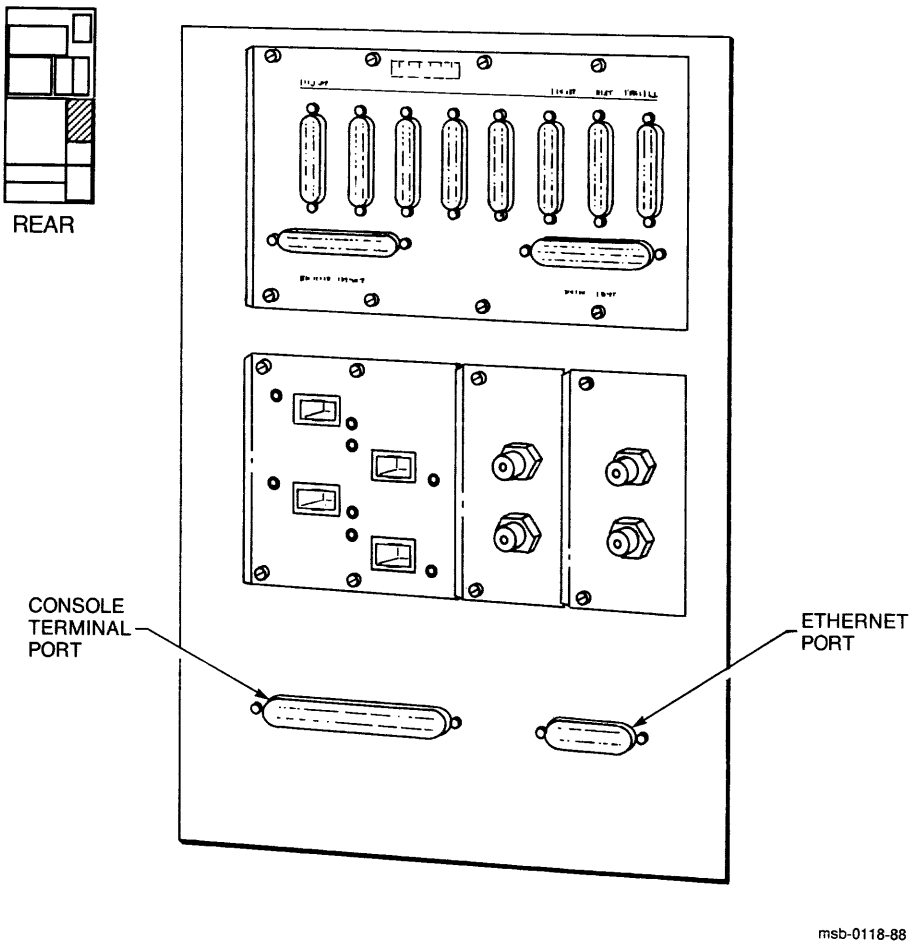
The TK tape drive is used for:

- Installing or updating software
- Loading diagnostics
- Interchanging user data
- Saving, restoring, and updating the contents of the EEPROM
- Loading stand-alone backup

1.12 Standard I/O Connections

The Ethernet and console terminal signal connections are located in the rear of the cabinet, above the AC power controller.

Figure 1–10 Console and Terminal Connectors



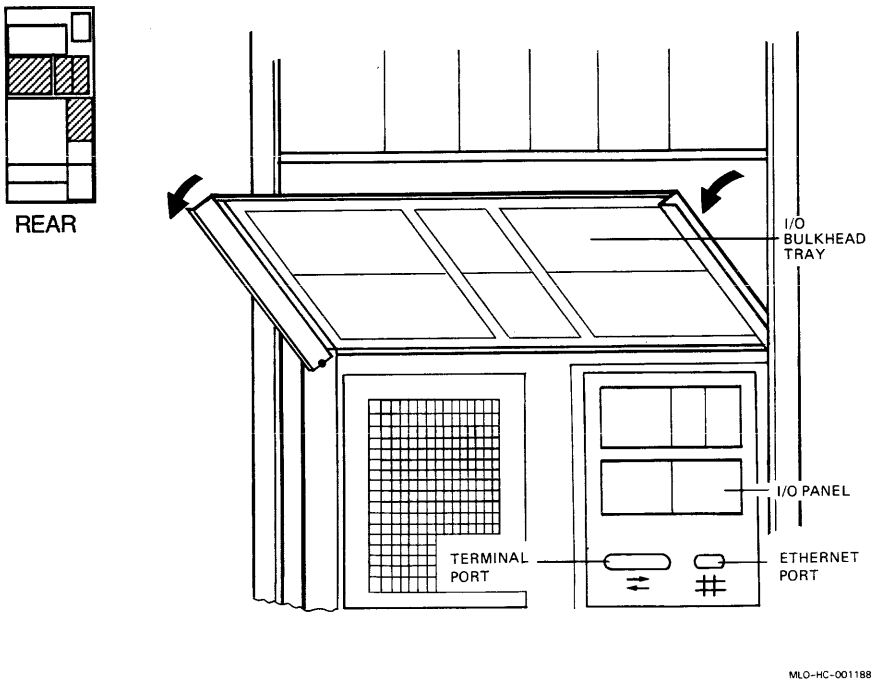
The Ethernet and console terminal ports are at the bottom of the I/O panel. The Ethernet DEBNA port is a 15-pin receptacle located on the bottom right, and the console terminal port is the 25-pin receptacle on the left.

This I/O panel also has two octal openings for additional I/O connections. Typical I/O panels installed in this connector panel may include the tape drive connect area and the KDB50 disk controller port.

1.13 I/O Bulkhead Connections

The I/O bulkhead tray is located in the rear of the cabinet, above the cooling system and standard I/O connection panel, and below the power regulators. The tray covers the XMI and VAXBI backplanes. It is hinged at the bottom and folds out and down for access to the card cages.

Figure 1–11 Sample I/O Bulkhead Connections



The tray is designed to accommodate a variety of I/O connections.

1.14 Power System

The power system consists of an H405E/F AC power controller, the H7206 power and logic unit, five power regulators for the XMI and VAXBI backplanes, and an optional battery backup unit.

Figure 1–12 Power System

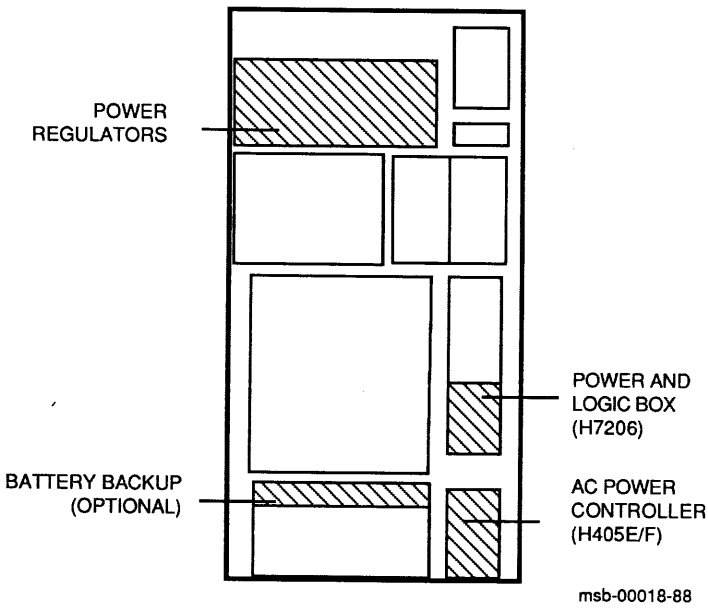


Table 1–3 Input Voltage

Model No.	Hz	Nominal	Phase
H405-E	60	208V	3
H405-F*	50	380V	3
H405-F	50	416V	3

*Change tap for 380V (nominal) operation.

Table 1–4 DC Power Distribution

Voltage	Current (Amps) Min. – Max.	Description
XMI		
+5V	1 – 120	Main logic supply
+5VBB	1 – 120	Memory supply
+12V	0 – 4	Communications devices and TK tape drive
–12V	0 – 2.5	Communications devices
–5.2V	0 – 20	ECL supply
–2V	0 – 7	ECL terminator voltage
VAXBI		
+5V	1 – 120	Main logic supply
+12V	0 – 4	Communications devices
–12V	0 – 2.5	Communication devices
–5.2V	0 – 20	ECL voltage
–2V	0 – 7	ECL teminator voltage
H7206-A power and logic module (PAL)		
+24V	0 – 4	Blowers and airflow sensor
Ethernet transceivers		
+13.5V	0 – 1.5	

Most of the power system is visible from the rear of the cabinet. The AC power controller is in the lower right corner. The power and logic box is above the AC power controller. Across the top of the cabinet are the power regulators for the XMI and VAXBI card cages.

Power is supplied by two H7215 power regulators and three H7214 power regulators. One H7215 and one H7214 supply the power to the VAXBI; one H7215 and two H7214s supply the power to the XMI. See Table 1–4.

The optional H7231 battery backup unit, if present, is located in the front left lower third of the cabinet, near the airflow plenum. This unit supplies 200 watts of +5VBB to system memory for 10 minutes following power interruption.

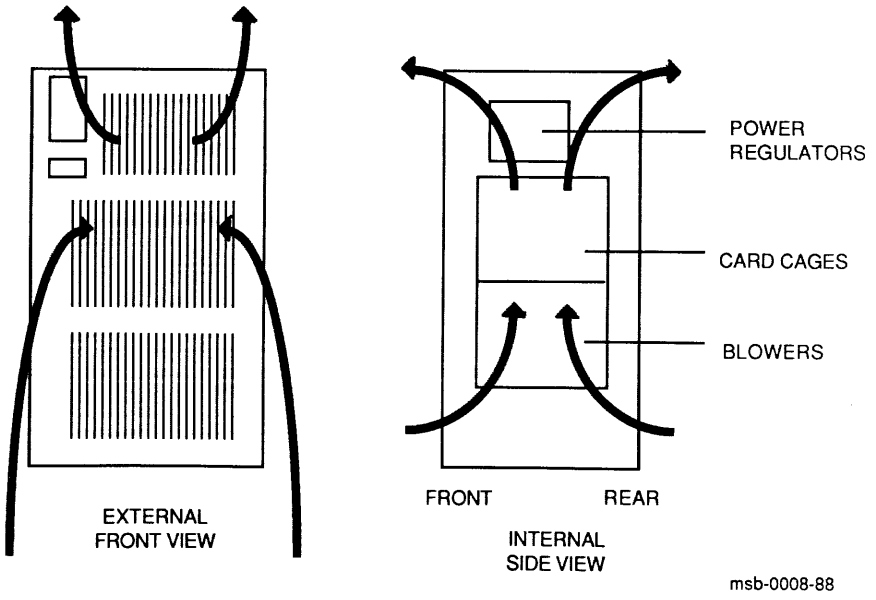
The H7231 battery backup unit power connection is on the H405 AC power controller and is fuse-protected at 10 A (60 Hz) or 6 A (50 Hz).

Three neon lights on the H405-E AC power controller indicate when AC power is present to the unit. The control panel on the front of the system indicates the status of the battery backup unit.

1.15 Cooling System

The cooling system consists of two blowers, an airflow sensor, a temperature sensor, and an airflow path through the card cages and up to the power regulators.

Figure 1–13 Airflow Pattern



The cooling system is designed to keep system components at an optimal operating temperature. It is important to keep the front and rear doors free of obstructions, leaving a clear space of 3 feet (1 meter) from the cabinet, so that air intake is kept at maximum capacity.

Air is taken in by the blowers, located in the lower half of the cabinet. The blowers push air up through the VAXBI and XMI card cages. The airflow continues through the top of the card cages, through the power regulators, and out the top of the front and rear doors.

The system has safety detectors for the cooling system: an airflow sensor and a temperature sensor installed above the power regulators in the top of the cabinet. Extreme conditions activate these detectors. The temperature sensor shuts off the power at the AC power controller if the unit experiences extreme temperatures. If the system has airflow seriously blocked for an extended period of time, then the airflow sensor will shut off power.

2 The XMI

This chapter describes the XMI, which includes a backplane and bus interconnect, protocol, and logic.

This chapter includes the following sections:

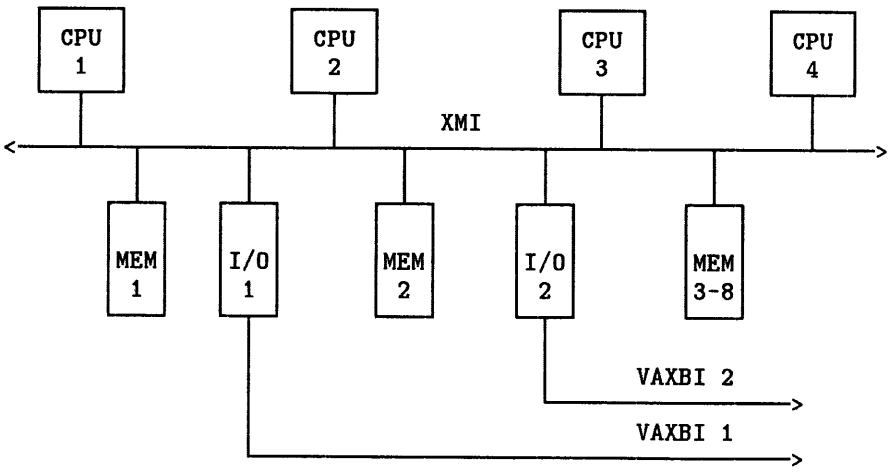
- XMI Overview
- XMI Addressing
- Arbitration Cycles
- XMI Cycles
- XMI Transactions
- XMI Initialization
- XMI Registers
- XMI Errors

2.1 XMI Overview

The XMI is the primary interconnect for the VAX 6200 system. The XMI supports multiple processors, multiple memory modules, and multiple I/O adapters. Figure 2–1 shows a four-processor VAX 6200 system.

2.1.1 XMI System Block Diagram Description

Figure 2–1 VAX 6200 System Block Diagram



The XMI consists of the electrical environment of the XMI bus, the protocol observed by a node on the bus, the backplane, and the logic used to implement the protocol.

The XMI bus is limited length, pended, and synchronous with centralized arbitration. Several transactions can be in progress at a given time, allowing highly efficient use of the bus bandwidth. Arbitration and data transfers can occur simultaneously. The bus supports:

- Quadword-, octaword-, and hexword-length reads to memory
- Quadword- and octaword-length memory writes
- Longword-length read and write operations to I/O space

The longword operations implement byte and word modes required by certain I/O devices. The XMI has a 64 ns bus cycle. The XMI has a bandwidth of 100 Mbytes per second; however, the usable bandwidth depends on transaction length (see Table 2–1).

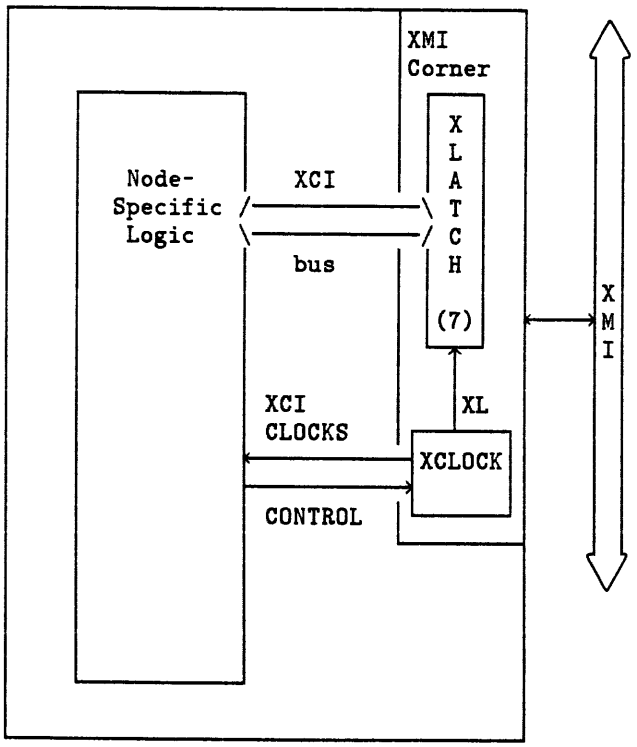
Table 2–1 Usable XMI Bandwidth

Operation	Bandwidth (Mbytes/seconds)
Longword (4 bytes) Read	31.25
Quadword (8 bytes) Read	62.50
Octaword (16 bytes) Read	83.30
Hexword (32 bytes) Read	100.00
Longword Write	31.25
Quadword Write	62.50
Octaword Write	83.30

2.1.2 XMI Corner

The XMI uses similar, but incompatible, connector and module technology as the VAXBI bus and, like the VAXBI, XMI modules have an area with predefined etch with custom components, which serves as the interface between the module and the XMI bus. This predefined etch and components is called the XMI Corner.

Figure 2-2 XMI Node Block Diagram Showing the XMI Corner



The XMI Corner has a predefined etch and parts placement. The custom components in the XMI Corner are called XLATCH and XCLOCK. Both components are implemented in CMOS and interface node-specific logic to the XMI Corner components over the XMI Corner interface (XCI) bus. The XMI Corner, in turn, interfaces directly to the XMI bus. (See Figure 2–2.)

Each node has a set of three clock signals, which are distributed radially to each node from a central source on the backplane. These clocks are received by the XCLOCK chip, which then provides a set of clock waveforms (XCI clocks) to the node-specific logic and the required control lines (XL lines) for the seven XLATCH chips. The XLATCH chips provide the interface to all the XMI lines except those directly interfaced to the XCLOCK chip.

2.1.3 XMI Data Transactions

The XMI supports various data transactions, as shown in Table 2–2.

Table 2–2 Data Transactions Supported by the XMI

Transaction	Length	I/O Space	Memory Space
Read	Longword	X	
	Quadword		X
	Octaword		X
	Hexword		X
Interlock Read	Longword	X	
	Quadword		X
	Octaword		X
	Hexword		X
Write Masked	Longword	X	
	Quadword		X
	Octaword		X
Unlock Write	Longword	X	
	Quadword		X
	Octaword		X

The following terms are used to describe XMI transactions:

Table 2–3 XMI Terms

Term	Description
Node	A hardware device that connects to the XMI backplane.
Transfer	The smallest quantum of work that occurs on the XMI. Typical examples of transfers are the command cycle of a read and the command cyles with the following data cycles of a write.
Transaction	The logical task being performed (such as a read). A transaction is composed of one or more transfers. As an example of a transaction, the read consists of a command transfer followed, some time later, by a return data transfer.
Commander	The node that initiated the transaction in progress. For example, the commander initiates a read transaction while the responder (data source) initiates the read data transfer. The responder is not the commander for the read data transfer because the transfer was requested by the commander node.
Responder	The node that responds to the commander in a transaction.
Transmitter	The node that is sourcing the information on the bus. For example, during a read transaction the commander is the transmitter during the command transfer but is the receiver during the return data transfer.
Receiver	The node that is the target during a transfer.
Naturally aligned	Describes a data quantity whose address could be specified as an offset, from the beginning of memory, of an integral number of data elements of the same size. The lower bits of a naturally aligned data item are zero. All XMI writes transfer a naturally aligned block of data.
Wraparound read	An octaword or hexword read where read data is returned with the specifically addressed quadword first, independent of alignment. The remaining data in the naturally aligned block of data containing the addressed quadword is returned in subsequent transfers.

Reads cause the transfer of data from the responder to the commander. Writes cause the transfer of data from the commander to the responder. Longword commands transfer 4 bytes while quadword, octaword, and hexword commands transfer 8, 16, and 32 bytes, respectively.

Interlocked variations of read commands are intended to do the same thing as the regular reads, but they also invoke a mutual exclusion mechanism where a lock flag associated with the location is set. Unlock writes cause the clearing of the lock flag. During periods when a location is locked, subsequent interlock reads to that location result in the responder returning a "locked" response instead of read data.

All writes are masked and are accompanied by a set of mask bits that specify which bytes of data are to be written. Any arbitrary pattern of bytes can be written with a write mask.

Longword-length transactions may only be used in I/O space (A<29> = 1). Quadword-, octaword-, and hexword-length transactions may only be used in memory space (A<29> = 0).

Addresses for memory and I/O space are given in Section 2.2.1 and Section 2.2.2.

2.1.4 XMI Interrupt Transactions

The XMI supports three types of interrupt transactions, listed in Table 2–4.

Table 2–4 XMI Interrupt Transactions

Type	Mnemonic
Interrupt Request	INTR
Identify (Interrupt Acknowledge)	IDENT
Implied Vector Interrupt	IVINTR

The INTR and IDENT transactions implement device interrupts. An I/O node issues an INTR transaction to a processor to interrupt the processor at a specified interrupt priority level (IPL). The processor responds to the INTR by issuing an IDENT transaction to the interrupting I/O node, soliciting an interrupt vector.

An INTR transaction can be broadcast to multiple processor nodes. The first processor to respond with IDENT receives the interrupt vector. All other processors, upon seeing the IDENT directed to the interrupting device, cease their interrupt-pending condition. If IDENTs are issued simultaneously by two or more processors, the first to gain the bus will service the interrupt while the other(s) force a microcode passive release.

The IVINTR transaction implements single-cycle interrupt transactions where the interrupt priority and the interrupt vector value are implied by bits in the interrupt type field. The IVINTR transaction implements VAX interprocessor interrupts (IPL = 14H, vector = 80H) and write error interrupts (IPL = 1DH, vector = 60H). Since the value of the interrupt vector is indicated by the value of the IPL field, IVINTR transactions do not require a corresponding interrupt acknowledge cycle.

See Section 2.5.5 and Section 2.5.6 for more information on interrupt transactions.

2.1.5 Arbitration

The XMI protocol includes arbitration because, at any time, any or all of the nodes may desire the use of the XMI. Arbitration determines which node gains the XMI when more than one node requests the XMI simultaneously.

Table 2–5 XMI Arbitration Lines

Name	Use
XMI CMD REQ L	Initiates XMI transactions
XMI RES REQ L	Returns data
XMI GRANT L	Indicates which node has been granted the XMI bus for the next cycle

The VAX 6200 supports an XMI bus of 14 nodes. Arbitration cycles occur in parallel with data transfer cycles, since the XMI has a set of lines dedicated to arbitration. These lines are listed in Table 2–5.

When a node desires ownership of the bus, it asserts one of its two request lines (XMI CMD REQ L or XMI RES REQ L) that are connected to the central arbiter. The XMI CMD REQ L line is used by nodes to initiate XMI transactions (that is, act as a commander) while the XMI RES REQ L line is used by nodes to return data to a commander (that is, act as a responder). The XMI arbiter maintains two independent round-robin queues, one for each request type. The responder requests are given higher priority than commander requests.

See Section 2.3 for more information on arbitration.

2.1.6 Bus Integrity

The XMI bus contains a number of features to enhance the integrity and reliability of the bus.

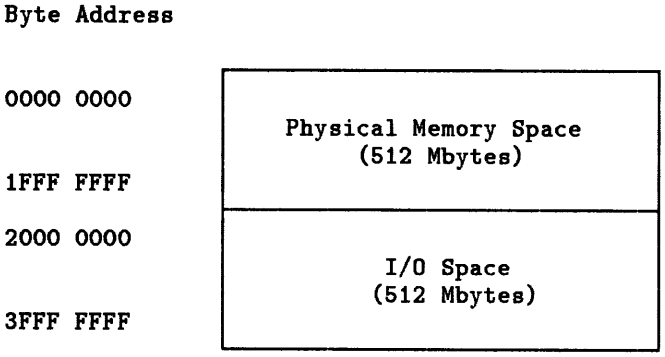
The features of the XMI that enhance bus integrity and reliability are:

- All bus information transfer lines are parity protected.
- Bus confirmation signals are ECC protected.
- XMI protocol permits detection and recovery of almost all single-bit errors on the information transfer lines and bus confirmation signal lines.
- XMI protocol defines timeout conditions that are used to detect failures.

2.2 XMI Addressing

The XMI supports memory with a gigabyte (2^{30} bytes) of address space. This memory space is divided into the physical memory space and I/O space, shown in Figure 2–3.

Figure 2–3 XMI Memory and I/O Address Space



2.2.1 XMI Memory Space

A/D<29> selects between the memory and I/O space. A/D<29> = 0 selects physical memory space, while A/D<29> = 1 selects I/O space.

The upper two bits of an XMI address are used to define transfer size.

2.2.2 XMI I/O Space

XMI I/O space is divided into private space, nodespace, and eight I/O adapter address space regions.

Figure 2–4 XMI I/O Space Address Allocation

Byte Address		Size
2000 0000	XMI Private Space	24 Mbytes
2180 0000	XMI Nodespace	16 x 512 Kbytes
2200 0000	I/O Adapter 1 Address Space	32 Mbytes
2400 0000	I/O Adapter 2 Address Space	32 Mbytes
2600 0000	I/O Adapter 3 Address Space	32 Mbytes
2800 0000	I/O Adapter 4 Address Space	32 Mbytes
2A00 0000	Reserved	192 Mbytes
3600 0000	I/O Adapter B Address Space	32 Mbytes
3800 0000	I/O Adapter C Address Space	32 Mbytes
3A00 0000	I/O Adapter D Address Space	32 Mbytes
3C00 0000	I/O Adapter E Address Space	32 Mbytes
3E00 0000	Reserved	32 Mbytes
3FFF FFFF		

2.2.2.1 XMI Private Space

References to XMI private space are serviced by resources local to a node, such as local device CSRs and boot ROM. The references are not broadcast on the XMI. XMI private space is a 24-Mbyte address region containing the reset address.

2.2.2.2 XMI Nodespace

The VAX 6200 XMI nodespace is a collection of 14 512-Kbyte regions located from 2188 0000 to 21F7 FFFF. Each XMI node is allocated one of the 512-Kbyte regions for its control and status registers. The starting address of the 512-Kbyte region associated with a given node is computed as 2180 0000 + Node ID * 80000.

Table 2–6 XMI Nodespace Addresses

Slot	Node	Nodespace	I/O Adapter Space
1	1	2188 0000 – 218F FFFF	2200 0000 – 23FF FFFF
2	2	2190 0000 – 2197 FFFF	2400 0000 – 25FF FFFF
3	3	2198 0000 – 219F FFFF	2600 0000 – 27FF FFFF
4	4	21A0 0000 – 21A7 FFFF	2800 0000 – 29FF FFFF
5	5	21A8 0000 – 21AF FFFF	N/A
6	6	21B0 0000 – 21B7 FFFF	N/A
7	7	21B8 0000 – 21BF FFFF	N/A
8	8	21C0 0000 – 21C7 FFFF	N/A
9	9	21C8 0000 – 21CF FFFF	N/A
10	A	21D0 0000 – 21D7 FFFF	N/A
11	B	21D8 0000 – 21DF FFFF	3600 0000 – 37FF FFFF
12	C	21E0 0000 – 21E7 FFFF	3800 0000 – 39FF FFFF
13	D	21E8 0000 – 21EF FFFF	3A00 0000 – 3BFF FFFF
14	E	21F0 0000 – 21F7 FFFF	3C00 0000 – 3DFF FFFF

2.2.2.3 I/O Adapter Address Space

I/O adapter address space consists of eight 32-Mbyte address regions used to access VAXBI I/O adapters. Longword-length references directed to a VAXBI's I/O adapter address space will be reissued on that VAXBI bus. XMI transactions are translated into a corresponding VAXBI transaction. The VAXBI address of the transaction is computed from XMI addresses as 2000 0000 + offset, where offset is the difference between the XMI address and the start of the appropriate DWMBA/A module's address space. XMI devices can only access VAXBI I/O space, as VAXBI memory space is not accessible to nodes on the XMI.

To calculate the address of the first register in nodespace (the DTYPE register):

- The base address of I/O space is 2000 0000 (hex).
- Bits<28:25> correspond to the XMI node number, which is the same as the slot number except that node numbers are in hexadecimal while slot numbers are in decimal. The VAX 6200 VAXBI nodes are 1, 2, 3, 4, B, C, D, and E. These are used for:

  E - the VAXBI in the middle of the system cabinet
  D - the VAXBI in the left of the system cabinet
  C - the first VAXBI in an expander cabinet; usually leftmost
  B - the second VAXBI in an expander cabinet; usually center-left
  1 - the third VAXBI in an expander cabinet; usually center-right
  2 - the fourth VAXBI in an expander cabinet; usually rightmost
  3 - the fifth VAXBI in an expander cabinet; not usually used
  4 - the sixth VAXBI in an expander cabinet; not usually used

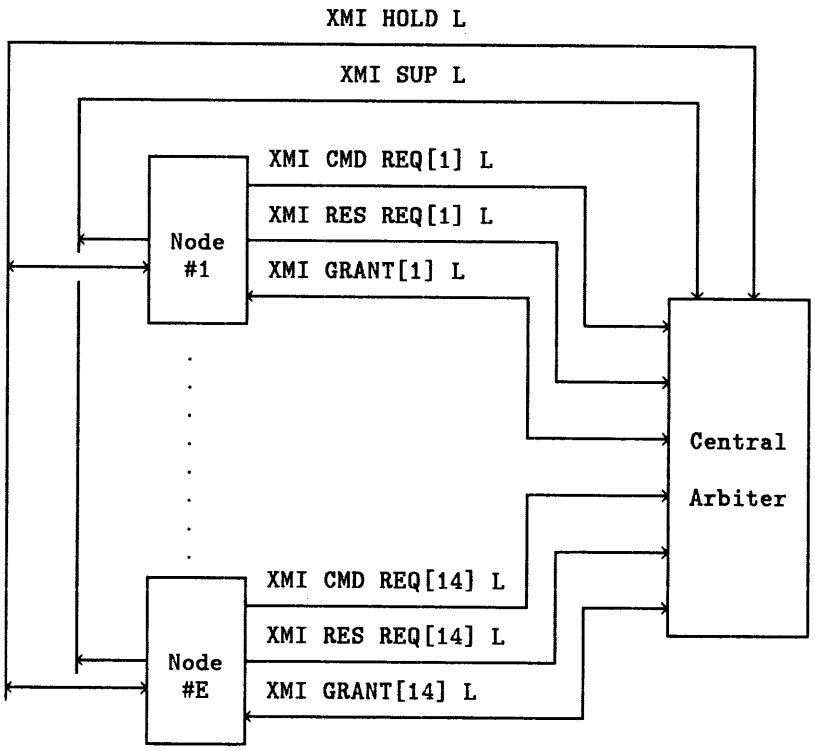
- Bits<16:13> correspond to the VAXBI node number. For the VAXBIs inside the system cabinet, the node number is usually the same as the slot number: 1, 2, 3, 4, 5, or 6. For the VAXBIs in a VAXBI expander cabinet, consult the system-specific configuration chart.

For example, the leftmost slot in the VAXBI in the left of the system cabinet, usually VAXBI node 6 would be connected to XMI node D. The DTYPE register for the VAXBI option in that slot would be addressed as 3A00 C000.

2.3 Arbitration Cycles

The XMI protocol includes arbitration because, at any time, any or all of the nodes may desire the use of the XMI. Arbitration determines which node gains the XMI when more than one node requests the XMI simultaneously. Arbitration cycles occur in parallel with data transfer cycles, since the XMI has a set of arbitration-dedicated lines.

Figure 2–5 XMI Arbitration Block Diagram



The XMI protocol architecturally supports up to 16 XMI nodes. However, the VAX 6200 implementation supports 14 nodes. Each node on the XMI bus has a hexadecimal identification number (1 through E) called the node ID, which is provided by the node's hardwired XMI NODE ID<3:0> H lines. The physical slot number equals the node ID. Slot 1 is the rightmost slot in the XMI card cage when viewed from the front of the cabinet.

Any or all nodes may desire the use of the XMI at any given time. Arbitration cycles occur in parallel with data transfer cycles by using a set of lines dedicated to arbitration. The XMI CMD REQ L line, the XMI RES REQ L line, and the XMI GRANT L line go between the central arbiter and each node. The XMI CMD REQ L line is used by nodes to initiate XMI transactions (to act as a commander), while the XMI RES REQ L line is used to return data to a commander (to act as a responder). The XMI arbiter maintains two independent round-robin queues, one for each of the request types. The responder requests have a higher priority than commander requests.

During any given cycle, all nodes have the opportunity to request the bus. The arbiter receives all the requests, decides which node will be granted the bus, and uses that node's XMI GRANT L line to tell the node that it has been selected. In the next cycle, the selected node begins its transfer.

The XMI has two additional arbitration control signals, XMI HOLD L and XMI SUP L. XMI SUP L suppresses all commander requests but allows responder requests to continue to be serviced. Assertion of XMI HOLD L guarantees that the current XMI transmitter will be granted ownership of the bus in the next cycle, independent of the value of any other outstanding requests. The XMI HOLD L signal is used for multicycle transfers, allowing the current transmitter to keep ownership of the bus for consecutive cycles.

A node can temporarily block the start of additional XMI transactions by asserting the XMI SUP L signal should it have difficulties in keeping up with bus traffic, such as a memory queue becoming full or a CPU backing up on cache invalidate operations due to XMI writes.

The XMI arbitration scheme consists of three priority classes:

- Hold, which has the highest priority and guarantees that the current transmitter will be granted the bus in the next cycle.
- Responder requests, the next highest priority.
- Commander requests, the lowest priority.

Within the responder and commander classes, priority is distributed in a round-robin manner.

2.4 XMI Cycles

The purpose of an XMI cycle is determined by four signal lines on the XMI backplane, XMI F<3:0> L.

2.4.1 Function Codes

The XMI uses four lines to encode the function being performed on the bus. Table 2–7 lists the function codes.

Table 2–7 XMI Function Codes

XMI F<3:0> L Logic Levels					
3	**2**	**1**	**0**	**Function**	**Mnemonic**
0	0	0	0	NULL cycle	NULL
0	0	0	1	Command cycle	CMD
0	0	1	0	Write Data cycle	WDAT
0	0	1	1	Reserved (decoded as NULL)	
0	1	0	0	Lock Response	LOC
0	1	0	1	Read Error Response	RER
0	1	1	0	Reserved (decoded as NULL)	
0	1	1	1	Reserved (decoded as NULL)	
1	0	0	0	Good Read Data 0	GRD0
1	0	0	1	Good Read Data 1	GRD1
1	0	1	0	Good Read Data 2	GRD2
1	0	1	1	Good Read Data 3	GRD3
1	1	0	0	Corrected Read Data 0	CRD0
1	1	0	1	Corrected Read Data 1	CRD1
1	1	1	0	Corrected Read Data 2	CRD2
1	1	1	1	Corrected Read Data 3	CRD3

2.4.2 Command Cycles

During XMI command cycles, commander nodes initiate XMI transactions. The commander drives its commander ID on XMI ID<5:0> L and drives command information on D<63:0> L, as shown in Figure 2–6 and Figure 2–7.

Figure 2–6 Data Transaction Command Cycle Format

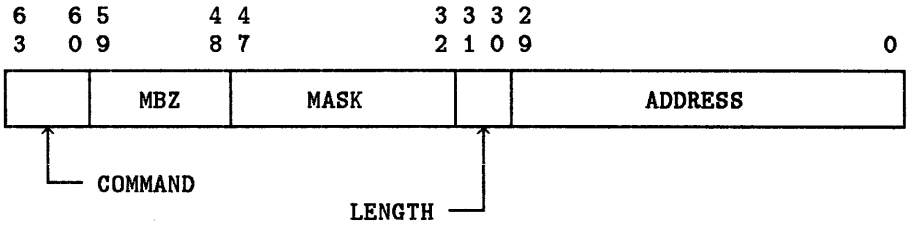
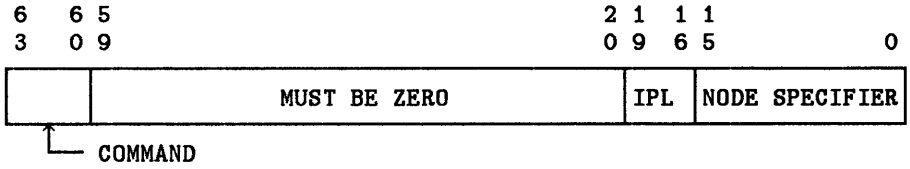


Figure 2–7 Interrupt Transaction Command Cycle Format



The command cycle has the command fields discussed in the following subsections:

- Command field
- Mask field
- Length field
- Address field
- Node Specifier field

2.4.2.1 Command Field

The Command field is XMI D<63:60> L. The Command field specifies the transaction being initiated in the command cycle. (See Table 2–8.)

Table 2–8 XMI Command Codes

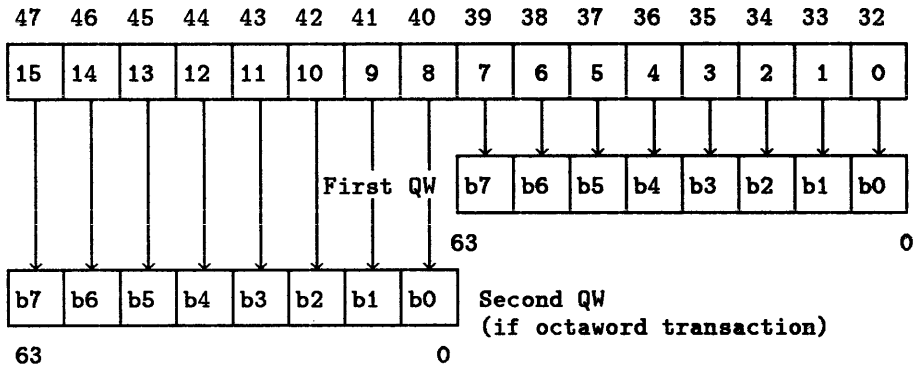
XMI D<63:60> L Logic Levels					
63	**62**	**61**	**60**	**Command**	**Mnemonic**
0	0	0	0	Reserved	
0	0	0	1	Read	READ
0	0	1	0	Interlock Read	IREAD
0	0	1	1	Reserved	
0	1	0	0	Reserved	
0	1	0	1	Reserved	
0	1	1	0	Unlock Write Mask	UWMASK
0	1	1	1	Write Mask	WMASK
1	0	0	0	Interrupt	INTR
1	0	0	1	Identify	IDENT
1	0	1	0	Reserved	
1	0	1	1	Reserved	
1	1	0	0	Reserved	
1	1	0	1	Reserved	
1	1	1	0	Reserved	
1	1	1	1	Implied Vector Interrupt	IVINTR

2.4.2.2 Mask Field

The Mask field is XMI D<47:43> L. The Mask field supplies byte-level mask information for the XMI Write Mask and Unlock Write Mask transactions. During nonwrite transactions this field is a "don't care," but proper parity is still generated. (See Figure 2–8.)

The maximum length of a write transaction is an octaword, which requires 16 mask bits in the upper longword of the command. The mask bits define which bytes of the following write data cycles are to be written to the specified locations. For longword- and quadword-length writes, the unused mask bits (D<47:36> L and D<47:40> L, respectively) are unspecified and are ignored by responders, other than to check parity.

Figure 2–8 Mask Field Bit Assignments



2.4.2.3 Length Field

The Length field is XMI D<31:30> L. The Length field is used to define the number of words in the XMI data transfer. Table 2–9 shows the Length field coding. Longword-length transactions are only used in I/O space. Quadword-, octaword-, and hexword-length transactions are only used in memory space. Hexword lengths are only used for Read or Interlock Read transactions.

Table 2–9 XMI Transaction Length Codes

XMI D<31:30> L Logic Levels		
31	**30**	**Size**
0	0	Hexword
0	1	Longword
1	0	Quadword
1	1	Octaword

2.4.2.4 Address Field

The Address field, XMI D<29:0> L, defines the address of an XMI read or write transaction. The number of significant bits in the address depends on the transaction type and length.

Quadword and octaword write transactions are assumed to be naturally aligned, allowing the lower bits of the address to be "don't care." Reads require that the lower bits be significant because memory does wraparound reads. All wrapped reads need to identify the quadword to be transferred first.

For longword-length transactions, XMI D<1:0> L are only significant for a VAXBI word-mode or byte-mode transaction in I/O space. XMI D<1> L is required for word mode and bits<1:0> are required for byte mode.

The relationship between the high and low words, the state of bit<1>, and the data bits is:

XMI D<1> = 1 ⇒ high word ⇒ D<31:16>
XMI D<1> = 0 ⇒ low word ⇒ D<15:0>

The data returned on the opposite word of the one specified will have correct parity, but its data is unspecified.

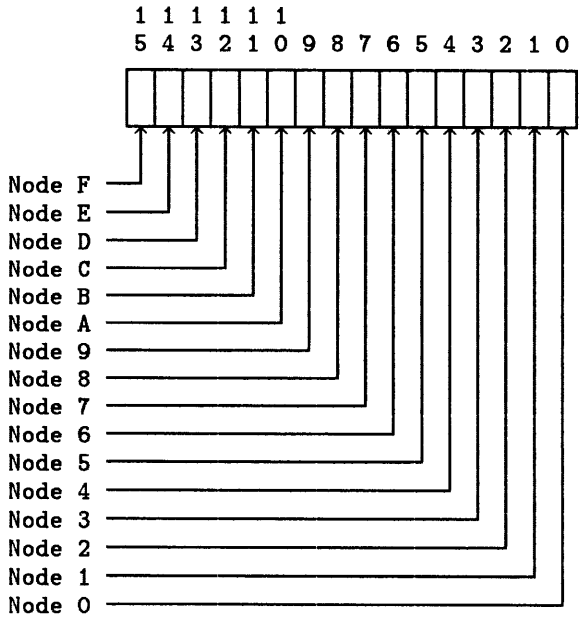
For a longword-oriented device, bit<1> is ignored as an address bit and a full longword of data is returned for a read operation.

2.4.2.5 Node Specifier Field

The Node Specifier field is XMI D<15:0> L. During command cycle interrupt transactions (INTR, IDENT, IVINTR), the Node Specifer field is used to specify the source or destination of an interrupt. (See Figure 2–7.) The relationship between bits in the Node Specifer field and the source/destination of an interrupt transaction is shown in Figure 2–9.

The VAX 6200 uses nodes 1 through E.

Figure 2–9 Node Specifier Field



2.4.3 Write Data Cycles

A function code of 0010 identifies an XMI write data cycle. Write data cycles immediately follow the XMI command cycle during an XMI write transfer. During this cycle, the commander drives its ID on XMI ID<5:0> L and drives write data on D<63:0> L. The full 64 bits of data are used during quadword-length or larger writes. For longword-length writes, only the lower longword D<31:0> L is used and the value of the upper longword is unspecified. In either case, the full 64 bits are used when checking XMI P<2:0> L.

2.4.4 Good Read Data (GRD) and Corrected Read Data Response (CRD) Cycles

Function codes 1000 through 1111 are used to identify return data in response to a Read, Interlock Read, or IDENT transaction. The Good Read Data response (GRDn, codes 1000 - 1011) indicates that the quadword of data is error-free. The Corrected Read Data response, CRDn, codes 1100 - 1111) indicate that the corresponding quadword of data stored in memory contained a single-bit error which was successfully corrected using ECC prior to shipment on the XMI. Both types of read data responses contain a sequence ID located in XMI F<1:0> L, which is used to identify when a read data cycle has been lost due to an XMI parity error.

During a read data response cycle, the responder drives the commander's ID on XMI ID<5:0> L and read data on D<63:0> L. All 64 bits of data are used during quadword- and octaword-length reads. For longword-length reads, only the lower longword (D<31:0> L) is used. In this case, the value of the upper longword is unspecified. In either case, the full 64 bits are used when checking XMI P<2:0> L.

2.4.5 Locked Response Cycle (LOC)

The Locked Response indicates that the location specified in an Interlock Read transaction was already locked. During this cycle the responder drives 0100 on XMI F<3:0> L and the commander's ID on XMI ID<5:0> L. The value of the data bits, D<63:0> L, is unspecified but must be consistent with P<2:0> L. A Locked Response signals the termination of an Interlock Read transaction. When issued, it is always the first and only read response to the transaction.

2.4.6 Read Error Response Cycle (RER)

The Read Error Response indicates that a Read, Interlock Read, or IDENT transaction completed unsuccessfully due to an error condition at the responder node. The Read Error Response is used for an uncorrectable memory error or a reference to a nonexistent location on the VAXBI. During this cycle the responder drives 0101 on XMI F<3:0> L and the commander's ID on XMI ID<5:0> L. The value of the data bits, D<63:0> L, is unspecified but must be consistent with XMI P<2:0> L. A Read Error Response signals the termination of the transaction, and no further read responses are provided.

2.4.7 The Null Cycle

A null cycle is an unused XMI cycle as no node has requested the bus. The null cycle is ignored by all XMI responders.

2.5 XMI Transactions

XMI transactions are listed in Table 2–10.

Table 2–10 XMI Transactions

Name	Mnemonic
Read	READ
Interlock Read	IREAD
Write Mask	WMASK
Unlock Write Mask	UWMASK
Interrupt	INTR
Identify	IDENT
Implied Vector Interrupt	IVINTR

2.5.1 Read Transaction

The Read transactions (READ) are used to transfer a longword, quadword, octaword, or hexword of data from the responder to the commander. A Read transaction is initiated by a commander driving the XMI address and function lines to represent a longword read, quadword read, octaword read, or hexword read. The Read command cycle is decoded by all responder nodes. The node that recognizes its own address latches that address and command. This node is the responder.

When the responder has the requested data, it initiates a return data transfer. Multiple transfers may be necessary to transfer all the quadwords in a given octaword or hexword transaction. The commander monitors the bus traffic waiting for its return data, and then latches the information. The commander issues its own ID in the ID field during the command cycle. The responder returns this same ID with the return read data so that the commander can recognize the return read data it had requested.

Longword-length transactions can only be used in I/O space while quadword-, octaword-, and hexword-length transactions can only be used in memory space.

2.5.2 Interlock Read Transaction

An Interlock Read transaction, combined with a corresponding Unlock Write transaction, permits mutually exclusive access to memory space locations.

The Interlock Read transaction (IREAD) works in memory space. This transaction gains access to a shared object in memory. The exact effect of the Interlock Read depends on the state of the memory's lock bit. Quadword-, octaword-, and hexword-length transactions are used in memory space.

If the memory is already locked, memory responds to IREAD with a Locked Response, and no data is returned. This tells the commander that the shared memory structure is not available at this time. The commander responds to the locked response by repeating the IREAD.

If the memory is not locked, memory locks itself to further IREADs upon receipt of an IREAD and provides the data contained in the addressed locations(s) to the commander. Unlocking the memory requires an UWMASK transaction.

The use of Interlock Read transactions in I/O space is implementation dependent. Most I/O locations treat an Interlock Read like a regular READ. Only longword-length transactions can be used in I/O space.

2.5.3 Write Mask Transaction

Write Mask transactions transfer data from the commander to the responder.

Write Mask transactions (WMASK) transfer quadwords or octawords from the commander to the memory-space responder, such as the MS62A memory module. The commander gains the XMI and sends a command cycle specifying the type of transaction (a longword Write Mask, quadword Write Mask, or an octaword Write Mask), a byte Write Mask, and the desired address. The commander immediately follows this with one or two cycles of write data. All nodes on the XMI decode the address, and the node that recognizes the address becomes the responder.

The responder accepts the command, address, and data and performs the requested write. The mask field that accompanies each command and address has 16 bits. Each bit corresponds to a byte of data in the associated one or two quadwords. If the bit is zero, then that byte is not written; if the bit is one, then that byte is written.

All cache-resident nodes on the XMI are required to monitor write traffic and perform cache invalidates if the XMI write "hits" a block stored in cache.

The XMI has the concept of a "cache invalidate" transaction that does not result in an update of main memory. A commander can perform an invalidate operation by issuing either a quadword Write Mask or octaword Write Mask command with the mask field equal to all zeros. The size of the region to be invalidated is specified in the Length field. Since an invalidate operation is a degenerate case of a Write Mask transaction, it obeys all the Write Mask transaction requirements, including supplying the appropriate write data cycles consistent with the transaction length. As the write data will be discarded by the responder, the value of XMI D<63:0> L during these cycles is unspecified but is consistent with XMI P<1:0> L.

2.5.4 Unlock Write Mask Transaction

The Unlock Write Mask transaction, combined with a corresponding Interlock Read transaction, is used to relinquish the locked memory location after an interlock read.

After a node successfully gains the lock in memory and finishes the required access to the shared structure, it then relinquishes the lock by performing an Unlock Write Mask (UWMASK) to the memory with appropriate data. The memory, which has been monitoring the bus traffic, reacts to the Unlock Write by unlocking memory and writing the data in the request.

If an Unlock Write Mask transaction is directed to a location that is not currently locked, the responder performs the write operation. An implementation might log this as an error in its CSRs.

Unlock Write Mask transactions to I/O space are implementation dependent and can only be longword length.

2.5.5 Interrupt and Identify Transactions

Any I/O device can send an interrupt to one or more processor nodes. A processor eventually issues an Identify and then performs the necessary service routine.

Any of the up to eight I/O devices on the XMI can send out an Interrupt transaction (INTR) to one or more CPU nodes, as designated by a destination mask. One of the processors eventually issues an Identify transaction (IDENT) at a selected level <7:4> and chooses one interrupting node to send it to. That processor then clears the I/O interrupt but other I/O interrupts (if any) remain in parallel to maintain the CPU interrupt request. An interrupt vector is eventually sent to the CPU, which then performs the necessary service routine and then sends out another IDENT or other transaction.

Interrupting nodes do not need to reissue their interrupts after one node/level is serviced. Each CPU monitors the XMI for IDENTs issued by another node. An IDENT issued by one CPU to an interrupting device causes the other processor nodes to clear their corresponding interrupt-pending flag. An interrupting node is not allowed to have more than one interrupt outstanding at a given level.

If more than one processor issues an IDENT for the same interrupt, the first processor node to win the XMI processes the interrupt and the other CPUs clear their corresponding interrupt-pending flags and abort the IDENT by taking a microcode passive release, which is not seen by the operating system.

2.5.6 Implied Vector Interrupt Transactions

The Implied Vector Interrupt is a single-cycle transfer used to implement VAX interprocessor interrupts and write error interrupts where the interrupt priority and interrupt vector are implied by the type of interrupt.

Interprocessor interrupts are issued at IPL 14H with a vector of 80H. Write error interrupts are issued at IPL 1DH with a vector of 60H. Since the value of the interrupt vector is indicated by the value of the Type field, IVINTR transactions do not require a corresponding IDENT (identify or interrupt acknowledge cycle).

The IVINTR transaction contains a 4-bit Type field used to specify the type of interrupt. Only two bits are used: <16> specifies an interprocessor interrupt, while <17> specifies a write error interrupt. The IVINTR transaction also contains a 16-bit Node Specifier field (one bit per node) indicating which nodes are to be interrupted. Interprocessor interrupt transactions can be directed to more than one node. Write error interrupt transactions are directed to only one node.

2.5.7 Transaction Examples

Examples are found in the following subsections:

- Single Data Cycle Reads
- Multiple Data Cycle Reads
- Longword and Quadword Writes
- Multiple Data Cycle Writes

2.5.7.1 Single Data Cycle Reads

The four types of single data cycle reads are:

- Longword Read
- Longword Interlock Read
- Quadword Read
- Quadword Interlock Read

Figure 2–10 Read Transaction

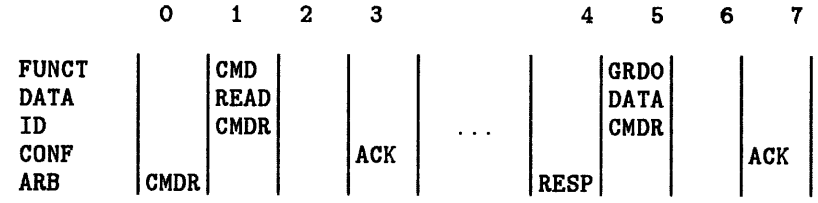
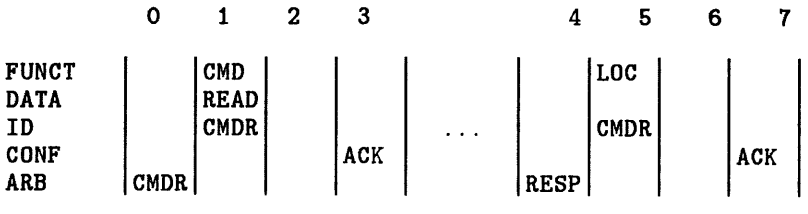


Figure 2–11 Interlock Read Transaction to a Locked Location



The Read transactions consist of a command transfer followed by a return data transfer, as shown in Figure 2–10. The two transfers are the command (FUNCT = CMD) and the read data response (FUNCT = GRD0). The commander arbitrates for the bus in cycle 0 and wins. In cycle 1, it drives the function, command, address of the read, and its own ID (for later use to identify the returning data). In cycle 3, the responder confirms receipt of the information.

Some variable time later, in this example at cycle 4, the return data transfer begins with the responder arbitration for the bus. Having won it, the responder drives the function, the data, and the commander's ID in cycle 5. The status of the returning data is specified in the read response function code, either Good Read Data, Corrected Read Data, or Read Error Response. The commander monitors the bus, checking for an ID match during read data cycles to indicate that the read data is meant for that commander.

If the particular transaction requested had been an Interlock Read, and if the memory was already interlocked, the commander would have provided a Locked Response in place of the returned data. (See Figure 2–11.)

2.5.7.2 Multiple Data Cycle Reads

The four types of multiple data cycle reads are:

- Octaword Read
- Octaword Interlock Read
- Hexword Read
- Hexword Interlock Read

Figure 2–12 Multiple Data Cycle Reads Command Cycle

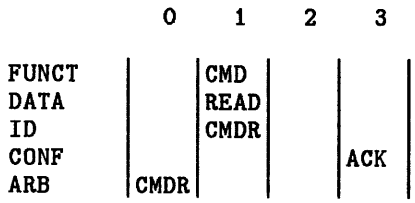


Figure 2–13 Read Data Cycles

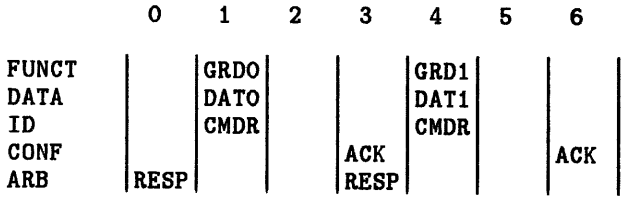
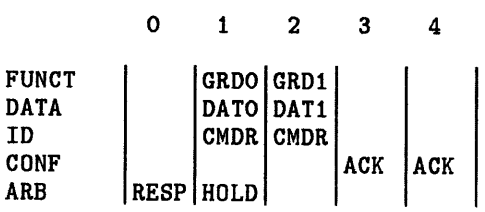


Figure 2–14 Read Data Cycles with HOLD



The four multiple data cycle read transactions move either 16 bytes (octaword) or 32 bytes (hexword) of data from the responder to the commander. Figure 2–12 is the command transfer of the transaction. The Interlock Read checks the state of the lock bit in the memory and qualifies the request, based on its state. This illustration applies to both octaword and hexword reads.

Figure 2–13 is a diagram of the return data transfer applicable to octaword reads, moving four longwords of data. The function field of the bus in cycle 1 indicates "good read data 0" with the ID field identifying the intended receiver (the transaction commander). Cycle 4 is a Good Read Data 1 cycle. Each cycle provides a new quadword of read data while the ID remains unchanged.

Read data may be returned in consecutive cycles through the use of HOLD, as shown in Figure 2–14. The transmitter asserts HOLD in the first cycle to ensure that it maintains the use of the bus until it completes the transfer. HOLD is the highest priority arbitration line and guarantees use for a maximum of four consecutive cycles. The confirmation is returned to the commander two cycles after the command cycle.

Bus usage during a hexword read with a single correctable read error is shown in Figure 2–15.

Figure 2–16 illustrates the events during a return data of hexword length containing an uncorrectable read error. When memory encounters an uncorrectable read error, it returns a Read Error Response and suppresses further read responses for that transaction.

Figure 2–15 Hexword Read with Single Correctable Read Error

	0	1	2	3	4	5	6	7
FUNCT		GRDO	GRD1	CRD2		GRD3		
DATA		DATO	DAT1	DAT2		DAT3		
ID		CMDR	CMDR	CMDR		CMDR		
CONF				ACK	ACK	ACK		ACK
ARB	RESP	HOLD	HOLD		RESP			

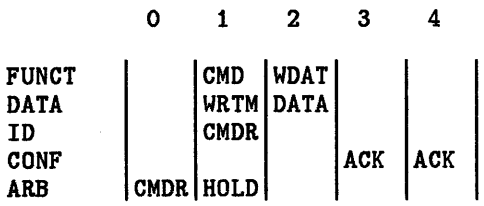
Figure 2–16 Hexword Data Return with Uncorrectable Read Error

	0	1	2	3	4	5
FUNCT		GRDO	GRD1	RER		
DATA		DATO	DAT1			
ID		CMDR	CMDR	CMDR		
CONF				ACK	ACK	ACK
ARB	RESP	HOLD	HOLD			

2.5.7.3 Longword and Quadword Writes

Longword and quadword writes can be either Write Mask or Unlock Write Mask transactions.

Figure 2–17 Longword and Quadword Writes

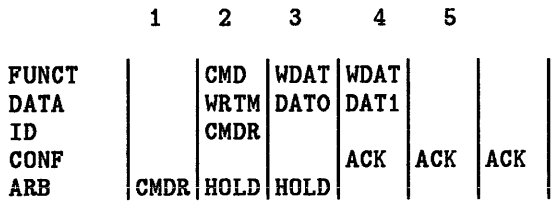


Longword and quadword writes move the number of bytes specified by the Mask field. The commander arbitrates for the XMI bus and, upon winning it, drives the appropriate write command, the intended address, the data mask, its own ID, and asserts HOLD to signal that it will need the next cycle as a Write Data cycle. It then provides the write data but no ID field, having identified itself in the command cycle. Cycles 3 and 4 show the confirmation from the responder.

2.5.7.4 Multiple Data Cycle Writes

The multiple data cycle writes are the octaword Write Mask and the octaword Unlock Write Mask transactions.

Figure 2–18 Multiple Data Cycle Writes



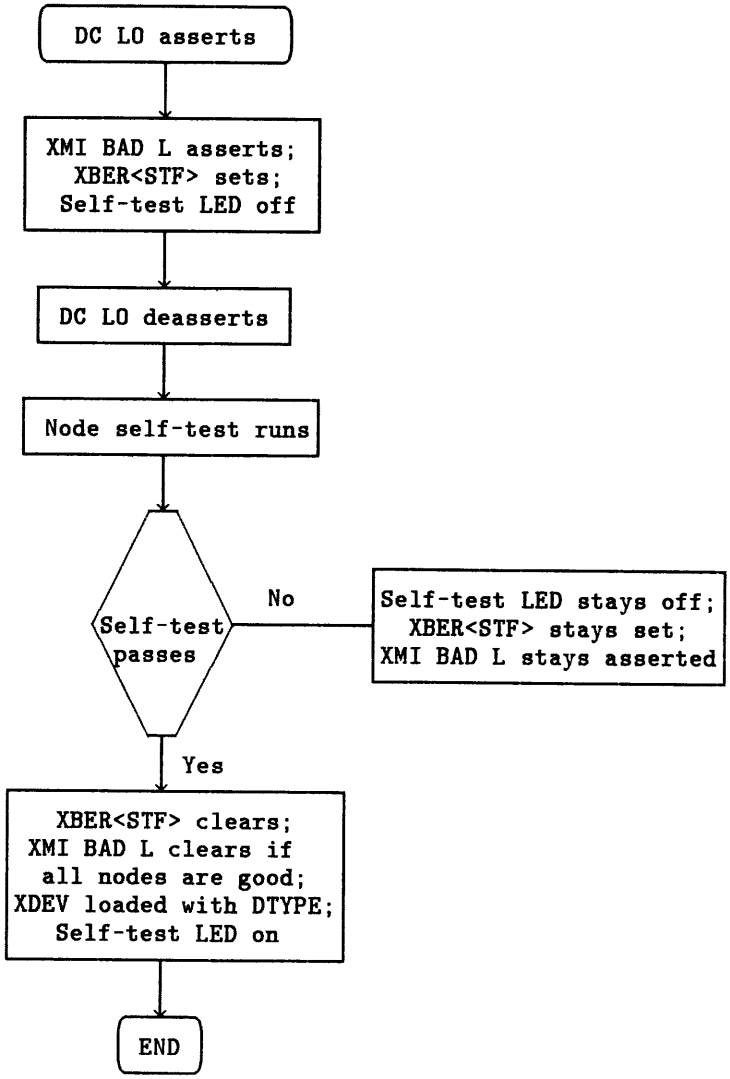
NOTE: The write data must immediately follow the comand cycle with no intervening null cycles.

Multiple data cycle writes identify the first cycle of the transfer with the desired write length. HOLD is asserted while successive cyles provide new data.

2.6 XMI Initialization

Regardless of the method used to cause a node to initialize, the initialization process consists of the same steps.

Figure 2-19 XMI Initialization Flowchart



2.6.1 Causes of an Initialization

Three causes of XMI initialization are:

- Power-down/power-up
- System reset
- Node reset

2.6.2 Power-Up

On power-up, the XMI AC LO L, XMI DC LO L, and XMI RESET L lines are sequenced to provide initialization of all nodes in the system.

During normal power-up, a node cannot access XMI-accessible memory space locations until the deassertion of XMI AC LO L. However, memory nodes clear memory locations following the deassertion of XMI DC LO L if a cold start is indicated. During a system reset sequence, it is possible for the resetting node to access memory prior to the deassertion of XMI AC LO L, but no other node can access memory prior to the deassertion of XMI AC LO L.

During brownout power conditions, XMI AC LO may assert and later deassert without an assertion of XMI DC LO L. XMI AC LO L remains asserted for a period of time after the deassertion of XMI DC LO L, allowing a node's internal initialization signals to be removed before a power restart interrupt is raised.

XMI DC LO L warns of the impending loss of DC power and is used for initialization on power-up. DC power and the XMI clock become valid before the deassertion of XMI DC LO L. XMI DC LO L is asserted after the assertion of XMI AC LO L, allowing the power-fail routine to save processor state in memory and to halt. The result of any XMI transaction in progress when XMI DC LO L asserts is indeterminate.

XMI DC LO L asserts before the loss of DC power so that nodes such as disk controllers can stop certain activities before the removal of power.

In a power outage, first AC power is lost, then (if not restored quickly), DC power falls below acceptable levels, asserting first XMI AC LO L and then XMI DC LO L.

During a power outage, systems with the battery backup option have their memory nodes supplied with battery power, allowing memory contents to be retained. After power is restored, the memory is not reinitialized. However, if the power outage is lengthy and exhausts the battery, the data in memory is no longer reliable. One function of the XTC power sequencer is to monitor the battery backup power voltage and assert the XMI RESET L line if the battery was exhausted.

2.6.3 System Reset

A power-down/power-up sequence can be emulated through the use of the XMI RESET L line, which causes the sequencing of XMI AC LO L and XMI DC LO L in the same way as a true power-down/power-up sequence. This allows all nodes in the system to be returned (or "reset") to their power-up state without cycling the power supplies. The XTC power sequencer is also used to carry out the reset sequence.

The XTC power sequencer monitors the XMI RESET L line and drives the XMI AC LO L, XMI DC LO L, and XMI RESET L lines. Upon detection of an asserted XMI RESET L line, the XTC begins the reset sequence. If XMI RESET L is asserted while XMI AC LO L and XMI DC LO L are deasserted, the XTC asserts XMI AC LO L first, then XMI DC LO L, and finally deasserts XMI DC LO L. In response, all XMI nodes perform self-test and initialization. When the RESET line is deasserted, the reset module deasserts XMI AC LO L, completing the emulation of the power-down/power-up sequence. If the RESET line remains asserted until after XMI DC LO L is deasserted, then all memory nodes reset, including those with battery backup.

2.6.4 Node Reset

A single node in a system can be reset without resetting the entire system by writing a one to the Node Reset bit (NRST) in the XMI Bus Error Register of that particular node. The node is inaccessible for the duration of its initialization and XMI BAD L is asserted. Accessing the node during self-test may cause a self-test failure. Software drivers that share a node must agree in advance that a node needs to be reset and lock the selection of that node.

2.7 XMI REGISTERS

This section describes the registers required for all XMI nodes.

Each XMI node is required to have a set of two or three registers in a specified location within the node's nodespace, as shown in Table 2–11. Table 2–12 defines the abbreviations used to describe the type of bits in the register descriptions.

Descriptions of module-specific XMI registers are given in the chapters on the XMI options.

Table 2–11 XMI Registers

Register	Mnemonic	Address	Node Requirements
Device Register	XDEV [1]	BB [2] + 0000 0000	All nodes
Bus Error Register	XBER	BB + 0000 0004	All nodes
Failing Address Register	XFADR	BB + 0000 0008	Commanders only

[1]X in the mnemonic indicates that this is an XMI register.

[2]BB = base address of a node, which is the address of the first location in nodespace.

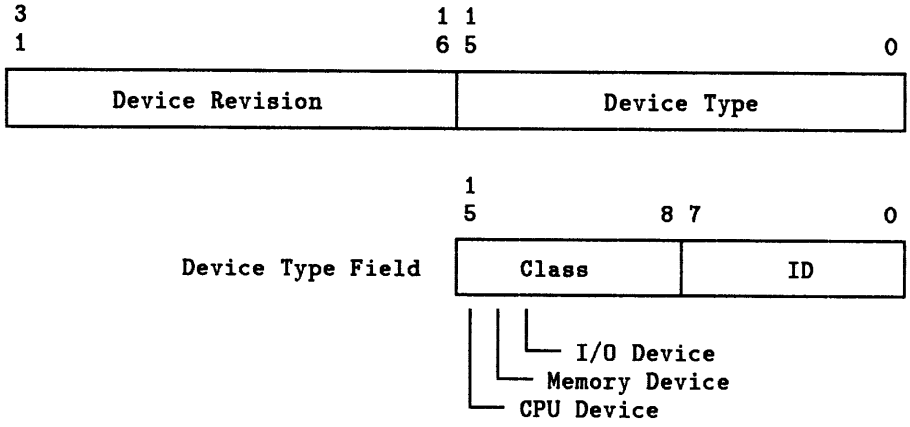
Table 2–12 Abbreviations for Bit Type

Abbreviation	Definition
0	Initialized to logic level zero
1	Initialized to logic level one
X	Initialized to either logic state
RO	Read only
R/W	Read/write
R/W1C	Read/cleared by writing a 1

Device Register (XDEV)

The Device Register contains information to identify the node. Both fields are loaded during node initialization. A zero value indicates an uninitialized node.

ADDRESS *Nodespace base address + 0000 0000*



Device Types are:

KA62A CPU Module:	8001
MS62A Memory Module:	4001
DWMBA/A XMI Module:	2001

bits<31:16>

Name: Device Revision

Mnemonic: DREV

Type: R/W, 0

Identifies the functional revision level of the device. The use of the Device Revision field is implementation dependent.

bits<15:0>

Name: Device Type

Mnemonic: DTYPE

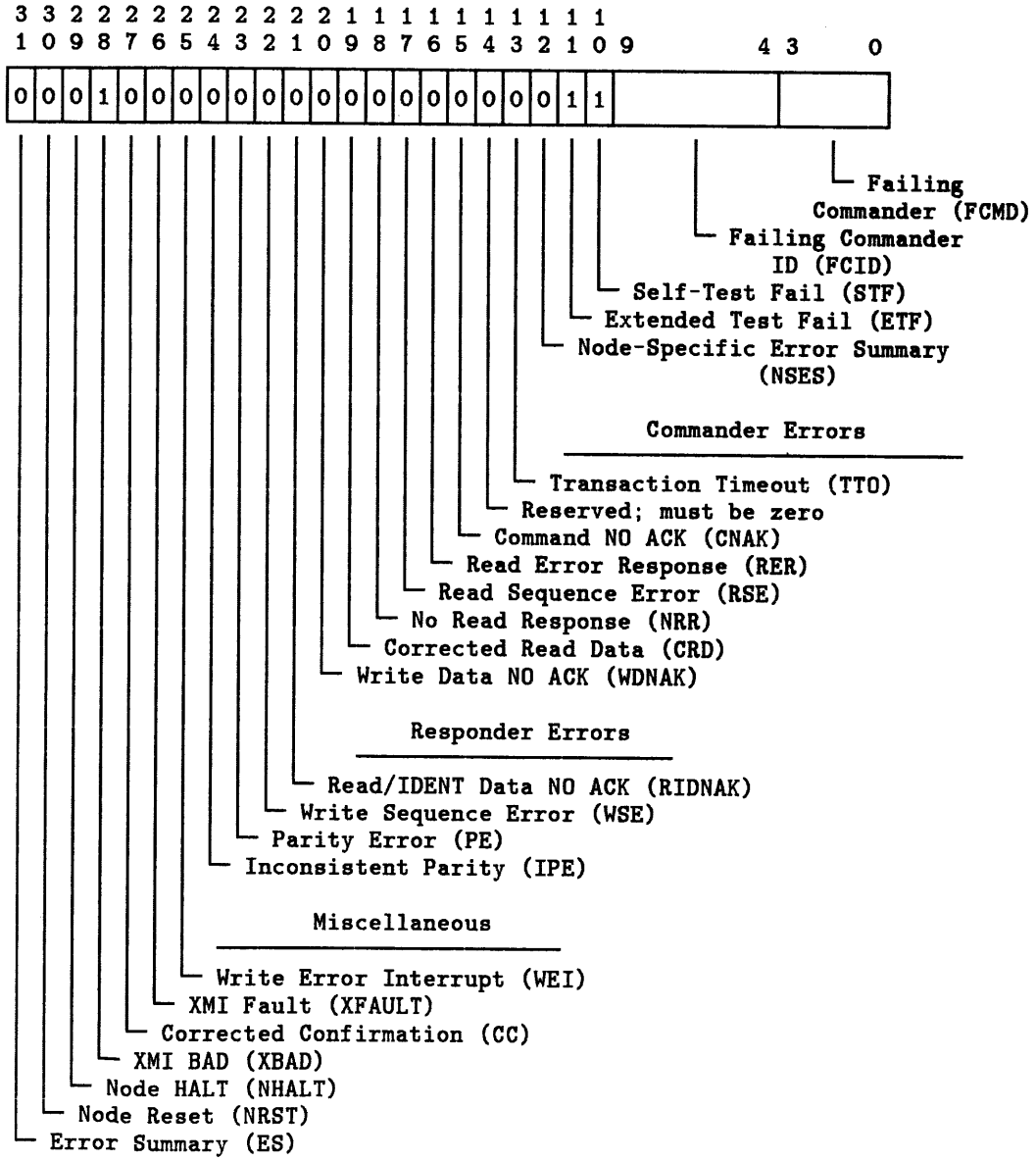
Type: R/W, 0

Identifies the type of node. The Device Type field is broken into two subfields: Class and ID. The Class field indicates the major category of the node. The currently defined classes are CPU, memory, and I/O. The ID field uniquely identifies a particular device within a specified class.

Bus Error Register (XBER)

The Bus Error Register contains error status on a failed XMI transaction. This status includes the failed command, commander ID, and an error bit that indicates the type of error that occurred. This status remains locked up until software resets the error bit(s).

ADDRESS *Nodespace base address + 0000 0004*



bit<31>

Name: Error Summary
Mnemonic: ES
Type: RO, 0

ES represents the logical OR of the error bits in this register. Therefore, ES asserts when any error bit asserts.

bit<30>

Name: Node Reset
Mnemonic: NRST
Type: R/W, 0

Writing a one to NRST initiates a complete power-up reset similar to the assertion and deassertion of XMI DC LO L (see note below); the node performs self-test and asserts XMI BAD L until it is successfully completed. Like power-up reset, nodes are precluded from accessing the node from the time it is node reset until it completes self-test (or the maximum self-test time is exceeded).

NOTE: During the time that a node is responding to node reset, the node does not access other nodes on the XMI bus. In response to a real power-up sequence (caused by XMI DC LO L), the NRST bit will be reset. Following a node reset sequence, it will remain set allowing the processor to recognize that it should not attempt to go through the normal boot process.

bit<29>

Name: Node HALT
Mnemonic: NHALT
Type: R/W, 0

Writing a one to NHALT forces the node to go into a "quiet" state while retaining as much state as possible. The KA62A CPU module will force the CVAX chip to HALT and go into console mode waiting for console commands.

bit<28>

Name: XMI BAD
Mnemonic: XBAD
Type: R/W, 1

On reads, XBAD indicates the state of the XMI BAD signal. A one indicates that BAD is asserted. Writes to this location supply the state to be driven on the wired-OR XMI BAD L line by this node; writing a one asserts XMI BAD L, while writing a zero releases it. Only XMI processor nodes are required to implement this bit. If not implemented, nodes return zero.

bit<27>

Name: Corrected Confirmation
Mnemonic: CC
Type: R/W1C, 0

CC sets when the node detects a single-bit CNF error. Single-bit CNF errors are automatically corrected by the XCLOCK chip.

bit<26>

Name: XMI FAULT
Mnemonic: XFAULT
Type: R/W1C, 0

When set, XFAULT indicates that the XMI FAULT signal has been asserted for at least one cycle. Only XMI processor nodes are required to implement this bit. If not implemented, nodes return zero.

bit<25>

Name: Write Error Interrupt
Mnemonic: WEI
Type: R/W1C, 0

When set, WEI indicates that the node has received a write error interrupt transaction. Only XMI processor nodes are required to implement this bit. If not implemented, nodes return zero.

bit<24>

Name: Inconsistent Parity Error
Mnemonic: IPE
Type: R/W1C, 0

When set, IPE indicates that the node has detected a parity error on an XMI cycle and the confirmation for the errored cycle was ACK. This indicates that at least one node (the responder) detected good parity during the cycle time that this node detected a parity error. Only XMI processor nodes are required to implement this bit. If not implemented, nodes return zero.

bit<23>

Name: Parity Error
Mnemonic: PE
Type: R/W1C, 0

When set, PE indicates that the node has detected a parity error on an XMI cycle.

bit<22>

Name: Write Sequence Error
Mnemonic: WSE
Type: R/W1C, 0

When set, WSE indicates that the node aborted a write transaction due to missing data cycles. Only XMI responder nodes are required to implement this bit. If not implemented, nodes return zero.

bit<21>

Name: Read/IDENT Data NO ACK
Mnemonic: RIDNAK
Type: R/W1C, 0

When set, RIDNAK indicates that a Read or IDENT data cycle (GRDn, CRDn, LOC, RER) transmitted by the node has received a NO ACK confirmation.

bit<20>

Name: Write Data NO ACK
Mnemonic: WDNAK
Type: R/W1C, 0

When set, WDNAK indicates that a Write data cycle (GRDn, CRDn, LOC, RER) transmitted by the node has received a NO ACK confirmation.

bit<19>

Name: Corrected Read Data
Mnemonic: CRD
Type: R/W1C, 0

When set, CRD indicates that the node has received a CRDn read response. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero.

bit<18>

Name: No Read Response
Mnemonic: NRR
Type: R/W1C, 0

When set, NRR indicates that a transaction initiated by the node failed due to a read response timeout. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero.

bit<17>

Name: Read Sequence Error

Mnemonic: RSE

Type: R/W1C, 0

When set, RSE indicates that a transaction initiated by the node failed due to a read sequence error. Only XMI commander nodes are required to implement this bit. This bit will be set only if the reattempt fails on commanders implementing error recovery. If this bit is not implemented, nodes return zero.

bit<16>

Name: Read Error Response

Mnemonic: RER

Type: R/W1C, 0

When set, RER indicates that a node has received a Read Error Response. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero.

bit<15>

Name: Command NO ACK

Mnemonic: CNAK

Type: R/W1C, 0

When set, CNAK indicates that a command cycle transmitted by the node has received a NO ACK confirmation caused by either a reference to a nonexistent memory location or a command cycle parity error. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero. For commanders implementing error recovery, this bit is set only if the reattempts fail.

bit<14>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bit<13>

Name: Transaction Timeout
Mnemonic: TTO
Type: R/W1C, 0

When set, TTO indicates that a transaction initiated by the node failed due to a transaction timeout. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero. For commanders implementing error recovery, this bit is set only if the reattempts fail.

bit<12>

Name: Node-Specific Error Summary
Mnemonic: NSES
Type: RO, 0

When set, NSES indicates that a node-specific error condition has been detected. The exact nature of the error is contained in node-specific registers.

bit<11>

Name: Extended Test Fail
Mnemonic: ETF
Type: R/W1C, 1 (processors), 0 (all others)

When set, ETF indicates that the node has not yet passed its extended test. This bit clears when the node passes its extended test. Only processor nodes implement extended test; all other nodes power up with ETF cleared.

bit<10>

Name: Selt-Test Fail
Mnemonic: STF
Type: R/W1C, 1

When set, STF indicates that the node has not yet passed its self-test. This bit is cleared by the user interface when the node passes its self-test.

bits<9:4>

Name: Failing Commander ID
Mnemonic: FCID
Type: RO

This field logs the commander ID of a failing transaction. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero.

bits<3:0>

Name: Failing Command

Mnemonic: FCMD

Type: RO

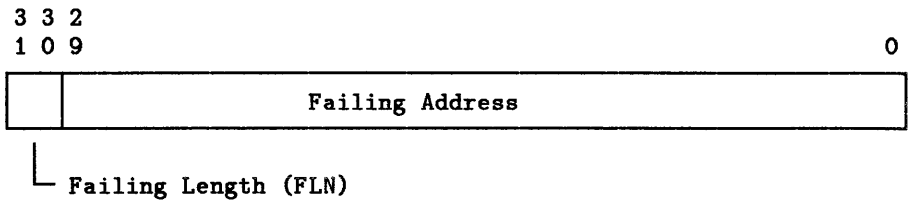
This field logs the command code of a failing transaction. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero.

Failing Address Register (XFADR)

The Failing Address Register logs address and length information associated with a failing transaction. Only XMI commander nodes are required to implement this register.

ADDRESS

Nodespace base address + 0000 0008



bits<31:30>

Name: Failing Length

Mnemonic: FLN

Type: RO

This field logs the value of XMI D<31:30> during the command cycle of a failing transaction.

bits<29:0>

Name: Failing Address

Mnemonic: None

Type: RO

This field logs the value of XMI D<29:0> during the command cycle of a failing transaction.

2.8 XMI Errors

The XMI bus detects all single-bit transmission-related errors on XMI D, XMI F, XMI ID, XMI P, and XMI CNF lines. The XMI protocol permits XMI commanders to recover from all transient memory space read/write transaction errors as well as from most I/O space read/write transaction errors.

2.8.1 Error Conditions

2.8.1.1 Parity Error

To detect single-bit errors, all nodes monitor parity of the bus. Any XMI receiver detecting bad parity ignores the cycle and returns a NO ACK confirmation.

2.8.1.2 Inconsistent Parity Error

Under certain error conditions, some nodes might detect bad parity while others compute proper parity. If the intended target of the transaction computes good parity, then the cycle may be ACKed (and assumed good by the commander), even if other nodes ignore the cycle due to bad parity. For XMI memory-space write transactions, this class of error may result in cache coherency problems due to cached processors failing to perform cache invalidates. For XMI IVINTR transactions, some destinations of the IVINTR transaction may not receive the interrupt. All other XMI transactions are insensitive to this class of error.

2.8.1.3 Transaction Timeout

The XMI protocol specifies that a timeout of 16 milliseconds be used by commanders to detect transaction failure. Responders ensure that transactions do not exceed these timeout values.

- Response Timeout—During an XMI Read, Interlock Read, or IDENT transaction, if a commander does not receive all read responses within a certain number of cycles after the transaction is issued, the transaction is considered to have failed. This does not imply that a responder has "died" since XMI receivers ignore cycles with bad parity and response timeouts can occur as a result of ignored cycles.
- Retry Timeout—An XMI commander needs to reissue an XMI transaction if it receives a NO ACK or a Locked Response. If the commander has not successfully completed the transaction within the timeout period, the transaction has failed.

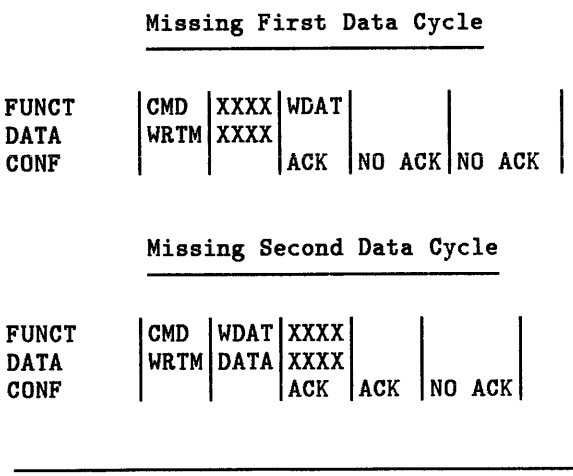
2.8.1.4 Sequence Error

Many transactions require that XMI cycles occur in a certain sequence. When the cycles occur out of sequence, the transaction is in error.

Read, Interlock Read, and IDENT transactions use sequence IDs embedded in the read data responses (GRDn, CRDn, RER—the sequence ID for RER is implicitly 0). The required order for read responses is 0, 0, 0...1, and 0...3 for longword (including IDENT), quadword, octaword, and hexword length transactions, respectively. For example, if the commander detects data returned out of sequence (such as GRD0, GRD2, GRD3), then it NO ACKs the out-of-order read response (GRD2) and the additional read response (GRD3) for that transaction.

Correct sequencing of write transactions is determined by the location of the write data cycles relative to the write command cycle rather than using sequence IDs. The write command cycle and associated write data cycles must occur in contiguous timeslots. If a responder detects missing data cycles in a write transaction, the incorrect cycle (and subsequent write data cycles) are NO ACKed. Figure 2–20 shows examples of failing octaword write transactions.

Figure 2–20 A Failed Octaword Write Transaction



2.8.2 Error Handling

XMI commanders and responders react to error conditons as follows:

- Receivers that detect bad parity ignore the cycle.
- Responders suppress any write transactions containing a sequence or parity error; that is, none of the data at the referenced location is modified as the entire write transaction is ignored.
- Responders receiving a NO ACK confirmation to a read response do not transmit further read responses associated with that transaction within 10 XMI cycles of the NO ACK.
- Memory nodes do not set a lock bit unless all read responses associated with an Interlock Read transaction receive an ACK confirmation.
- Memory nodes do not clear a lock bit unless all write data cycles associated with the Unlock Write Mask transaction are properly received.
- Cached processors detecting an inconsistent parity error either flush their caches or perform a machine check.

2.8.3 Error Recovery

Error recovery involves one or more reattempts of the failed transaction before reporting a hard error. A failed XMI transaction is retried under the following circumstances:

- All transactions receiving a NO ACK confirmation for the command cycle are retried. The NO ACK can result from either a reference to nonexistent memory locations (NXM) or from bus parity errors. Transactions failing the retry are assumed to be to an NXM.
- Failing XMI Write transactions are retried.
- XMI IDENT transactions receiving a response timeout are retried. Since this may result in a lost interrupt vector, the consequences are implemented by software.
- Failing XMI I/O space Write Mask or Unlock Write Mask transactions are retried.
- Failing DWMBA I/O space Read or Interlock Read transactions receiving a response timeout are NOT retried since some I/O devices might have read side effects.

2.8.4 Error Reporting

The XMI bus protocol supports two mechanisms that signal error conditions to processors if normal transaction-level error reporting cannot be used.

Normal transaction-level error reporting mechanisms include NO ACK, Read Error Response (RER), and timeout. The mechanisms that signal error conditions to processors if normal transaction-level error reporting cannot be used are:

- Write error interrupt—This transaction is directed to one or more CPU nodes, resulting in each targeted CPU taking an IPL 29 (decimal) error interrupt. The CPU then identifies the source of the write error interrupt.
- XMI FAULT—When XMI FAULT is asserted, all XMI CPUs take an IPL 29 (decimal) error interrupt .

An example of a write error interrupt is if the DWMBA is unable to complete either an XMI-to-VAXBI windowed write operation or a VAXBI-to-XMI windowed write operation. Then the DWMBA issues a write error IVINTR transaction to the nodes designated in the DWMBA AIVINTR destination register.

3 KA62A CPU Module

This chapter describes the KA62A CPU module, the 32-bit, virtual memory microprocessor for the VAX 6200.

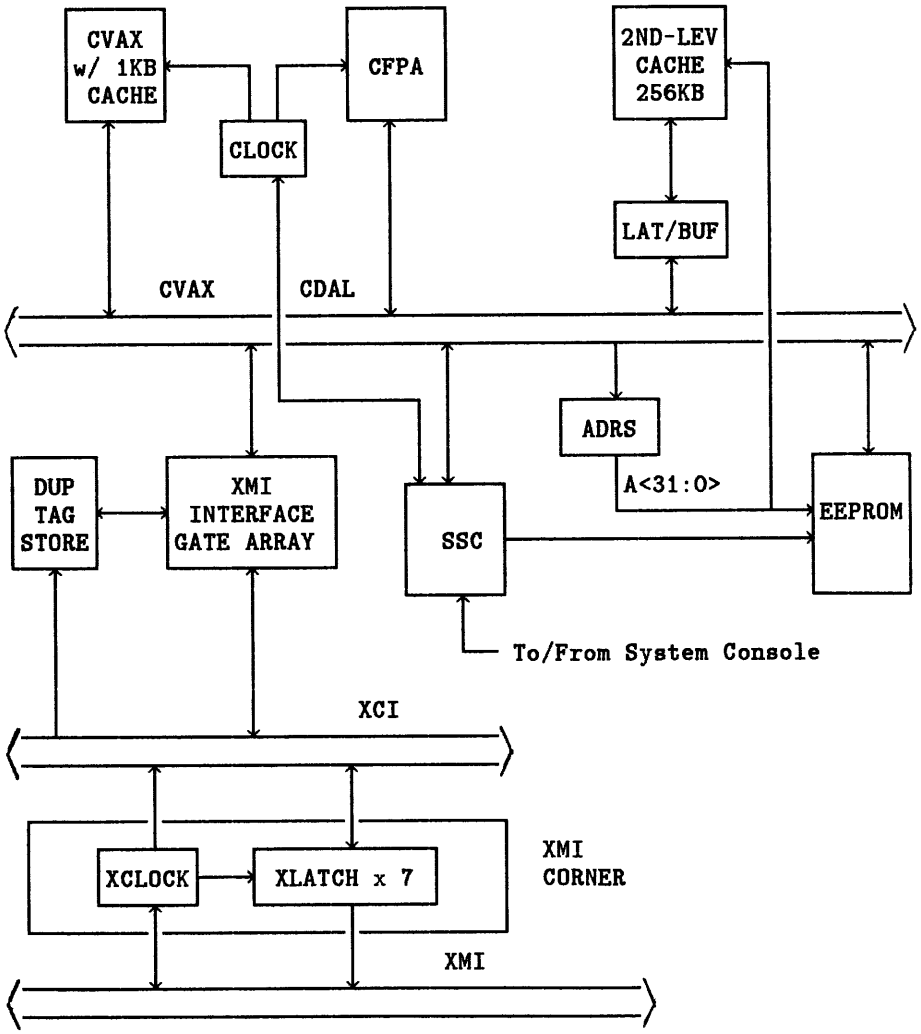
This chapter includes the following sections:

- Module Features
- KA62A Private I/O Address Space Map
- CPU Section of the Module
- Cache Memory
- XMI Corner-to-CPU Interface
- KA62A Registers
- Initialization, Self-Test, and Booting
- Error Handling
- Interprocessor Communication through the Console Program
- Error Handling

3.1 KA62A CPU Module Features

The KA62A CPU module implements a 32-bit, virtual memory microprocessor conforming to the MicroVAX subset including floating-point instructions. Multiple KA62A CPU modules can be installed in a VAX 6200.

Figure 3–1 KA62A CPU Module Block Diagram



The KA62A CPU module includes the following:

1 The CPU section, which contains:

- The CVAX processor chip, which supports the MicroVAX subset of the VAX instruction set and data types. It has full VAX memory management including demand paging and 4 Gbytes of virtual memory. The CPU chip includes the first-level cache for I-stream (instruction) storage only. The first-level cache is one Kbyte, organized with 128 tags. The cache is write-through, two-way associative, and filled eight bytes at a time. The cache provides parity protection on both the tag and data stores.
- The CFPA floating-point accelerator chip, with the MicroVAX subset of the VAX floating-point instruction set and data types.
- The system support chip (SSC), which incorporates in one integrated circuit required functions to support the MicroVAX system environment.
- The clock chip, which includes a VAX standard time-of-year (TOY) clock with battery backup, an interval timer with 10 millisecond interrupts, and two programmable timers.

2 The 256-Kbyte direct-mapped second-level cache, which contains both I- and data-stream (D-stream) data. The second-level cache is write-through and organized with 4096 tags. If a processor read misses an entry in the cache, or if the entry is invalid, the XCP gate array reads the data from main memory. The cache is filled 32 bytes at a time; the first longword read satisfies the processor's read request. The cache provides parity protection on both the tag and data stores.

3 The XCP gate array (XCPGA) chip, which handles the task of interfacing the CDAL bus to the XMI Corner. The XCPGA chip controls a duplicate TAG store that monitors XMI write transactions and filters out all addresses that are not in the second-level cache. The processor performance is only affected by XMI write activity when the write address from another node hits in the cache. When this occurs, the second-level cache block corresponding to the XMI write is invalidated.

4 An XMI interface, which includes:

- An octaword write buffer that decreases bus and memory controller bandwidth needs by packing writes into larger, more efficient blocks prior to sending them to main memory
- Hexword cache fill logic that loads the second-level cache with eight longwords of data on each cache miss
- XMI write monitoring logic that uses a duplicate tag store to efficiently flag write addresses that must be invalidated in the second-level cache
- Full set of error recovery and logging capabilities

5 The console and diagnostics section, which includes:

- A console read-only memory (ROM), which contains the code for initialization, executing console commands, and bootstrapping the system.
- A diagnostic ROM, which contains the power-up self-test and extended diagnostics.
- An electrically-erasable ROM (EEPROM), which contains system parameters and boot code. The parameters can be modified by using the console SET commands. The SSC contains circuits for writing the EEPROM and controlling the console.

3.2 KA62A CPU Module Private I/O Address Space Map

Figure 3–2 shows the private I/O address space map for the KA62A CPU module.

Figure 3–2 KA62A CPU Module Private I/O Address Space Map

Byte Address		Size
2000 0000	CSR1	4 Bytes
2000 0004 2000 3FFF	RESERVED	approx. 256 Kbytes
2004 0000 2007 FFFF	Self-test/Console/Boot Code implemented in two (2) 128KB X 8 PROMs	256 Kbytes
2008 0000 2008 7FFF	Self-test/Console/Boot Code implemented in one (1) 32KB X 8 EEPROM	32 Kbytes
2008 8000 200B FFFF	RESERVED	224 Kbytes
200C 0000 200F FFFF	Non-HALT Protected Self-test/Console/Boot Code implemented in two (2) 128KB X 8 EEPROMs (double-mapped addresses — same physical ROMs as accessed by 2004 0000 to 2007 FFFF)	256 Kbytes
2010 0000 2013 FFFF	RESERVED	256 Kbytes
2014 0000 2014 03FF	SSC CSRs	1 KByte
2014 0400 2014 07FF	SSC Battery-backed Up RAM	1 Kbyte
2014 0800 2100 FFFF	RESERVED	approx. 14.8 Mbytes
2101 0000 2101 FFFF	Interprocessor IVINTR Generation "Virtual" Registers	64 Kbytes
2102 0000 2102 FFFF	Write Error IVINTR Generation "Virtual" Registers	64 Kbytes
2103 0000 217F FFFF	RESERVED	approx. 7.75 Mbytes

3.3 CPU Section of the Module

The KA62A CPU module central processor supports full VAX memory management and the MicroVAX subset of the VAX instruction set and data types. Most of the CPU is implemented in a VLSI chip called the CVAX. The KA62A CPU module floating-point accelerator is implemented in a VLSI chip called the CFPA.

3.3.1 Data Types

The CPU supports the following subset of the VAX data types:

- Byte
- Word
- Longword
- Quadword
- Character string
- Variable-length bit field
- F_floating
- D_floating
- G_floating

Support of the remaining VAX data types is provided by macrocode emulation.

3.3.2 Instruction Set Types

The CPU implements the following subset of the VAX instruction set types in microcode:

- Integer arithmetic and logical
- Address
- Variable-length bit field
- Control
- Procedure call
- Miscellaneous
- Queue
- Character string moves
  - CMPC3/CMPC5
  - LOCC
  - MOVC3/MOVC5
  - SCANC
  - SKPC
  - SPANC
- Operating system support
- F_floating
- D_floating
- G_floating

The CVAX chip provides special microcode assistance to aid the macrocode emulation of the following instruction groups:

- Character string moves not included in microcode
- Decimal string
- CRC
- EDITPC

The following instruction groups are not implented but are emulated by macrocode:

- Octaword
- H_floating
- Compatibility mode instructions

.3.3 Memory Management

The KA62A CPU module implements full VAX memory management. System space addresses are virtually mapped through single-level page tables, and process space addresses are virtually mapped through two-level page tables.

The KA62A CPU module uses a 28-entry, fully associative, translation buffer for caching modified VAX page table entries (PTEs). Each entry stores a modified PTE for translating virtual addresses in either the VAX process space or VAX system space. Each entry is divided into two parts: a 23-bit tag register and a 32-bit PTE register.

The tag register stores the virtual page number (VPN) of the virtual page that the corresponding PTE register maps. The PTE register stores the 21-bit PFN field, the PTE.V bit, the PTE.M bit, and an 8-bit partially decoded representation of the 4-bit VAX PTE PROT field, from the corresponding VAX PTE and a Translation Buffer Valid (TB.V) bit.

During virtual to physical address translation, the contents of the 28 Tag registers are compared with the Virtual Page Number field (bits <31:9>) of the virtual address of the reference. If there is a match with one of the Tag registers, then a translation buffer "hit" has occurred and the contents of the corresponding PTE register is used for the translation.

If there is no match, the translation buffer does not contain the necessary VAX PTE information to translate the address of the reference, and the PTE must be fetched from memory. Upon fetching the PTE, the translation buffer is updated by replacing the entry that is selected by the replacement pointer. Since this pointer is moved to the next sequential translation buffer entry whenever it is pointing to an entry that is accessed, the replacement algorithm is Not Last Used.

Both exceptions and interrupts divert execution from the normal flow of control. An exception is caused by the execution of the current instruction and is typically handled by the current process, such as an arithmetic overflow, while an interrupt is caused by some activity outside the current process and typically transfers control outside the process, such as an interrupt from an external hardware device.

3.3.4 Interrupts

Table 3-1 lists the CPU's use of the 31 interrupt levels that the VAX architecture specifies.

Table 3-1 KA62A CPU Module Interrupts

Interrupt Level	Interrupt Condition
Nonmaskable	CTRL/P typed at the console
	Node HALT bit (XBER<29>) set
1F	Unused
1E	XMI AC LO L assertion
1D (MEMERR) "Hard" Errors	Write Data NO ACK (XBER<20>) set
	Command NO ACK (XBER<15>) set
	All forms of IDENT errors
	CDAL Write Parity Error (CSR2<28>) set
	XMI Write Error Interrupt (XBER<25>) set
	XMI FAULT assertion (XBER<26>) set
1C - 1B	Unused
1A (CRD) "Soft" Errors	Correctable Main Memory Errors (XBER<19>) set (hardware disable supported)
	2nd-level Cache Valid Bit Parity Error (CSR2<31>) set
	2nd-level Cache Tag Parity Error (CSR2<30>) set
	2nd-level Cache Invalidate Queue Overflow (CSR2<29>) set
	Duplicate Tag Store Parity Error (CSR2<26>) set
	Cache Fill Error (CSR2<27>) set
	Corrected XMI CNF Error (XBER<27>) set (hardware disable supported)
	Inconsistent Parity Error (XBER<24>) set
	Parity Error (XBER<23>) set
19 - 18	Unused
17	XMI Level 7 INTR

Table 3-1 (Cont.) KA62A CPU Module Interrupts

Interrupt Level	Interrupt Condition
16	Interval Timer Interrupt[1]
	XMI interprocessor IVINTRs
	XMI Level 6 INTR
15	Console Terminal Interrupts[1]
	Programmable Timer Interrupts
	XMI Level 5 INTR
14	XMI Level 4 INTR
13 - 10	Unused
0F - 01	Software interrupt request

[1]At a given IPL, the priority of interrupts is shown in descending order.

3.3.5 Exceptions

The VAX architecture recognizes six classes of exceptions, as shown in Table 3–2.

Table 3–2 KA62A CPU Module Exceptions

Exception Class	Instances
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
	Translation not valid fault
Operand reference exceptions	Reserved addressing mode fault
	Reserved operand fault or abort
Instruction execution exception	Reserved/privileged instruction fault
	Emulated instruction fault
	Extended function fault
	Breakpoint fault
Tracing exception	Trace fault
System failure exception	Machine-check abort including:
	1. CDAL bus parity errors on demand reads (detected by CVAX chip)
	2. 1st- and 2nd-level cache parity errors on demand reads (detected by CVAX)
	3. Main memory uncorrectable read errors on demand reads (XCPGA asserts ERR)
	4. Nonexistent XMI memory errors on demand reads (XCPGA asserts ERR)
	5. CDAL bus timeout errors (SSC asserts ERR)
	Kernel stack not valid abort
	Interrupt stack not valid abort

3.3.6 Machine Checks

A machine check is an exception that indicates a processor-detected error. Machine checks are taken regardless of the current IPL. The machine check exception vector bits (<1:0>) specify one, or the operation of the processor is UNDEFINED. The exception is taken on the interrupt stack and the IPL is raised to 1F (hex).

Figure 3–3 shows the parameters that are pushed on the stack in response to a machine check. Table 3–3 lists these parameters.

Figure 3–3 The Stack in Response to a Machine Check

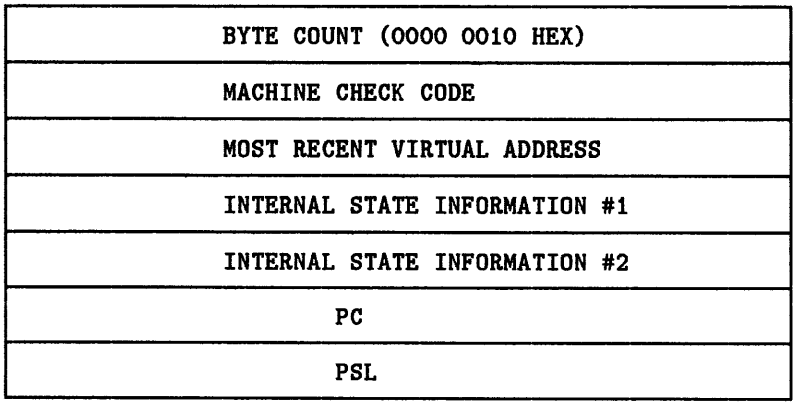


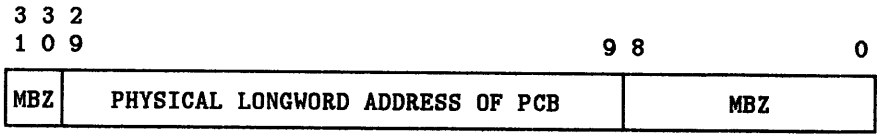
Table 3–3 Machine Check Parameters

Parameter	Value	Description
Machine check code (hex)	1	Floating-point protocol error
	2	Floating-point reserved instruction
	3	Floating-point unknown error
	4	Floating-point unknown error
	5	Process PTE in P0 space during TB miss flows
	6	Process PTE in P1 space during TB miss flows
	7	Process PTE in P0 space during M = 0 flows
	8	Process PTE in P1 space during M = 0 flows
	9	Undefined INT.ID value
	A	Undefined MOVCx state
	80	Memory read error
	81	SCB, PCB, or SPTE read error
	82	Memory write error
	83	SCB, PCB, or SPTE write error
Most recent virtual address	<31:0>	Current contents of VAP register
Internal state information #1	<31:24>	Opcode
	<23:16>	1110, highest priority software interrupt <3:0>
	<15:8>	CADR<7:0>
	<7:0>	MSER<7:0>
Internal state information #2	<31:24>	Most recent contents of SC register <7:0>
	<23:16>	11, state flags <5:0>
	<15:8>	Restart flag, 111, ALU CC flags <3:0>
	<7:0>	Offset from saved PC to PC at time of machine check
PC	<31:0>	PC at the start of the current instruction
PSL	<31:0>	Current contents of PSL

3.3.7 System Control Block (SCB)

The SCB is a page containing the vectors for servicing interrupts and exceptions. The SCB is pointed to by IPR17, the System Control Block Base Register (SCBB). See Figure 3-4.

Figure 3-4 System Control Block Base Register



The system control block format is shown in Table 3-4.

Table 3-4 System Control Block Format

Vector	Name	Type	Param	Notes
00	Unused	–	–	IRQ passive release on other VAXes
04	Machine check	Abort	4	Parameters depend on error type
08	Kernel stack not valid	Abort	0	Must be serviced on interrupt stack
0C	Power fault	Interrupt	0	IPL is raised to 1E
10	Reserved/privileged intstruction	Fault	0	
14	Customer reserved instruction	Fault	0	XFC instruction
18	Reserved operand	Fault/abort	0	Not always recoverable
1C	Reserved addressing mode	Fault	0	
20	Access control violation	Fault	2	Parameters are virtual address, status code
24	Translation not valid	Fault	2	Parameters are virtual address, status code
28	Trace pending (TP)	Fault	0	
2C	Breakpoint instruction	Fault	0	
30	Unused	–	–	Compatibility mode in other VAXes
34	Arithmetic	Trap/ fault	1	Parameter is type code
38 – 3C	Unused	–	–	–
40	CHMK	Trap	1	Parameter is sign-extended operand word
44	CHME	Trap	1	Parameter is sign-extended operand word
48	CHMS	Trap	1	Parameter is sign-extended operand word
4C	CHMU	Trap	1	Parameter is sign-extended operand word
50	Unused	–	–	–
54	Corrected read data	Interrupt	0	IPL is 1A (CRD L)

Table 3–4 (Cont.) System Control Block Format

Vector	Name	Type	Param	Notes
58 – 5C	Unused	–	–	–
60	Memory error	interrupt	0	IPL is 1D (MEMERR L)
64 – 74	Unused	–	–	–
78	SSC programmable timer 0	interrupt	0	Reserved for DEC use only
7C	SSC programmable timer 1	interrupt	0	Reserved for DEC use only
80	Interprocessor interrupt	Interrupt	0	IPL is 16
84	Software level 1	Interrupt	0	–
88	Software level 2	Interrupt	0	Ordinarily used for AST delivery
8C	Software level 3	Interrupt	0	Ordinarily used for process scheduling
90 – BC	Software levels 4–15	Interrupt	0	–
C0	Interval timer	Interrupt	0	IPL is 16 (INTTIM L)
C4	Unused	–	–	–
C8	Emulation start	Fault	10	Same mode exception, FPD = 0; parameters are opcode, PC, specifiers
CC	Emulation continue	Fault	0	Same mode exception, FPD = 1;no parameters
D0 – F4	Unused	–	–	–
F8	Console receiver	Interrupt	0	IPL is 15
FC	Console transmitter	Interrupt	0	IPL is 15
10D – FFFC	XMI/VAXBI device vectors	Interrupt	0	DWMBA appends bits <15:9> to VAXBI vectors that have <13:9> = 0 before transmission on the XMI

3.3.8 Hardware Restart Sequence

The CPU enters the hardware restart process upon the occurrence of one of several events:

- Following an XMI power-up sequence.
- Following an XMI system reset sequence, an "emulated" power-up sequence that is initiated by asserting the XMI RESET L line. This can be accomplished by writing to IPR55 (IORESET).
- When node reset (XBER<30>) is set from the XMI.
- When HALTs are enabled and a CTRL/P is generated by the console or node HALT (XBER<29>) is set from the XMI.
- When the hardware or kernel software environment becomes severely corrupted and the CPU cannot continue normal processing.
- When a HALT instruction is executed in kernel mode.

When the hardware restart process begins, the CPU executes a microcode restart sequence and passes control to console code beginning at physical address 2004 0000 (hex). The current value of the PC is stored in IPR42 (SAVPC). The PSL, MAPEN<0>, and the restart code are saved in IPR43 (SAVPSL). The current stack pointer is saved in the appropriate internal register. The PSL is set to 041F 0000 (hex), and the current stack pointer is loaded from the interrupt stack pointer. The restart process sets the initial state of the CPU.

3.3.9 CPU References

All references by the CPU are classified as request instruction-stream read references, demand data-stream read references, or write references. Request reads are generated when the data is not immediately needed by the CPU, while demand reads are generated when the data is immediately needed by the CPU.

Request read errors do not affect program flow; however, demand read errors cause a machine check abort.

Instruction-Stream Read References. The CPU has an instruction prefetcher with a 12-byte (3 longwords) Instruction Prefetch Queue (IPQ) for prefetching program instructions from either cache or main memory. Whenever there is an empty longword in the IPQ and the prefetcher is not halted due to an error, the instruction prefetcher generates an aligned longword, request instruction-stream (I-stream) read reference.

Data-Stream Read References. Whenever data is immediately needed by the CPU to continue processing, demand data-stream (D-stream) read references are generated on operand, page table entry (PTE), system control block (SCB), and process control block (PCB) references. When interlocked instructions, such as Branch on Bit Set and Set Interlock (BBSSI), are executed, a demand D-stream Read-Lock reference is generated.

Write References. Whenever data is stored or moved, a Write Reference is generated.

Since the CPU does not impose any restrictions on data alignment other than the aligned operands of the ADAWI and Interlocked Queue instructions and since memory can only be accessed one aligned longword at a time, all data read references and write references are translated into an appropriate combination of masked and unmasked aligned longword read references. If the required data is a byte, a word within a longword, or an aligned longword, then a single aligned longword demand D-stream read reference or write reference is generated. If the required data is a word that crosses a longword boundary or an unaligned longword, then two successive aligned-longword demand D-stream read references or write references are generated. Data larger than a longword is divided into a number of successive aligned-longword demand D-stream reads, with no optimization, or writes.

3.3.10 System Support Chip (SSC)

The SSC incorporates in one integrated circuit the following required functions to support the MicroVAX system environment:

- ROM support by performing ROM address decoding and providing chip select signals. It returns RDY to the CVAX chip for ROM access and performs byte packing of ROM data to longwords for the CVAX.
- 1 Kbyte of internal, battery-backed-up RAM
- Console support by providing a VAX SRM-compatible terminal UART that supports a full set of baud rates and BREAK-detect functions.
- A 100 Hz interval timer.
- A battery-backed-up 32-bit counter time-of-year (TOY) clock.
- Two programmable address strobes, used by the KA62A CPU module to access CSR1 and to write the EEPROM.
- A 4-bit general purpose output port used to control the console line multiplexer and to drive a status LED.
- Two programmable timers and a CDAL bus timeout register.

3.3.11 EEPROM

The EEPROM stores parameters for initialization of the KA62A CPU module and patches to the ROM code which does VAX standard console emulation, module self-tests, and boot code.

The EEPROM can be read with byte, word, or longword references and is coordinated by the SSC. If the READ is word or longword, the SSC reads a byte at a time from the EEPROM and returns the full word or longword to the CVAX chip.

Console (initialization) code sets the ROM Size field in the SSC Configuration Register (SSCR<22:20>) to the 1-Mbyte block 2004 0000 to 2013 FFFF (hex). The halt protect field (SSCR<18:16>) is set to map the 512-Kbyte block from 2004 0000 to 200B FFFF (hex). This double maps the ROM and EEPROM to provide halt-protected and unprotected images of the contents. Writes to the ROM portion of this address space result in a machine check.

Console code also sets the EEPROM Address Decode Mask Register (EEADMR) and the EEPROM Base Address Register (EEBADR).

Writes to the remainder of the EEPROM address space must follow these rules:

- Write only a byte of data at a time. The write data must be driven on CDAL<7:0>.
- The botton two address bits for the EEPROM are provided by CSR1<1:0> (EEADR). These bits must be set to the proper state before the EEPROM write is issued.
- A front panel switch provides write enable protection for the EEPROM by controlling the XMI UPDATE EN H line. The state of this line is read as SSCCR<5> (FPEEUE). Console code confirms that this bit is set before updating the EEPROM.
- EEPROM updates are controlled by console software. Console code sets SSCCR<6:4>, the EEPROM enable field, to 101 just before the write and then clears the field immediately following the update.
- Console code delays the return prompt until an internal counter expires to prevent accesses immediately after a write.

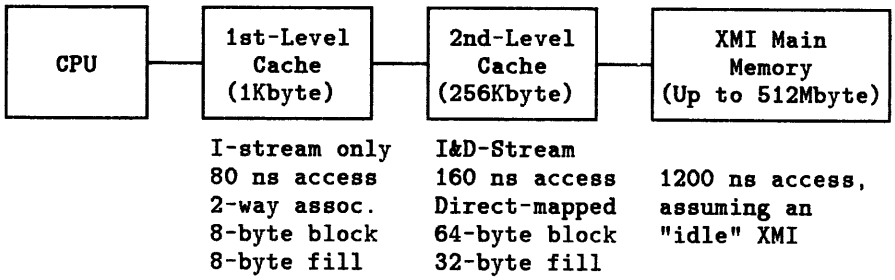
3.3.12 Floating-Point Accelerator

The KA62A CPU module floating-point accelerator (FPA) is implemented in a VLSI chip called the CFPA. The FPA processes F_floating, D_floating, and G_floating format instructions and accelerates the execution of MULL, DIVL, and EMUL integer instructions. The FPA supports byte, word, longword, quadword, F_floating, D_floating, and G_floating data types. The H_floating data type is not supported but is emulated by macrocode.

3.4 Cache Memory

The KA62A CPU module has a two-level cache to maximize CPU performance. The first-level cache is implemented in the CVAX chip. The second-level cache is implemented on the KA62A CPU module using 64K x 4-bit static RAMs.

Figure 3–5 Simplified Block Diagram of KA62A CPU Module Memory



3.4.1 First-Level Cache

The first-level cache is 1-Kbyte, two-way associative, and write-through with an 80 ns cycle time. CPU read references access one longword at a time, while CPU writes can access as little as one byte at a time. A single parity bit is generated, stored, and checked for each byte of data and each tag. Only I-stream references to VAX memory space are stored in the first-level cache.

3.4.1.1 First-Level Cachable References

Any reference stored by the first-level cache is called a "first-level cachable reference" ("FL cachable reference"). For cache coherency to be maintained, the first-level cache is configured by initialization code to store only I-stream CPU read references made to VAX memory space (physical address<29> = 0) by writing CADR<5:4> = 10.

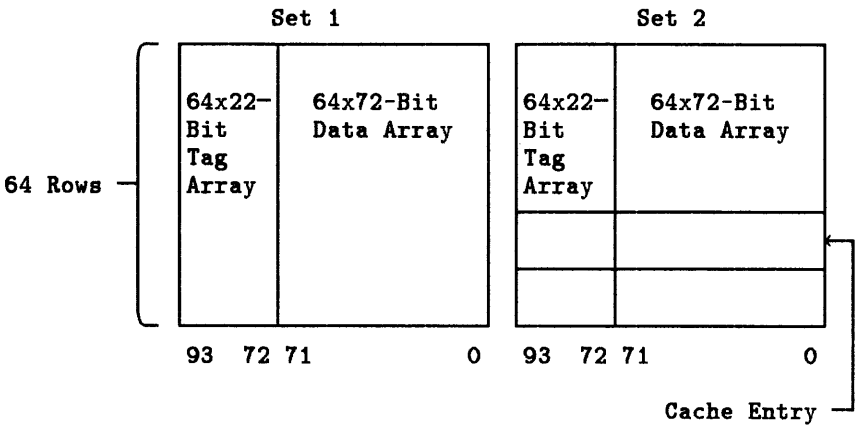
The CPU generates CDAL references, depending on the reference type, as follows:

- Whenever the CPU generates a non-FL cachable reference, a single longword reference of the same type is generated on the CDAL bus.
- Whenever the CPU generates an FL cachable reference that is already stored in the first-level cache, no reference is generated on the CDAL bus.
- Whenever the CPU generates an FL cachable reference that is not stored in the first-level cache, a quadword transfer is generated on the CDAL bus. On the KA62A CPU module, all FL cachable references consist of two indivisible longword transfers, the first being a Request I-stream Read (prefetch) and the second being a Request I-stream Read (fill).

3.4.1.2 First-Level Cache Organization

The first-level cache is divided into two independent storage arrays called Set 1 and Set 2. Each set contains a 64-row x 22-bit tag array and a 64-row x 72-bit data array. The organization of the two sets is shown in Figure 3–6.

Figure 3–6 First-Level Cache Organization



A row within a set corresponds to a cache entry, resulting in 64 entries in each set and 128 entries in the entire cache. Each entry contains a 22-bit tag block and a 72-bit (8-byte) data block. Figure 3–7 shows the format of a cache entry.

A tag block consists of a parity bit, a valid bit, and a 20-bit tag, as shown in Figure 3–8.

A data block consists of eight bytes of data, each with an associated parity bit. The total data capacity of the cache is 128 8-byte blocks (1024 bytes). A data block is organized as shown in Figure 3–9.

Figure 3–7 Cache Entry

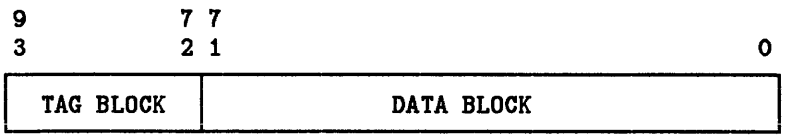


Figure 3–8 Tag Block

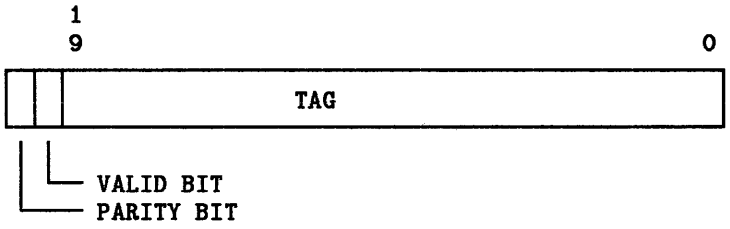
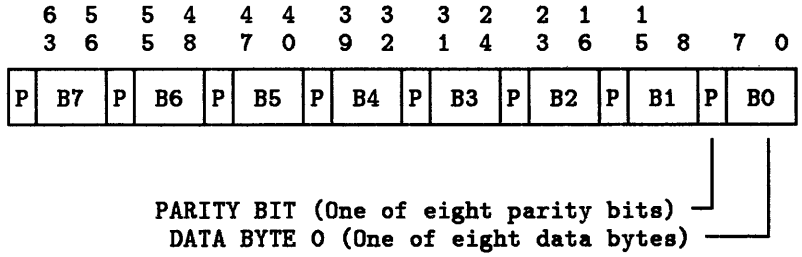


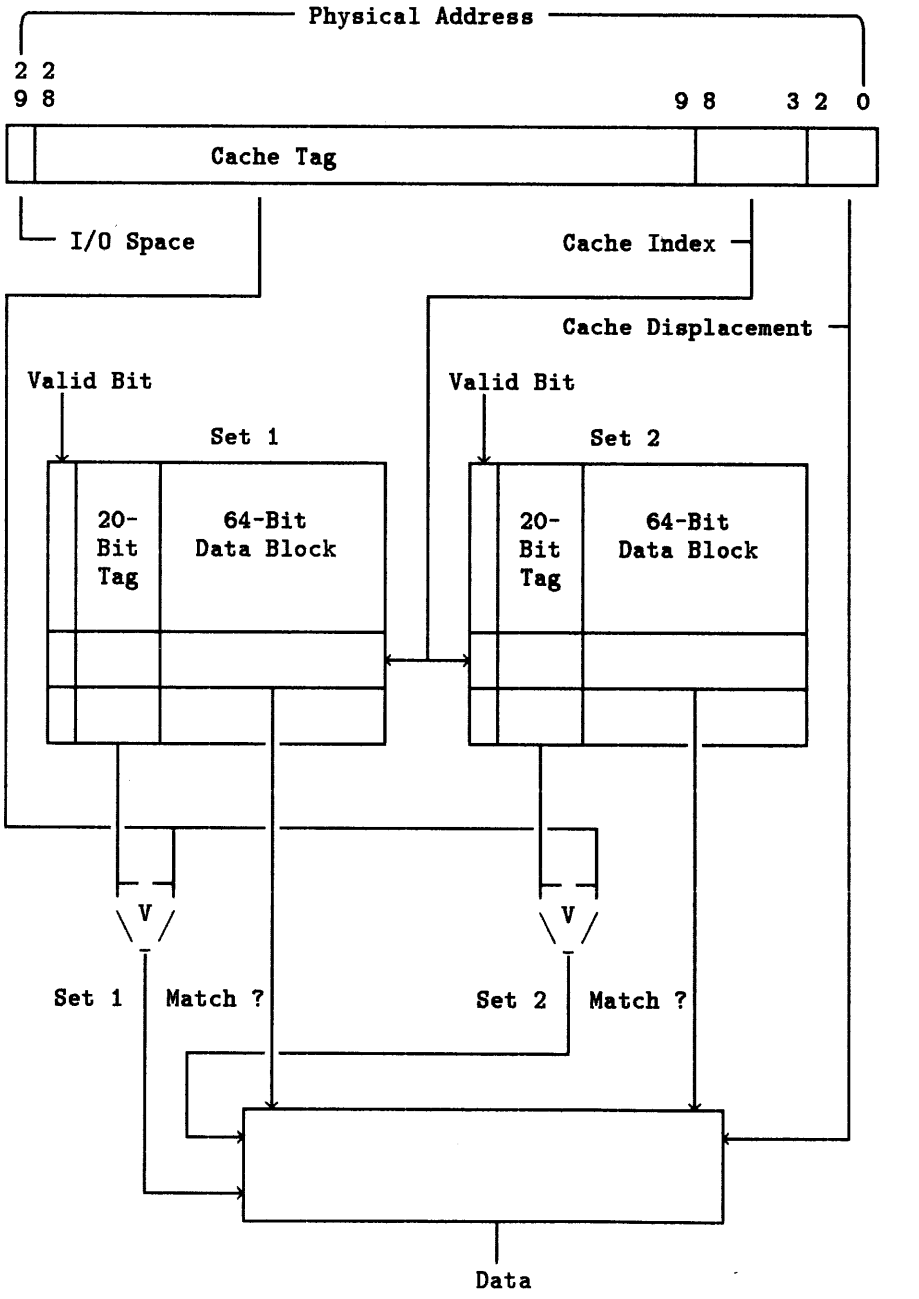
Figure 3–9 Data Block



3.4.1.3 First-Level Cache Address Translation

Whenever the CPU requires I-stream data, the first-level cache is checked to determine if the referenced location is stored there.

Figure 3–10 First-Level Cache Address Translation



The first-level (FL) cache is checked by translating the physical address (see Figure 3-10) as follows:

- On non-FL cachable references, the reference is never stored in the cache, so a first-level "miss" occurs and a single longword reference is generated on the CDAL bus.
- On FL cachable references, the physical address must be translated to determine if the contents of the referenced location is resident in the cache. The Cache Index field, bits<8:3> of the address, is used to select one of the 64 rows of the cache, with each row containing a single entry from each set. The Cache Tag field, bits<28:9> of the physical address, is then compared to the Tag Block of the entry from both sets in the selected row.
- If a match occurs with the Tag Block of one of the set entries, and the valid bit within the entry is set, the contents of the referenced location is contained in the cache and a cache "hit" occurs. On a cache hit, the Set Match Signals generated by the compare operation select the data block from the appropriate set. The Cache Displacement field, bits<2:0> of the physical address, is used to select the byte(s) within the block. No CDAL bus transfers are initiated on CPU references that "hit" the first-level cache.
- If no match occurs, then the contents of the referenced location is not contained in the cache and a cache "miss" occurs. The data must be obtained from either the second-level cache or from XMI memory. If the reference is cachable, then a quadword transfer is initiated on the CDAL bus. If the reference is not cachable, then a single longword transfer is initiated on the CDAL bus.

3.4.1.4 First-Level Cache Data Allocation

FL cachable references that "miss" the first-level cache cause a quadword read to be initiated on the CDAL bus. When the requested quadword is supplied by either the second-level cache or the main memory controller, the requested longword is passed to the CPU, and a data block is allocated in the cache to store the entire quadword.

Since the cache is two-way associative, there are two data blocks (one in each set) that can be allocated to a given quadword. These two data blocks are determined by the Cache Index field of the address of the quadword, which selects a unique row within the cache. Selection of a data block within the row (that is, set selection) for storing the new entry is random.

3.4.1.5 First-Level Cache Behavior on Writes

On CPU-generated write references, the first-level cache is "write through." All CPU write references that "hit" the first-level cache cause the contents of the referenced location in main memory to be updated as well as the copy in the cache.

3.4.1.6 First-Level Cache Coherency

VAX architecture requires that a Return from Exception or Interrupt (REI) instruction be executed between procedure instructions and writable data. Since the KA62A CPU module FL-cache is operated in "I-stream-only" mode, the coherency of the first-level cache is maintained by invalidating the cache on every REI instruction.

CAUTION: **If the following rules are not followed, it is possible that stale data will be read from the cache, causing UNPREDICTABLE results.**

In order for the I-cache to remain coherent, the following requirements are placed on software:

1. A native mode procedure may not write data that is to be subsequently executed as an instruction without an intervening REI instruction being executed.
2. A new process is invoked by executing a LDPCTX (Load Process Context) instruction. A LDPCTX must be followed by an REI instruction.
3. When using the operating system to generate I/O, the process must execute a CHMK (Change Mode to Kernel) instruction. An REI will be generated when the operating system returns to user mode after initiating the I/O request. The process must assure that the data is not accessed until the I/O has completed.
4. When a privileged process does not use the operating system to initiate I/O but initiates I/O directly, the privileged process must perform an REI after initiating the I/O request. The process must assure that the data is not accessed until the I/O has completed.

3.4.1.7 First-Level Cache Control

The first-level cache is *enabled* by writing 0000 00EC to CADR (IPR37). It is *disabled* by writing 0000 00CC to CADR. Any writes to CADR cause the first-level cache to flush.

3.4.1.8 First-Level Cache Error Detection

Both the tag and data arrays in the first-level cache are protected by parity. Each 8-bit byte of cache data and the 20-bit tag are stored with an associated parity bit. The Valid bit in the tag is not covered by parity. Odd data bytes are stored with odd parity, and even data bytes are stored with even parity. The tag is stored with odd parity. The stored parity is valid only when the Valid bit associated with the cache entry is set. Tag and data parity (on the entire longword) are checked on read references that hit the cache, while only tag parity is checked on CPU write references that hit the cache.

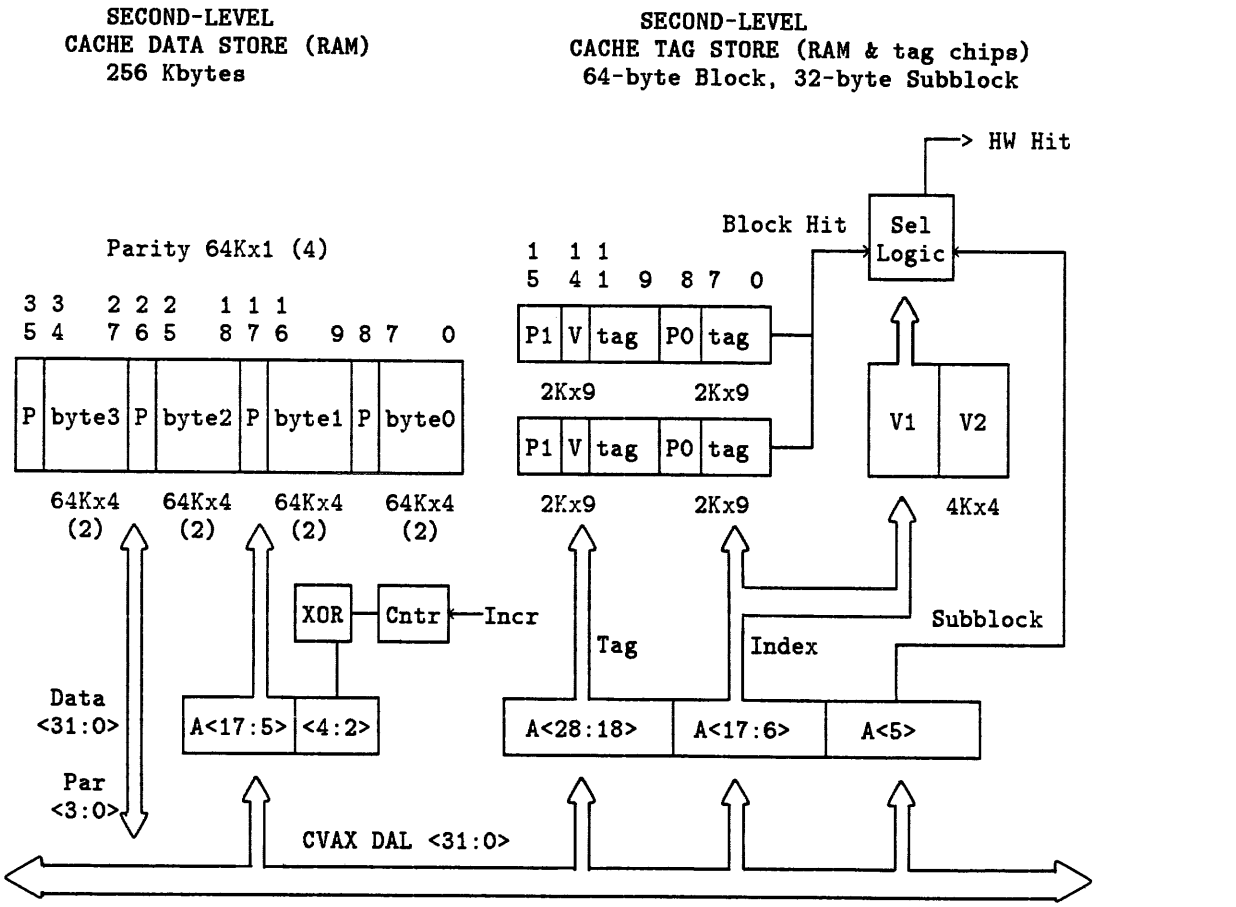
The action taken following the detection of a cache parity error depends on the reference type:

- During a request I-stream reference, the entire cache is flushed, the cause of the error is logged in MSER<1:0> (DAT and TAG), the prefetch is halted, but no machine check abort occurs. The cache remains enabled.
- During a demand D-stream reference, the entire cache is flushed, the cache is disabled (CADR is cleared), the cause of the error is logged in MSER<5,3:0>, and a machine-check abort is initiated.
- During a request D-stream reference, the entire cache is flushed, the cause of the error is logged in MSER<3:0>, but no abort occurs and the cache remains enabled.

3.4.2 Second-Level Cache

The second-level cache is a 256-Kbyte, direct-mapped, write-through cache with a 160-ns cycle time. All VAX memory space read references made by the CPU except Interlock Reads, including both I-stream and D-stream references, are stored in the second-level cache.

Figure 3-11 Second-Level Cache Block Diagram



3.4.2.1 Second-Level Cache Description

The 256-Kbyte, direct-mapped, second-level cache supplements the 1-Kbyte first-level cache that is internal to the CVAX chip. The second-level cache is located on the CDAL bus and is partitioned into 64-byte blocks that are subdivided into two 32-byte (hexword) subblocks. Both the data and tag stores are protected with parity. An entire 64-byte block is invalidated on a device write to memory. A duplicate tag store is maintained by the XCPGA interface to reduce unnecessary CVAX invalidate traffic.

The second-level cache memory array is a direct mapped 64K x 36-bit array which stores up to 256 Kbytes of data. The data is physically stored in eight 64K x 4-bit and four 64K x 1-bit static MOS RAM chips. A parity bit is included with each byte.

The cache tags are stored in four 2K x 9 cache address comparator chips which contain 4096 tag entries. This write-through cache is updated if there is a cache hit during a write transaction. If the longword being written into memory has not been cached, only main memory is written; that is, there are no "write allocates."

Each of the 4096 tag entries of two hexwords each (64 bytes per block) has two valid bits stored with each tag in the tag store to specify the validity of the two 32-byte hexwords in that block.

Whenever an Invalidate transaction occurs on the XMI, or when an XMI memory write transaction is initiated by another node, the duplicate tag store performs a tag lookup. If the data for that location is cached, then the duplicate tag store logic immediately clears the appropriate valid bit of the cache tag and generates a second-level cache invalidate request. An XMI quadword write to a cached location in XMI memory results in an entire 64-byte block being invalidated in the cache.

The 16-bit second-level data cache address lines directly address the data within the cache memory array. The data cache address lines are driven with the address latched for the CDAL lines as shown in Figure 3–12. During cache fill operations, they are driven by latched CDAL lines as shown in Figure 3–13.

Figure 3–12 Cache Address Line Contents During a Cache Read

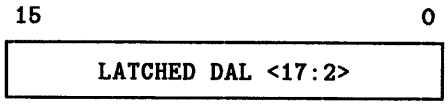
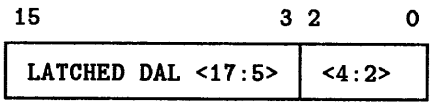


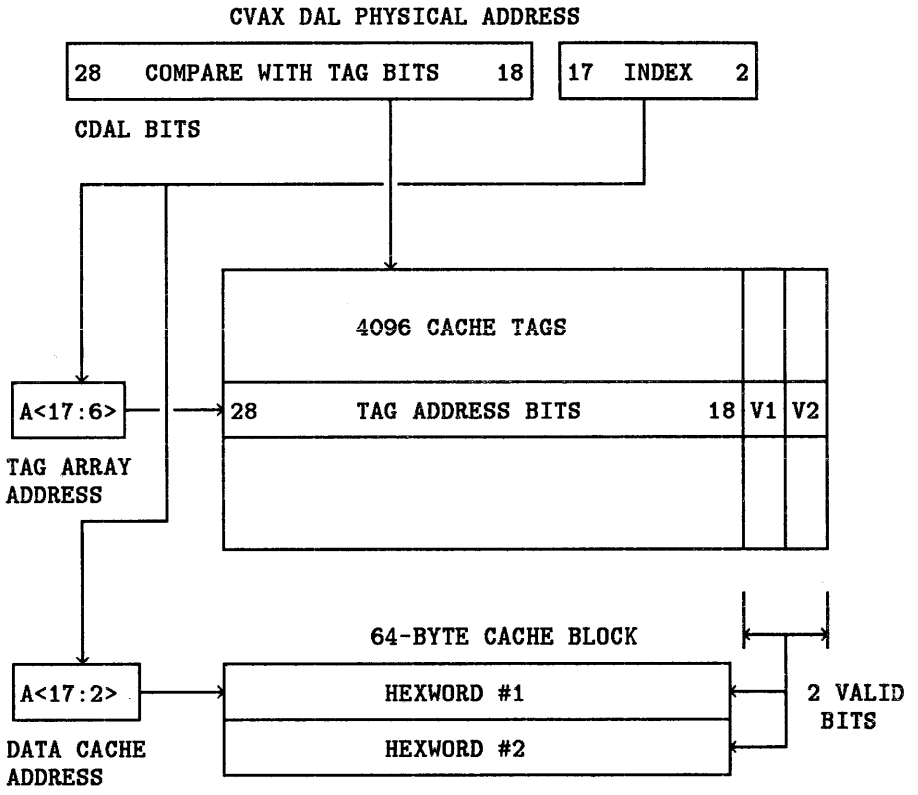
Figure 3–13 Cache Address Line Contents During a Cache Fill



Bits<2:0> are XORed with a straight 3-bit up counter to select the longword within the eight-longword cache allocate block. These bits always start with the addressed longword, then are wrapped within a quadword, and the quadwords wrapped within an octaword to fill the hexword subblock. For example, if the bits initially address 0228, they will wrap around in the following order: 0228, 022C, 0220, 0224, 0238, 023C, 0230, 0234.

Figure 3–14 shows how the lower bits of the CDAL physical address are used to access the cache tag addresses that are compared with the physical address on the CDAL bus. The CDAL address bits are also used to drive the data store address lines for addressing the data cache RAMs.

Figure 3–14 Second-Level Cache Addressing



Each of the 4096 entries in the tag store contains an 11-bit tag address, a valid bit, and a parity bit, combined with a separate RAM containing two valid bits. There is a tag address for each 64-byte block within the cache data RAMS, and a (logical) valid bit that is actually two bits to support single-bit error detection for each 32-byte hexword within the block. The cache tag array is addressed by the physical address from the CDAL, and a comparator generates a hit signal if the data is both resident and valid within the cache data RAMs.

A CVAX memory read which results in a second-level cache miss will cause the CDAL/XMI interface to begin a hexword read transaction to update the cache. The first quadword fetched contains the longword (quadword, if an I-stream request) requested by the CPU; the remaining seven longwords (six longwords, if an I-steam request) comprise the cache fill of the second-level cache only. During memory writes, a cache hit results in both the cache and the main memory being written. This is controlled by the second-level cache logic, which inhibits the write enables to the cache RAMs if the write location was not cached.

3.4.2.2 Controlling the Second-Level Cache

The second-level cache is controlled by the Control and Status Register 1 (CSR1).

The second-level cache is *flushed* by the following sequence:

1 Perform BIT SET of FMISS (CSR1<18>) and FCI (CSR1<20>)

2 Perform BIT CLEAR of FCI (CSR1<20>)

The second-level cache is *enabled* by first flushing the cache (above) and then performing BIT CLEAR of FMISS (CSR1<18>).

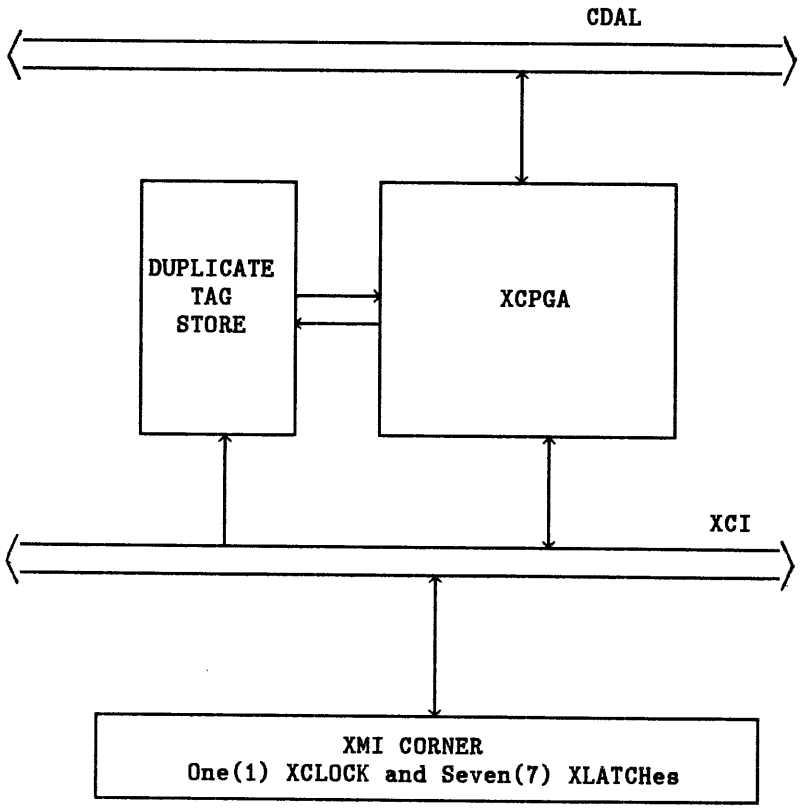
CAUTION: **The second-level cache must always be flushed immediately before enabling.**

It is *disabled* by performing BIT SET of FMISS (CSR1<18>).

3.5 XMI Corner-to-CPU Interface

The KA62A CPU module's XMI Corner is a predefined interface to the XMI bus. Refer to Chapter 2 for a discussion of the XMI.

Figure 3–15 XMI Corner-to-KA62A CPU Module Interface



The KA62A CPU module generates the following XMI transaction types:

- Hexword memory reads
- Quadword memory interlock reads
- Quadword memory write masks
- Octaword memory write masks
- Quadword memory unlock write masks
- Longword I/O reads
- Longword I/O interlock reads
- Longword I/O write masks
- Longword I/O unlock write masks
- Write error IVINTRs
- Interprocessor IVINTRs
- IDENTs (in response to CVAX interrupt acknowlege)

The KA62A CPU module responds to the following XMI transactions:

- Longword nodespace reads
- Longword nodespace write masks
- Interrupts

In addition, the KA62A CPU module monitors all memory writes for cache invalidates.

The duplicate tag store logic and the XCPGA chip provide the functionality required to interface the CVAX chip to the XMI Corner.

Table 3-5 Mapping of CVAX Operations to XMI Transactions

CVAX Operation	Resulting XMI Transaction[1]
Memory Space References	
Read (misses both caches)	Hexword Read
Interlock Read (forced to miss both caches)	Quadword Interlock Read
Write Mask	No XMI transaction generated. Data is loaded in the write buffer. If this is a write buffer miss, then the "old" write buffer data is flushed to main memory with either a Quadword or an Octaword Write Mask.
Unlock Write Mask (forced to miss the write buffer)	Quadword Unlock Write Mask
I/O Space References	
Read (forced to miss both caches)	Longword Read
Interlock Read (forced to miss both caches)	Longword Interlock Read.
Write Mask (forced to miss write buffer)	Longword Write
Unlock Write Mask (forced to miss write buffer)	Longword Unlock Write Mask
Miscellaneous References	
Interrupt Acknowledge	XMI IDENT (assuming that an XMI interrupt is pending and no SSC (only for IPL 15) or IP IVINTR (IPL 16) interrupts are pending)
I/O Space Write to IVINTR Generation Space	XMI IVINTR

[1]Under certain conditions defined in Section 3.5.2, the KA62A CPU module may be required to flush its write buffer prior to performing this resulting XMI transaction.

Table 3–6 Detailed CVAX Read Operation to XMI Map

Request	Command	CDAL Length	CSR1:FMISS	XMI Transaction	2nd-Level Cache
I-Request	Read or Read-Modify	LW	X	HW Read	Fill
I-Request	Read or Read-Modify	QW	X	HW Read	Fill
D-Demand	Read or Read-Modify	LW	X	HW Read	Fill
D-Demand	Read or Read-Modify	QW	–	Illegal	–
D-Demand	Read Lock	LW	X	QW Read	No Fill
D-Demand	Read Lock	QW	–	Illegal	–
I-Request	Read Lock	X	–	Illegal	–

3.5.1 The XCPGA Chip

The XCPGA is a gate array that serves as the interface between the XMI bus and the KA62A CPU module's CDAL bus.

Figure 3–16 XCPGA Block Diagram

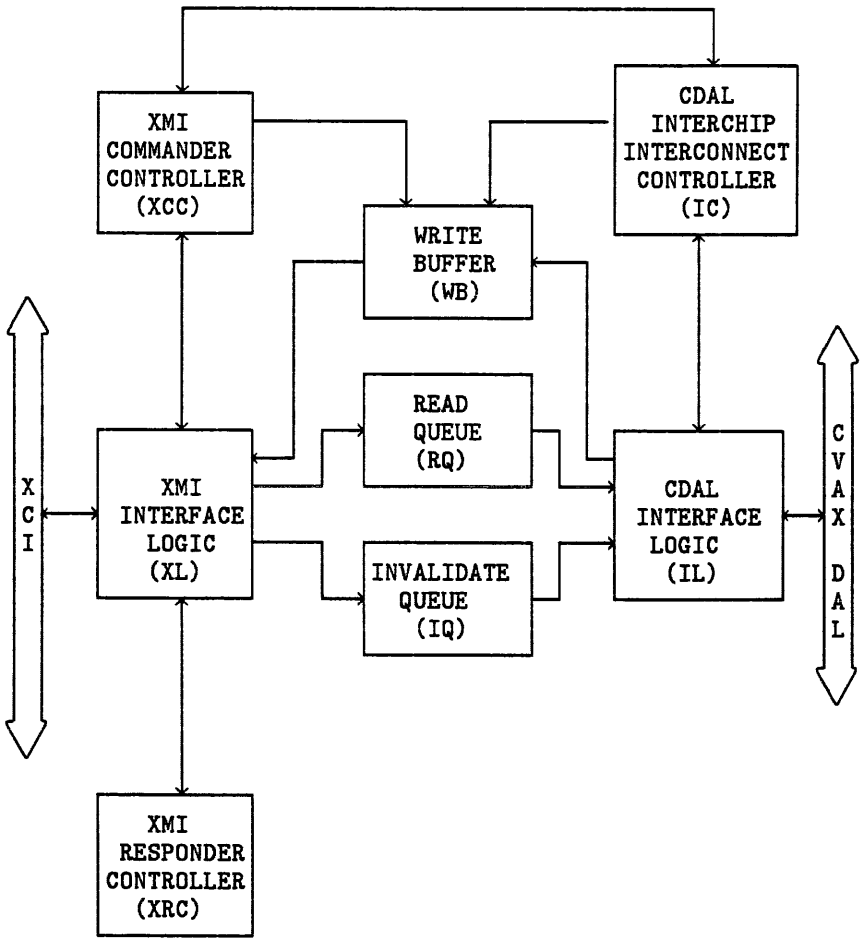


Figure 3-16 is a block diagram of the XCPGA chip. The following is a description of each block:

The **XMI Commander Controller (XCC)** performs the control functions of the XMI commander. These consist of bus arbitration, issuance of command/address and data, and control of returning read data.

The **XMI Interface Logic (XL)** is the data path logic needed to interface to the XMI. It contains the 64-bit input and output registers and multiplexers. It also contains all XMI registers and the XMI interrupt and invalidate support logic.

The **XMI Responder Controller (XRC)** performs the control functions of the XMI responder. These consist of CSR reads and writes.

The **Invalidate Queue (IQ)** stores up to eight invalidate addresses to be sent to the external cache located on the CDAL bus.

The **Read Queue (RQ)** stores up to four quadwords from each read command issued to the XMI. The queue is unloaded, one longword at a time, for placement on the CDAL bus.

The **CDAL Interchip Interconnect Controller (IC)** performs the control functions of the interface to CVAX chip. It handles both master and slave functions.

The **CDAL Interchip Interface Logic (IL)** is the data path logic needed to interface to the CDAL bus. It also implements the 32 interrupt-pending bits.

The **Write Buffer (WB)** is used to combine longword writes from the CVAX to octaword write transactions, reducing XMI bus traffic.

3.5.2 The Write Buffer

The Write Buffer (WB) is a single-entry, double octaword write buffer used to combine longword writes from the CVAX to octaword write transactions, reducing XMI bus traffic. The write buffer also causes fewer bytes to be written to memory by allowing the most recent write reference to the same location to overwrite earlier ones. Read requests bypass the write buffer if the address does not fall within the hexword boundary of the presently active octaword. The write buffer accumulates data in one octaword buffer until a memory write address falls outside the natural octaword boundary. Then the second octaword buffer starts to fill while the first is emptied with an XMI write transaction.

The KA62A CPU module automatically flushes the write buffer in response to the following conditions:

1. In response to a write that misses the currently active write buffer. The current write buffer is flushed while the new write is accepted by the alternate buffer.
2. Before performing an XMI I/O space read or write reference, except for XMI private space references. However, writes to IVINTR generation space do cause a flush of the write buffer.
3. Before performing an Interlock Read or Unlock Write reference.
4. Before issuing an XMI read to a hexword location that includes the data contained in the write buffer. The write buffer contents are flushed to main memory and then the XMI read is issued. Reads that "miss" the write buffer do not force a write buffer flush.
5. Following the assertion of the CVAX Clear Write Buffer (CWB) signal. The XCPGA requests the CDAL bus and, when granted, will flush the write buffer to main memory. MEMERR asserts if the write fails on the XMI.

   The CWB signal asserts:

   - At the start of instructions or sequences that can change processor state: CHMx; REI; and the start of interrupts, exceptions, or aborts (including machine check, BPT, and so forth).
   - As part of instructions or sequences that can change context: end of LDPCTX; end of SVPCTX.
   - As part of instructions or sequences involved in error recovery: write to MAPEN; write to CADR; write to MSER.

3.5.3 Duplicate Tag Store

A duplicate tag store is located on the multiplexed XCI bus. It contains a duplicate copy of the 4096 tag entries in the second-level cache located on the CDAL bus. The duplicate tag store tracks the primary tag store on allocates by monitoring XMI read transactions. Whenever an XMI memory space read is initiated by this node (an XMI read has to occur whenever a second-level cache fill is performed), it allocates the cache block that corresponds to the read address.

The duplicate tag store also monitors all XMI write transactions and performs a duplicate tag store lookup. If a "hit" occurs and the write was not from this node, then the tag is invalidated and the address is loaded into an 8-entry invalidate queue implemented in the XCPGA. Cache invalidates are not performed in response to a KA62A CPU module's own writes since the write-through second-level cache always contains the most recent data. A KA62A CPU module can be forced to look up and conditionally invalidate data on all XMI memory writes (including those generated by itself) by setting CSR2<10>, the Enable Self-Invalidates (ESI) bit.

When an entry has been loaded into the invalidate queue, the CDAL interface logic arbitrates for the CDAL bus and performs an invalidate of the full 64-byte block in which the write address was located. The use of a duplicate tag store reduces CDAL traffic to only necessary invalidate transactions. After performing an invalidate, the XCPGA checks for any additional invalidates that may have accumulated while the previous invalidate was being serviced. If another invalidate request exists, then the XCPGA services it prior to releasing the CDAL bus.

The CVAX chip provides the KA62A CPU module an opportunity to be granted the CDAL bus between every bus operation to perform an invalidate, ensuring a "no stale data" race condition with the invalidate logic. Since it is possible for the XMI bus to issue writes quickly enough to overflow the KA62A CPU module's invalidate queue, the second-level cache is disabled (CRS1<18> (FMISS) is set), an error bit (CSR2<29> (IQO)) is set, and a soft-error interrupt (Level 1A) is generated. While the IQO bit is set, the invalidate queue is held cleared and FMISS stays set. Cache Disable is also generated by CRS2<31,30,26> (VPE, TPE, and DTPE) and XBER<4> (IPE).

The soft-error interrupt routine that handles the IQO error must do the following to return the system to normal operation:

1. Flush the second-level cache.
2. Clear the IQO bit.
3. Enable the second-level cache.

3.5.4 XMI Interrupt Operation

The XMI has an INTR and an IDENT command. Only I/O devices can generate interrupts to interrupt one or more CPU nodes, as designated by a destination mask.

The KA62A CPU module's XMI receiver logic monitors each XMI cycle. If it detects an interrupt command targeted to its node ID, it sets the interrupt-pending bit corresponding to the interrupt level (IPL 17, 16, 15, or 14) and the interrupting node's node ID (E, D, C, B, 4, 3, 2, or 1). The interrupt logic then posts an interrupt request at the appropriate level. Since the CPU has only four interrupt request lines (one for each level), the eight interrupt-pending bits at each level are ORed together to form a set of four composite interrupt requests, one for each level.

Eventually the CPU drops its IPL low enough to recogize the interrupt. The CPU then issues an interrupt acknowledge which is translated by the XCPGA into an XMI IDENT. There can be up to eight outstanding XMI interrupts at any given level (one from each of the maximum of eight devices that can interrupt). The KA62A CPU module gives priority to the highest node ID request within a given level.

Each CPU monitors the XMI for IDENT transactions. When an IDENT is detected, the interrupt-pending bit at the corresponding level and node is cleared, assuring that multiple interrupt-fielding nodes will not attempt to service the same interrupt. Once the first CPU sends an IDENT to a given node at a given level, all nodes clear the corresponding interrupt-pending bit. After the transmission of the IDENT, the interrupting device returns an interrupt vector to the CPU. The CPU then executes the appropriate interrupt service routine.

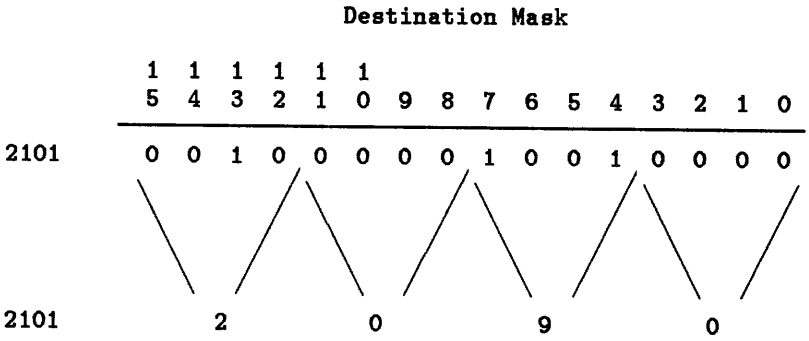
The interrupt-pending bits are controlled as follows:

- All KA62A CPU modules targeted by an XMI interrupt unconditionally set the corresponding interrupt-pending bits.
- All KA62A CPU modules unconditonally reset the corresponding interrupt-pending bit whenever an IDENT is transmitted on the XMI. For the KA62A CPU module generating the IDENT, the interrupt-pending state is cleared before the KA62A CPU module knows that it has successfully transmitted the IDENT to the interrupting node as it takes two cycles after the IDENT for the confirmation to be returned. The XMI interface stores the IDENT command so that, if the IDENT transmission fails, it can be reattempted. The interrupt-pending bits are not reexamined after a failed IDENT.
- The XMI interface arbitrates for the bus and, when granted, drives several null cycles to ensure that the interrupt-pending bits are quiesent during generation of the IDENT command. These null cycles are used to allow the interrupt-pending bits to become stable since the bits can only change state in response to an XMI transaction. After the required number of null cycles, the interrupt-pending bits are sampled and used to generate the proper IDENT destination field.
- It is possible that two nodes will attempt to service an interrupt at about the same time, as more than one CPU can be interrupted for a single interrupt condition. Only one processor wins the bus and transmits the IDENT. In response to the IDENT, all processors reset their corresponding interrupt-pending bits. It is possible, however, that a second CPU will issue an interrupt acknowledgment before its interrupt-pending bit resets. The second CPU module, once granted the XMI, drives the required number of null cycles, samples the interrupt-pending bits, finds none set, releases the bus, and issues an ERR response to the CPU. The CVAX treats the ERR assertion to an interrupt acknowledge as a "microcode passive release," and the operating system never knows that the interrupt happened.
- It is possible that during the time between receipt of an IPL 16 XMI interrupt and the generation of the corresponding IDENT that an interprocessor interrupt (IP) IVINTR could be received, as IP IVINTRs interrupt at IPL 16. Then the XMI logic performs the same XMI arbitration/null process in response to the CPU's interrupt acknowledge except, when the interrupt-pending bits are sampled, it will find the IP IVINTR bit set. Instead of sending an XMI IDENT it returns a vector of 80H to the CPU. Since no IDENT was transmitted, the interrupt-pending bit at IPL 16 is still set, and after servicing the IP IVINTR, the CPU services the XMI device.

3.5.5 Implied Vector Interrupts (IVINTR)

The IVINTR is a single-cycle XMI transaction used to implement VAX interprocessor interrupts (IP) and write error (WE) interrupts. For both WE and IP interrupts, the interrupt priority level and interrupt vector are implied by the type of interrupt.

Figure 3–17 Interprocessor IVINTR Generation Address Example



2101 2090 (I/O address for IP IVINTR that targets nodes C, 7, and 4)

The KA62A CPU module can generate and respond to IP and WE IVINTRs. WE IVINTRs are issued by I/O nodes that are unable to complete an I/O write transaction ("disconnected" transfers on the XMI).

The KA62A CPU module has a fixed range of I/O space addesses in XMI private space that, when written to, cause the generation of an XMI IVINTR transaction. The XMI interface handles the tranaction as if it were a write for error reporting.

NOTE: The write that generates the IVINTR must be generated by a byte-type macro instruction. MOVB is recommended.

The IVINTR generation address ranges are:

- 2101 0000 to 2101 FFFF for IP IVINTR
- 2102 0000 to 2102 FFFF for WE IVINTR

For both types of IVINTRs, A<15:0> are used as the XMI destination mask to indicate which nodes(s) are targeted by the IVINTR. Figure 3-17 gives an example of the address needed to send an IP IVINTR to XMI nodes 4, 7, and C.

The receipt of an IP IVINTR with a destination mask that has the corresponding node ID bit set causes the XMI interface logic to set an internal IP IVINTR pending bit and generate an IPL 16 device interrupt to the CPU. When the CPU acknowledges an IPL 16 interrupt, the XMI interface checks the IP IVINTR pending bit and, if set, returns a vector of 80H. The XMI interface logic resets the IP IVINTR pending bit during the XMI null cycles that precede each IDENT to ensure that no IP IVINTRs are "lost."

The receipt of a WE IVINTR with a destination mask that has the corresponding node ID bit set causes the XMI interface logic to set XBER<25> (WEI) and generate a MEMERR interrupt to the CPU. The CPU vectors directly to 60H in the SCB for a MEMERR interrupt. XBER<25> is cleared by the MEMERR interrupt service routine prior to servicing the write error interrupt. Software then polls all XMI devices to determine which device sent the WE IVINTR.

3.6 KA62A CPU Module Registers

The KA62A CPU module registers consist of internal processor registers, KA62A CPU module registers in XMI private space, and XMI required registers.

Table 3–7 KA62A CPU Module Internal Processor Registers

Address	Name	Mnemonic	Type[1]	Class[2]	Location
IPR0	Kernel Stack Pointer	KSP	R/W	1	
IPR1	Executive Stack Pointer	ESP	R/W	1	
IPR2	Supervisor Stack Pointer	SSP	R/W	1	
IPR3	User Stack Pointer	USP	R/W	1	
IPR4	Interrupt Stack Pointer	ISP	R/W	1	
IPR5 – IPR7	Reserved			3	
IPR8	P0 Base Register	P0BR	R/W	1	
IPR9	P0 Length Register	P0LR	R/W	1	
IPR10	P1 Base Register	P1BR	R/W	1	
IPR11	P1 Length Register	P1LR	R/W	1	
IPR12	System Base Register	SBR	R/W	1	
IPR13	System Length Register	SLR	R/W	1	
IPR14 – IPR15	Reserved			3	
IPR16	Process Control Block Base	PCBB	R/W	1	
IPR17	System Control Block Base	SCBB	R/W	1	
IPR18	Interrupt Priority Level	IPL	R/W	1 I	
IPR19	AST Level	ASTLVL	R/W	1 I	
IPR20	Software Interrupt Request	SIRR	W	1	
IPR21	Software Interrupt Summary	SISR	R/W	1 I	
IPR22 – IPR23	Reserved			3	
IPR24	Interval Clock Control and Status[3]	ICCS	R/W	2 I	CVAX

[1]See Table 3–8.

[2]Key to Classes:

1–Implemented by the KA62A CPU module (as specified in the *VAX Architecture Reference Manual*).
2–Implemented uniquely by the KA62A CPU module.
3–Not implemented. Read as zero; NOP on write.
4–Access not allowed; accesses result in a reserved operand fault.
5–Accessible, but not fully implemented; accesses yield UNPREDICTABLE results.
n I–the register is initalized on a KA62A CPU module reset (power-up, system reset, and node reset).

[3]Interval timer requests are posted at IPL 16 with a vector of C0 (hex). The interval timer is the lowest priority device at the IPL.

Table 3–7 (Cont.) KA62A CPU Module Internal Processor Registers

Address	Name	Mnemonic	Type[1]	Class[2]	Location
IPR25	Next Interval Count	NICR	W	3	
IPR26	Interval Count	ICR	RO	3	
IPR27	Time-of-Year Clock[4]	TODR	R/W	1	
IPR28	Console Storage Receiver Status	CSRS	R/W	5 I	
IPR29	Console Storage Receiver Data	CSRD	RO	5 I	
IPR30	Console Storage Transmitter Status	CSTS	R/W	5 I	
IPR31	Console Storage Transmitter Data	CSTD	W	5 I	
IPR32	Console Receiver Control/Status	RXCS	R/W	2 I	SSC
IPR33	Console Receiver Data Buffer	RXDB	RO	2 I	SSC
IPR34	Console Transmit Control/Status	TXCS	W	2 I	SSC
IPR35	Console Transmit Data Buffer	TXDB	W	2 I	SSC
IPR36	Translation Buffer Disable	TBDR	R/W	3	
IPR37	Cache Disable	CADR	R/W	2 I	CVAX
IPR38	Machine Check Error Summary	MCESR	R/W	3	
IPR39	Memory System Error	MSER	R/W	2 I	CVAX
IPR40 – IPR41	Reserved			3	
IPR42	Console Saved PC	SAVPC	RO	2	
IPR43	Console Saved PSL	SAVPSL	RO	2	
IPR44 – IPR47	Reserved			3	
IPR48	SBI Fault/Status	SBIFS	R/W	3	
IPR49	SBI Silo	SBIS	RO	3	
IPR50	SBI Silo Comparator	SBISC	R/W	3	
IPR51	SBI Maintenance	SBIMT	R/W	3	
IPR52	SBI Error	SBIER	R/W	3	
IPR53	SBI Timeout Address	SBITA	RO	3	
IPR54	SBI Quadword Clear	SBIQC	W	3	
IPR55	I/O Bus Reset	IORESET	W	2	
IPR56	Memory Management Enable	MAPEN	R/W	1	
IPR57	Translation Buffer Invalidate All	TBIA	W	1	

[1]See Table 3–8.

[2]Key to Classes:

1–Implemented by the KA62A CPU module (as specified in the *VAX Architecture Reference Manual*).
2–Implemented uniquely by the KA62A CPU module.
3–Not implemented. Read as zero; NOP on write.
4–Access not allowed; accesses result in a reserved operand fault.
5–Accessible, but not fully implemented; accesses yield UNPREDICTABLE results.
n I–the register is initalized on a KA62A CPU module reset (power-up, system reset, and node reset).

[4]TODR is maintained during power failure by the XMI TOY BBU PWR line on the XMI backplane.

Table 3-7 (Cont.) KA62A CPU Module Internal Processor Registers

Address	Name	Mnemonic	Type[1]	Class[2]	Location
IPR58	Translation Buffer Invalidate Single	TBIS	W	1	
IPR59	Translation Buffer Data	TBDATA	R/W	3	
IPR60	Microprogam Break	MBRK	R/W	3	
IPR61	Performance Monitor Enable	PMR	R/W	3	
IPR62	System Identification	SID	RO	1	CVAX
IPR63	Translation Buffer Check	TBCHK	W	1	
IPR64 - IPR127	Reserved			4	

[1]See Table 3-8.

[2]Key to Classes:

1-Implemented by the KA62A CPU module (as specified in the *VAX Architecture Reference Manual*).
2-Implemented uniquely by the KA62A CPU module.
3-Not implemented. Read as zero; NOP on write.
4-Access not allowed; accesses result in a reserved operand fault.
5-Accessible, but not fully implemented; accesses yield UNPREDICTABLE results.
n I-the register is initalized on a KA62A CPU module reset (power-up, system reset, and node reset).

Table 3-8 Types of Registers and Bits

Type	Description
0	Initialized to logic level zero
1	Initialized to logic level one
X	Initialized to either logic level
RO	Read only
R/W	Read/write
R/Cleared on W	Read/cleared on write
R/W1C	Read/cleared by writing a one

3.6.1 Processor Registers

The processor state is stored in processor registers rather than in memory. The processor state is composed of 16 general purpose registers (GPRs), the processor status longword (PSL), and internal processor registers (IPRs).

Nonprivileged software can access the GPRs and the lower half of the processor status longword (PSL<15:0>). The IPRs and PSL<31:16> can only be accessed by privileged software. The IPRs are explicitly accessible only by the Move To Processor Register (MTPR) and Move From Processor Register (MFPR) instructions, which require kernel mode privileges.

In addition to the internal processor registers, the KA62A CPU module contains registers in XMI private space and XMI required registers in XMI nodespace. These registers are listed in Table 3–10 and Table 3–9.

3.6.2 XMI Registers and Control and Status Register 1 Characteristics

The KA62A CPU module's XMI registers have the following characteristics (items 1 and 2 also apply to CSR1):

1 The Mask bits are ignored on writes to the KA62A CPU module's Control and Status 1 and 2 registers. The CPU always performs a full longword write.

2 Interlocks are not supported. Interlock Read and Unlock Write Mask transactions are treated as Read and Write Mask transactions, respectively.

3 The XMI responder queue is only one deep so the KA62A CPU module will NO ACK subsequent CSR references until the read data for the queued CSR read has been returned.

Table 3-9 KA62A CPU Module Registers in XMI Private Space

Register	Mnemonic	Address	Location
XCP Control and Status 1	CSR1	2000 0000	External logic
XCP ROM	ROM	2004 0000 - 2007 FFFF	
XCP EEPROM	EEPROM	2008 0000 - 2008 7FFF	
SSC Base Address	SSCBR	2014 0000	SSC
SSC Configuration	SSCCR	2014 0010	SSC
CDAL Bus Timeout Control	CBTCR	2014 0020	SSC
Console Select	CONSEL	2014 0030	SSC
Timer Control Register 0	TCR0	2014 0100	SSC
Timer Interval Register 0	TIR0	2014 0104	SSC
Timer Next Interval Register 0	TNIR0	2014 0108	SSC
Timer Interrupt Vector Register 0	TIVR0	2014 010C	SSC
Timer Control Register 1	TCR1	2014 0110	SSC
Timer Interval Register 1	TIR1	2014 0114	SSC
Timer Next Interval Register 1	TNIR1	2014 0118	SSC
Timer Interrupt Vector Register 1	TIVR1	2014 011C	SSC
CSR1 Base Address	CSR1BADR	2014 0130	SSC
CSR1 Address Decode Mask	CSR1ADMR	2014 0134	SSC
EEPROM Base Address	EEBADR	2014 0140	SSC
EEPROM Address Decode Mask	EEADMR	2014 0144	SSC
SSC BBU RAM	BBURAM	2014 0400 - 2014 07FF	
IP IVINTR Generation	IPIVINTRGEN	2101 0000 - 2101 FFFF	
WE IVINTR Generation	WEIVINTRGEN	2102 0000 - 2102 FFFF	

Table 3-10 XMI Registers for the KA62A CPU Module

Name	Mnemonic	Address	Location
XMI Device Register	XDEV	BB[1] + 0000 0000	XCPGA
XMI Bus Error Register	XBER	BB + 0000 0004	XCPGA
XMI Failing Address Register	XFADR	BB + 0000 0008	SSC
XMI XGPR	XGPR	BB + 0000 000C	XCPGA
XCP Control and Status 2	CSR2	BB + 0000 0010	SSC

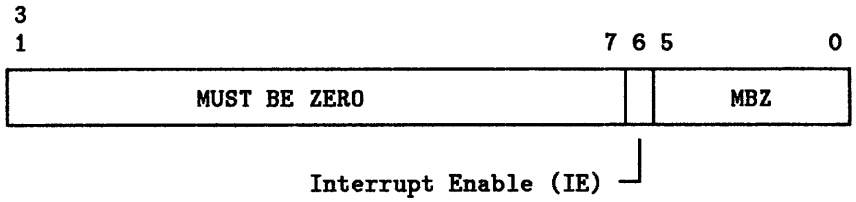
[1] "BB" = base address of a node, which is the address of the first location in nodespace.

Interval Clock Control and Status Register (ICCS)

The ICCS contains control information for the interval clock. The interval clock is used for accounting, for time-dependent events, and for maintaining the software date and time.

ADDRESS

IPR24 (CVAX)



bits<31:7>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<6>

Name: Interrupt Enable

Mnemonic: IE

Type: R/W,0

IE enables and disables interval timer interrupts. When IE is set, an interval timer interrupt is requested every 10 milliseconds. When IE is clear, interval timer interrupts are disabled.

bits<5:0>

Name: Reserved

Mnemonic: None

Type: –

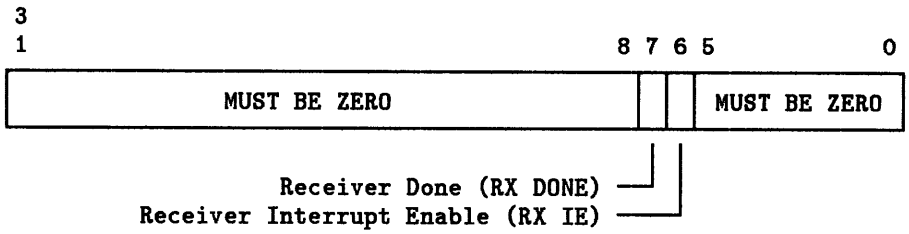
Reserved; must be zero.

Console Receiver Control and Status (RXCS)

The RXCS controls and reports the status of incoming data on the console serial line.

ADDRESS

IPR32 (SSC)



bits<31:8>

Name: Reserved
Mnemonic: None
Type: –

Reserved; returns zero.

bit<7>

Name: Receiver Done
Mnemonic: RX DONE
Type: RO, 0

RX DONE is set when an entire character has been received and is ready to be read from RXDB<7:0> (RBUF). RX DONE is automatically cleared when RXDB<7:0> is read.

bit<6>

Name: Receiver Interrupt Enable
Mnemonic: RX IE
Type: R/W, 0

If RX DONE and RX IE are both set, a program interrupt is requested.

bits<5:0>

Name: Reserved
Mnemonic: None
Type: –

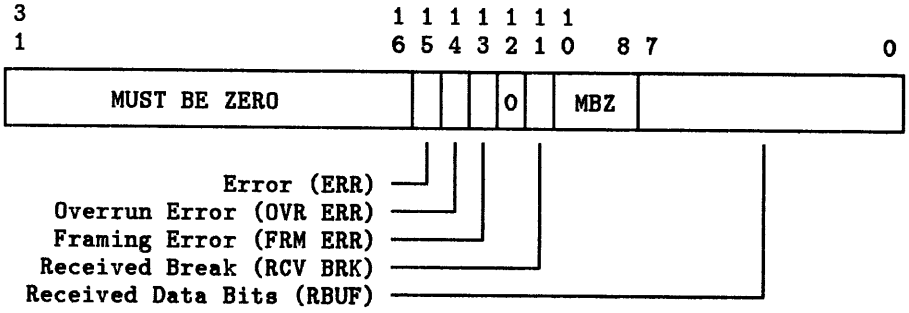
Reserved; returns zero.

Console Receiver Data Buffer (RXDB)

RXDB buffers incoming serial-line data and captures error information. Error conditions remain until the next character is received, at which point the error bits are updated.

ADDRESS

IPR33 (SSC)



bits<31:16>

Name: Reserved

Mnemonic: None

Type: –

Reserved; returns zero.

bit<15>

Name: Error

Mnemonic: ERR

Type: RO, 0

ERR is set if either bit<14> or <13> is set. ERR is clear if both bits are clear. ERR does not generate a program interrupt.

bit<14>

Name: Overrun Error

Mnemonic: OVR ERR

Type: RO, 0

OVR ERR is set if a previously received character was not read before being overwritten by the present character.

bit<13>

Name: Framing Error
Mnemonic: FRM ERR
Type: RO, 0

FRM ERR is set if the present character did not have a valid stop bit.

bit<12>

Name: Reserved
Mnemonic: None
Type: -

Reserved; returns zero.

bit<11>

Name: Received Break
Mnemonic: RCV BRK
Type: RO, 0

RCV BRK is set following the receipt of a CTRL/P character and remains set until the register is read.

bits<10:8>

Name: Reserved
Mnemonic: None
Type: -

Reserved; returns zero.

bits<7:0>

Name: Received Data Bits
Mnemonic: RBUF
Type: RO

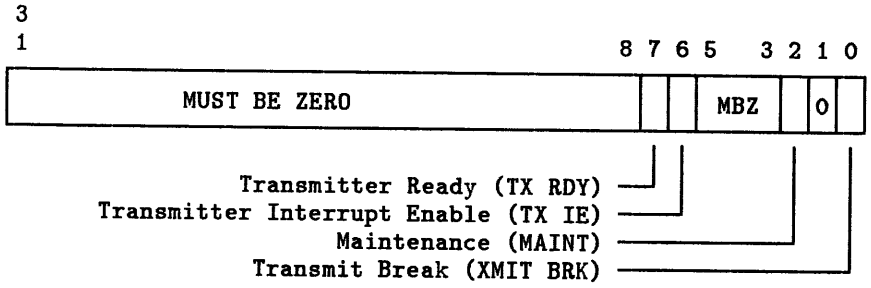
The RBUF field contains the last character received from the console.

Console Transmitter Control and Status (TXCS)

TXCS controls and reports the status of outgoing data on the console serial line.

ADDRESS

IPR34 (SSC)



bits<31:8>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<7>

Name: Transmitter Ready

Mnemonic: TX RDY

Type: RO,1

TX RDY is cleared when TXDB<7:0> (TBUF) is loaded and is set when TBUF can receive another character.

bit<6>

Name: Transmitter Interrupt Enable

Mnemonic: TX IE

Type: R/W,0

If both TX RDY and TX IE are set, a program interrupt is requested.

bits<5:3>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<2>

Name: Maintenance
Mnemonic: MAINT
Type: R/W,0

MAINT facilitates a maintenance self-test. When MAINT is set, the external serial output is set to MARK and the serial output is used as the serial input (a loopback).

bit<1>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<0>

Name: Transmit Break
Mnemonic: XMIT BRK
Type: R/W,0

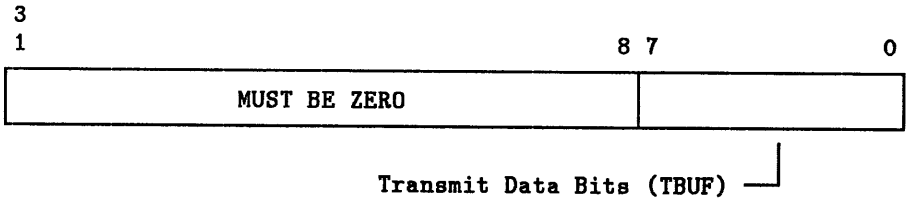
When XMIT BRK is set, the serial output is forced to the SPACE condition. While XMIT BRK is set, the transmitter operates normally, but the output line remains low so that software can transmit dummy characters to time the break.

Console Transmitter Data Buffer (TXDB)

TXDB buffers outgoing data on the console serial line.

ADDRESS

IPR35 (SSC)



bits<31:8>

Name: Reserved
Mnemonic: None
Type: –

Reserved; returns zero.

bits<7:0>

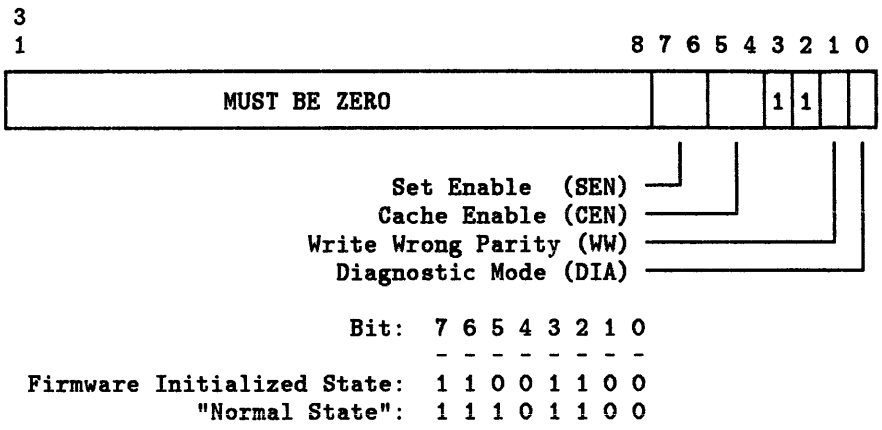
Name: Transmit Data Bits
Mnemonic: TBUF
Type: WO

TBUF loads the character to be transmitted on the console serial line.

Cache Disable Register (CADR)

CADR is used by the CVAX chip to control the first-level cache.

ADDRESS *IPR37 (CVAX)*



bits<31:8>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bits<7:6>

Name: Set Enable

Mnemonic: SEN

Type: R/W, 0

SEN is used to selectively enable or disable each set within the cache. CADR<7> sets to enable Set 2 of the cache and clears to disable Set 2. CADR<6> sets to enable Set 1 of the cache and clears to disable Set 1.

bits<5:4>

Name: Cache Enable

Mnemonic: CEN

Type: R/W, 0

CEN is used to selectively enable or disable the storing of I-stream and D-stream references in the cache. CADR<5> sets to enable I-stream memory space reference storage in the cache and clears to disable I-stream storage in the cache. CADR<4> sets to enable D-stream memory space reference storage in the cache and clears to disable D-stream storage in the cache.

Configuration Rule: The first-level cache MUST be configured for I-stream only operation (CADR<5:4> = 10).

NOTE: The first-level cache can be disabled by either disabling both Set 1 and Set 2 or by not storing both I-stream and D-stream references.

bits<3:2>

Name: Reserved

Mnemonic: None

Type: RO, 1

Reserved; both bits must be one.

bit<1>

Name: Write Wrong Parity

Mnemonic: WW

Type: R/W, 0

WW sets to require that incorrect parity be stored in the first-level cache whenever it is written. When cleared, correct parity is stored in the cache whenever the cache is written.

bit<0>

Name: Diagnostic Mode

Mnemonic: DIA

Type: R/W, 0

DIA causes writes to the CADR to flush the first-level cache (all valid bits set to the invalid state). The cache is configured for "normal" write-through operation when cleared.

When DIA is set:

- Writes to the CADR will not cause the first-level cache to be flushed.
- All CPU write references write the data into the cache as well as main memory, irrespective of whether or not a cache hit occurred.
- Errors are ignored (they do not cause a machine-check trap to be generated or prevent data from being stored in the cache).
- There is no effect on read references.

Diagnostic mode blocks the flush of the cache when CADR is written. Diagnostic mode is subject to the following restrictions:

- Diagnostic mode must only be selected when one and only one of the two Sets is enabled. Operation of diagnostic mode with both sets or neither set enabled yields UNPREDICTABLE results.
- A valid write allocation occurs only if a specific sequence of instructions is followed. The first instruction must be a quadword write (MOVQ) to a quadword-aligned destination. This instruction writes the second longword of the source operand to the first longword of the cache entry selected by the destination address. The first longword of the source is not used and the second longword of the cache entry remains unchanged. The cache tag and valid bits are set so that subsequent reads and writes to either longword in the destination report a cache hit.
- The second instruction must be a cachable read operation.
- The third instruction must be a longword write to the address corresponding to the second longword in the cache entry. The following is a sample macrocode listing showing this sequence:

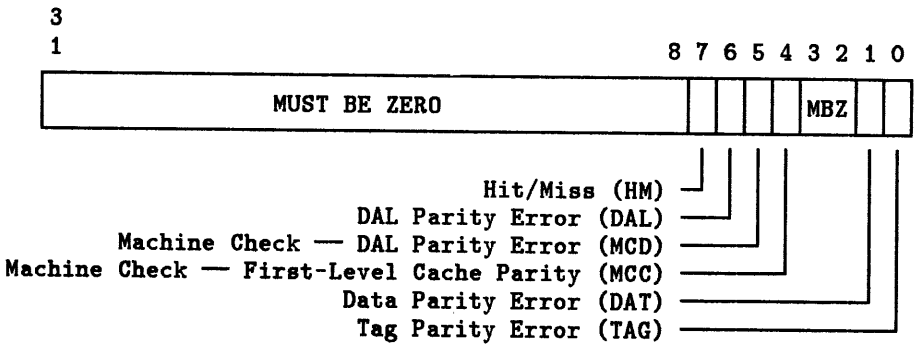
```
    .
    .
    .
MOVQ    #quadscr, @#quaddst     ; writes longword quadsrc+4 into
                                ; longword quaddst
MOVL    #quaddst, R0            ; reads allocated longword quaddst
MFPR    #mser, R1               ; get MSER in order to look at H/M
                                ; bit later
MOVE    #longsrc, @#(quaddst+4); writes 2nd longword quaddst+4
    .
    .
    .
```

When this sequence is followed, each cache entry can be allocated with any arbitrary address.

Memory System Error Register (MSER)

The memory system error register records the occurrence of first-level cache hits and parity errors on the CDAL bus, in the first-level cache, and in the second-level cache. The MSER is explicitly cleared via the MTPR MSER instruction irrespective of the write data.

ADDRESS *IPR39 (CVAX)*



bits<31:8>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<7>

Name: Hit/Miss
Mnemonic: HM
Type: RO, 0

HM sets on all cachable references that miss the first-level cache and clears on all cachable references that hit the first-level cache.

bit<6>

Name: DAL Parity Error
Mnemonic: DAL
Type: R/Cleared on W, 0

DAL sets whenever a CDAL bus or second-level cache data store parity error is detected.

bit<5>

Name: Machine Check - DAL Parity Error
Mnemonic: MCD
Type: R/Cleared on W, 0

MCD sets whenever a machine check is caused by a CDAL bus or second-level cache parity error.

bit<4>

Name: Machine Check - First-Level Cache Parity Error
Mnemonic: MCC
Type: R/Cleared on W, 0

MCC sets whenever a machine check is caused by a first-level cache parity error in the tag or data store.

bit<3:2>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<1>

Name: Data Parity Error
Mnemonic: DAT
Type: R/Cleared on W, 0

DAT sets when a parity error is detected in the data store of the first-level cache unless there is a simultaneous tag and data parity error. Only TAG (bit<0>) sets when there are simultaneous tag and data parity errors.

bit<0>

Name: Tag Parity Error
Mnemonic: TAG
Type: R/Cleared on W, 0

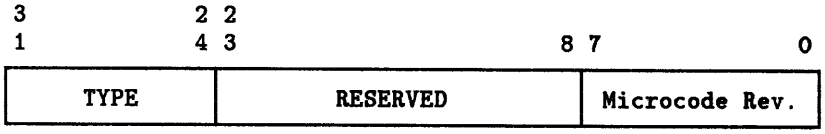
TAG sets when a parity error is detected in the tag store of the first-level cache. Only TAG sets when there are simultaneous tag and data parity errors.

System Identification Register (SID)

SID specifies the processor type and includes an implementation-dependent field. It can only be accessed locally. Other devices on the XMI determine the nature of a node by reading its XMI Device Register (XDEV).

ADDRESS

IPR62 (CVAX)



bits<31:24>

Name: Processor Type
Mnemonic: TYPE
Type: RO

This field is always 0A (hex), indicating the CVAX chip.

bits<23:8>

Name: Reserved
Mnemonic: None
Type: RO

Reserved.

bits<7:0>

Name: Microcode Revision

Mnemonic: None

Type: RO

This field shows the microcode revision level of the CVAX chip, at the time of printing, as follows:

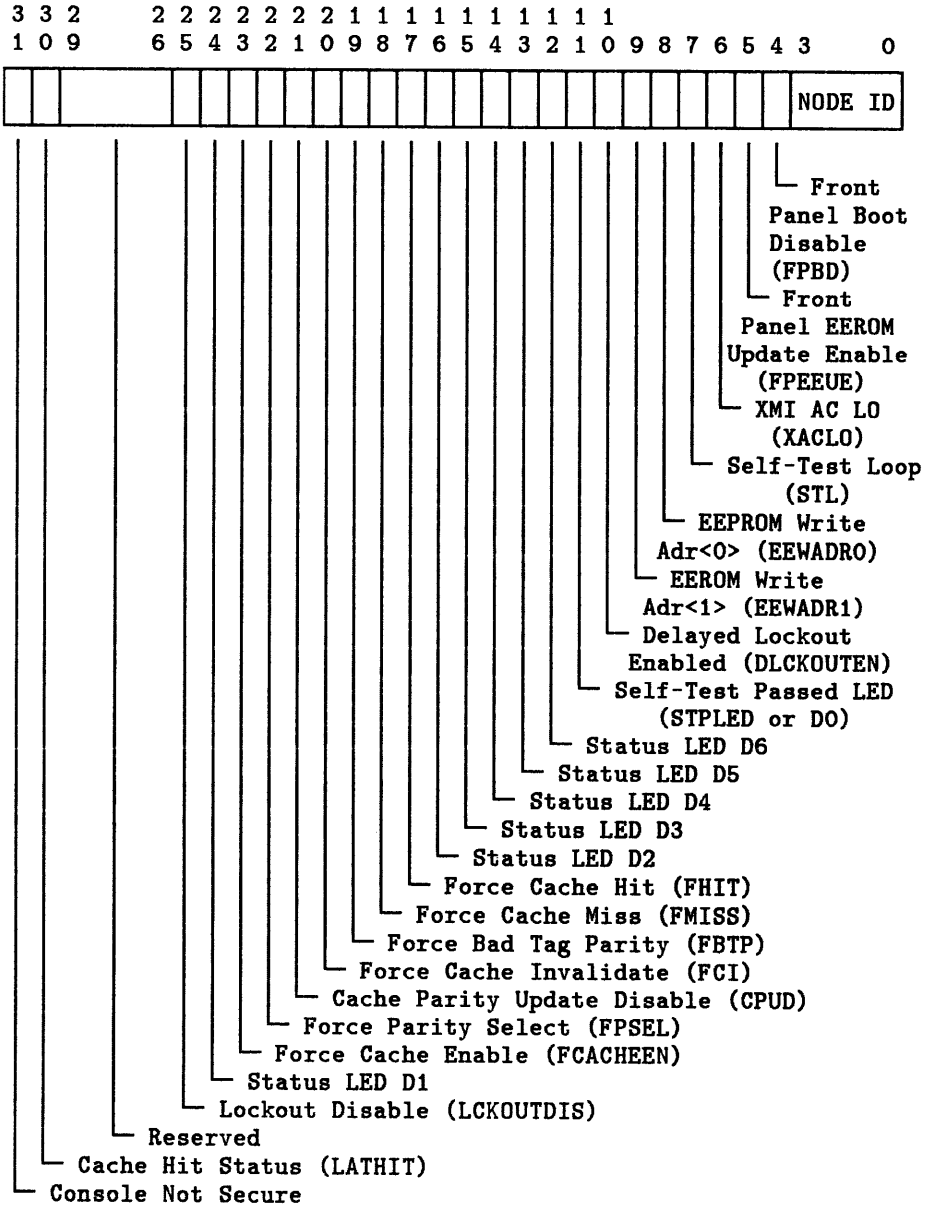
CVAX Revision	<7:0> (hex)
3.0	02
3.1	02
3.2	02
4.2	03
4.4	04

Control and Status Register 1 (CSR1)

CSR1 provides KA62A CPU module control and status. Since most bits in CSR1 power up in an indeterminate state, console code initializes CSR1 very early in the power-up sequence.

ADDRESS

2000 0000 (External logic)



bit<31>

Name: Console Not Secure
Mnemonic: None
Type: RO, 1

Console Not Secure reflects the received state of the XMI CON SECURE L line that is driven from the XMI backplane. When this bit is deasserted (a HIGH voltage level), the console is secure.

bit<30>

Name: Cache Hit Status
Mnemonic: LATHIT
Type: RO, X

LATHIT is used by cache coherency diagnostics running out of I/O space (that is, the on-board ROM) to determine if a cache hit has occurred. LATHIT is first cleared by writing a zero to CSR1<10> (DLCKOUTEN) and then releasing the clear by writing a one to the same location. The next cache hit (meaning TAG address and VALID bit match) causes LATHIT to be set. Once set, this bit remains set until explicitly cleared by writing a zero to CSR1<10>.

bits<29:26>

Name: Reserved
Mnemonic: None
Type: RO, 1

Reserved; returns a one.

bit<25>

Name: Lockout Disable
Mnemonic: LCKOUTDIS
Type: R/W, 0

LCKOUTDIS disables all assertions of XMI LOCKOUT (CSR2<22:21>). Normally, the KA62A CPU module asserts LOCKOUT even if interlock lockout avoidance is disabled.

bit<24>

Name: Status LED D1
Mnemonic: SLED1
Type: R/W, X

Status LED D1 is used with Status LED D2 through D6. See CSR1<16:12>.

bit<23>

Name: Force Cache Enable

Mnemonic: FCACHEEN

Type: R/W, X

Setting FCACHEEN causes the cache to remain active after error conditions. When cleared, certain errors will disable the cache.

bit<22>

Name: Force Parity Select

Mnemonic: FPSEL

Type: R/W, X

When FPSEL is set, the KA62A CPU module does not generate parity for the XMI P<2:0> L lines but, instead, drives Force Parity <2:0> (CSR2<6:4>). FPSEL is used only during diagnostic testing; remains cleared during normal system operation.

bit<21>

Name: Cache Parity Update Disable

Mnemonic: CPUD

Type: R/W, X

When CPUD is set, the KA62A CPU module second-level cache does not update its data parity RAMs. Bad parity can be forced by first writing cache while CPUD is set. Then, after clearing CPUD, subsequent writes to cache have correct/incorrect parity, depending on the data pattern written.

When CPUD is set, CDAL parity checking is disabled for second-level cache references, allowing operating system and diagnostic software to capture data from a second-level cache location that contains a parity error.

bit<20>

Name: Force Cache Invalidate

Mnemonic: FCI

Type: R/W, X

When FCI is set, the entire second-level cache and duplicate tag store are held invalidated. The cache should be first disabled by setting Force Miss, bit <18>, before setting FCI.

bit<19>

Name: Force Bad Tag Parity
Mnemonic: FBTP
Type: R/W, X

When FBTP is set, the parity enable (PE) line on each of the KA62A CPU module tag chips is asserted during operations that write the tag, forcing bad parity to be written by the tag chips for the current tag entry. Subsequent reads of the tag entry cause parity errors.

bit<18>

Name: Force Miss
Mnemonic: FMISS
Type: R/W, 0

When FMISS is set, the KA62A CPU module second-level cache and XMI interface behave as though a cache miss occurred, regardless of the state of the tag and valid bits. Setting both FHIT CSR1<17> (FHIT) and FMISS results in the disabling of both cache and XCPGA, which should be avoided.

FMISS is also set by various error conditions that generate cache disable. The error conditions must be removed before FMISS can be cleared. Cache disable is inhibited when CSR1<23>=1 (FCACHEEN), as this is used for diagnostic purposes only (that is, cache remains active after error conditions).

Operating system software is required to flush the second-level cache (CSR1<FCI>) before resetting FMISS to ensure that the cache state is consistent when the cache is reenabled. This is required since the KA62A CPU module performs cache fills while FMISS is asserted but does not update the cache on CVAX writes that "hit" (that is, write-throughs are disabled), which could cause the state of the cache to become inconsistent while FMISS is asserted.

bit<17>

Name: Force Hit
Mnemonic: FHIT
Type: R/W, X

When FHIT is set, the KA62A CPU module second-level cache and XMI interface behave as though a cache hit occurs for each memory-space reference regardless of the state of the tag and valid bits. Associated XMI writes are suppressed and only the cache location will be updated. I/O space references are disabled as FHIT causes the XCPGA chip to ignore CVAX transactions. To maintain the FHIT functionality regardless of errors, the CSR1<23> (FCACHEEN) is also set. Setting both FMISS and FHIT results in the disabling of both cache and XCPGA, which should be avoided.

bits<16:12>

Name: Status LEDs D2–D6

Mnemonic: SLED2–SLED6

Type: R/W, X

SLED1 (CSR1<24>) and SLED2–SLEDD6 bits drive the six LEDs used to report status of self-test as it runs and the failing test number if self-test fails.

bit<11>

Name: Self-Test Pass LED

Mnemonic: STPLED

Type: R/W, 0

STPLED drives the self-test pass LED (D8) on the KA62A CPU module. It is set following the successful completion of self-test.

bit<10>

Name: Delayed Lockout Enable

Mnemonic: DLCKOUTEN

Type: R/W, X

DLCKOUTEN enables an optional delay between the time that the XCPGA chip asserts LOCKOUT until XMI LOCKOUT is asserted. DLCKOUTEN is also used to clear the LATHIT latch (CSR1<30>) during cache testing. The two functions of DLCKOUTEN are never used as the same time.

bits<9:8>

Name: EEPROM Write Address <1:0>

Mnemonic: EEWADR

Type: R/W, X

The KA62A CPU module always provides write data on DAL<7:0>. EEWADR gives the programmer the ability to write the data to any byte address within the EEPROM since the EEPROM data path is a byte wide.

Before updating an EEPROM location, the software must first load the correct byte address into EEWADR<1:0>. Then the write to the EEPROM can be started.

bit<7>

Name: Self-Test Loop
Mnemonic: STL
Type: RO

When STL is set, the KA62A CPU module continually reruns its self-test sequence. STL is driven by an I/O pin and can be used to implement a manufacturing "burn-in" test. This bit is "low true."

bit<6>

Name: XMI AC LO
Mnemonic: XACLO
Type: RO

XACLO shows the state of the XMI AC LO L line. The KA62A CPU module should not access main memory until the bit is a one, indicating that XMI AC LO L is deasserted.

bit<5>

Name: Front Panel EEROM Update Enable
Mnemonic: FPEEUE
Type: RO

FPEEUE shows the received state of the XMI BOOT EN L line that is driven by the front panel switch.

bit<4>

Name: Front Panel Boot Disable
Mnemonic: FPBD
Type: RO

FPBD shows the received state of the XMI BOOT EN L line that is driven by the front panel switch.

bits<3:0>

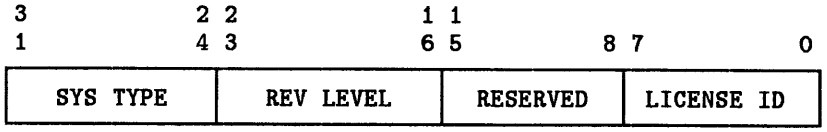
Name: Node ID
Mnemonic: NID
Type: RO

NID contains the node ID as received from the XMI backplane.

System Type (SYSTYPE)

SYSTYPE is a 32-bit register implemented in the KA62A CPU module ROM. It can only be accessed locally. Other devices on the XMI determine the nature of a node by reading its XMI Device Register (XDEV).

ADDRESS *2004 0004 (EEPROM)*



bits<31:24>

Name: System Type
Mnemonic: SYS TYPE
Type: RO

SYS TYPE is 02 (hex) for the KA62A CPU module.

bits<23:16>

Name: Revision Level
Mnemonic: REV LEVEL
Type: RO

REV LEVEL shows the revision level of the KA62A CPU module console code. REV LEVEL is encode in the form x.y where x is encoded into <23:20> and y is encoded into <19:16>. Therefore, a console revision of 2.1 would be encoded as 21 (hex) while a console revision of 2.10 would be 2A (hex).

bits<15:8>

Name: Reserved
Mnemonic: None
Type: RO

Reserved.

bits<7:0>

Name: License Identifier

Mnemonic: LICENSE ID

Type: RO

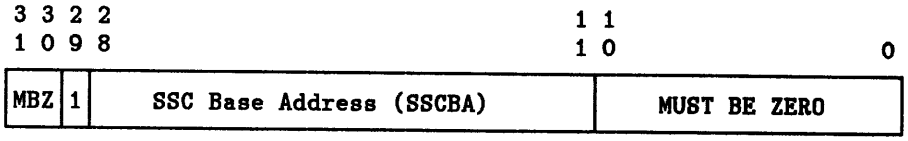
LICENSE ID is set (logic level of one) to allow the processor to be part of a timesharing system.

SSC Base Address Register (SSCBR)

SSCBR controls the base address of a 2-Kbyte block of the local I/O space that includes the battery-backed-up RAM, the registers for the programmable timers, the CSR1, the EEPROM Address Decode Match and Mask Registers, the Diagnostic LED Register, the CDAL Bus Timeout Register, and diagnostic registers that allow several IPRs to be accessed by means of I/O page addresses.

ADDRESS

2014 0000 (SSC)



bits<31:30>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bit<29>

Name: Reserved
Mnemonic: None
Type: R/W, 1

Reserved; must be one.

bits<28:11>

Name: SSC Base Address
Mnemonic: SSCBA
Type: R/W

SSCBA controls the base address of the 2-Kbyte block and is set to 2014 0000 (hex) by console code during processor initialization.

bits<10:0>

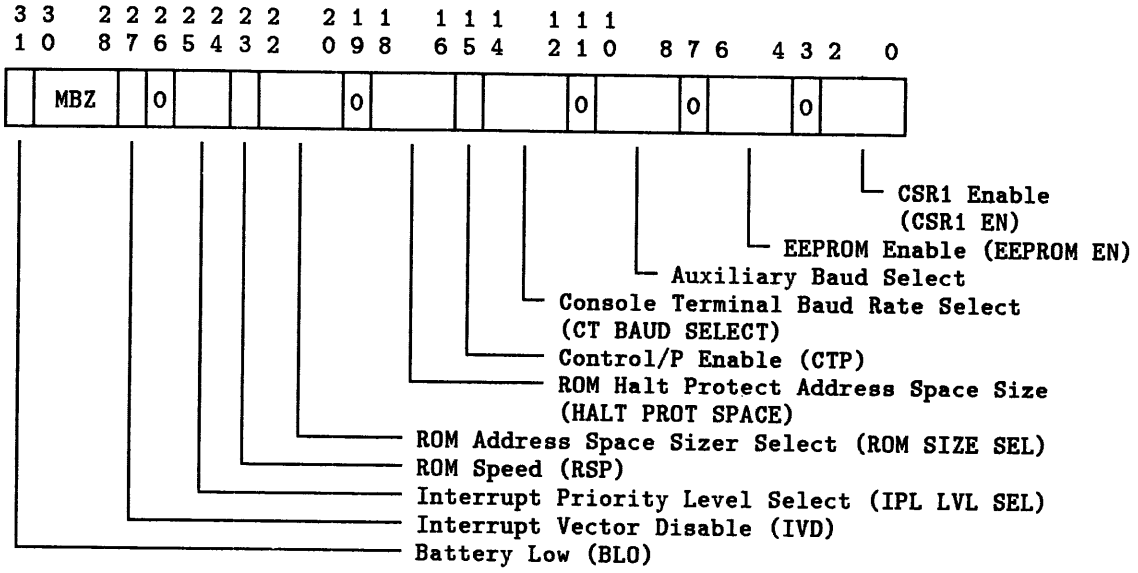
Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

SSC Configuration Register (SSCCR)

SSCCR controls the initialization parameters for the console serial line, programmable timers, ROM, EEPROM, TOY clocks, and CSR1.

ADDRESS *2014 0010 (SSC)*



Firmware Initialized State:

Binary:	0	0	0	0	0	0	0	1	0	1	1	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1
Hex:		0				1				7				6				2				0				0				7		

bit<31>

Name: Battery Low

Mnemonic: BLO

Type: R/W1C

BLO is set if the battery voltage goes below threshold while the module is powered down. Once set, BLO can only be cleared by software writing a zero to it. If set, the TOY clocks are cleared on KA62A CPU module reset.

bits<30:28>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bit<27>

Name: Interrupt Vector Disable
Mnemonic: IVD
Type: R/W, 0

When IVD is set, the console serial line and programmable timers do not respond to interrupt acknowledge cycles.

bit<26>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bits<25:24>

Name: IPL Level Select
Mnemonic: IPL LVL SEL
Type: R/W, 0

IPL LVL SEL specify the IPL level of interrupt acknowledge cycles that the console serial line and programmable timers respond to. On the KA62A CPU module, this field is set to 01 (IPL 15) by console code.

bit<23>

Name: ROM Speed
Mnemonic: RSP
Type: R/W, 0

RSP selects the ROM access time. 0=450 ns; 1=250 ns.

bits<22:20>

Name: ROM Address Space Size Select
Mnemonic: ROM SIZE SEL
Type: R/W, 0

ROM SIZE SEL controls the size of the range of addresses to which the ROM responds. ROM SIZE SEL is always 111, yielding an address range of 1 Mbyte (2004 0000 to 2013 FFFF).

bit<19>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bits<18:16>

Name: ROM Halt Protect Address Space Size Select
Mnemonic: HALT PROT Space
Type: R/W

The halt protect address space is set to 512 Kbytes by console code during processor initialization.

bit<15>

Name: Control/P Enable
Mnemonic: CTP
Type: R/W, 0

When CTP is set, it causes the CPU to be halted, if halts are enabled (XMI CON SECURE reset), when CTRL/P is typed at the console. When CTP is clear, it causes the CPU to be halted, if halts are enabled, when BREAK is typed at the console.

bits<14:12>

Name: Console Terminal Baud Rate Select
Mnemonic: CT BAUD SELECT
Type: R/W, 0

CT BAUD SELECT use the following codes to select the console baud rate:

CT BAUD SELECT<14:12>			
14	**13**	**12**	**Baud Rate**
0	0	0	300
0	0	1	600
0	1	0	1200
0	1	1	2400
1	0	0	4800
1	0	1	9600
1	1	0	19200
1	1	1	38400

bit<11>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bits<10:8>

Name: Auxiliary Baud Select
Mnemonic: None
Type: R/W, 0

Unused; read as written.

bit<7>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bits<6:4>

Name: EEPROM Enable
Mnemonic: EEPROM EN
Type: R/W, 0

EEPROM EN is set to 000 (binary) by console code during processor initialization. When set to 101 (binary), updates to the EEPROM are enabled.

bit<3>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bits<2:0>

Name: CSR1 Enable
Mnemonic: CSR1 EN
Type: R/W, 0

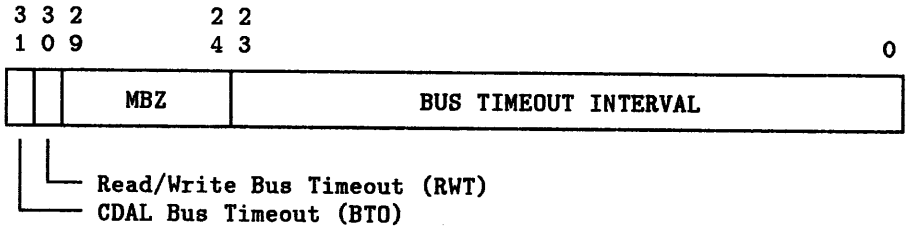
CSR1 EN enables CSR1 when set to 111 (binary) by a processor initialization.

CDAL Bus Timeout Control Register (CBTCR)

CBTCR controls the amount of time (timeout) allowed to elapse before a CDAL bus cycle is aborted. This prevents unanswered CPU read or write accesses or interrupt acknowledge cycles (IDENT) from hanging the system longer than the timeout interval.

ADDRESS

2014 0020 (SSC)



bit<31>

Name: CDAL Bus Timeout

Mnemonic: BTO

Type: R/Cleared on W, 0

BTO is set when the Bus Timeout Interval (CBTCR<23:0>) has expired during a CPU read, write, or interrupt acknowledge cycle.

bit<30>

Name: Read/Write Bus Timeout

Mnemonic: RWT

Type: R/Cleared on W, 0

RWT is set when the Bus Timeout Interval (CBTCR<23:0>) has expired during a CPU read or write cycle but not during an interrupt acknowledge cycle.

bits<29:24>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bits<23:0>

Name: Bus Timeout Interval

Mnemonic: None

Type: R/W, 0

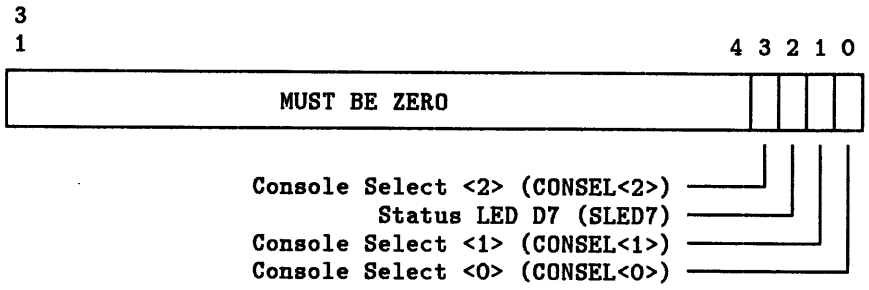
Bus Timeout Interval gives the desired timeout period. The available range of 1 to FFFFFF (hex) corresponds to a selectable timeout range of 1 microsecond to 16.77 seconds in 1 microsecond increments. Writing a zero to this field disables the bus timeout function.

Console Select Register (CONSEL)

The CONSEL register is used to select which console lines are attached to the console transmit and receive register.

ADDRESS

2014 0030 (SSC)



bits<31:4>

Name: Reserved

Mnemonic: None

Type: –

Reserved; returns zero.

bit<2>

Name: Status LED D7

Mnemonic: SLED7

Type: R/W, 0

SLED7 powers up cleared, which causes LED D7 to be on. Writing a one to this bit turns LED D7 off.

bits<3> and <1:0>

Name: Console Select<2:0>

Mnemonic: CONSEL<2:0>

Type: R/W, 0

The CONSEL field selects the operational mode for the console attached to the console transmit and receive register. The modes are as follows:

CONSEL<2:0>				RECV		
2	**1**	**0**	**Drive**	**XMI**	**Data**	**Mode**
0	0	0	No	AUX	AUX	Power-up state
0	0	1	No	XMI	AUX	All fail on power-up state
0	1	0	No	LB	AUX	Loopback at XMI XMIT driver input
0	1	1	No	XMI/AUX	Unused	
1	0	0	Yes	AUX	XMI/AUX	Unused
1	0	1	Yes	XMI	XMI/AUX	Boot processor state
1	1	0	Yes	LB	XMI/AUX	Unused
1	1	1	Yes	LB	XMI/AUX	Loopback on XMI

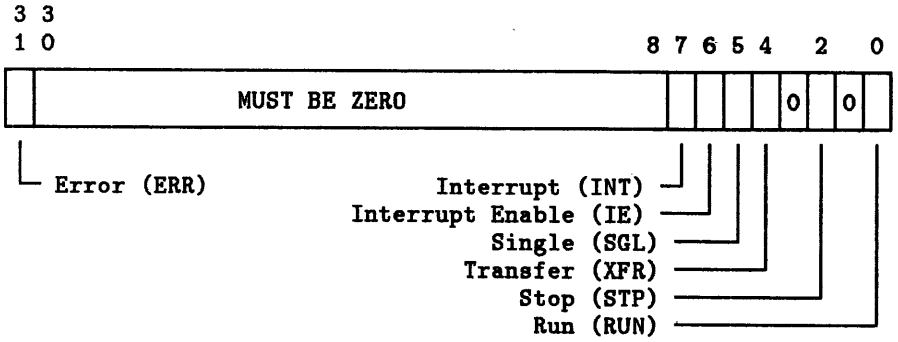
It is possible to receive data on XMI CON RECV without having the XMI CON XMIT driver enabled. This mode is used when no CPU becomes the boot processor on power-up; all monitor the XMI console lines for further commands.

Timer Control Register 0 (TCR0)

TCR0 controls timer 0.

ADDRESS

2014 0100 (SSC)



bit<31>

Name: Error

Mnemonic: ERR

Type: R/W1C, 0

ERR is set whenever the Timer Interval Register overflows and INT is already set, indicating a missed overflow.

bits<30:8>

Name: Reserved

Mnemonic: None

Type: R/W

Reserved; must be zero.

bit<7>

Name: Interrupt

Mnemonic: INT

Type: R/W1C, 0

INT is set whenever the Timer Interval Register overflows. If IE is set when INT is set, an interrupt is posted at IPL 15.

bit<6>

Name: Interrupt Enable
Mnemonic: IE
Type: R/W, 0

When IE is set, the timer interrupts at IPL 15 when INT is set.

bit<5>

Name: Single
Mnemonic: SGL
Type: R/W, 0

Setting SGL causes the Timer Interval Register to increment by one if the RUN bit is cleared. If RUN is set, then writes to SGL are ignored. SGL is always read as zero.

bit<4>

Name: Transfer
Mnemonic: XFR
Type: R/W, 0

Setting XFR causes the Timer Next Interval Register to be copied into the Timer Interval Register.

bit<3>

Name: Reserved
Mnemonic: None
Type: R/W

Reserved; must be zero.

bit<2>

Name: Stop
Mnemonic: STP
Type: R/W, 0

STP determines whether the timer stops after an overflow. If STP is set at overflow, RUN is cleared by the hardware at overflow and counting stops.

bit<1>

Name: Reserved
Mnemonic: None
Type: R/W

Reserved; must be zero.

bit<0>

Name: Run

Mnemonic: RUN

Type: R/W, 0

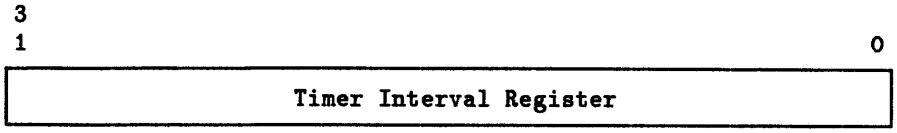
When RUN is set, the Timer Interval Register is incremented once every microsecond. INT is set when the timer overflows. If STP is set at overflow, RUN is cleared by the hardware at overflow and counting stops. When RUN is clear, the Timer Interval Register is not incremented automatically.

Timer Interval Register 0 (TIR0)

TIR0 contains the interval count for timer 0.

ADDRESS

2014 0104 (SSC)



bits<31:0>

Name: Timer Interval Register 0

Mnemonic: TIR0

Type: RO, 0

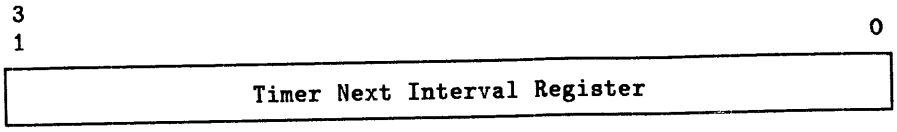
When TCR0<0> (RUN) is one, the register is incremented once every microsecond. When the counter overflows, TCR0<7> is set, and an interrupt is posted at IPL 15 if TCR0<6> is set. Then, if TCR0<2> is zero, TCR0<0> is cleared and counting stops.

Timer Next Interval Register 0 (TNIR0)

TNIR0 is for timer 0.

ADDRESS

2014 0108 (SSC)



bits<31:0>

Name: Timer Next Interval Register 0

Mnemonic: TNIR0

Type: R/W, 0

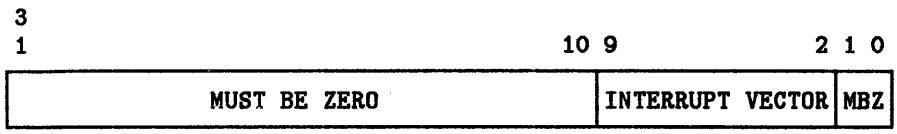
TNIR0 contains the value that is written into TIR0 after an overflow or in response to TCR0<4> (XFR).

Timer Interrupt Vector Register 0 (TIVR0)

TIVR0 is used by timer 0.

ADDRESS

2014 010C (SSC)



bits<31:10>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bits<9:2>

Name: Interrupt Vector

Mnemonic: IV

Type: R/W, 0

When TCR0<6> (IE) and TCR0<7> (INT) transition to a one, an interrupt is posted at IPL 15. When a timer's interrupt is acknowledged, the contents of IV are passed to the CPU and TCR0<7> is cleared. Interrupt requests are also cleared by clearing TCR0<6> or TCR0<7>. IV is set to 78 (hex) by console code during processor initialization.

bits<1:0>

Name: Reserved

Mnemonic: None

Type: R/W, 0

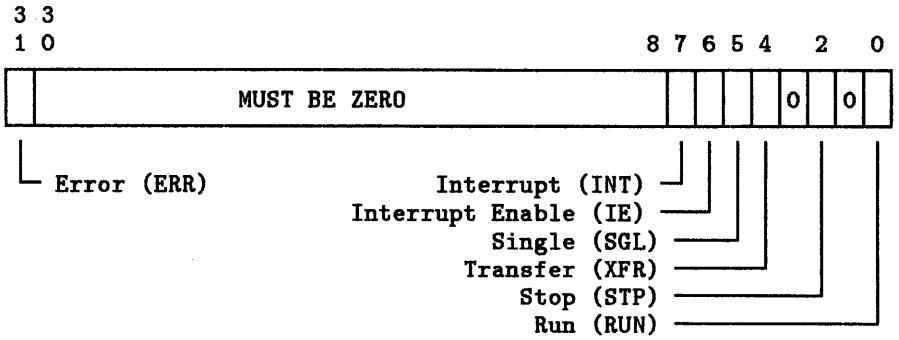
Reserved; must be zero.

Timer Control Register 1 (TCR1)

TCR1 controls timer 1, which is used by the console code.

ADDRESS

2014 0110 (SSC)



bit<31>

Name: Error

Mnemonic: ERR

Type: R/W1C, 0

ERR is set whenever the Timer Interval Register overflows and INT is already set, indicating a missed overflow.

bit<30:8>

Name: Reserved

Mnemonic: None

Type: R/W

Reserved; must be zero.

bit<7>

Name: Interrupt

Mnemonic: INT

Type: R/W1C, 0

INT is set whenever the Timer Interval Register overflows. If IE is set when INT is set, an interrupt is posted at IPL 15.

bit<6>

Name: Interrupt Enable
Mnemonic: IE
Type: R/W, 0

When IE is set, the timer interrupts at IPL 15 when INT is set.

bit<5>

Name: Single
Mnemonic: SGL
Type: R/W, 0

Setting SGL causes the Timer Interval Register to increment by one if the RUN bit is cleared. If RUN is set, then writes to SGL are ignored. SGL is always read as zero.

bit<4>

Name: Transfer
Mnemonic: XFR
Type: R/W, 0

Setting XFR causes the Timer Next Interval Register to be copied into the Timer Interval Register.

bit<3>

Name: Reserved
Mnemonic: None
Type: R/W

Reserved; must be zero.

bit<2>

Name: Stop
Mnemonic: STP
Type: R/W, 0

STP determines whether the timer stops after an overflow. If STP is set at overflow, RUN is cleared by the hardware at overflow and counting stops.

bit<1>

Name: Reserved
Mnemonic: None
Type: R/W

Reserved; must be zero.

bit<0>

Name: Run

Mnemonic: RUN

Type: R/W, 0

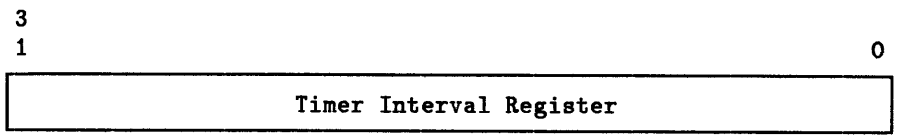
When RUN is set, the Timer Interval Register is incremented once every microsecond. INT is set when the timer overflows. If STP is set at overflow, RUN is cleared by the hardware at overflow and counting stops. When RUN is clear, the Timer Interval Register is not incremented automatically.

Timer Interval Register 1 (TIR1)

TIR1 contains the interval count for timer 1, which is used by console code.

ADDRESS

2014 0114 (SSC)



bits<31:0>

Name: Timer Interval Register 1

Mnemonic: TIR1

Type: RO, 0

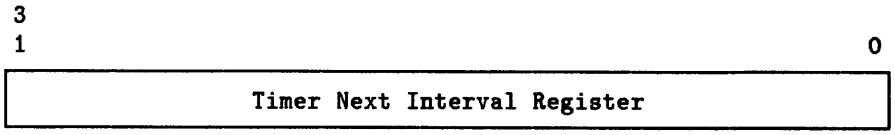
When TCR1<0> (RUN) is one, the register is incremented once every microsecond. When the counter overflows, TCR1<7> is set, and an interrupt is posted at IPL 15 if TCR1<6> is set. Then, if TCR1<2> is zero, TCR1<0> is cleared and counting stops.

Timer Next Interval Register 1 (TNIR1)

TNIR1 is for timer 1, which is used by console code.

ADDRESS

2014 0118 (SSC)



bits<31:0>

Name: Timer Next Interval Register 1

Mnemonic: TNIR0

Type: R/W, 0

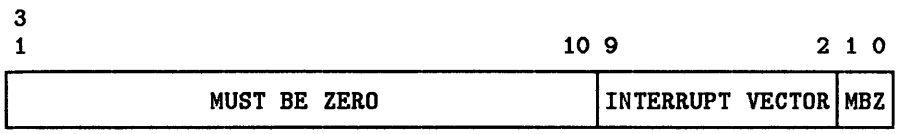
TNIR1 contains the value that is written into TIR1 after an overflow or in response to TCR1<4> (XFR).

Timer Interrupt Vector Register 1 (TIVR1)

TIVR1 is used by timer 1, which is used by console code.

ADDRESS

2014 011C (SSC)



bits<31:10>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bits<9:2>

Name: Interrupt Vector

Mnemonic: IV

Type: R/W, 0

When TCR1<6> (IE) and TCR1<7> (INT) transition to a one, an interrupt is posted at IPL 15. When a timer's interrupt is acknowledged, the contents of IV are passed to the CPU and TCR1<7> is cleared. Interrupt requests are also cleared by clearing TCR1<6> or TCR1<7>. IV is set to 7C (hex) by console code during processor initialization.

bits<1:0>

Name: Reserved

Mnemonic: None

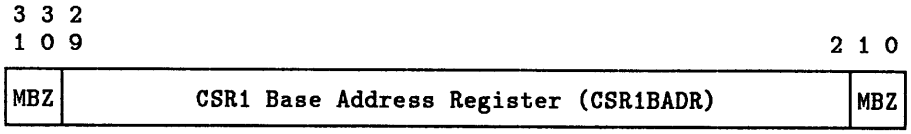
Type: R/W, 0

Reserved; must be zero.

CSR1 Base Address Register (CSR1BADR)

CSR1BADR controls the address of CSR1.

ADDRESS *2014 0130 (SSC)*



bits<31:30>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bits<29:2>

Name: CSR1 Base Address Register

Mnemonic: CSR1BADR

Type: R/W, 0

CSR1BADR controls the address of CSR1 and is set to 2000 0000 (hex) by console code during processor initialization.

bits<1:0>

Name: Reserved

Mnemonic: None

Type: R/W, 0

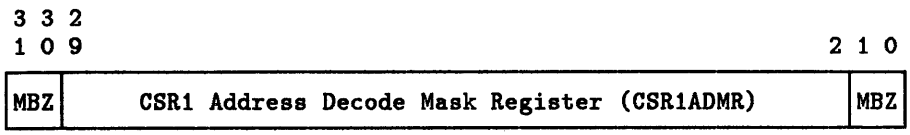
Reserved; must be zero.

CSR1 Address Decode Mask Register (CSR1ADMR)

CSR1ADMR controls the addresses that select CSR1.

ADDRESS

2014 0134 (SSC)



bits<31:30>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bits<29:2>

Name: CSR1 Address Decode Mask Register

Mnemonic: CSR1ADMR

Type: R/W, 0

CSR1ADMR controls the addresses that select CSR1 and is set to 0000 0004 (hex) by console code during processor initialization.

bits<1:0>

Name: Reserved

Mnemonic: None

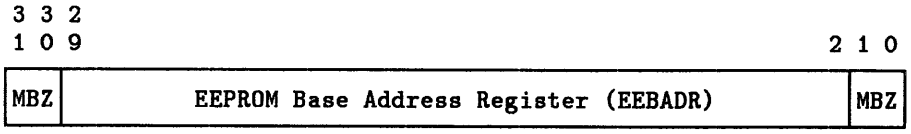
Type: R/W, 0

Reserved; must be zero.

EEPROM Base Address Register (EEBADR)

EEBADR specifies the base address of the EEPROM.

ADDRESS *2014 0140 (SSC)*



bits<31:30>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bits<29:2>

Name: EEPROM Base Address Register
Mnemonic: EEBADR
Type: R/W, 0

EEBADR specifies the base address of the EEPROM and is set to 2008 0000 (hex) by console code during processor initialization.

bits<1:0>

Name: Reserved
Mnemonic: None
Type: R/W, 0

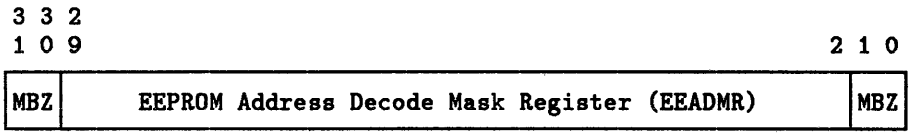
Reserved; must be zero.

EEPROM Address Decode Mask Register (EEADMR)

EEADMR specifies the addresses that select the EEPROM.

ADDRESS

2014 0144 (SSC)



bits<31:30>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bits<29:2>

Name: EEPROM Address Decode Mask Register
Mnemonic: EEADMR
Type: R/W, 0

EEADMR specifies the addresses that select the EEPROM and is set to 0000 7FFF (hex) by console code during processor initialization.

bits<1:0>

Name: Reserved
Mnemonic: None
Type: R/W, 0

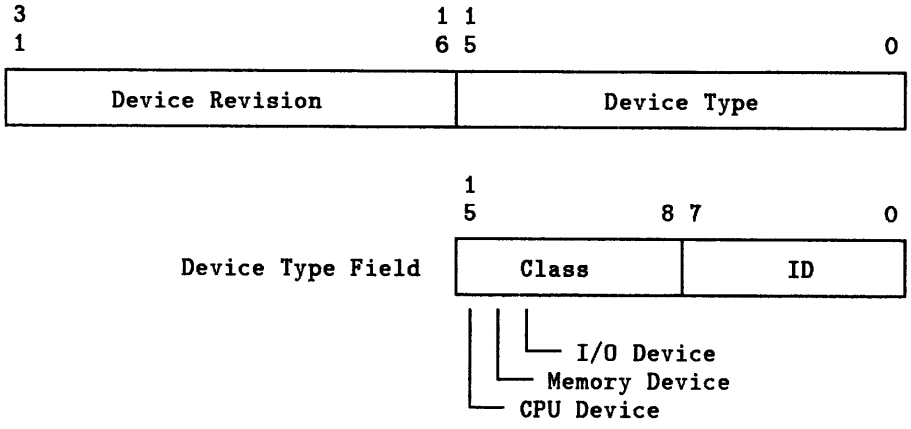
Reserved; must be zero.

Device Register (XDEV)

The Device Register contains information to identify the node. Both fields are loaded during node initialization. A zero value indicates an uninitialized node.

ADDRESS

Nodespace base address + 0000 0000 (XCPGA)



bits<31:16>

Name: Device Revision

Mnemonic: DREV

Type: R/W, 0

Identifies the functional revision level of the module in hexadecimal. The DREV field always reflects the letter revision of the module as follows:

KA62A CPU Module Revision	DREV (decimal)	DREV (hex)
A0	1	0001
A1	1	0001
B0	2	0002
B1	2	0002
.		
.		
.		
Z0	26	001A

bits<15:0>

Name: Device Type

Mnemonic: DTYPE

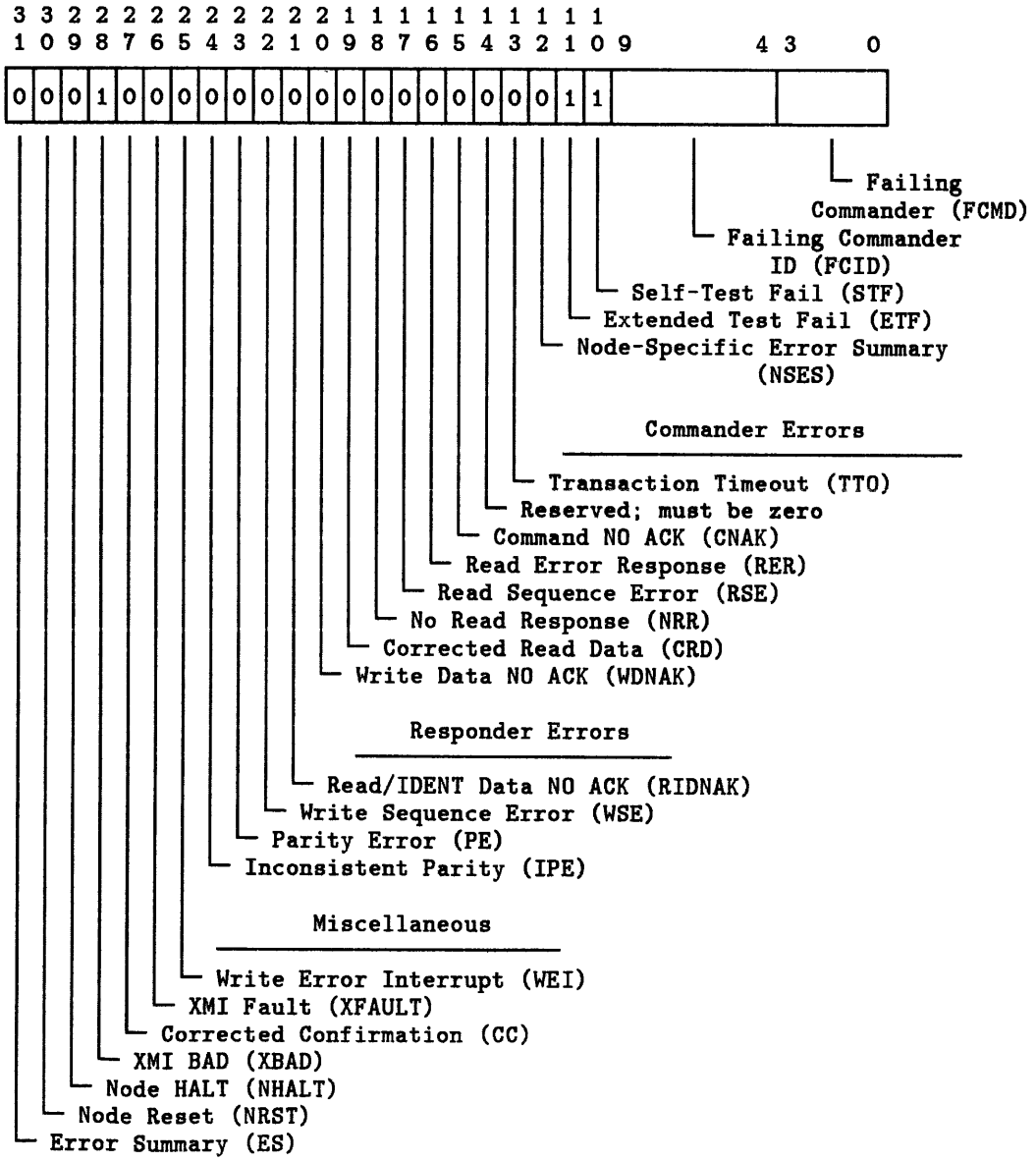
Type: R/W, 0

Identifies the type of node. The Device Type field is broken into two subfields: Class and ID. The Class field indicates the major category of the node. The ID field uniquely identifies a particular device within a specified class. DTYPE contains 8001 (hex) for the KA62A CPU module.

Bus Error Register (XBER)

The Bus Error Register contains error status on a failed XMI transaction. This status includes the failed command, commander ID, and an error bit that indicates the type of error that occurred. This status remains locked up until software resets the error bit(s).

ADDRESS *Nodespace base address + 0000 0004 (XCPGA)*



bit<31>

Name: Error Summary
Mnemonic: ES
Type: RO, 0

The state of ES represents the logical-OR of the error bits in this register. Therefore, ES is asserted if any error bit is asserted.

bit<30>

Name: Node Reset
Mnemonic: NRST
Type: R/W, 0

Writing a one to NRST initiates a complete power-up reset similar to the assertion and deassertion of XMI DC LO L (see note below); the node performs self-test and asserts XMI BAD L until self-test is successfully completed. Like power-up reset, nodes are precluded from accessing the node from the time it is node reset until it completes self-test (or the maximum self-test time is exceeded).

NOTE: During the time that a node is responding to node reset, the node does not access other nodes on the XMI. In response to a real power-up sequence (caused by XMI DC LO L), the NRST bit resets. Following a node reset sequence, NRST remains set, allowing the processor to recognize that it should not attempt to go through the normal boot process.

bit<29>

Name: Node HALT
Mnemonic: NHALT
Type: R/W, 0

Writing a one to NHALT forces the node to go into a "quiet" state while retaining as much state as possible. The KA62A CPU module will force the CVAX chip to HALT and go into console mode waiting for console commands.

bit<28>

Name: XMI BAD
Mnemonic: XBAD
Type: R/W, 1

On reads, XBAD indicates the state of the XMI BAD signal. A one indicates that BAD is asserted. Writes to XBAD cause the state to be driven on the wired-OR XMI BAD L line by this node; writing a one asserts XMI BAD L, while writing a zero releases it.

bit<27>

Name: Corrected Confirmation

Mnemonic: CC

Type: R/W1C, 0

CC sets when the node detects a single-bit CNF error. Single-bit CNF errors are automatically corrected by the XCLOCK chip.

Error Flag Asserted: CRD

Additional Status Stored: None

Action: Since the ACK/NAK is usable, no further action is needed.

bit<26>

Name: XMI FAULT

Mnemonic: XFAULT

Type: R/W1C, 0

When set, XFAULT indicates that the XMI FAULT signal has been asserted for at least one cycle. An XMI node asserts FAULT to indicate that it has sensed a Transmit Error (data transmitted onto the XMI does not compare with data received during the same cycle) on a cycle that was ACKed.

Error Flag Asserted: MEMERR

Additional Status Stored: None

Action: MEMERR is also generated if XFAULT is asserted by another XMI node, providing systemwide coverage of a connector or multiple transmitter failure.

bit<25>

Name: Write Error Interrupt

Mnemonic: WEI

Type: R/W1C, 0

When set, WEI indicates that the node has received a write error interrupt transaction (IVINTR).

Error Flag Asserted: MEMERR

Additional Status Stored: None

Action: CVAX polls nodes to determine source and cause.

bit<24>

Name: Inconsistent Parity Error

Mnemonic: IPE

Type: R/W1C, 0

When set, IPE indicates that the node has detected a parity error on an XMI cycle and the confirmation for the errored cycle was ACK. This indicates that at least one node (the responder) detected good parity during the cycle time that this node detected a parity error. If this was a successful write to memory, it could leave the cache incoherent.

Error Flag Asserted: CRD

Additional Status Stored: None

Action: KA62A CPU module hardware disables the second-level cache by asserting CSR1<18> (Force Miss, FMISS). Software flushes the second-level cache by writing a one, then a zero, to CSR1<20> (Force Cache Invalidate, FCI)

bit<23>

Name: Parity Error

Mnemonic: PE

Type: R/W1C, 0

When set, PE indicates that the node has detected a parity error on an XMI cycle.

Error Flag Asserted: CRD

Additional Status Stored: None

Action: Appropriate error recovery is initiated when PE is set.

bit<22>

Name: Write Sequence Error

Mnemonic: WSE

Type: R/W1C, 0

Node aborted write transaction due to missing data cycles.

Error Flag Asserted: No Interrupt

Additional Status Stored: None

Action: Write to CSR is not performed. WSE bit sets but the commander of the issuing node is responsible for error recovery.

bit<21>

Name: READ/IDENT Data NO ACK

Mnemonic: RIDNAK

Type: R/W1C, 0

When set, RIDNAK indicates that a read data cycle (GRDn, CRDn, LOC, RER) transmitted by the node has received a NO ACK confirmation. The KA62A CPU module does not respond to IDENT transactions.

Error Flag Asserted: No Interrupt

Additional Status Stored: None

Action: When read data sent by the responder does not get ACKed, the responder causes RIDNAK to set; but it is the commander of the issuing node that is responsible for error recovery.

bit<20>

Name: Write Data No Ack

Mnemonic: WDNAK

Type: R/W1C, 0

When set, WDNAK indicates that a write data cycle transmitted by the node has received a NO ACK confirmation. WDNAK sets only if the reattempt fails.

Error Flag Asserted: MEMERR

Additional Status Stored: Failing Address (XFADR), Commander ID, and Command.

Action: The transaction is reattempted until a timeout occurs. Failed address is saved.

bit<19>

Name: Corrected Read Data

Mnemonic: CRD

Type: R/W1C, 0

When set, CRD indicates that the node has received a CRDn read response, meaning that the read transaction was received by memory with bad parity but memory corrected it.

Error Flag Asserted: CRD

Additional Status Stored: None

Action: Since the data is usable, no further action is necessary.

bit<18>

Name: No Read Response

Mnemonic: NRR

Type: R/W1C, 0

When set, NRR indicates that a transaction initiated by the node failed due to a read response timeout.

Error Flag Asserted (READ): ERR if during the first quadword, then CFE during cache fill.

Error Flag Asserted (IDNET): MEMERR

Additional Status Stored: Failing Address (XFADR), Command ID, and Command.

Action: No retry is attempted. CVAX does a machine check. Failed address is saved.

bit<17>

Name: Read Sequence Error

Mnemonic: RSE

Type: R/W1C, 0

When set, RSE indicates that a transaction initiated by the node failed due to a read sequence error, meaning that data which is returned as the result of a read transaction or an interrupt vector which is returned in an IDENT transaction is identified as being out of sequence.

Error Flag Asserted (READ): ERR if during the first quadword, then CFE during cache fill.

Error Flag Asserted (IDENT): MEMERR

Additional Status Stored: Failing Address (XFADR), Commander ID, and Command.

Action: CVAX does a machine check. Failed address is saved. No retry is attempted.

bit<16>

Name: Read Error Response

Mnemonic: RER

Type: R/W1C, 0

When set, RER indicates that a node has received a Read Error Response, meaning that the result of a read transaction or an interrupt vector returned in an IDENT transaction is uncorrectable.

Error Flag Asserted (READ): ERR if during the first quadword, then CFE during cache fill.

Error Flag Asserted (IDENT): MEMERR

Additional Status Stored: Failing Address (XFADR), Command ID, and Command.

Action: CVAC does a machine check. Failed address is saved. No retry is attempted.

bit<15>

Name: Command NO ACK

Mnemonic: CNAK

Type: R/W1C, 0

When set, CNAK indicates that a command cycle transmitted by the node has received a NO ACK confirmation caused by either a reference to a nonexistent memory location or a command cycle parity error. This bit is set only if the error recovery reattempts fail.

Error Flag Asserted (READ): ERR

Error Flag Asserted (WRITE/IDENT): MEMERR

Additional Status Stored: Failing Address (XFADR), Commander ID, and Command.

bit<14>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bit<13>

Name: Transaction Timeout
Mnemonic: TTO
Type: R/W1C, 0

When set, TTO indicates that a transaction initiated by the node failed due to a transaction timeout. This bit is set only if the error recovery reattempt fails.

Error Flag Asserted: Varies, depends on the transaction causing the error.

Additional Status Stored: Failing Address (XFADR), Command ID, and Command.

Action: Depends on whether a read or write error caused TTO to set. TTO always sets in conjunction with another error, and the other error bit determines the appropriate action.

bit<12>

Name: Node-Specific Error Summary
Mnemonic: NSES
Type: RO, 0

When set, NSES indicates that a node-specific error condition has been detected. The exact nature of the error is contained in node-specific registers.

bit<11>

Name: Extended Test Fail
Mnemonic: ETF
Type: R/W1C, 1

When set, ETF indicates that the node has not yet passed its extended test. This bit clears when the node passes its extended test.

bit<10>

Name: Self-Test Fail
Mnemonic: STF
Type: R/W1C, 1

When set, STF indicates that the node has not yet passed its self-test. This bit is cleared by the user interface when the node passes its self-test.

bits<9:4>

Name: Failing Commander ID
Mnemonic: FCID
Type: RO

FCID logs the commander ID of a failing transaction.

bits<3:0>

Name: Failing Command
Mnemonic: FCMD
Type: RO

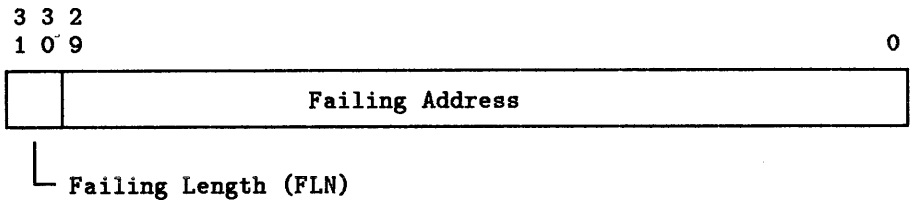
FCMD logs the command code of a failing transaction.

Failing Address Register (XFADR)

The Failing Address Register logs address and length information associated with a failing transaction. The XFADR has an undetermined value on power-up.

ADDRESS

Nodespace base address + 0000 0008 (SSC)



bits<31:30>

Name: Failing Length

Mnemonic: FLN

Type: RO

FLN logs the value of XMI D<31:30> during the command cycle of a failing transaction.

bits<29:0>

Name: Failing Address

Mnemonic: None

Type: RO

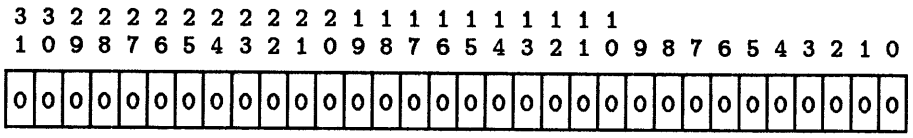
The Failing Address field logs the value of XMI D<29:0> during the command cycle of a failing transaction.

XMI General Purpose Register (XGPR)

The XGPR is a general purpose register that is visible to the XMI. This register is used during self-test and by the ROM-based diagnostics.

ADDRESS

Nodespace base address + 0000 000C (XCPGA)



bits<31:0>

Name: XMI General Purpose Register

Mnemonic: XGPR

Type: R/W 0

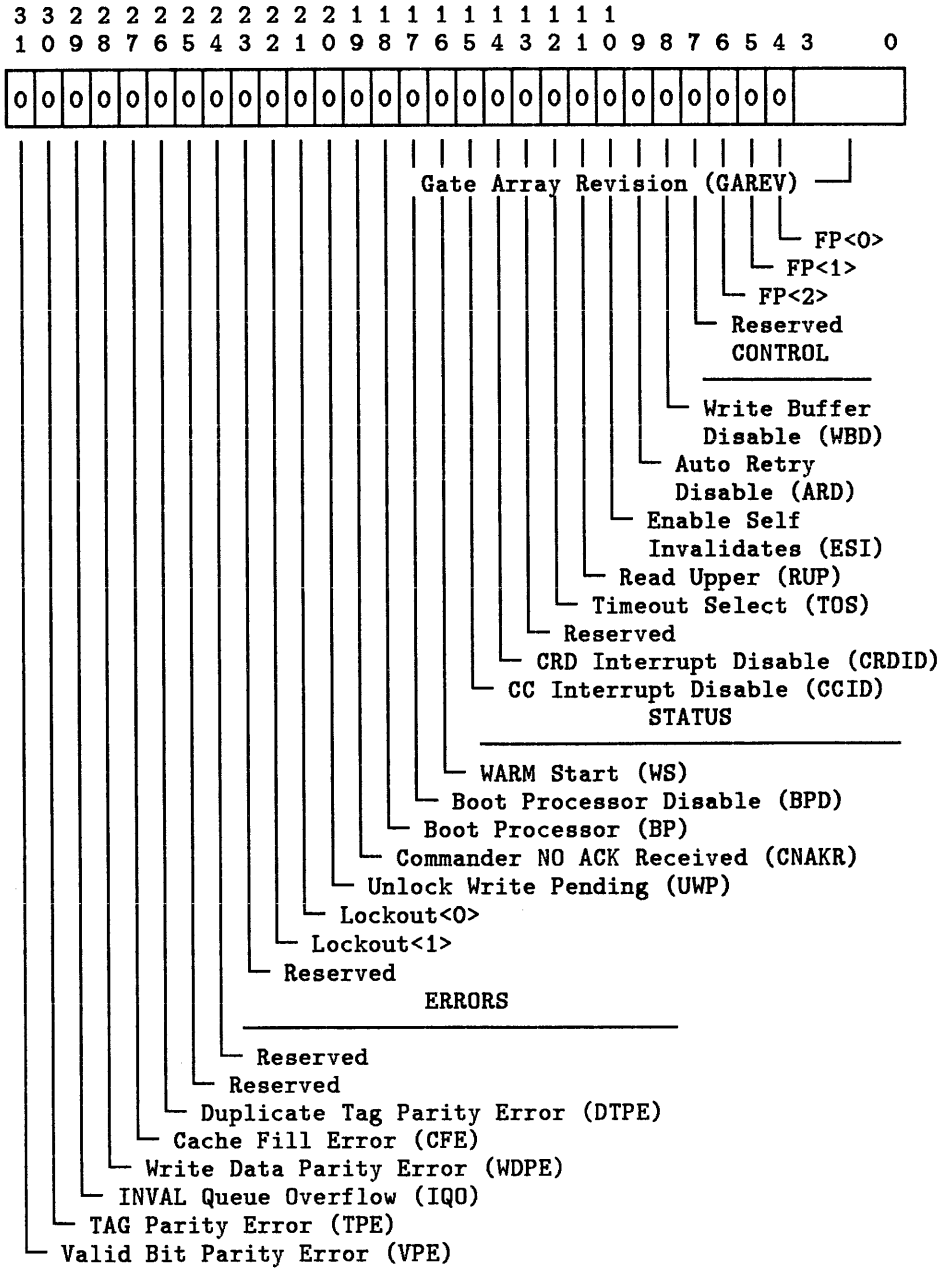
The general purpose register is used by self-test and during ROM-based diagnostics.

Control and Status Register 2 (CSR2)

CSR2 provides KA62A CPU module control and status to the XMI.

ADDRESS

Nodespace base address + 0000 0010 (SSC)



bit<31>

Name: Valid Bit Parity Error

Mnemonic: VPE

Type: R/W1C, 0

VPE is set when a second-level cache subblock valid bit parity error is detected.

Error Flag Asserted: CRD

Additional Status Stored: None

Action: VPE causes a cache miss, resulting in a hexword fill on a read. On a write, VPE results in a failue to update the cache. The KA62A CPU module hardware flushes the second-level cache by asserting Force Miss (CSR1<18> (FMISS)). Software flushes the second-level cache by writing a one, then a zero, to Force Cache Invalidate (CSR1<20> (FCI)).

bit<30>

Name: Tag Parity Error

Mnemonic: TPE

Type: R/W1C, 0

TPE is set when an external cache tag parity error is detected.

Error Flag Asserted: CRD

Additional Status Stored: None

Action: TPE causes a cache miss, resulting in a hexword fill on a read. On a write, TPE results in a failure to update the cache. The KA62A CPU module hardware disables the second-level cache by asserting Force Miss (CSR1<18> (FMISS)). Software flushes the second-level cache by writing a one, then a zero, to Force Cache Invalidate (CSR1<20> (FCI)).

bit<29>

Name: INVAL Queue Overflow

Mnemonic: IQO

Type: R/W1C, 0

IQO is set whenever the INVAL queue overflows. The KA62A CPU module's cache is flushed when this error occurs to ensure cache coherency. When IQO is set, the INVAL queue in the processor is held clear.

Error Flag Asserted: CRD

Additional Status Stored: None

Actions: KA62A CPU module hardware disables the second-level cache by asserting Force Miss (CSR1<18> (FMISS)). Software flushes the second-level cache by writing a one, then a zero, to Force Cache Invalidate (CSR1<20> (FCI)).

bit<28>

Name: Write Data Parity Error

Mnemonic: WDPE

Type: R/W1C, 0

WDPE is set whenever a parity error is detected on write data driven by the processor on the CDAL bus.

Error Flag Asserted: MEMERR

Additional Status Stored: None

Actions: The write transaction is not allowed to proceed onto the XMI. If a write buffer hit, then data is not loaded into the write buffer. The failing address is not saved by the pinout error logic.

bit<27>

Name: Cache Fill Error

Mnemonic: CFE

Type: R/W1C, 0

CFE is set whenever a second-level cache fill error occurs. Second-level cache fill errors are soft errors that occur on the 2nd, 3rd, or 4th quadword of the second-level cache fills. CFE is always set in conjunction with other error bits.

Whenever an error occurs on the data being returned to the CPU, the second-level cache is disabled because CSR1<FMISS> asserts.

Error Flag Asserted: CRD

Additional Status Stored: Failing Address (XFADR), Command ID, and Command (XBER)

Action: The Valid Parity Error bit is not set at the completion of a hexword read. The resulting invalid subblock causes a cache miss when addressed.

bit<26>

Name: Duplicate Tag Parity Error

Mnemonic: DTPE

Type: R/W1C, 0

DTPE is set whenever the duplicate tag store detects a parity error on lookup. Since this error could result in a second-level cache coherency problem (the write might have hit if the parity error had not occurred and resulted in the generation of an invalidate) the KA62A CPU module hardware disables the second-level cache when this error occurs and posts a soft-error interrupt.

Error Flag Asserted: CRD

Additional Status Stored: None

Action: DTPE causes a miss, which if a memory write, results in a potential second-level cache coherency problem. KA62A CPU module hardware disables the second-level cache by asserting Force Miss (CSR1<18> (FMISS)). Software flushes the second-level cache by writing a one, then a zero, to Force Cache Invalidate (CSR1<20> (FCI))

bits<25:23>

Name: Reserved

Mnemonic: None

Type: –

Reserved.

bits<22:21>

Name: Lockout<1:0>

Mnemonic: None

Type: R/W, 01

The KA62A CPU module supports a lockout avoidance mechanism that assures access to interlock variables. Lockout<1:0> controls these mechanisms as follows:

Bits<22:21>

22	21	Description
0	0	Interlock lockout avoidance is disabled but XMI LOCKOUT L is still asserted as defined for Lockout<1:0> = 01.
0	1	Interlock lockout avoidance is enabled.
1	0	Reserved
1	1	Reserved

bit<20>

Name: Unlock Write Pending
Mnemonic: UWP
Type: R/W1C, 0

UWP is set whenever an Interlock Read is generated and is cleared on the subsequent Unlock Write from the same node. The setting and clearing of this bit is not gated by the successful transmission of the XMI transaction.

bit<19>

Name: Commander NO ACK Received
Mnemonic: CNAKR
Type: R/W1C, 0

CNAKR is set whenever a command/address NO ACK is received to an XMI commander transfer. A NO ACK is not necessarily an error on the XMI as it is used for retries, but this status bit is used by diagnostics that wish to know whether a transfer was NO ACKed. The KA62A CPU module automatically reattempts all XMI transfers that are NO ACKed until a timeout occurs, unless CSR2<9> (ARD) is set.

bit<18>

Name: Boot Processor
Mnemonic: BP
Type: R/W, 0

BP is used to indicate the boot processor. The console code sets this bit after self-test if it determines that this KA62A CPU module is the CPU with the lowest node ID number with its CSR2:BPD bit clear.

bit<17>

Name: Boot Processor Disable
Mnemonic: BPD
Type: R/W, 0

BPD is used to indicate that a KA62A CPU module is ineligible to become the boot processor. It is loaded by console code on power-up with a state stored in EEPROM.

bit<16>

Name: Warm Start
Mnemonic: WS
Type: RO

When WS is set, it indicates that memory was battery backed up and that the system should attempt a warm start. WS is loaded with the state of the XMI RESET L line when the XMI DC LO L line deasserts (a deasserted XMI RESET L indicates warm start). WS is not valid after a node reset.

bit<15>

Name: CC Interrupt Disable
Mnemonic: CCID
Type: R/W, 0

CCID disables the generation of error interrupts to the KA62A CPU module processor in response to corrected confirmation indications from the XMI. While CCID is set, XBER<27> (CC) bit will still be set on the receipt of a corrected confirmation code but the processor will not be interrupted. When reset, the CRD line asserts when a corrected confirmation code is received from the XMI (XBER <27> also sets).

bit<14>

Name: CRD Interrupt Disable
Mnemonic: CRDID
Type: R/W, 0

CRDID disables the generation of error interrupts to the processor in response to Corrected Read Data responses from memory. While CRDID is set, the XBER<19> (CRD) bit will still be set on the receipt of a Corrected Read Data response but the processor will not be interrupted. When reset, the CRD line will assert when a Corrected Read Data response is received from the XMI (XBER<19> (CRD) bit will also be set). Software should clear XBER<19> (CRD) before clearing CRDID to ensure that only newly generated CRD responses cause interrupts.

bit<13>

Name: Reserved
Mnemonic: None
Type: -

Reserved.

bit<12>

Name: Timeout Select
Mnemonic: TOS
Type: R/W, 0

TOS selects one of two timeout values (0 selects ≈16.77 ms; 1 selects ≈16.38 us). This timeout value is used to detect both Response and Reattempt Timeout conditions. This bit remains clear during normal system operation.

bit<11>

Name: Enable Read Upper
Mnemonic: ERUP
Type: R/W, 0

When ERUP is set, the upper longword of the data driven on the XMI is returned in response to an I/O space read. Normally, the lower longword is returned. ERUP is used during self-test to test the logic and pins associated with the upper longword of the XMI data path.

bit<10>

Name: Enable Self-Invalidates
Mnemonic: ESI
Type: R/W, 0

When ESI is set, the processor will also invalidate cache entries matching its own XMI write addresses. Normally, since the cache is write through, only writes from other XMI nodes generate invalidates. ESI is used for testing because it permits a single processor, in conjunction with XMI memory, to verify the operation of its invalidate logic.

bit<9>

Name: Auto Retry Disable
Mnemonic: ARD
Type: R/W, 0

ARD disables auto retry of NO ACKed XMI commander transfers and causes the immediate return of an error response after the receipt of a NO ACK confirmation to a commander transfer. ARD is only used by diagnostics and must be clear during normal operation.

bit<8>

Name: Write Buffer Disable

Mnemonic: WBD

Type: R/W, 0

WBD disables the write buffer so that all writes are written directly to main memory. Logically, the write logic is forced to assume that all writes are to I/O space and this automatically forces the write buffer function to be bypassed.

bit<7>

Name: Reserved

Mnemonic: None

Type: –

Reserved.

bits<6:4>

Name: Force Parity <2:0>

Mnemonic: FP

Type: R/W, 0

FP is used to provide the parity states for XMI P<2:0> when CSR1<22> (Force Parity Select) is set.

bits<3:0>

Name: Gate Array Revision

Mnemonic: GAREV

Type: RO

GAREV contains the revision level of the KA62A CPU module gate array.

3.7 KA62A CPU Module Initialization, Self-Test, and Booting

This section give the KA62A CPU module initialization overview; describes the initialization in detail; and then discusses the bootstrapping or restarting of the operating system.

3.7.1 Initialization Overview

The three ways to reset the KA62A CPU module are:

- Power-Up Sequence—When the VAX 6200 is powered up, XMI AC LO L and XMI DC LO L are sequenced so that all XMI nodes are reset.
- System Reset—The XMI emulates a power-up sequence by asserting the XMI RESET L line, causing the power supply to sequence XMI AC LO L and XMI DC LO L as in a "real" power-up. Software asserts XMI RESET L by writing to IPR55. The XMI does not differentiate between a "real" power-up and a system reset. The console INITIALIZE command generates a system reset if no argument is given.
- Node Reset—Any CPU can be "node reset" by setting its XBER<30> (NRST) bit. The console INITIALIZE command generates a node reset if a node ID argument is provided. For the KA62A CPU module the difference between the node reset and a system reset is that XMI AC LO L is not sequenced during node reset.

3.7.1.1 Initialization Description

In response to a power-up or system reset, the KA62A CPU module(s) perform the following sequence:

1. Reset(s) to a known state. (Refer to the individual registers and their bits for their state on reset.)
2. All CPUs start executing code at 2004 0000 in ROM and execute a complete self-test.
3. Select a boot CPU, which prints self-test results and configures memory for the CPU/MEM test.
4. All KA62A CPU modules execute a CPU/MEM test that uses memory, unless this is a warm start, where the memory was maintained by battery backup.
5. Again select a boot CPU. The boot CPU:
   - Prints CPU/MEM test results.
   - Tests the DWMBA.
   - Prints the DWMBA test and corresponding VAXBI self-test results.
   - Configures memory.
   - Prints the memory configuration.
   - Starts the operating system if the front panel key switch is in the Restart position; otherwise, the boot CPU enters console input mode.

If an individual KA62A CPU module node is reset, only steps 1 and 2 apply, except a boot CPU also prints its self-test results.

See Section 3.7.2 for a detailed flowchart and summary of the initialization program.

3.7.1.2 CPU Self-Test

Self-test verifies all KA62A CPU module logic that does not require access to other XMI nodes. XMI intranode transactions to the KA62A CPU module's I/O registers are used extensively to verify the XMI data path logic.

CSR1<7> (STL) may be set by grounding an I/O pin. The Self-Test Loop bit is used to implement a manufacturing "burn-in" test.

3.7.1.3 CPU/MEM Test

While CPU self-test does not access other XMI nodes, the CPU/MEM test does. Read and Writes are done to a defined section of XMI memory as blocks are allocated based on node ID. Reads/Writes to the MS62A memory module's XMI registers are performed. This ability to access memory allows the testing of logic that cannot be tested in self-test. The areas tested in the CPU/MEM test but not in the CPU self-test are:

- The MS62A memory module's memory and XMI Corner. Self-test is limited to I/O space.
- Quad-, octa-, and hexword XMI transactions.
- Write Buffer/read queue functionality.
- Duplicate tag store/inval queue functionality.
- Parts of the second-level cache functionality.
- Certain error conditions.

3.7.2 Detailed Initialization Description

The following is a flowchart and summary of the initialization program:

Figure 3–18 Initialization Flowchart, Part 1 of 2

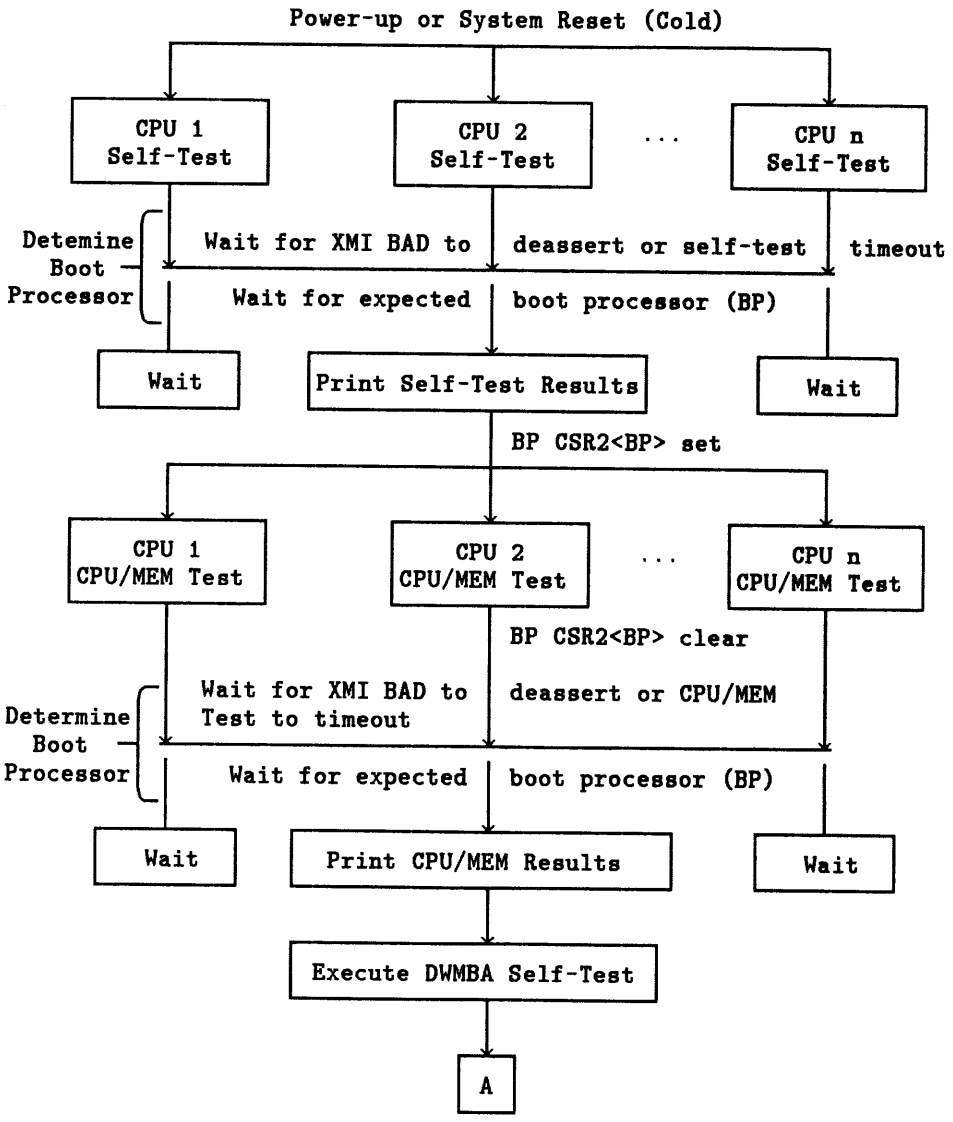
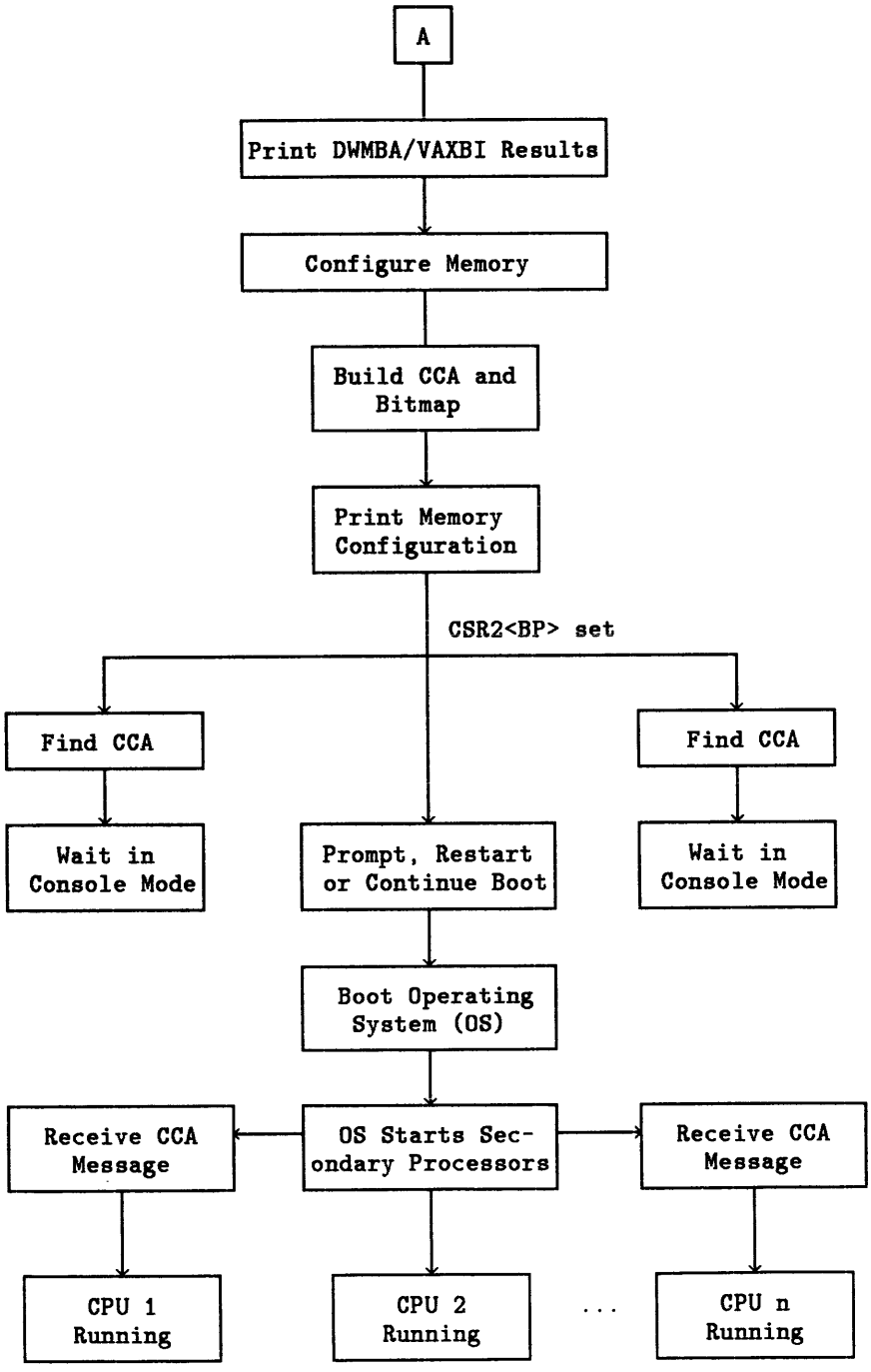


Figure 3–19 Initialization Flowchart, Part 2 of 2



3.7.2.1 Determine Type of Restart

- If entering console because of anything other than XCP RESET, then print error code and wait in console input loop.

  else continue

3.7.2.2 CPU Self-Test

All processors perform the following in parallel:

- The following states are initialized by node hardware in response to XCP Reset: Set Self-Test Failed and Extended Test Failed bits (XBER<10> (STF) and XBER<11> (ETF)). Assert XMI BAD L (XBER<28> (XBAD)) and extinguish the Self-Test Passed LED (STPLED).
- Start self-test duration timer.
- Execute normal CPU self-test.
- Load XDEV's device type and device revision fields (this can be done anytime while XBER<20> (XBAD) is asserted and the self-test timer has not expired, but is attempted regardless of whether the self-test passes or fails).
- If self-test passes, clear XBER<10> (STF) and light STPLED.
- If self-test fails, then self-test leaves the results in a CPU location.
- Load the XMI-visible Boot Processor Disable Bit with state stored in the EEPROM's boot processor enable flag.
- If self-test passes, deassert XMI BAD L driver (XBER<28> (XBAD)).

The last step is performed since processors begin polling nodespace registers after XMI BAD L deasserts and expect node state to be consistent (for example, a node would not deassert XMI BAD L prior to the loading of the Device Register or the XMI-visible Boot Processor Disable Bit).

3.7.2.3 Determine the Boot Processor

All processors perform the following in parallel:

- Wait for deassertion of XMI BAD L (XBER<28> (XBAD)) or for self-test timer to reach its programmed expiration (60 seconds by default).
- If this node is the lowest node ID CPU class node (indicated by set XDEV<15>) which cleared its XBER<10> (STF) bit and cleared its Boot Processor Disable bit, this processor enables its tristate system console.
- If this is not the boot processor (BP), wait for expected BP to set its CSR2<BP> bit.

NOTE: If all CPUs have failed their self-test or are ineligible to be the BP, they will all be sitting in a console "hang" loop. In this case, none will be driving the console line, but all can still receive the console command, ">>x", where x is the node ID, which forces a CPU to become boot processor.

This "hang" loop alternates the display of one with the previous contents of the auxiliary LEDs. The previous contents would be either a self-test error code or no code.

- The BP prints a message indicating the status of testing so far.
- The BP set its boot processor bit indicating to other CPUs that this KA62A CPU module will be the BP.

3.7.2.4 CPU/MEM Test

Boot processor prints self-test results and all processors perform the following in parallel:

- All processors wait for CSR2<BP> to clear before continuing.
- If memory battery voltage OK, then clear XBER<11> (ETF) bits (since the CPU/MEM test will not be run) and go to FINAL-STEPS.
- Start new CPU/MEM test timer.
- Assert XMI BAD L (XBER<28> (XBAD)), disable its tristate system console, and extinguish STPLED.
- Execute CPU/MEM test.
- If the CPU/MEM test fails, then leave code in CPU register.
- If CPU/MEM test passes, then clear XBER<11> (ETF), light STPLED to reflect the successful test, and deassert the XMI BAD L driver.
- After CPU/MEM test is complete, wait for all XMI BAD L to clear or for timer to expire.

FINAL-STEPS:

- Check if this node is the lowest node ID CPU class node (indicated by set XDEV<15> (CPU Device)) which cleared its XBER<10> (STF) and XBER<11> (ETF) bits and cleared its Boot Processor Disable bit. This processor enables its tristate system console.
- If this is not the boot processor, wait for expected BP to set its CSR2<BP> bit.

NOTE: **If all CPUs have failed their self-test or CPU/MEM test or are ineligible to be the BP, they will all be sitting in a console "hang" loop. In this case, none will be driving the console line, but all can still receive the console command, ">>x", where x is the node ID, which forces a CPU to become boot processor.**

This "hang" loop alternates the display of one with the previous contents of the auxiliary LEDs. The previous contents would be either a CPU/MEM test error code or no code.

- If this is not the boot processor, wait for expected BP to set its CSR2<BP> bit. When the bit sets, find the console communications area (CCA) and then wait in console mode.

NOTE: **If the processor fails to find the CCA, it will sit in a console "hang" loop. In this case, the processor will not be driving the console line but can still receive the console command.**

else continue

3.7.2.5 Execute DWMBA XMI-to-VAXBI Adapter Self-Test

Performed only by the boot processor:

- If memory battery voltage OK, go to PRINT.
- Assert XMI BAD L from the boot processor.
- Execute DWMBA self-test. If successful, then clear the DWMBA's XBER<10> (STF) bit and light the XBI STPLED.

  else store code in CPU register, display error number in auxiliary LEDs, and begin testing the next DWMBA.
- If all DWMBAs pass self-test, then deassert XMI BAD L.

PRINT:

- Print test results.

3.7.2.6 Boot Processor Sets Up Memory

Performed only by the boot processor:

- Set address and interleave parameters in all XMI memories.
- Search for 256 Kbytes X number of CPUs + COMM block size bytes of good memory. If no block of good memory can be found, then print error message, set CSR<2> (<BPD>), and go into "hang" loop.

NOTE: **This "hang" loop alternates the display of two with the previous contents of the auxiliary LEDs. The previous contents would be either a CPU/MEM test error code or no code.**

- Write console communications area at good memory.
- Set CSR2<BP> to signal other processors to search for CCA.
- Set the "restart in progress" bit.
- Search for restart parameter block (RPB) in memory.
- If RPB found, restart operating system, otherwise restart fails.

Start the operating system (performed only by the boot processor)

- If the key switch is in restart position, pass parameters to VMB, set the "Bootstrap in Progress" flag, and boot the operating system. Operating system passes START commands to all secondary processors with XBER<10> (STF) and XBER<11> (ETF) clear and clears the "Bootstrap in Progress" flag when boot is complete.

  else enter console input routine.

3.7.3 Bootstrapping or Restarting the Operating System

The console code bootstraps a copy of the operating system from a tape or disk device and can attempt to restart an existing memory-resident copy of the operating system ("warm start").

Only the primary processor initiates a bootstrap. A secondary processor cannot execute a BOOT command or perform the automatic bootstrap sequence but remains in console mode awaiting further commands.

Only the primary processor attempts a restart following a power-up. The operating system restarts any secondary processors by passing START commands through the console communications area (CCA). Following an error halt, such as nested machine checks, the halting processor always attempts to restart, regardless of whether it is a primary or secondary.

3.7.3.1 Operating System Restart

The primary processor console code attempts to restart the operating system whenever one of the following events occurs:

- Power is restored to the processor.
- A system reset occurs.
- The running processor halts due to an error halt. Neither a CTRL/P from the console terminal nor a node halt (NHALT) is considered an error halt.

Restart is suppressed if the control panel key switches are set to Enabled and Halt.

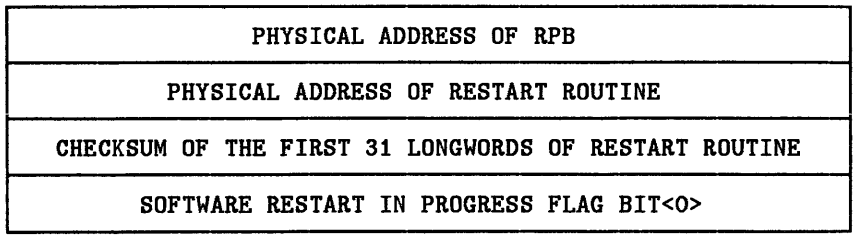
A secondary processor console attempts a restart only following an error halt. For all other halt conditions, the primary processor is responsible for restarting the secondary.

Restart of the operating system is controlled by a memory data structure called the restart parameter block (RPB), constructed by the operating system. Console restart code searches memory for an RPB and, if a valid RPB is found, restarts the operating system at an address stored in the RPB.

The console code also keeps internal flags to indicate that a restart is in progress. There is one flag for each processor, located at CCA$Q_RESTARTIP, in the CCA. These flags allow the console to avoid repeated attempts to restart a failing system. The operating system clears these flags following the successful restart of a processor.

The RPB is a page-aligned structure with the format shown in Figure 3-20.

Figure 3–20 Restart Parameter Block Format



The algorithm used to locate the RPB is:

1. Examine the first longword of each page of memory for a location which contains its own physical address. If none is found, the search fails.
2. Test that the second longword of the page contains a valid non-zero physical address. If this test fails, resume Step 1.
3. Obtain the restart address from the second longword. Calculate the signed longword sum of the first 31 longwords of the restart routine, ignoring overflows. If this value does not match the contents of the third longword of the page, resume Step 1.
4. If all the above tests pass, a valid RPB has been found.

3.7.3.2 Failing Restart

If the restart of a primary processor fails, a message is displayed on the console terminal and a bootstrap is attempted. A failed restart is a serious condition and causes the other processors to abandon whatever is still running.

If a secondary processor's restart fails, the console code examines the CCA$_SECSTART field of the CCA. The console code forces a bootstrap in the same manner as for a primary processor if the bit corresponding to the failing processor is clear. If this bit is set, the console code does not force a bootstrap and the failing processor enters console mode.

The CCA$Q_SECSTART bits are set by the operating system when it is attempting to start a secondary processor. The operating system clears these bits when it is satisfied that the secondary processor has successfully started executing.

This extra state avoids the following scenario peculiar to multiprocessors:

1. A secondary processor encounters an error halt and then fails to restart.
2. The console code forces a bootstrap.
3. The primary processor boots and begins running the operating system.
4. The primary processor starts the defective secondary processor if the secondary processor passed CPU self-test.
5. The secondary processor repeats its error halt and fails restart.
6. The console code again forces a bootstrap and the sequence repeats.

NOTE: A secondary processor cannot directly perform a reboot because it cannot notify the other secondary processors that an expected entry is planned. If the location of the primary processor changed during the system reset, the fact that a boot was in progress could be lost. To avoid this problem, a secondary processor forces the primary processor into console mode (via NHALT) and then signals through the CCA that a bootstrap is needed.

3.7.3.3 Restart Parameters

The console code transfers control to the restart address once a valid RPB has been found. The console code passes the following restart parameters in the GPRs, as specified by the *VAX Architecture Reference Manual.*

R10—Halt PC

R11—Halt PSL

AP—Halt code

SP—Address of the RPB + 512

3.7.3.4 Operating System Bootstrap

The console code attempts to bootstrap the operating system from the primary processor whenever one of the following events occurs:

- The control panel is Enabled and the BOOT command is typed on the console terminal.
- A restart is attempted and fails.

The console code's goal in a bootstrap is to load the primary system bootstrap program, VMB, into memory and begin its execution. VMB is loaded from the device specified by the BOOT command or from a default device recorded in EEPROM. The VAX 6200 uses a set of minimal device handler routines, called boot primitives, to read VMB from the boot device, a technique called "bootblock" booting.

The console searches tables in EEPROM and ROM, in that order, to locate a boot primitive that matches the specified device as the first phase of bootstrap. If a suitable primitive is found, the target device information is saved in the SSC RAM of all CPUs. Then the console code forces a system reset that could change the location of the primary processor.

The system reset causes all processors, memories, and I/O adapters to perform self-test with memory tested in the fastest possible manner.

The second phase of bootstrap begins as the console code is reentered following the reset. The console code examines the SSC RAM to determine that the entry was an "expected entry," and then continues with the bootstrap. The boot parameters are transferred from SSC RAM to the GPRs, the boot primitive is again located, and control transfers to the primitive.

The boot primitive initializes the boot device and reads the first logical block from the device into the first page of good memory. The block contains information about the location of VMB on the device and a program that copies VMB into memory. The program begins at offset 12 in the block. The boot primitive provides a read-block routine that the bootblock program uses to read a block from the device.

The boot devices supported are determined by the boot primitives stored in EEPROM and ROM, and by devices supported in VMB. The KDB50 VAXBI disk adapter, the TBK50 tape adapter, the DEBNA Ethernet adapter, and the CIBCA-A and CIBCA-B VAXBI-CI adapters are supported. The table in EEPROM allows new primitives to be added as new devices are developed.

If the boot device is a disk, the primitive loads logical block zero (the "bootblock") into memory and transfers control to it. The bootblock contains code that knows the location and size of the VMB image on disk. The bootblock code uses a service routine in the boot primitive to read each block of VMB into memory. If the target device is not a disk, the boot primitive must know how to ask for VMB from the device.

3.7.3.5 Parameters Passed to the Boot Primitive

The console code passes parameters to the boot primitive through the GPRs. The boot primitive must preserve all the nonreserved registers so that they can be passed to VMB. These parameters describe the boot device and any bootstrap options that are to be used.

Table 3–11 show how the registers are used.

Table 3–11 Boot Parameters Loaded into GPRs

Register	Bits	Description
GPR0	<7:0>	VMB device type code, supplied by the boot primitive
GPR1	<7:4>	XMI node number of the desired DWMBA
	<3:0>	VAXBI node number
	<31:28>	When loading VMB from the system TK tape drive, the XMI node number of the DWMBA controlling the tape drive.
	<27:24>	When loading VMB from the system TK tape drive, the VAXBI node number of the DEBNA tape adapter.
GPR2	<15:0>	The remote (HSC) node numbers, if the Boot/Node qualifier was specified.
GPR3		Boot device unit number
GPR4		Reserved, the LBN of the secondary bootstrap
GPR5		Software boot control flags
GPR6		Used by the boot primitive to pass information to the bootblock program
GRP7		Physical address of the CCA
GPR8		Reserved
GPR9		Reserved
GPR10		The halt PC
GPR11		The halt PSL
AP		The halt code
FP		Used by boot primitive to pass information to the bootblock program.
SP		The address of the 256-Kbyte block of good memory + 512

3.7.3.6 Parameters Passed to the Bootblock Program

The parameters passed to the bootblock program are the same as those passed to the boot primitive plus the contents of GPR6. GPR6 has the physical address of the read-block routine provided by the primitive. The bootblock program must preserve all parameters except GPR6 so that they can be passed to VMB.

3.7.3.7 Parameters Required by the Boot Primitive

When the bootblock program calls the read-block routine in the boot primitive, it must supply the input parameters shown in Table 3–12 and the output parameters shown in Table 3–13.

Table 3–12 Input Parameters Required by the Boot Primitive

Register	Bits
GPR1	XMI and VAXBI node numbers of the boot device, as passed by the console code
GPR3	Unit number of the boot device, as passed by the console code
GPR8	LBN to be read or, if a tape drive using non-ANSI labeled tape, the length of the block that was just read
FP	Address of the data structures set in memory by the boot primitive when it is first invoked
SP	Physical address to receive the transfer

Table 3–13 Output Parameters Required by the Boot Primitive

Register	Bits
GPR0	SS$_NORMAL if successful, the low bit clears on error.
GPR7 through GPR10	May be modified

3.7.3.8 Considerations for Tape Drives

The boot primitive rewinds the tape before it performs the first read and before transferring control to the loaded image.

The boot primitive checks the length of the first block read from the tape. If the block is 80 bytes long, the tape is assumed to be ANSI labeled and VMB is assumed to be the first file on the tape. The boot primitive then skips to the first tapemark, reads blocks into memory by storing them, beginning at the address passed in the SP. Blocks are loaded until a tapemark is encountered, and then control is passed to the first byte in the loaded image.

If the first block of the tape is not 80 bytes long, the remaining contents of the first file are loaded and control is transferred to the loaded image at offset 12 from the base of good memory.

The read-block routine also supports rewinding the tape. GPR0 must contain IO$_READPBLK for a read operation or IO$_REWIND for a rewind. The read-block routine always reads the next block from the tape and ignores any logical block number (LBN) passed in GPR8. Instead, GPR8 returns the length of the block just read.

3.7.3.9 Considerations for Ethernet Devices

The boot primitive for Ethernet devices uses the automatic load feature of the DEBNA Ethernet adapter. The boot primitive signals the adapter to request a tertiary load starting at the base of memory. If the load succeeds, control is passed to the loaded image at the transfer offset supplied with the image.

3.8 Interprocessor Communication through the Console Program

Each CPU of a multiprocessor system must communicate with the other CPUs and the operating system. This section describes the interprocessor communication for the VAX 6200.

The console program runs on each processor of a multiprocessor VAX 6200. These copies of console code must be able to communicate with each other and with the operating system.

When two processors needing to communicate are running, that is, not in console mode, the communications take place using mechanisms provided by the operating system. When one, or both, of the processors is in console mode, communications take place using a shared data structure called the console communications area (CCA).

The primary processor controls the console terminal and, therefore, most of the communication in the VAX 6200. There is no communication between secondary processors.

3.8.1 Required Communications Paths

A processor can be in one of four communication states: a running primary processor, a primary processor in console mode, a running secondary processor, or a secondary processor in console mode. The following communication paths are provided.

1 Running processor to running processor, independent of primary or secondary.

The console program is not involved. The processors are supported by the communications mechanisms within the operating system. These paths are used even when the communication is related to the console program. For example, when the system time is modified, the new time must be stored in the time-of-year clocks on each processor. The operating system uses its own methods to examine or propagate this information.

A special case of communications on these paths involves the XDELTA system debugger when it is entered on a secondary processor. The operating system is responsible for passing characters to and from the primary processor and, thus, to the console terminal.

2 Running primary console to/from secondary console.

The operating system on the primary processor must send complete console commands to the secondary console, such as to start or stop the secondary processor. The secondary console program must be able to send responses (human readable messages) to the operating system on the primary, such as when the secondary processor encounters an error halt. The secondary processor can send these responses at any time.

The secondary processor does not send commands to the primary processor, and the primary processor does not send responses to the secondary processor.

3 Console mode primary processor to/from running secondary processor.

Whenever the primary processor halts, the secondary processors eventually block while waiting for resources locked by the primary. The primary console supports receiving complete responses from the running secondary processor.

4 Primary console to/from secondary console requires two different types of communication.

The primary console sends complete commands to the secondary, allowing the primary console to update the copy of a parameter stored on each processor. An example of this type of communication is to synchronize the console terminal baud rate whenever it is changed on the primary. The secondary consoles send complete responses to the primary console to report, for example, a processor halt. Since responses arrive complete, there are no interleaving messages on the console terminal. The secondary processor does not send commands, and the primary processor does not send responses.

The consoles support character-at-a-time communications to implement the "Z" command, which transfers characters to and from a secondary node so that the secondary processor appears to be directly connected to the console terminal. The primary processor sends single characters of a command to the secondary processor. The receiving secondary processor performs all the processing of the input characters, including echoing and line editing. The secondary processor sends single characters of a response to the primary processor for immediate display on the console terminal. The "Z" command also extends to communication with VAXBI devices and, potentially, to non-processor XMI nodes.

3.8.2 Console Communications Area

The Console Communications Area (CCA) is the shared data structure in high physical memory used for communications between console programs. It consists of a one-page header followed by a variable number of pages containing buffers. The header contains status information that must be visible systemwide. The buffers, used for passing messages between processors, are allocated one set for each XMI node that could be in the system.

The CCA is initialized by the primary (boot) processor at system reset. It is allocated beginning on a page boundary from the highest addressed page of system memory that can be located by the primary processor. The header lies in the lowest addressed page of the CCA, followed by buffers.

The CCA is not initialized under any other console entry conditions (node reset or halts). The address of the CCA is obtained from the console state remaining in SSC RAM.

Diagnostic tests that must test or reconfigure memory could overwrite the CCA. If this should happen, the diagnostic tests must observe the following conventions:

- The diagnostic tests can only be run from the primary processor.
- The diagnostic tests must force the secondary processors to stop polling the CCA.
- The diagnostic tests must rebuild the CCA after completing testing.
- The secondary processors must wait for a signal passed through the XGPR register before locating the new CCA.

The location of the CCA is passed to the operating system at bootstrap time through GPR7. During system initialization, each processor is triggered to search for the CCA. This search starts at the highest addressed memory that can be located by each processor and then works backward. If a processor cannot locate the CCA, it enters an endless loop and cannot participate in the system. The algorithm used by the console code to locate the existing CCA is as follows:

1 Next = highest memory address in system + 1 - 512.

2 If next < 0, then "Failed to find CCA."

3 If (next + CCA$L_BASE) < > next, then goto Step 7.

4 If (next + CCA$W_IDENT) < > "CC", then goto Step 7.

5 Compute sum of bytes at (next) through (next + CCA$B_CHKSUM - 1) ignoring overflow.

6 If sum = (next + CCA$B_CHKSUM), then "Exit with CCA found at next."

7 Next = next - 512.

8 Goto Step 2.

The overall layout of the CCA is shown in Figure 3-21 and Figure 3-22. The contents of the fields are described in Table 3-14.

Figure 3-21 CCA Layout, Part 1

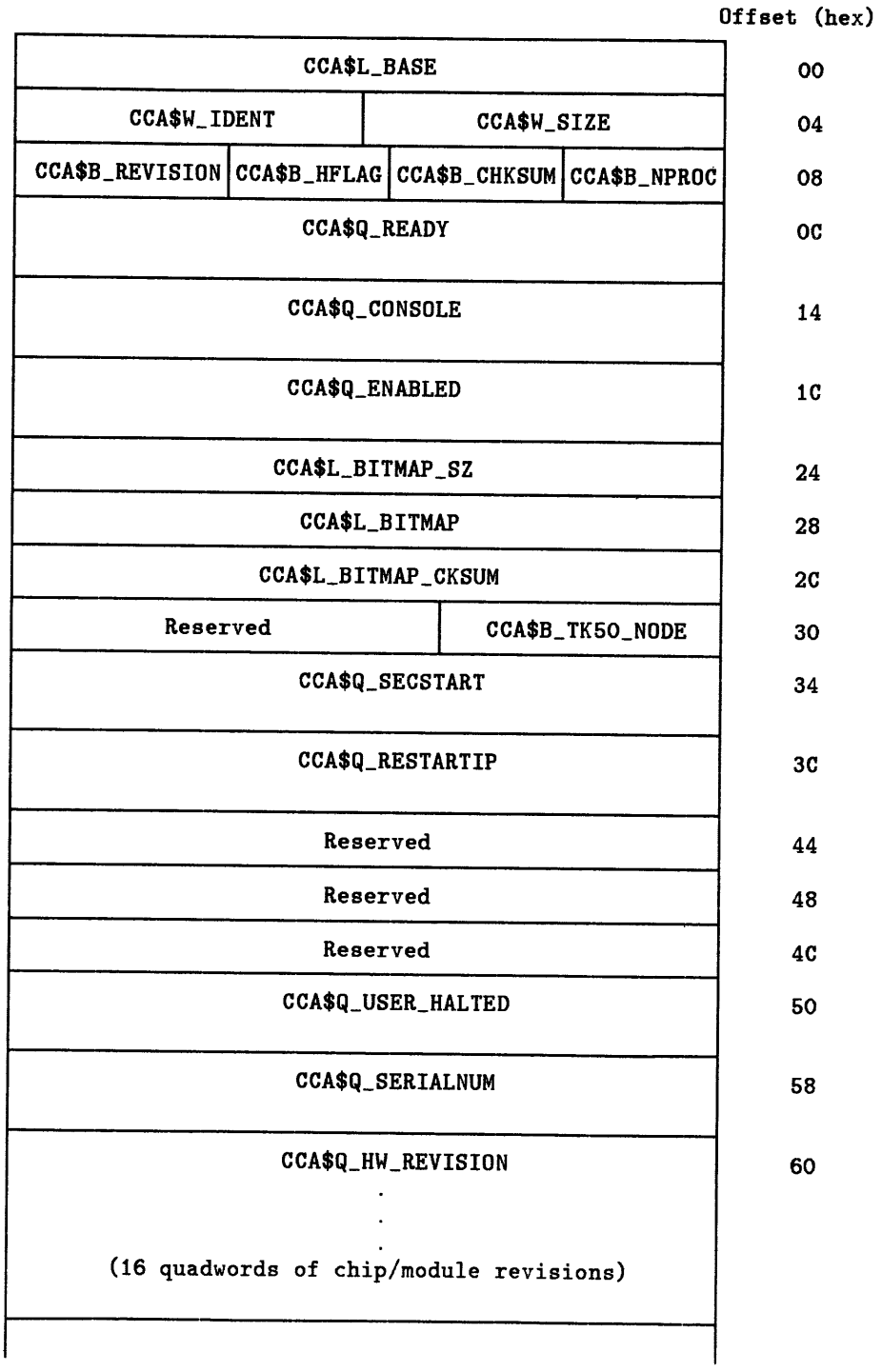


Figure 3–22 CCA Layout, Part 2

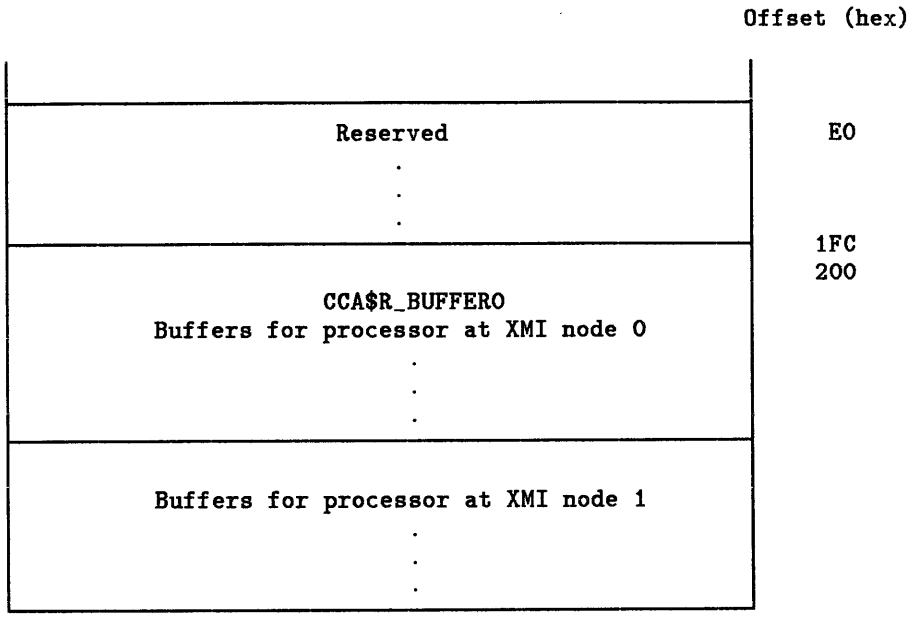


Table 3-14 CCA Fields

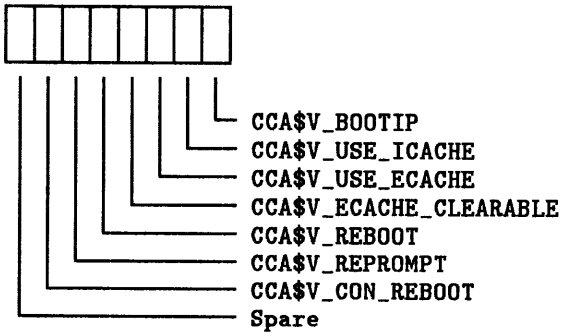
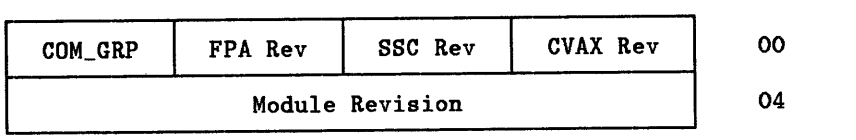
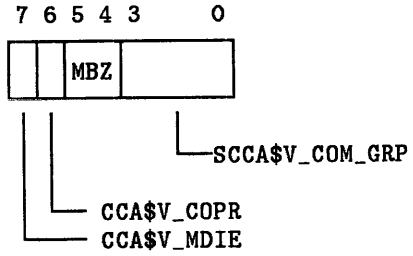
Field	Description
CCA$L_BASE	Physical address of the base of the CCA.
CCA$W_SIZE	The size, in bytes, of the CCA, usually 3200.
CCA$W_IDENT	The ASCII characters "CC".
CCA$B_NPROC	The number of processors supported by the CCA.
CCA$B_CHKSUM	Checksum of the first CCA$B_CHKSUM-1 bytes of the CCA. Computed by doing signed, byte addition, ignoring any overflow.
CCA$B_HFLAGS	Systemwide status flags: 
	CCA$V_BOOTIP — When set, a bootstrap is being attempted. This prevents repeated attempts to bootstrap after a failure.
	CCA$V_USE_ICACHE — When set, the CVAX chip internal (first-level) cache is to be enabled by the operating system.
	CCA$V_USE_ECACHE — When set, the external (second-level) cache is to be enabled by the operating system.
	CCA$V_ECACHE_CLEARABLE — When set, the external cache clear operation can be used successfully. Some operating system error recovery is needed to clear the cache.
	CCA$V_REBOOT — This bit is tested whenever the console is entered as a result of a CTRL/P or a node halt. If the bit is set, the operating system is requesting a reboot. The system is rebooted from the default boot device regardless of the front panel switch settings.
	CCA$V_REPROMPT — This bit is used internally by the console to support the SET CPU command.
CCA$B_REVISION	The revision number for the CCA.
CCA$Q_READY	A bitmask of the processors that have data posted in their transmit buffer for processing by the primary processor. This field allows the operating system to use a Find First Set (FFS) instruction to locate any pending messages. The bits and nodes are numbered, starting with zero.
CCA$Q_CONSOLE	A bitmask indicating the processors known to be in console mode. The appropriate bit is set and cleared by each processor as it enters and leaves console mode.
CCA$Q_ENABLED	A bitmask indicating which processors are enabled to leave console mode. A processor sets or clears its bit during console initialization, based on a bit stored in EEPROM. The EEPROM bit is set with the SET CPU command.

Table 3-14 (Cont.) CCA Fields

Field	Description
CCA$L_BITMAP_SZ	The size, in bytes, of the physical memory bitmap. The bitmap is always an even number of longwords in length.
CCA$L_BITMAP	The physical address of the physical memory bitmap. The bitmap contains one bit for each page of physical memory present on the system. The bit is clear if the page contains a hard error or if the page is in use by the bitmap or CCA. The bitmap is always page aligned.
CCA$L_BITMAP_ CKSUM	Reserved; not used.
CCA$B_TK50_NODE	This field is used to pass to the operating system the XMI (in bits<7:4>) and VAXBI (in bits<3:0>) node numbers of the adapter that controls the TK tape drive.
CCA$Q_SECSTART	A bitmask indicating which processors are currently being started by the primary processor. The console code uses this information to avoid repeatedly forcing a bootstrap. This field is set and cleared by the operating system.
CCA$Q_RESTARTIP	A bitmask indicating which processors are currently attempting restarts. Multiple flags are maintained to allow simultaneous error restarts to be performed. The operating system clears these fields if restart succeeds.
CCA$Q_USER_HALTED	A bitmask indicating which processors entered console mode as a result of user intervention (CTRL/P or STOP command). This information allows the operating sytem to make decisions about timeouts in a symmetric multiprocessing configuration.
CCA$Q_SERIALNUM	The system serial number.
CCA$Q_HW_REVISION	Consists of a 16-quadword array containing the chip and module revision information for the processors. Module revisions are an ASCII string; chip revisions consist of two digits with an implied decimal point. The quadword is zero for non-processor nodes. The layout of this quadword is:



The layout of the COM_GRP byte is:



CCA$V_MDIE — When set, this non-boot processor receives interrupts. If this bit is clear, interrupts are directed only to the boot processor.

Table 3–14 (Cont.) CCA Fields

Field	Description	
	CCA$V_COPR	When set, this bit indicates that the processor can correctly perform a passive release on an interrupt acknowledge cycle. If this bit is clear, data corruption results from performing a passive release.
	CCA$V_COM_GRP	This binary field is used by the operating system to determine if all processors in the system are hardware compatible. Any processors not in the same group as the boot processor are inhibited from starting.

The CCA contains a buffer area for each possible XMI node. Each buffer area contains fields to support both message oriented and character-at-a-time communications.

The address of the buffer area for XMI node *n* is given by:

$$Buffer_n = \text{Base address of CCA} + 512 + (n * 168)$$

The layout of the buffer area is shown in Figure 3–23, and the contents of the field are described in Table 3–15.

Figure 3–23 Layout of XMI Node Buffers

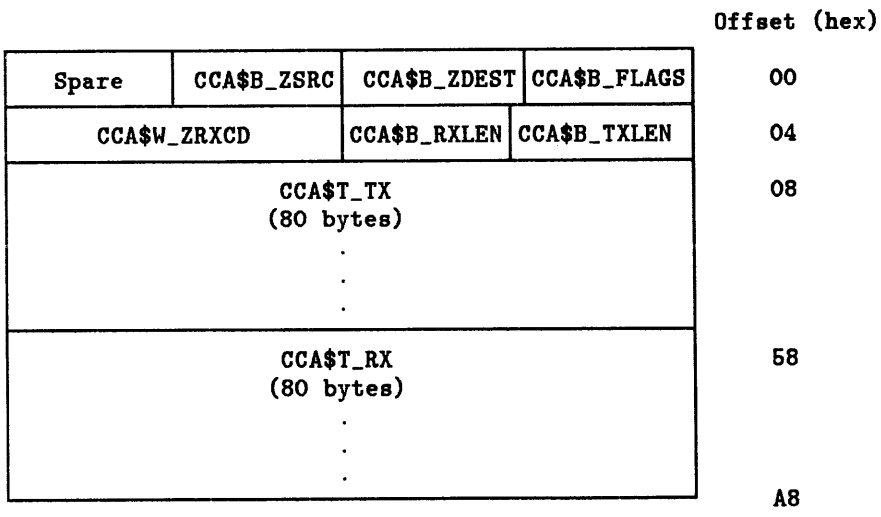
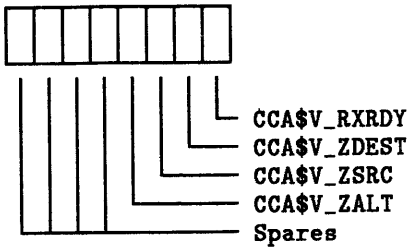


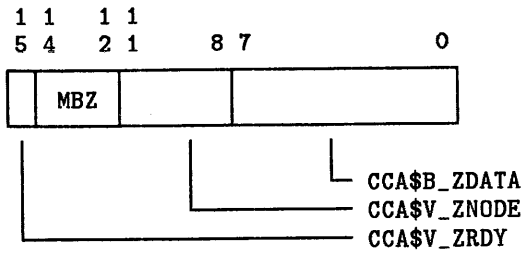
Table 3–15 Buffer Fields

Field	Description
CCA$B_FLAGS	Status flags:



CCA$V_RXRDY	When set, there is a complete message in the CCA$T_RX buffer. The equivalent bit for CCA$T_TX is in CCA$Q_READY of the CCA header.
CCA$V_ZDEST	When set, this node is sending "Z" command data to the node listed in CCA$B_ZDEST.
CCA$V_ZSRC	When set, this node is receiving "Z" command data from the node listed in CCA$B_ZSRC. This bit is always set or cleared by the node originating the "Z" command.
CCA$V_ZALT	When set, the target of the current "Z" command cannot communicate through the CCA. The target is either a non-processor XMI node or a VAXBI node and must be accessed using alternate RXCD protocol, as described in the *VAXBI System Reference Manual.*

CCA$B_ZDEST	When CCA$V_ZDEST is set, this field contains the XMI node number of the node receiving the "Z" command data that this node is sending. If the low four bits of this field identify a node that is a DWMBA, the high order four bits contain the destination VAXBI node number.
CCA$B_ZSRC	If CCA$V_ZSRC is set, this field contains the XMI node number of the node transmitting "Z" command data to this node.
CCA$B_TXLEN	If the bit corresponding to this node is set in CCA$Q_READY, then this field contains the length, in bytes, of the message in CCA$T_TX.
CCA$B_RXLEN	If CCA$V_RXRDY is set in CCA$Q_READY, then this field contains the length, in bytes, of the message in CCA$T_RX.
CCA$W_ZRXCD	This field is used for character-at-a-time communication in the same manner as a VAXBI RXCD Register. The layout is:



CCA$B_ZDATA	When CCA$V_ZRDY is set, this field contains one byte of "Z" command data being sent to this node.

Table 3–15 (Cont.) Buffer Fields

Field	Description	
	CCA$V_ZNODE	When CCA$V_ZRDY is set, this four-bit field contains the XMI node number of the node that transmitted the data in CCA$B_ZDATA.
	CCA$V_ZRDY	When this bit is set, there is valid data in the other CCA$W_ZRXCD fields.
CCA$T_TX	This buffer is used by the node to transmit a response to the primary processor. Only response data is passed through this buffer since a secondary processor does not send commands to the primary processor.	
CCA$T_RX	This buffer is used by the node to receive a command from the primary processor. Only command data is passed through this buffer since a secondary processor does not receive responses from the primary processor.	

3.8.3 Sending a Message to Another Processor

The following two examples show how the CCA is manipulated when a complete message is sent between two processors.

For the first example, the primary processor, located at XMI node 1, sends a START command to the secondary processor, located at XMI node 4.

1. Node 1 examines the CCA$V_RXRDY bit in the CCA buffer area for node 4. If the bit is clear, then goto Step 3.
2. Node 1 polls the bit until it clears or until a timeout of 12 seconds is reached. If a timeout occurs, an error is reported.
3. Node 1 moves the text of the START command into the CCA$T_RX buffer for node 4.
4. Node 1 sets the length of the command into the CCA$B_RXLEN field for node 4.
5. Node 1 sets the CCA$V_RXRDY bit for node 4 to indicate that a command is waiting.
6. Whenever node 4 enters its main console loop, it will eventually check for commands to execute. It will examine its local command buffer and then check its CCA$V_RXRDY bit for a command from another node.
7. Node 4 will now process the command contained in its CCA$T_RX buffer.
8. After reading the command, node 4 then clears its CCA$V_RXRDY bit, indicating that the buffer is again available.

For the second example, the secondary processor, which is located at XMI node 4, halts, enters console mode, and sends a "halted" message to the primary processor, located at XMI node 1.

1. Node 4 examines bit 4 of the CCA$Q_READY field. If the bit is clear, then goto Step 3.
2. Node 4 polls this bit until it clears.
3. Node 4 moves the text of its response into its CCA$T_TX buffer.
4. Node 4 sets the length of the response in its CCA$B_TXLEN field.
5. Node 4 sets bit 4 in CCA$Q_READY to indicates that a response is waiting.
6. Node 4 issues an IVINTR interrupt to node 1. If node 1 is running, this alerts the operating system that a response is waiting. Node 4 polls CCA$Q_READY until bit 4 clears or until a timeout of 60 seconds expires, preventing the secondary node from performing any action that might cause the response to be lost before the primary can display it.
7. If node 1 is running, it responds to the IVINTR and eventually checks for console responses, using an FFS instruction to check CCA$Q_READY. If node 1 was in console mode, it would be polling CCA$Q_READY and discover bit 4 set.
8. Node 1 (either the operating system or the console code) processes the response from the CCA$T_TX buffer for node 4. If the console code is running, it displays the response on the console terminal.
9. Node 1 clears bit 4 in CCA$Q_READY, indicating that the buffer is again available.

3.9 KA62A CPU Module Error Handling

This section describes the error handling features of the KA62A CPU module.

The KA62A CPU module hardware provides automatic reattempts of many XMI bus transfer failures:

- All XMI command/address transfers are reattempted until acknowledged or a transaction timeout occurs (when XBER<13> (TTO) asserts).
- All XMI write transactions are reattempted until acknowledged or a transaction timeout occurs.

All second-level cache errors, except data parity errors on CPU demand reads, are "soft" and are signaled by asserting CRD to the CVAX chip. KA62A CPU module hardware automatically disables the second-level cache following a cache error that has the potential to leave the second-level cache incoherent, such as tag or valid bit parity errors on a write-through.

All XMI memory reads are "connected"; the CPU waits for all demand-requested data to be returned and, if it cannot be delivered from the XMI, it signals with ERR, resulting in a machine check. Failures on delivery of non-demand data (such as cache fill data) results in a "soft" error. A memory read "hit" in the active WB causes the WB to be purged. If the purge results in an XMI memory write failure, the read is suppressed and an ERR response is returned to the CVAX.

All XMI memory writes are "disconnects." They are acknowledged by the XMI interface and data is placed in the write buffer to be written later. If a subsequent WB unload or purge results in an XMI write failure, it is signaled to the CPU by posting a MEMERR interrupt.

All XMI I/O reads and writes are "connected"; they cause purging of the WB prior to their initiation on the XMI and they are not acknowledged until all XMI transactions are successfully completed. If the WB purge results in an XMI memory write failure, the I/O transaction is suppressed and an ERR response is returned to the CVAX chip.

For error handling purposes, XMI IVINTR transactions are treated as I/O writes.

For error handling purposes, XMI IDENT transactions are treated as I/O reads except that errors are reported with a MEMERR interrupt, since an ERR assertion during a CVAX Interrupt Acknowledge cycle is interpreted as a passive release.

The XMI interface maintains complete error status on a failed XMI transaction that was initiated by its node. This status includes the failed command, commander ID, address, and an error bit that indicates the type of error that had occurred. This status remains locked-up until software resets the error bit(s).

3.9.1 Parity Generation and Checking for Error Detection

Parity generation and check characteristics of the KA62A CPU module follows:

- The CVAX chip's CPU generates parity on write data and checks parity on read data. The CVAX does not generate parity on command/address data.
- The first-level cache, contained in the CVAX chip, supports parity on both the tag and data store.
- The second-level cache supports parity on the tag bits, valid bits, and data store. On cache fills and writes, parity is stored and then checked by the CVAX chip's CPU on reads.
- The XCPGA detects CDAL parity errors on writes.
- The XMI supports three parity bits covering both data and command information. The KA62A CPU module generates and checks XMI parity.
- The CFPA does not generate or check parity.
- Since the SSC does not support parity, the internal battery-backed-up 1 Kbyte of RAM and the internal registers are not protected.
- KA62A CPU module CSR1 and CSR2 are not parity protected.

3.9.2 Error Interrupt Service Routines

Interrupt service routines use the following sequence when an error occurs:

1. Read XBER to determine the type of error.
2. If XBER<ES> is set, then find more specific error information in CSR2.
3. Service the error condition. In many cases, the second-level cache must be flushed by setting and then clearing CSR1<FCI>.
4. Clear only the individual error bits that were serviced after the error condition has been handled. All error bits are write-one-to-clear.
5. Read XBER to ensure that no new errors have been detected. A new error condition cannot generate a new interrupt unless all other error bits are clear, since the MEMERR and CRD interrupt lines are edge-sensitive. If this read indicates that no error bits are set, then exit the interrupt service routine; else loop to step 1.

3.9.3 KA62A CPU Module Error Matrix

Table 3–16 CDAL Bus Parity Errors[1]

Reference Type	Effect on CPU Execution	Effect on Prefetcher	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	State Captured on Error	Notes
Demand D-Stream Read	Abort Cycle, Machine Check 80 or 81	–	–	Store Bad Data[2]	–	MSER<5> set; MSER<6> set	Machine Check Routine Must Flush 2nd-Level Cache. Use CSR1<FCI>
Request D-Stream Read (Fill)	–	–	–	Store Bad Data[2]	–	MSER<6> set	
Request I-Stream Read (Prefetch)	–	Abort Prefetch	Invalidate Row	Store Bad Data[2]	–	MSER<6> set	
Request I-Stream Read (Fill)	–	–	Invalidate Row, Abort Fill	Store Bad Data[2]	–	MSER<6> set	
Read-Lock	Abort Cycle, Machine Check 80 or 81	–	–	–	–	CSR2<UWP> set;MSER<6> set	Failed address in FADR. Memory CSR can be used to unlock location.
Masked Write	Interrupt at IPL1D (MEMERR) Vector of 60 (next cycle)	–	–	Store Bad Data[2]	XMI write suppressed, memory not updated.	CSR2<WDPE> set	MEMERR interrupt routine must flush 2nd-level cache CSR1<FCI>
Unmasked Write	Interrupt at IPL1D (MEMERR) Vector of 60 (next cycle)	–	–	Store Bad Data[2]	XMI write suppressed, memory not updated.	CSR2<WDPE> set	MEMERR interrupt routine must flush 2nd-level cache CSR1<FCI>
Unlock Write	Interrupt at IPL1D (MEMERR) Vector of 60 (next cycle)	–	–	Store Bad Data[3]	XMI write suppressed, memory not updated.	CSR2<WDPE> set	MEMERR interrupt routine must flush 2nd-level cache CSR1<FCI>
DMA Read	– – – – – – – – Never Performed on CDAL – – – – – – – –						
DMA Write (External Cache Fill)	–	–	–	Store Bad Data	–	–	Parity not checked.

[1]CDAL parity is only checked on transfers between the CPU and XCPGA.

[2]On cachable miss references only (that is, when second-level cache allocates a block).

[3]On second-level cache hits only.

Table 3–17 First-Level Cache Parity Errors[1]

Reference Type	Effect on CPU Execution	Effect on Prefetcher	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	State Captured on Error	Notes
Demand D-Stream Read	–	–	–	–	–	–	D-Stream references are forced to miss the 1st-level cachec (I-stream only mode).
Request D-Stream Read (Fill)	–	–	–	–	–	–	Same
Request I-Stream Read (Prefetch)	–	Abort Prefetch	Flushes Cache[2]	–	–	MSER<3> set if data error in set 2 MSER<2> set if data error in set 1 MSER<1> set if data error MSER<0> set if tag error	
Request I-Stream Read (Fill)	–	–	–	–	–	–	Reference type never "seen" by 1st-level cache
Read-Lock	–	–	–	–	–	–	Parity not checked
Masked Write	–	–	Flushes Cache[2]	–	–	MSER<0> set	Only tag parity is checked on writes that hit 1st-level cache
Unmasked Write	–	–	Flushes Cache[2]	–	–	MSER<0> set	Only tag parity is checked on writes that hit 1st-level cache
Unlock Write	–	–	Flushes Cache[2]	–	–	MSER<0> set	Only tag parity is checked on writes that hit 1st-level cache
DMA Read	– – – – – – – – Never Performed on CDAL – – – – – – – –						
DMA Write (External Cache Fill)	–	–	–	–	–	–	–

[1]First-level cache parity errors can be detected only on references that hit the cache.

[2]The first-level cache is flushed only if CADR<0> (Diagnostic Mode) is cleared.

Table 3–18 Second-Level Cache Data Parity Errors[1]

Reference Type	Effect on CPU Execution	Effect on Prefetcher	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	State Captured on Error	Notes
Demand D-Stream Read	Abort Cycle, Machine Check 80 or 81	–	–	Bad Data Unaltered	–	MSER<5> set; MSER<6> set	Machine Check Routine Must Flush 2nd-Level Cache. Use CSR1<FCI>
Request D-Stream Read (Fill)	–	–	–	Bad Data Unaltered	–	MSER<6> set	
Request I-Stream Read (Prefetch)	–	Abort prefetch	Invalidate Row	Bad Data Unaltered	–	MSER<6> set	
Request I-Stream Read (Fill)	–	–	Invalidate Row, Abort Fill	Bad Data Unaltered	–	MSER<6> set	
Read-Lock	–	–	–	–	–	–	Read-locks are forced to miss the cache.
Masked Write	–	–	–	–	–	–	Parity not checked, entry updated on all CPU writes that hit cache.
Unmasked Write	–	–	–	–	–	–	Parity not checked, entry updated on all CPU writes that hit cache.
Unlock Write	–	–	–	–	–	–	Parity not checked, entry updated on all CPU writes that hit cache.
DMA Read	– – – – – – – – Never Performed on CDAL – – – – – – – –						
DMA Write (External Cache Fill)	–	–	–	–	–	–	Parity not checked.

[1]Second-level cache data parity errors can be detected only on references that hit the cache. These errors look like CDAL parity errors.

Table 3–19 Second-Level Cache Tag Parity Errors[1]

Reference Type	Effect on CPU Execution	Effect on Prefetcher	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	State Captured on Error	Notes
Demand D-Stream Read	Interrupt at IPL1A (CRD) Vector of 54	–	–	Force Cache Miss. Disable Cache with CSR1<FMISS>	–	CSR2<TPE> or CSR2<VPE>	CRD Interrupt
Request D-Stream Read (Fill)	Same Behavior as Demand D-Stream Read						
Request I-Stream Read (Prefetch)	Same Behavior as Demand D-Stream Read						
Request I-Stream Read (Fill)	Same Behavior as Demand D-Stream Read						
Read-Lock	–	–	–	–	–	–	Read-Locks are forced to bypass the cache
Masked Write	Same Behavior as Demand D-Stream Read						
Unmasked Write	Same Behavior as Demand D-Stream Read						
Unlock Write	Same Behavior as Demand D-Stream Read						
DMA Read	— — — — — — — — Never Performed on CDAL — — — — — — — —						
DMA Write (External Cache Fill)	–	–	–	–	–	–	Parity not checked during cache full
DMA Write (Invalidate)	Same Behavior as Demand D-Stream Read						

[1]Second-level cache tag parity errors can be detected on all I- and D-stream references to VAX memory space except read-lock.

Table 3–20 XMI Bus Timeout Errors

Reference Type	Effect on CPU Execution	Effect on Prefetcher	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	State Captured on Error	Notes
Demand D-Stream Read	Abort Cycle, Machine Check 80 or 81	–	–	–	–	XBER<NRR> set XBER<TTO> set XFADR<31:0>	16.7ms Timer - NXM Reattempt T/O XMI Failing Adr/Len
Request I- or D-Stream Read (Fill)	Abort Cycle, Machine Check 80 or 81	–	–	–	–	XBER<NRR> set XBER<TTO> set XFADR<31:0>	16.7ms Timer - NXM Reattempt T/O XMI Failing Adr/Len
Request I-Stream Read (Prefetch)	–	Abort Prefetch	Invalidate Row	–	–	–	16.7ms Timer - NXM
Read-Lock	Abort Cycle, Machine Check 80 or 81	–	–	–	–	XBER<NRR> set XBER<TTO> set XFADR<31:0>	16.7ms Timer - NXM Reattempt T/O XMI Failing Adr/Len
Masked Write	Interrupt at IPL1D (MEMERR) Vector of 60 (next cycle)	–	–	–	–	XBER<TTO> set XFADR<31:0>	Reattempt T/O XMI Failing Adr/Len
Unmasked Write	Interrupt at IPL1D (MEMERR) Vector of 60 (next cycle)	–	–	–	–	XBER<TTO> set XFADR<31:0>	Reattempt T/O XMI Failing Adr/Len
Unlock Write	Interrupt at IPL1D (MEMERR) Vector of 60 (next cycle)	–	–	–	–	XBER<TTO> set XFADR<31:0>	Reattempt T/O XMI Failing Adr/Len
DMA Read	— — — — — — — — Never Performed on XMI — — — — — — — —						
DMA Write	— — — — — — — — Never Performed on XMI — — — — — — — —						
Interrupt Acknowledge (IDENT)	Interrupt at IPL1D (MEMERR) Vector of 60	–	–	–	–	XBER<NRR> set XBER<TTO> set XFADR<31:0>	16.7ms Timer - NXM Reattempt T/O XMI Failing Adr/Len

Table 3–21 XMI Bus Parity Errors

Reference Type	Effect on CPU Execution	Effect on Prefetcher	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	State Captured on Error	Notes
Demand D-Stream Read	Abort Cycle, Machine Check 80 or 81	–	–	HW not cached	–	XBER<PE> set XFADR<31:0>	NRR, Seq Err could also set XMI Failing Adr/Len
Request I- or D-Stream Read (Fill)	–	–	–	HW not cached	–	XBER<PE> set XFADR<31:0>	NRR, Seq Err could also set XMI Failing Adr/Len
Request I-Stream Read (Prefetch)	–	Abort Prefetch	Invalidate Row	HW not cached	–	XBER<PE> set XFADR<31:0>	NRR, Seq Err could also set XMI Failing Adr/Len
Read-Lock	Abort Cycle, Machine Check 80 or 81	–	–	–	–	XBER<PE> set XFADR<31:0>	NRR, Seq Err could also set XMI Failing Adr/Len
Masked or Unmasked Write	–	–	–	–	–	–	Parity not checked
DMA Read	— — — — — — — — Never Performed on XMI — — — — — — — —						
DMA Write	— — — — — — — — Never Performed on XMI — — — — — — — —						
Interrupt Acknowledge (IDENT)	Interrupt at IPL1D (MEMERR) Vector of 60	–	–	–	–	XBER<PE> set XFADR<31:0>	NRR, Seq Err could also set XMI Failing Adr/Len

Table 3–22 CDAL Bus Timeout Errors

Reference Type	Effect on CPU Execution	Effect on Prefetcher	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	State Captured on Error	Notes
Demand D-Stream Read	Abort Cycle, Machine Check 80 or 81	–	–	–	–	BTCR<31> set	36.9ms Timer - NXM
Request I- or D-Stream Read (Fill)	–	–	Invalidate Row, Abort Fill	–	–	BTCR<31> set	36.9ms Timer - NXM
Request I-Stream Read (Prefetch)	–	Abort Prefetch	Invalidate Row	–	–	BTCR<31> set	36.9ms Timer - NXM
Read-Lock	Abort Cycle, Machine Check 80 or 81	–	–	–	–	BTCR<31> set	36.9ms Timer - NXM
Masked or Unmasked Write	Abort Cycle, Machine Check 80 or 81	–	–	–	–	BTCR<31> set	36.9ms Timer - NXM
DMA Read	— — — — — — — — Never Performed on CDAL — — — — — — — —						
DMA Write	–	–	–	Cache Fill or Invalidate	–	–	No Timer
Interrupt Acknowledge (IDENT)	Abort Cycle	–	–	–	–	BTCR<31> set	36.9ms Timer - NXM
DMA Grant	Hangs	–	–	–	–	–	No Timer

Table 3-23 Main Memory Correctable Errors

Reference Type	Effect on CPU Execution	Effect on Prefetcher	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	State Captured on Error	Notes
Demand D-Stream Read	Interrupt at IPL1A (CRD) Vector of 54	-	-	-	Read & Correct Data, Bad Data in Memory Unaltered	MEMCSR4<29> set MEMCSR4<28:9> set MEMCSR4<7:0> set	Main Memory Page Adr. Identity Bit Position CRD Interrupt Routine Must Flush Main Memory Page
Request D-Stream Read (Fill)	Same Behavior as Demand D-Stream Read						
Request I-Stream Read (Prefetch)	Same Behavior as Demand D-Stream Read						
Request I-Stream Read (Fill)	Same Behavior as Demand D-Stream Read						
Read-Lock	Same Behavior as Demand D-Stream Read						
Masked Write	-	-	-	-	-	-	CRD Logged in MEMCSR4<29>
Unmasked Write	-	-	-	-	-	-	ECC Not Checked

Table 3-24 Main Memory Uncorrectable Errors

Reference Type	Effect on CPU Execution	Effect on Prefetcher	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	State Captured on Error	Notes
Demand D-Stream Read	Abort Cycle, Machine Check 80 or 81	-	-	HW not cached	-	MEMCSR4<31> set MEMCSR4<28:9> set MEMCSR4<7:0> set	 Main Memory Page Adr. Identity Bit Position
Request I- or D-Stream Read (Fill)	-	-	Invalidate Row, Abort Fill	HW not cached	-	MEMCSR4<31> set MEMCSR4<28:9> set MEMCSR4<7:0> set	 Main Memory Page Adr. Identity Bit Position
Request I-Stream Read (Prefetch)	-	Abort Prefetch	Invalidate Row	HW not cached	-	MEMCSR4<31> set MEMCSR4<28:9> set MEMCSR4<7:0> set	 Main Memory Page Adr. Identity Bit Position
Read-Lock	Abort Cycle, Machine Check 80 or 81	-	-	-	-	MEMCSR4<31> set MEMCSR4<28:9> set MEMCSR4<7:0> set	 Main Memory Page Adr. Identity Bit Position
Masked Write	-	-	-	-	-	MEMCSR4<31> set MEMCSR4<28:9> set MEMCSR4<7:0> set	 Main Memory Page Adr. Identity Bit Position
Unmasked Write	-	-	-	-	-	-	ECC Not Checked

4 MS62A Memory Module

The MS62A memory module is a metal-oxide semiconductor (MOS), dynamic random access memory (DRAM), that provides 32 Mbytes of data storage. The memory array is designed for use in the VAX 6200 system and communicates over the XMI bus.

This chapter contains the following sections:

- Features
- Technical Description
- Self-Test and Initialization
- Starting Address and Interleaving
- Control and Status Registers
- Error Handling and Command Responses

4.1 Module Features

The MS62A memory module is a dynamic random access memory (DRAM) that communicates through the XMI bus to provide VAX 6200 system memory.

The MS62A memory module has the following features:

- The memory module contains MOS dynamic RAM (DRAM) arrays, a CMOS gate array (that contains error correction code (ECC) logic and control logic), and an XMI interface (the XMI Corner).
- Storage arrays are made up of four banks of 72 DRAMs.
- ECC logic detects single-bit and double-bit errors and corrects single-bit errors.
- Memory self-test checks all RAMS, the data path, and control logic on power-up.
- Quadwords, octawords, and hexwords can be read from memory.
- Quadwords and octawords can be written to memory.
- The memory can be configured by the system for 1-, 2-, 4-, 8-way or no interleaving.

4.2 Technical Description

The MS62A memory module uses XMA logic, DRAM arrays, and a PROM to provide 32-Mbytes of memory to the VAX 6200 system.

The MS62A memory module consists of the following major components:

- XMI Corner
- XMA gate array
- Address and control logic
- DRAMs

The **XMI Corner** is the module's interface to the XMI bus and contains CMOS gate arrays and interface logic. Its primary purpose is to transfer data between the MS62A memory module and the KA62A CPU module.

The **XMA gate array** transfers data between the XMI Corner and the DRAMs. The gate array also controls address multiplexing, command decoding, arbitration, and CSR logic functions.

Address and control logic modifies address bits received from the XMI Corner. These modified address bits are used to control the selection of the DRAMs during reading and writing.

All power for the XMI memory array is supplied from +5VBB. If the optional battery backup unit (BBU) is not installed and VAX 6200 system power is lost, memory is lost as well.

If the optional BBU is installed, it takes over, ensuring that no data is lost during the power interruption. The BBU supplies power to memory for approximately 10 minutes.

4.3 Self-Test and Initialization

The MS62A memory module performs an initialization and self-test sequence on a cold power-up or when the sequence is requested by a console command.

During a cold power-up the gate array chip is initialized, all memory locations are tested, and the control and status registers are initialized.

A warm power-up occurs when the system (excluding memory if a battery backup unit (BBU) is present) looses power. During a warm power-up, self-test is not run and memory contents are unmodified. However, any data in the data path is lost.

Memory self-test takes about 60 seconds to run. While self-test runs, the Fault light on the system front panel is on. When self-test completes, the Fault light goes off and the console printout of self-test begins. For details on the self-test console printout, refer to chapter 6 in the *VAX 6200 Owner's Manual.*

4.4 Starting Address and Interleaving

On power-up the VAX 6200 console firmware loads the Starting and Ending Address Register (SEADR) with the starting address, the interleave mode, and the ending address. The following paragraphs describe how to set the SEADR for proper system operation. Section 4.5 gives a description of the SEADR.

4.4.1 Starting and Ending Addresses

The memory responds to starting addresses on any 2-Mbyte boundary. The ending address is also on any 2-Mbyte boundary. The ending address must be greater than the starting address to ensure that data will not be overwritten. The ending address minus the starting address must be equal to or less than the memory size multiplied by the number of ways interleaved.

EA - SA = Memory Size X (# of ways interleaved)

Starting addresses for memory can be in the range from 0 to 510 Mbytes and ending addresses in the range from 0 to 512 Mbytes. Ending addresses greater than 512 Mbytes are not permitted. The area above 512 Mbytes is reserved for CSR addresses.

4.4.2 Interleaving

Interleaving achieves greater throughput to memory by optimizing memory access time and increasing the effective memory transfer rate. This is done by operating memory modules in parallel.

The memory array supports 1-way, 2-way, 4-way, 8-way or no interleaving at the system level. Up to eight memory array modules can be interleaved. Interleaving is done on hexword boundaries.

4.5 Control and Status Registers

The CSR names and their relative addresses are shown in Table 4–1. Descriptions of the CSRs are also included in this section.

Table 4–1 MS62A Memory Module Control and Status Registers

CSR Name	Mnemonic	Address
Device Register	XDEV	BB[1] + 0000 0000
Bus Error Register	XBER	BB + 0000 0004
Starting and Ending Address Register	SEADR	BB + 0000 0010
Memory Control Register 1	MCTL1	BB + 0000 0014
Memory ECC Error Register	MECER	BB + 0000 0018
Memory ECC Error Address Register	MECEA	BB + 0000 001C
Memory Control Register 2	MCTL2	BB + 0000 0030
TCY Register	TCY	BB + 0000 0034
Interlock Flag Status Registers	IFLG*n*	BB + 0000 000*n*[2]

[1] "BB" refers to the base address of an XMI node (2180 0000 + (node ID x 8000)).

[2] Refer to the Interlock Flag Status Register description for the relative address of the Interlock Flag Status Registers.

The memory contains 24 control and status registers (CSRs) to control the memory and log errors. All CSRs are 32 bits long and respond only to longword read and write transactions. When writing to the CSRs, only full writes are performed. If a parity error occurs during a write operation, the operation is aborted and the contents of the CSRs are unchanged.

Some bits in the registers are cleared on power-up, while others need a one written to them to clear. Only the Interlock Flag Status Registers initialize on a warm start.

The CSRs start at an address dependent upon the node ID. All CSR addresses are designated as BB + *n*, where *n* is the relative offset of the register.

The following definitions apply to the descriptions of the control and status registers.

CRD error – A correctable single-bit error.

RDS error – An uncorrectable double-bit error that occurs when the syndrome bits represent an unused ECC code.

RER error – A general uncorrectable double-bit error indicator that includes an RDS error, a row parity error, a column parity error, or a byte write error.

RO – Indicates a read-only register.

RO, 0 – Indicates a read-only register, cleared on power-up.

R/W – Indicates a read and write register.

R/W, 0 – Indicates a read and write register, cleared on power-up.

R/W1C – Indicates a read and write register, write a one to clear.

R/W1C, 0 – Indicates a read and write register, write a one to clear, and cleared on power-up.

R/W1C, 1 – Indicates a read and write register, set on power-up.

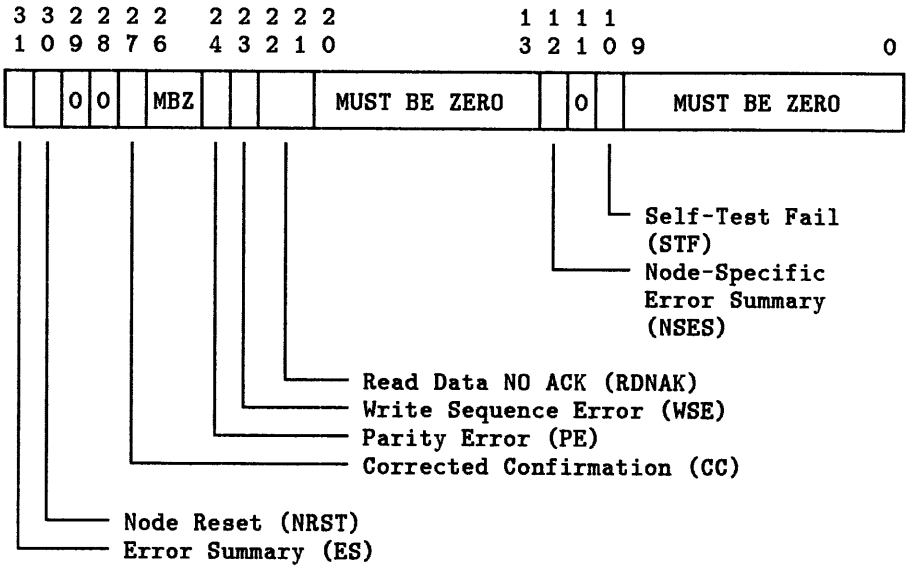
W/O, 0 – Indicates a write only register, cleared on power-up.

Bus Error Register (XBER)

The Bus Error Register records error and status information about the XMI bus.

ADDRESS

Nodespace base address + 0000 0004



bit<31>

Name: Error Summary
Mnemonic: ES
Type: RO, 0

This bit state represents the logical OR of the error bits in this register.

bit<30>

Name: Node Reset
Mnemonic: NRST
Type: W/O, 0

Writing a one to this location initiates a complete node reset, including self-test.

bits<29:28>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<27>

Name: Corrected Confirmation
Mnemonic: CC
Type: R/W1C, 0

This bit is set when the XMI Corner interface (XCI) bus detects a single-bit error on the XMI CNF bits.

bits<26:24>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<23>

Name: Parity Error
Mnemonic: PE
Type: R/W1C, 0

This bit is set when the node detects a parity error on an XMI cycle.

bit<22>

Name: Write Sequence Error
Mnemonic: WSE
Type: R/W1C, 0

When set, indicates that the node aborted a write transaction due to one or more missing data cycles.

bit<21>

Name: Read Data NO ACK
Mnemonic: RDNAK
Type: R/W1C, 0

When set, indicates that the node received a NO ACK confirmation for a data cycle it transmitted.

bits<20:13>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<12>

Name: Node-Specific Error Summary
Mnemonic: NSES
Type: RO, 0

When set, this bit indicates that a node-specific error condition has been detected. The exact nature of the error is located in the memory error status registers.

bit<11>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<10>

Name: Self-Test Fail
Mnemonic: STF
Type: R/W1C, 1

While set, this bit indicates that the node has not yet passed its self-test. This bit is cleared when self-test successfully completes. This bit also drives XMI BAD (an XMI bus signal that reports node failures). Clearing this bit also clears XMI BAD.

bits<9:0>

Name: Reserved
Mnemonic: None
Type: RO

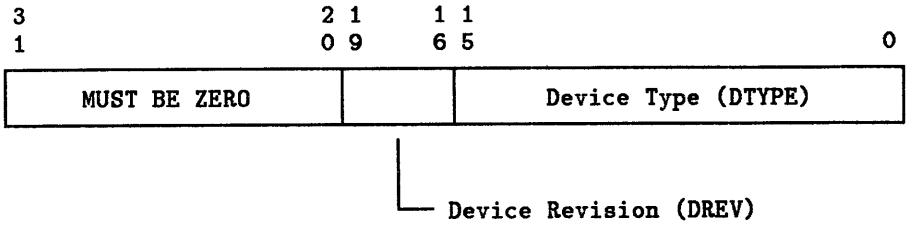
Reserved; must be zero.

Device Register (XDEV)

The Device Register contains information to identify the MS62A memory module. Both fields are loaded during node initialization. A zero value indicates an uninitialized node.

ADDRESS

Nodespace base address + 00000 0000



bits<31:20>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bits<19:16>

Name: Device Revision
Mnemonic: DREV
Type: RO

Identifies the revision level of the MS62A memory module. The use of the Device Revision field is implementation dependent. The field does not indicate the hardware revison level, only the functional level.

bits<15:0>

Name: Device Type
Mnemonic: DTYPE
Type: RO

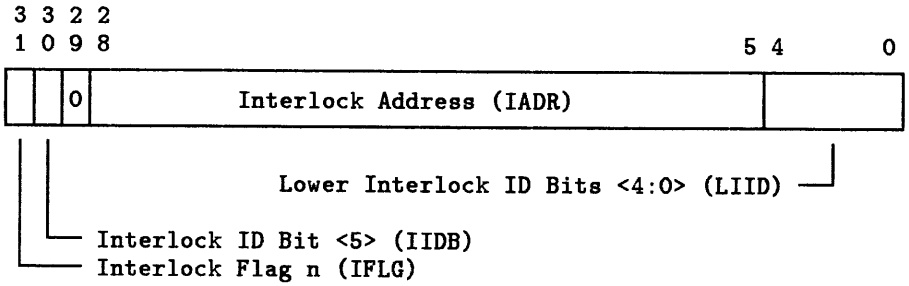
Identifies the type of node. The device type for an MS62A memory module is 4001 (hex). This value is hardwired in the Device Register.

Interlock Flag Register (IFLG*n*)

The Interlock Flag *n* Register (IFLG*n*) (where *n* is 0 to 15) holds the address and ID of the last interlock flag only if all lower interlock flags are set. The locations of IFLG*n* flags are shown in the relative address table.

ADDRESS

Nodespace base address + (relative address)



where n is the number of the Interlock Flag Register Number (0-15)

Interlock Flag Register	Relative Address
Interlock Flag 0 Status Register	BB + 20
Interlock Flag 1 Status Register	BB + 24
Interlock Flag 2 Status Register	BB + 28
Interlock Flag 3 Status Register	BB + 2C
Interlock Flag 4 Status Register	BB + 40
Interlock Flag 5 Status Register	BB + 44
Interlock Flag 6 Status Register	BB + 48
Interlock Flag 7 Status Register	BB + 4C
Interlock Flag 8 Status Register	BB + 80
Interlock Flag 9 Status Register	BB + 84
Interlock Flag 10 Status Register	BB + 88
Interlock Flag 11 Status Register	BB + 8C
Interlock Flag 12 Status Register	BB + 100
Interlock Flag 13 Status Register	BB + 104
Interlock Flag 14 Status Register	BB + 108
Interlock Flag 15 Status Register	BB + 10C

bit<31>

Name: Interlock Flag *n*
Mnemonic: IFLGn
Type: R/W1C, 0

This bit is Interlock Flag *n*, where n = (0-15). If asserted, the Interlock Address and Interlock ID are valid and the lock is set. The lock cannot be set by writing directly to IFLGn. Writing a one to IFLG*n* clears the lock.

bit<30>

Name: Interlock ID <5>
Mnemonic: IIDB
Type: RO, 0

IIDB is the most significant ID bit of the Interlock Read transaction. This bit is valid only if Interlock Flag *n* is set.

bit<29>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<28:5>

Name: Interlock Address
Mnemonic: IADR
Type: RO, 0

IADR gives the address of the Interlock Read transaction. It is valid only if Interlock Flag *n* is set.

bits<4:0>

Name: Lower Interlock ID <4:0>
Mnemonic: LIID
Type: RO, 0

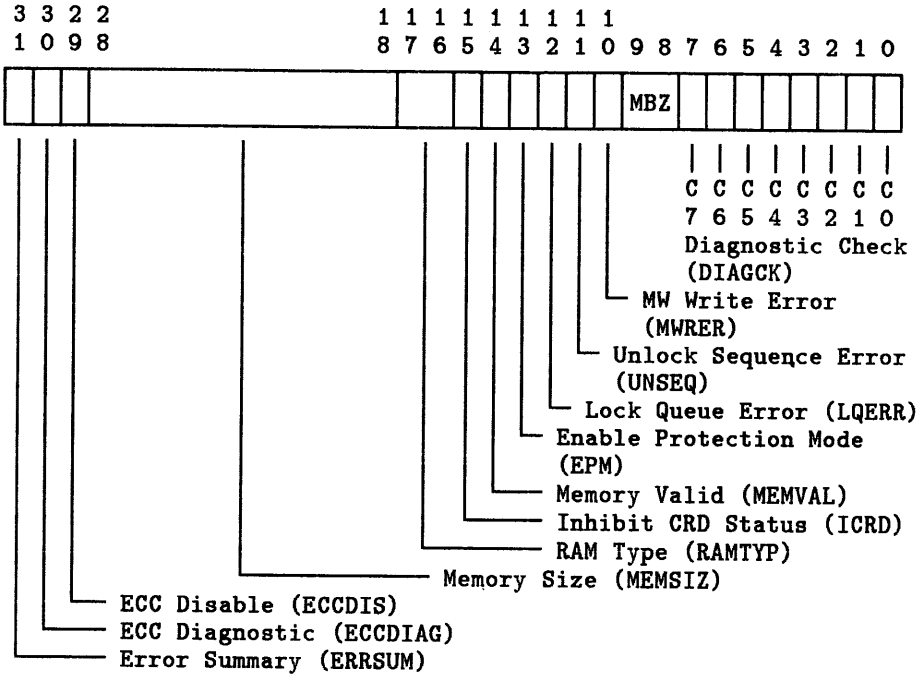
LIID are the lower four ID bits of the Interlock Read transaction. These bits are valid only if Interlock Flag *n* is set.

Memory Control Register 1 (MCTL1)

The Memory Control Register 1 along with the Memory Control Register 2 contains memory-specific control, status, and error bits. The MCTL1 Register also controls the diagnostic modes of the memory module.

ADDRESS

Nodespace base address + 0000 0014



bit<31>

Name: Error Summary

Mnemonic: ERRSUM

Type: RO

This bit contains the ORed sum of error bits in MCTL1, MCTL2, and Memory ECC Error Registers.

bit<30>

Name: ECC Diagnostic
Mnemonic: ECCDIAG
Type: R/W, 0

This bit is used for diagnostic purposes.

bit<29>

Name: ECC Disable
Mnemonic: ECCDIS
Type: R/W, 0

This bit is used for diagnostic purposes.

bits<28:18>

Name: Memory Size
Mnemonic: MEMSIZ
Type: RO

These bits contain the memory module size in 256-Kbyte increments, where 00000011000 = 6 Mbytes, 00000100000 = 8 Mbytes, and 00010000000 = 32 Mbytes.

bits<17:16>

Name: RAM Type
Mnemonic: RAMTYP
Type: RO

These bits contain the size of the RAM.

bit<15>

Name: Inhibit CRD Status
Mnemonic: ICRD
Type: R/W, 0

This bit inhibits the reporting of CRD status to the commander on read cycles. When this bit is set, any CRD response is changed to a GRD response. The CRD errors are still logged and RER errors are logged and reported normally.

bit<14>

Name: Memory Valid
Mnemonic: MEMVAL
Type: RO, 0

This bit indicates that valid data is stored in memory. The bit is set on the first write to the module memory space.

bit<13>

Name: Enable Protection Mode
Mnemonic: EPM
Type: R/W, 0

When this bit is set, the operation of the ECC Diagnostic<30> and ECC Disable <29> bits are inhibited in the first 2 Mbytes of memory space, starting address to starting address plus 2 Mbytes.

bit<12>

Name: Lock Queue Error
Mnemonic: LQERR
Type: R/W1C, 0

This bit is set if a data word is sent as a response to an Interlock Read and no lock is pending in the memory.

bit<11>

Name: Unlock Sequence Error
Mnemonic: UNSEQ
Type: R/W1C, 0

This bit is set if an Unlock Write transaction is accepted and no corresponding matching location is marked as locked. Either an Interlock Read was never performed to this location, the lock did not set, or the lock might have been cleared by another source.

bit<10>

Name: MWrite Error
Mnemonic: MWRER
Type: R/W1C, 0

This bit is set on an RDS error during a partial write cycle.

bits<9:8>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<7:0>

Name: Diagnostic Check
Mnemonic: DIAGCK
Type: R/W, 0

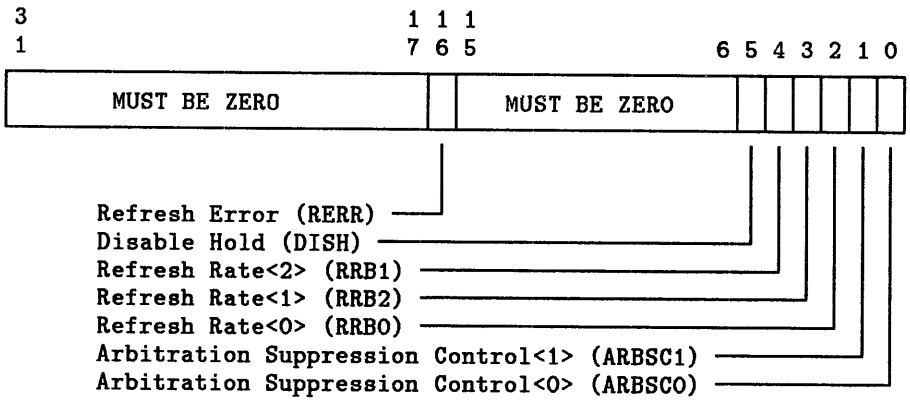
These bits are used during ECC diagnostic mode as substitute check bits.

Memory Control Register 2 (MCTL2)

The second memory control register contains additional control and error status information.

ADDRESS

Nodespace base address + 0000 0030



bits<31:17>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<16>

Name: Refresh Error
Mnemonic: RERR
Type: R/W1C, 0

This bit is set if a refresh request is set, and a second refresh request is asserted before the first one is implemented, meaning that a refresh was missed.

bits<15:6>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<5>

Name: Disable Hold
Mnemonic: DISH
Type: R/W, 0

This bit is used by memory arbitration logic to disable the use of XMI HOLD L.

bits<4:2>

Name: Refresh Rate
Mnemonic: RRB
Type: R/W

This bit controls the module's DRAM refresh rate.

bits<1:0>

Name: Arbitration Supression Control
Mnemonic: ARBSCn
Type: R/W, 0

These bits control the Arbitration Supression mode.

Memory ECC Error Address Register (MECEA)

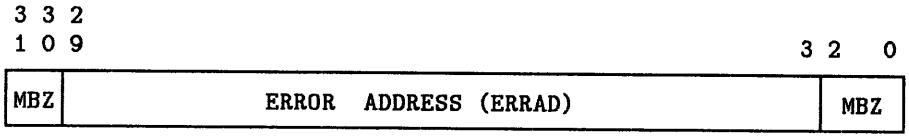
The Memory ECC Error Address Register logs the address of correctable and uncorrectable errors logged in the Memory ECC Error Register.

For read accesses, this register logs the address of the first corrected read data (CRD) error and holds it until a double-bit uncorrectable error (RER) occurs or the error is cleared. An RER error causes a logged CRD error address to be overwritten. A CRD will not overwrite a logged RER error address. If multiple RER errors occur, only the first error address is logged.

This register logs errors during self-test.

ADDRESS

Nodespace base address + 0000 001C



bits<31:30>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

bits<29:3>

Name: Error Address

Mnemonic: ERRAD

Type: RO, 0

The error address of the RER or CRD error logged in the Memory ECC Error Register. This register is valid only if the RER or CRD Error log bits are set in the Memory ECC Error Register. This address is the bus address of the cycle that was being performed at the time of the error.

bits<2:0>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

Memory ECC Error Register (MECER)

The Memory ECC Error Register logs ECC error status. The MECER also logs uncorrectable error codes for row parity error, column parity errors, and byte write errors. The MECER logs ECC error information during read cycles only. If an RER error occurs during a Write Mask cycle, the MWRITE error bit in the MCTL1 Register is set.

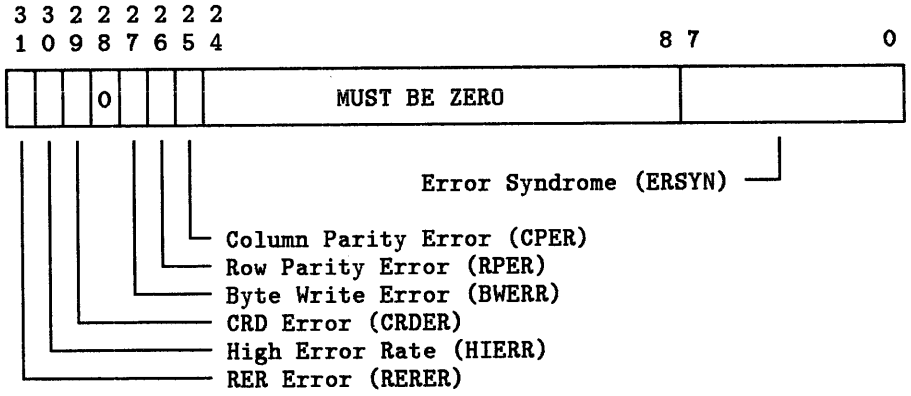
This register logs ECC error type and error syndrome information when correctable and uncorrectable errors occur during Read transactions. During a Write Mask transaction, only the MWRITE error bit logs the fact that the ECC error occurred.

For read accesses, the register logs the first correctable error and holds it until either an uncorrectable error occurs or the error is cleared. Additional correctable errors are only reported and are not logged. An uncorrectable error will overwrite a logged correctable error. A correctable error will not overwrite a logged uncorrectable error or a previously logged correctable error until the error has been cleared.

This register logs errors during module self-test.

ADDRESS

Nodespace base address + 0000 0018



bit<31>

Name: Uncorrectable Double-Bit (RER) Error

Mnemonic: RERER

Type: R/W1C, 0

This bit indicates that an uncorrectable error occurred during a read transaction. The Error Address and Error Syndrome are valid for the uncorrectable double-bit error. If the Column Parity Error bit, the Row Parity Error bit, and the Byte Write Error bit are not all set, then the uncorrectable double-bit error is an RDS error.

bit<30>

Name: High Error Rate
Mnemonic: HIERR
Type: R/W1C, 0

This bit indicates that another error, RER or CRD, occurred before the previous one was cleared from the register.

bit<29>

Name: CRD Error
Mnemonic: CRDER
Type: R/W1C, 0

This bit indicates that a CRD error occurred during a read transaction. This includes a single-bit error in the check bits, even though no correction is done on the data bits. The error address and error syndrome are valid if no RER error log exists.

bit<28>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<27>

Name: Byte Write Error
Mnemonic: BWERR
Type: RO, 0

This bit indicates that the RER error was due to reading a location that was marked bad during a partial write cycle that had previously detected an RER error. Cleared when MECER<31> is cleared.

bit<26>

Name: Row Parity Error
Mnemonic: RPER
Type: RO, 0

This bit indicates that the RER error is due to a row address parity error. Cleared when MECER<31> is cleared.

bit<25>

Name: Column Parity Error
Mnemonic: CPER
Type: RO, 0

This bit indicates that the RER error is due to a column address parity error. Cleared when MECER<31> is cleared.

bits<24:8>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<7:0>

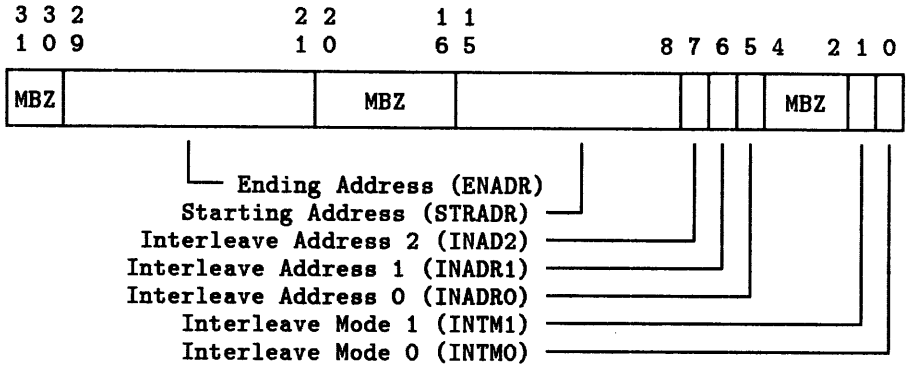
Name: Error Syndrome
Mnemonic: ERSYN
Type: RO, 0

These bits are the syndrome bits of the location in an RER or CRD error.

Starting and Ending Address Register (SEADR)

The Starting and Ending Address Register contains the memory starting and ending addresses. See Section 4.4.1 for a description of the rules that must be followed when setting these addresses. This register also sets the interleave mode.

ADDRESS *Nodespace base address + 0000 0010*



bits<31:30>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<29:21>

Name: Ending Address
Mnemonic: ENDADR
Type: R/W, 0

The Ending Address for the memory on 2-Mbyte boundaries. The memory is enabled if the ending address is greater than the starting address. The ending address range is from 0 to (510 Mbytes + 2 Mbytes).

bits<20:16>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<15:8>

Name: Starting Address
Mnemonic: STRADR
Type: R/W, 0

The Starting Address for the memory on 2-Mbyte boundaries. The starting address range is from 0 to 510 Mbytes.

bits<7:5>

Name: Interleave Address
Mnemonic: INADn
Type: R/W, 0

The address bits used for interleaving. This address determines to what address the module will respond.

bits<4:2>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<1:0>

Name: Interleave Mode
Mnemonic: INTLMn
Type: R/W, 0

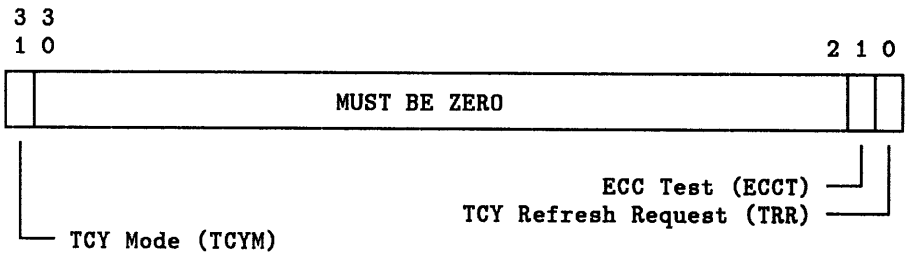
These bits show how many ways the module is being interleaved and are used to determine the addresses that the module will respond to.

TCY Tester Register (TCY)

The TCY Tester Register contains control bits to implement manufacturing tests.

ADDRESS

Nodespace base address + 0000 0034



4.6 Error Handling and Command Responses

The following paragraphs describe how the memory responds to an error condition. The memory performs single-bit correction and double-bit detection on the data stored.

4.6.1 Read Errors

If no errors occur during a read operation, a Good Data (GRDn) function code is returned with the data. If a correctable error occurs during the read operation, a Corrected Read Data (CRDn) function code is returned. If an uncorrectable error occurs, a Read Error Response (RER) is returned in place of the data.

The lock bit is not set if:

- An RER error occurs during an Interlock Read transaction.
- The confirmation of Interlock Read data is missing or bad.

A locked response is sent if:

- The address of an Interlock Read transaction matches a locked hexword.
- All locks are set and memory receives an Interlock Read request.

4.6.2 Full Write Errors

A full write is performed on a quadword or octaword, dependent on the number of mask bits that are set. If mask bits <47:32> are set, an octaword write transaction takes place. If mask bits <39:32> are set, a quadword write transaction takes place. Write data is written into memory with the generated ECC check bits. The write transaction does not begin until all the write data is received from the XMI bus and checked for parity.

If an XMI parity error occurs on one or more quadwords of received data, the write will not begin and a NO ACK response is returned.

4.6.3 Partial Write Errors

If the mask bits for a quadword or octaword are not all set, a partial write is performed. After write data is merged with read data, the write data is written into memory. If the read data is correct, the write is completed. If a correctable read error occurs, the write continues to completion with the corrected data. Uncorrectable read data causes the old data to be rewritten with a Byte Write Error ECC code to mark the location defective. If the cycle is an Unlock Write cycle, an uncorrectable error causes the location to be marked bad and the interlock flag cleared.

If an XMI parity error occurs on one or more quadwords of received data, the write does not begin. If the parity error occurs during an Unlock Write command or data cycle, the lock is not reset.

5 DWMBA XMI-to-VAXBI Adapter

The DWMBA XMI-to-VAXBI adapter provides an information path between the XMI bus and I/O devices on the VAXBI bus.

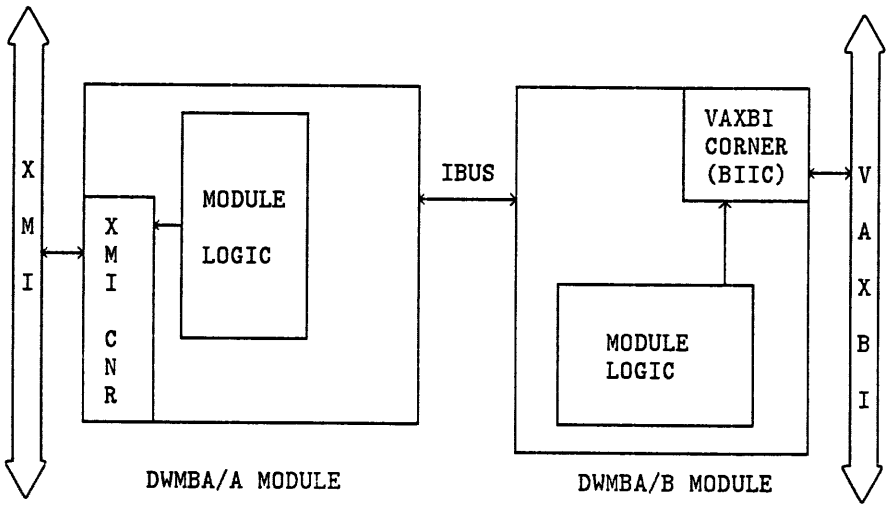
This chapter contains the following sections:

- DWMBA Overview
- CPU Transactions
- DMA Transactions
- DWMBA Registers
- Interrupts
- Error Reporting
- DWMBA Initialization, Self-Test, and Booting

5.1 DWMBA Overview

The DWMBA XMI-to-VAXBI adapter provides an information path between the XMI bus and I/O devices on the VAXBI bus. The DWMBA consists of two modules: the DWMBA/A XMI module and the DWMBA/B VAXBI module. The IBUS connects the two modules.

Figure 5–1 DWMBA XMI-to-VAXBI Adapter Block Diagram



The DWMBA/A module contains an XMI Corner, register files, XMI required registers, DWMBA-specific registers, and control sequencers for the XMI interface.

The DWMBA/B module contains a BIIC, interconnect drivers, control sequencers to handle the control of the data transfer, status bits to/from the DWMBA/A module's register files and the BIIC, DWMBA/B module specific registers, decode logic for DMA operations, and VAXBI clock-generation circuitry.

These two modules are connected by four cables of 30 wires each. The 120 wires make up the IBUS, which transfers data and control information between the two modules.

The DWMBA uses CPU and DMA transactions to exchange information. CPU transactions originate from the KA62A CPU module(s) and are presented to the DWMBA from the XMI bus with the CPU as the XMI commander and the DWMBA as the XMI responder.

DMA transactions originate from VAXBI nodes that select the DWMBA as the VAXBI slave. These are read or write transactions targeted to XMI memory space or are VAXBI-generated interrupt transactions that target a KA62A CPU module. For DMA transactions, the DWMBA is the XMI commander and the MS62A memory module is the XMI responder.

Write transactions, whether DMA or CPU, are always disconnected. This means that as soon as either the CPU or the VAXBI master issues the write, it waits for an ACK confirmation that the command and write data was accepted but not necessarily completed at the destination. If the write fails, an IVINTR is returned.

The VAX 6200 system uses a 30-bit physical address. Chapter 2 describes the XMI address space. The *VAXBI Options Handbook* describes the VAXBI address space. The DWMBA can be both a master and a slave on the VAXBI. As a master, it carries out transactions requested by its XMI devices. As a slave, it responds to VAXBI transactions that select its node.

5.2 CPU Transactions

The DWMBA XMI-to-VAXBI adapter translates XMI transactions into equivalent VAXBI transactions. Regardless of whether the transaction is a read, write, or IDENT, software need not concern itself with the details, as the XMI transaction behaves as it would if it were directed to memory or other XMI devices.

Table 5–1 XMI-to-VAXBI Command Translations

XMI	VAXBI
Longword Read	Longword Read
Quadword Read	Illegal
Octaword Read	Illegal
Hexword Read	Illegal
Longword Interlock Read	Longword Interlock Read (IRCI)
Quadword Interlock Read	Illegal
Octaword Interlock Read	Illegal
Hexword Interlock Read	Illegal
Longword Mask Write	Longword Write Mask (WMCI)
Quadword Mask Write	Illegal
Octaword Unlock Write Mask	Illegal
Longword Unlock Write Mask	Longword Unlock Write Mask (UWMCI)
Quadword Unlock Write Mask	Illegal
Octaword Unlock Write Mask	Illegal
Interrupt Request (INTR)	Illegal
Indentify (IDENT)	IDENT
Implied Vector Interrupt (IVINTR)	Illegal

5.2.1 General Operation

The DWMBA responds to XMI longword transactions. When an XMI commander issues a Read, Interlock Read, Write Mask, Unlock Write Mask, or IDENT targeting the DWMBA, the XMI commander arbitrates for the XMI bus, wins the bus, sends out the function, command, address, ID, and parity. The targeted DWMBA recognizes its ID and returns ACK or NO ACK (for busy, an error, or illegal transaction). Once the DWMBA accepts a CPU transaction from an XMI commander, it asserts the NO ACK confirmation code to all subsequent XMI commanders that attempt a CPU transaction until the current transaction completes.

For Read transactions, the DWMBA decodes the XMI command and determines if the address references VAXBI I/O space or a DWMBA register. If VAXBI address space is referenced, the DWMBA generates a VAXBI Read transaction and waits for the return of read data from the VAXBI. Upon receiving the read data from either the VAXBI or a DWMBA register, the DWMBA arbitrates for the XMI bus as a responder and returns the requested data to the commander. The XMI commander sends confirmation of the receipt of data back to the DWMBA. If the Read fails, the XMI commander retries the Read.

Interlock Read transactions are handled the same as Reads except:

- DWMBA registers do not support Interlock Reads and handle them the same as Reads.
- If the Interlock Read command that targets the VAXBI bus gets a RETRY CNF from the VAXBI, the DWMBA returns the Lock Response back to the XMI commander.

Write transactions to the VAXBI are disconnected. The CPU continues on after the DWMBA/A ACKs the Mask Write and Unlock Write Mask transaction if the command/address (C/A) and data received from the XMI bus is error free. The DWMBA decodes the XMI command and determines if the address references VAXBI I/O space or a DWMBA register. If VAXBI address space is referenced, the DWMBA generates the corresponding VAXBI write transaction. If a DWMBA register is referenced, it is written with the write data. Write errors cause an IVINTR to be returned to the CPU.

5.2.2 VAXBI I/O Space Reads

The two XMI read transactions are Read and Interlock Read. The XMI Interlock Read is translated to a VAXBI IRCI transaction while the XMI Read is translated to a VAXBI Read transaction.

The length of the generated VAXBI transaction must be a longword (D<31:30> = 01 in the VAXBI command/address cycle). XMI address bits<28:25> are forced to zero to map XMI addresses to VAXBI addresses and passed onto the VAXBI. The DWMBA ignores with a NO ACK confirmation any targeted transaction longer than a longword.

If the VAXBI issues a RETRY on an XMI Interlock Read request to VAXBI I/O address space due to the resource being locked by a previous Interlock Read request, the DWMBA issues a Locked Response to the XMI commander.

5.2.3 VAXBI I/O Space Writes

The two XMI writes are Mask Write and Unlock Write Mask. The Mask Write is translated to a VAXBI Write Mask with Cache Intent (WMCI), while the Unlock Write Mask is translated to a VAXBI Unlock Write Mask with Cache Intent (UWMCI).

The length of the generated VAXBI transaction must be a longword (D<31:30> = 01 in the VAXBI command/address cycle). XMI address bits<28:25> are forced to zero and passed onto the VAXBI. The DWMBA ignores with a NO ACK confirmation any targeted XMI transaction longer than a longword. The DWMBA supports interlocked instructions even though the KA62A CPU module never issues interlocked instructions to I/O space.

5.2.4 Interrupts

5.2.4.1 XMI IDENT to VAXBI IDENT

When an XMI CPU issues an XMI IDENT, the DWMBA issues a VAXBI IDENT if the DWMBA does not have a pending interrupt at the IDENT level. The DWMBA/B module fetches the IDENT command from the DWMBA/A module's register file and clears the corresponding level and interrupt sent flip-flops that were previously set by the VAXBI-initiated interrupt, providing that no IBUS parity errors are detected.

The DWMBA/B module writes the received vector data into the CPU read data buffer and notifies the DWMBA/A module that the vector is available. The DWMBA/A module then issues an IDENT response cycle on the XMI (with a Good Read Data response where the function code = 100 and the vector is in bits<15:2>).

5.2.4.2 XMI IDENT with DWMBA Adapter Pending Interrupt

If an XMI IDENT is decoded with an IPL matched by the DWMBA/B module while the DWMBA's interrupt-pending flip-flop is set, the interrupt vector of the DWMBA is issued to the XMI. The IDENT clears both the IPL level 17 sent flip-flop and the DWMBA interrupt-pending flip-flop. The corresponding level 17 VAXBI interrupt-pending flip-flop, if also set, is not cleared, resulting in the DWMBA issuing an XMI INTR transaction.

5.2.4.3 Passive Release of VAXBI Interrupts

If the requesting VAXBI node aborts its interrupt request before the XMI CPU generates an IDENT transaction at that level, the resulting IDENT on the VAXBI gets NO ACKed. The DWMBA then issues a Read Error Response (RER) to the XMI commander.

If an XMI CPU issues an IDENT to the VAXBI and the DWMBA has no pending flip-flops set, the DWMBA issues the IDENT to the VAXBI. The resulting IDENT on the VAXBI gets NO ACKed. The DWMBA then issues a Read Error Response (RER) to the XMI commander and sets the IDENT Error bit in the DWMBA/B module's Error Summary Register (BESR<1>).

5.3 DMA Transactions

The DWMBA XMI-to-VAXBI adapter translates a VAXBI transaction into an XMI bus transaction when a VAXBI node selects the DWMBA as the slave node for a VAXBI transaction. The XMI bus transaction is serviced by a memory node, and the requested data is then read from or written to XMI memory.

Table 5–2 VAXBI-to-XMI Command Translations

VAXBI	XMI
Read	Read
Interlock Read with Cache Intent	Interlock Read
Read with Cache Intent	Read
Write (LW)	Write Mask on the unused longword within the XMI quadword
Write (QW)	Write Mask (QW)
Write (OW)	Write Mask (OW)
Write with Cache Intent (LW)	Write Mask on the unused longword within the XMI quadword
Write with Cache Intent (QW)	Write Mask
Write with Cache Intent (OW)	Write Mask
Unlock Write Mask with Cache Intent (LW)	Unlock Write Mask
Unlock Write Mask with Cache Intent (QW)	Unlock Write Mask
Unlock Write Mask with Cache Intent (OW)	Unlock Write Mask
Write Mask with Cache Intent (LW)	Write Mask on the unused longword within the XMI quadword
Write Mask with Cache Intent (QW)	Write Mask (QW)
Write Mask with Cache Intent (OW)	Write Mask (OW)
Interrupt (INTR)	Interrupt
Identify (IDENT)	Not supported (NO ACK to VAXBI)[1]
Invalidate (INVAL)	Not supported (NO ACK to VAXBI)
Broadcast (BDCST)	Not supported (NO ACK to VAXBI)
Interprocessor Interrupt (IPINTR)[2]	Interrupt at IPL 16
Stop	Not supported (NO ACK to VAXBI)

[1]The DWMBA does not process VAXBI IDENTs onto the XMI bus but the DWMBA's BIIC responds to VAXBI IDENTs that are directed to it if:

- The BIIC detects an error condition that results in a generated interrupt.
- The user sets the force interrupt bits in the appropriate BIIC register.
- External logic such as the IPINTR decode logic asserts the BCI INT signal (pins<7:4> on the BIIC).

[2]See Section 5.3.3.

A VAXBI transaction can reference an address between the addresses in the Starting and Ending Address Registers in the DWMBA's BIIC. VAXBI transactions cannot access DWMBA-specific registers.

5.3.1 VAXBI-to-XMI Memory Space Reads

If the incoming VAXBI transaction is a read-type transaction and the address falls between the address in the DWMBA's BIIC Starting and Ending Address Registers, the slave sequencer determines if a DMA buffer is available for use. If so, the slave sequencer moves the C/A data to the DMA(x) buffer, where x indicates either DMA-A or DMA-B, and notifies the DWMBA/A module that VAXBI C/A data has been loaded and the DWMBA/A module should request the XMI bus. The slave sequencer then issues a STALL response to the VAXBI until the transaction completes.

Later, the DWMBA/A module receives a Read response cycle from XMI memory with the requested data. The DWMBA/A module loads the data into the DMA data buffer and notifies the slave sequencer in the DWMBA/B module that the requested data is available. The slave sequencer then moves the data to the VAXBI, completing the request.

The DWMBA does not support the caching of memory on VAXBI nodes so VAXBI reads are always answered with the VAXBI "don't cache" read status.

5.3.1.1 VAXBI-to-XMI Memory Space Interlock Reads

VAXBI interlock reads (IRCI) behave the same as reads except if a VAXBI node references a location in XMI memory that is locked. Then the memory returns a Locked Response (LOC) to the DWMBA, and the DWMBA issues a RETRY confirmation code to the VAXBI commander, which then releases the VAXBI. The DWMBA returns to idle and awaits the next VAXBI request.

5.3.2 VAXBI-to-XMI Memory Writes

The disconnected write mode of operation is used for VAXBI-to-XMI memory writes, allowing use of the VAXBI by other devices while the DWMBA completes the write on the XMI.

The DWMBA's slave sequencer moves the C/A and write data (whether longword, quadword, or octaword) to an available DMA buffer location when the incoming write-type VAXBI transaction's address falls between the addresses in the DWMBA'S Starting and Ending Address Registers in its BIIC. The slave sequencer then issues an ACK confirmation to the VAXBI.

When the buffer load completes, the slave sequencer notifies the DWMBA/A module's XMI transmit logic that it should request the XMI bus. Upon receiving an XMI grant, the DWMBA transmits the write data transaction and waits for an ACK response.

The DWMBA has two sets of register files, DMA-A and DMA-B, which allows the DWMBA to accept either a second VAXBI write transaction or a VAXBI read transaction before the previous XMI write completes. The DWMBA performs the operations on the XMI in the order that the VAXBI issues the transactions to ensure that out-of-order sequences do not occur.

If a third VAXBI write transaction occurs before the first and second XMI writes complete, the DWMBA stalls this VAXBI transaction until the first XMI write completes successfully.

5.3.3 VAXBI-Generated Interrupts

Interrupts can either be (1) generated by the DWMBA if there is a status change or an error condition or (2) passed through the DWMBA to the XMI bus if generated by various I/O devices on the VAXBI bus. These interrupts are translated into the appropriate XMI interrupt transactions. If a DWMBA and a VAXBI device interrupt are both pending at the same IPL when an XMI IDENT transaction is issued, the DWMBA returns its vector to ensure that DWMBA error interrupts are serviced first.

5.4 DWMBA XMI-to-VAXBI Adapter Registers

Two sets of registers are used by the DWMBA: DWMBA registers (residing on both modules of the DWMBA) and VAXBI registers (residing in the BIIC) . The DWMBA registers include the XMI required registers and DWMBA-specific registers in DWMBA private space.

Table 5–3 lists the DWMBA/A module XMI module registers. Table 5–4 lists the DWMBA/B module VAXBI module registers. Table 5–5 lists the VAXBI registers. See Chapter 5 of the *VAXBI Options Handbook* for a description of the VAXBI registers, except for the VAXBI Device Register. The remainder of Section 5.4 gives detailed descriptions of the DWMBA registers. The DWMBA/A module registers are presented first, followed by the DWMBA/B module registers and the VAXBI Device Register.

See Section 2.2.2.3 for details of I/O addressing.

Table 5–3 XMI Registers on the DWMBA/A Module

Name	Mnemonic[1]	Address[2]
Device Register	XDEV	BB+0000 0000
Bus Error Register	XBER	BB+0000 0004
Failing Address Register	XFADR	BB+0000 0008
Responder Error Address Register	AREAR	BB+0000 000C
Error Summary Register	AESR	BB+0000 0010
Interrupt Mask Register	AIMR	BB+0000 0014
Implied Vector Interrupt Destination/Diagnostic Register	AIVINTR	BB+0000 0018
Diag 1 Register	ADG1	BB+0000 001C

[1]The first letter of the mnemonic indicates the following:

X=XMI register, resides on the DWMBA/A XMI module
A=Resides on the DWMBA/A XMI module
B=Resides on the DWMBA/B VAXBI module

[2]The abbreviation "BB" refers to the base address of an XMI node (the address of the first location of the nodespace).

Table 5–4 XMI Registers on the DWMBA/B Module

Name	Mnemonic[1]	Address[2]
Control and Status Register	BCSR	BB+0000 0040
Error Summary Register	BESR	BB+0000 0044
Interrupt Destination Register	BIDR	BB+0000 0048
Timeout Address Register	BTIM	BB+0000 004C
Vector Offset Register	BVOR	BB+0000 0050
Vector Register	BVR	BB+0000 0054
Diagnostic Control Register 1	BDCR1	BB+0000 0058
Reserved Register	–	BB+0000 005C

[1]The first letter of the mnemonic indicates the following:

X=XMI register, resides on the DWMBA/A module
A=Resides on the DWMBA/A module
B=Resides on the DWMBA/B module

[2]The abbreviation "BB" refers to the base address of an XMI node (the address of the first location of the nodespace).

Table 5–5 VAXBI Registers

Name	Mnemonic	Address[1]
Device Register	DTYPE[2]	bb + 00
VAXBI Control and Status Register	VAXBICSR	bb + 04
Bus Error Register	BER	bb + 08
Error Interrupt Control Register	EINTRSCR	bb + 0C
Interrupt Destination Register	INTRDES	bb + 10
IPINTR Mask Register	IPINTRMSK	bb + 14
Force-Bit IPINTR/STOP Destination Register	FIPSDES	bb + 18
IPINTR Source Register	IPINTRSRC	bb + 1C
Starting Address Register	SADR	bb + 20
Ending Address Register	EADR	bb + 24
BCI Control and Status Register	BCICSR	bb + 28
Write Status Register	WSTAT	bb + 2C
Force-Bit IPINTR/STOP Command Register	FIPSCMD	bb + 30
User Interface Interrupt Control Register	UINTRCSR	bb + 40
General Purpose Register 0	GPR0	bb + F0
General Purpose Register 1	GPR1	bb + F4
General Purpose Register 2	GPR2	bb + F8
General Purpose Register 3	GPR3	bb + FC
Slave-Only Status Register	SOSR	bb + 100
Receive Console Data Register	RXCD	bb + 200

[1]The abbreviation "bb" refers to the base address of a VAXBI node (the address of the first location of the nodespace).

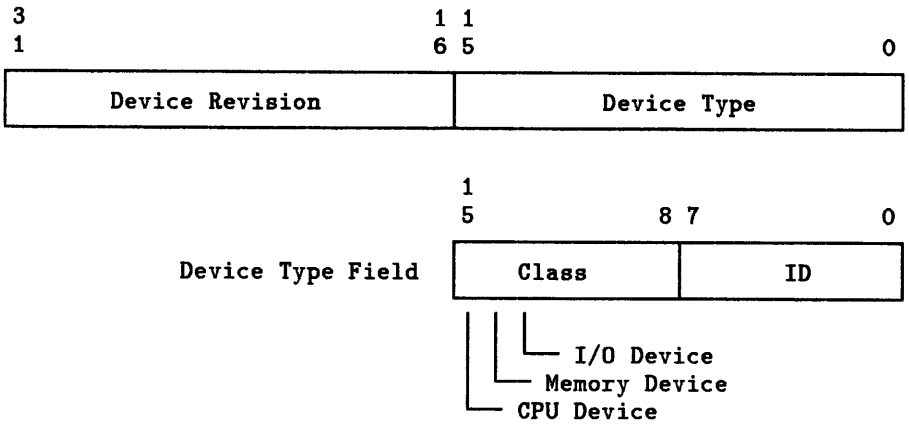
[2]Described in this section.

Device Register (XDEV)

The Device Register contains information to identify the node and is loaded during node initialization. A zero value indicates an uninitialized node.

ADDRESS

XMI nodespace base address + 0000 0000



bits<31:16>

Name: Device Revision

Mnemonic: DREV

Type: RO, 0

Identifies the functional revision level of the module in hexadecimal. The DREV field always reflects the letter revision of the module as follows:

DWMBA/A Adapter Revision	DREV (decimal)	DREV (hex)
A0	1	0001
A1	1	0001
B0	2	0002
B1	2	0002
.		
.		
.		
Z0	26	001A

bits<15:0>

Name: Device Type

Mnemonic: DTYPE

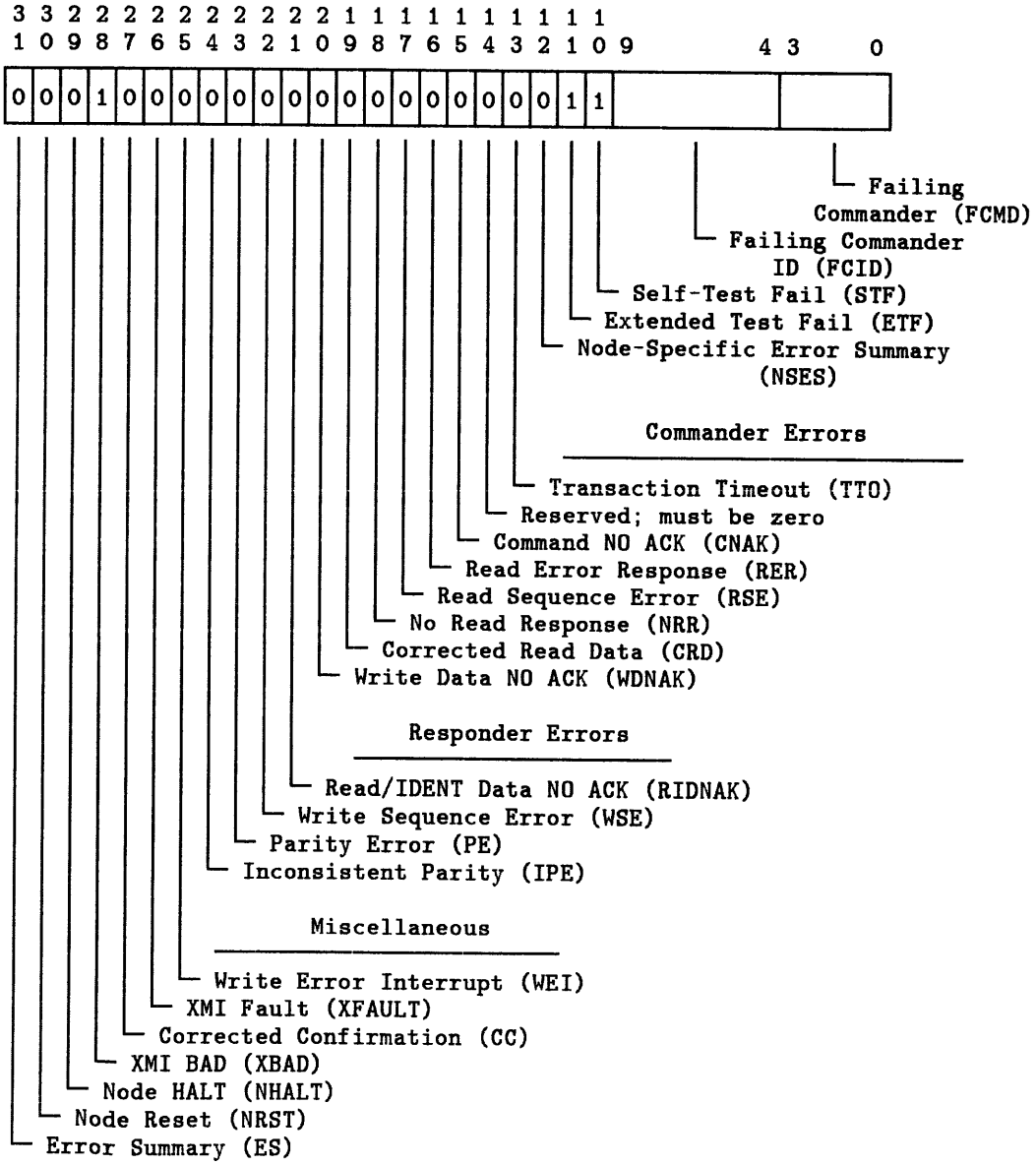
Type: RO, 0

Identifies the type of node. The Device Type field is broken into two subfields: Class and ID. The Class field indicates the major category in which the node falls. The ID field uniquely identifies a particular device within a specified class. DTYPE is 2001 (hex) for the DWMBA/A module.

Bus Error Register (XBER)

The Bus Error Register contains error status on a failed XMI transaction. This status includes the failed command, commander ID, and an error bit that indicates the type of error that occurred. This status remains locked up until software resets the error bit(s).

ADDRESS *XMI nodespace base address + 0000 0004*



bit<31>

Name: Error Summary
Mnemonic: ES
Type: RO, 0

ES represents the logical OR of the error bits in this register. Therefore, ES asserts whenever any error bit asserts.

bit<30>

Name: Node Reset
Mnemonic: NRST
Type: R/W, 0

Writing a one to NRST initiates a power-up reset of the system. Reads to this bit location return zero. When NRST has a one written to it, the DWMBA:

- Resets all logic on the DWMBA/A module to an initialized (power-up) state.
- Asserts the RESET control signal to the DWMBA/B module, sequencing the VAXBI power supply(s). The assertion of RESET to the DWMBA/B causes the DWMBA/B to sequence BI AC LO, and BI DC LO. The assertion of BI DC LO causes the DWMBA/B module to reset to an initialized (power-up) state.

When NRST is set, it remains asserted for six to eight XMI cycles, after which it is cleared by logic on the DWMBA/A module. During the time that the DWMBA is performing its node reset, it does not affect the operation of the XMI bus.

bit<29>

Name: Node HALT
Mnemonic: NHALT
Type: R/W, 0

Unused; must be zero.

bit<28>

Name: XMI BAD
Mnemonic: XBAD
Type: R/W, 1

Unused; must be zero.

bit<27>

Name: Corrected Confirmation
Mnemonic: CC
Type: R/W1C, 0

CC sets when the DWMBA detects a single-bit CNF error. Single-bit CNF errors are automatically corrected by the XCLOCK chip in the XMI Corner.

bit<26>

Name: XMI FAULT
Mnemonic: XFAULT
Type: R/W1C, 0

Unused; must be zero.

bit<25>

Name: Write Error Interrupt
Mnemonic: WEI
Type: R/W1C, 0

Unused; must be zero.

bit<24>

Name: Inconsistent Parity Error
Mnemonic: IPE
Type: R/W1C, 0

Unused; must be zero.

bit<23>

Name: Parity Error
Mnemonic: PE
Type: R/W1C, 0

When set, PE indicates that the DWMBA has detected a parity error on an XMI cycle.

bit<22>

Name: Write Sequence Error
Mnemonic: WSE
Type: R/W1C, 0

When set, WSE indicates that the DWMBA aborted a write transaction directed to it due to missing data cycles.

bit<21>

Name: Read/IDENT Data NO ACK
Mnemonic: RIDNAK
Type: R/W1C, 0

When set, RIDNAK indicates that a Read or IDENT data cycle (GRDn, CRDn, LOC, RER) transmitted by the DWMBA has received a NO ACK confirmation.

bit<20>

Name: Write Data NO ACK
Mnemonic: WDNAK
Type: R/W1C, 0

When set, WDNAK indicates that a Write data cycle (GRDn, CRDn, LOC, RER) transmitted by the DWMBA has received a NO ACK confirmation.

bit<19>

Name: Corrected Read Data
Mnemonic: CRD
Type: R/W1C, 0

When set, CRD indicates that the DWMBA has received a CRDn read response.

bit<18>

Name: No Read Response
Mnemonic: NRR
Type: R/W1C, 0

When set, NRR indicates that a read transaction initiated by the DWMBA failed due to a read response timeout.

bit<17>

Name: Read Sequence Error
Mnemonic: RSE
Type: R/W1C, 0

When set, RSE indicates that a transaction initiated by the DWMBA failed due to a read sequence error.

bit<16>

Name: Read Error Response
Mnemonic: RER
Type: R/W1C, 0

When set, RER indicates that a DWMBA has received a Read Error Response.

bit<15>

Name: Command NO ACK
Mnemonic: CNAK
Type: R/W1C, 0

When set, CNAK indicates that a command cycle transmitted by the DWMBA has received a NO ACK confirmation caused by either a reference to a nonexistent memory location or a command cycle parity error. This bit is set only if the reattempts fail.

bit<14>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bit<13>

Name: Transaction Timeout
Mnemonic: TTO
Type: R/W1C, 0

When set, TTO indicates that a transaction initiated by the DWMBA failed due to a transaction timeout. This bit is set only if the reattempts fail.

bit<12>

Name: Node-Specific Error Summary
Mnemonic: NSES
Type: RO, 0

When set, NSES indicates that a node-specific error condition has been detected. The exact nature of the error is contained in DWMBA-specific registers.

bit<11>

Name: Extended Test Fail
Mnemonic: ETF
Type: R/W1C, 0

Unused; must be zero.

bit<10>

Name: Selt-Test Fail
Mnemonic: STF
Type: R/W1C, 1

When set, STF indicates that the DWMBA has not yet passed its self-test. This bit is cleared by the CPU node that executed the DWMBA self-test when the DWMBA passes its self-test.

bits<9:4>

Name: Failing Commander ID
Mnemonic: FCID
Type: RO

The Failing Commander ID field logs the commander ID of a failing transaction. FCID sets only if the retried transaction fails.

bits<3:0>

Name: Failing Command
Mnemonic: FCMD
Type: RO

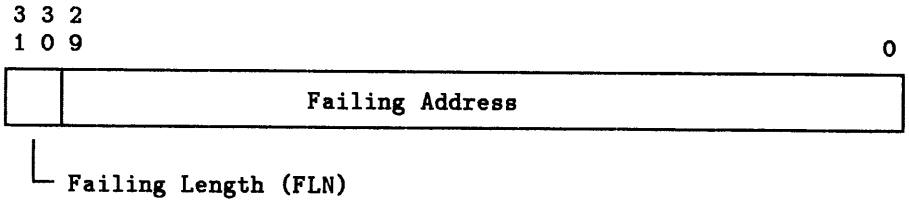
The Failing Command field logs the command code of a failing transaction. FCMD sets only if the retried transaction fails.

Failing Address Register (XFADR)

The Failing Address Register logs address and length information associated with a failing transaction.

ADDRESS

XMI nodespace base address + 0000 0008



bits<31:30>

Name: Failing Length
Mnemonic: FLN
Type: RO

This field logs the value of XMI D<31:30> during the command cycle of a failing transaction.

bits<29:0>

Name: Failing Address
Mnemonic: None
Type: RO

This field logs the value of XMI D<29:0> during the command cycle of a failing transaction.

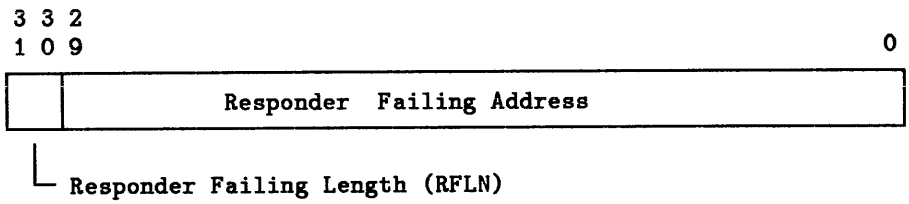
Responder Error Address Register (AREAR)

AREAR logs the failing address received from a CPU node initializing an I/O write, read, or IDENT transaction to the DWMBA or the VAXBI. AREAR is loaded when the DWMBA/A module ACKs the XMI's C/A cycle.

AREAR is locked when the DWMBA is unable to complete the requested operation, either a CPU write transaction that fails, resulting in the I/O Write Failure bit in the DWMBA/A module's Error Summary Register being set or a CPU read or IDENT transaction that results in the setting of the Data NO ACK bit in the DWMBA/A module's XBER register.

ADDRESS

XMI nodespace base address + 0000 000C



bits<31:30>

Name: Responder Failing Length

Mnemonic: RFLN

Type: RO

RFLN logs the value of XMI D<31:30> during the cycle that the DWMBA accepts the C/A cycle for the XMI commander.

bits<29:0>

Name: Responder Failing Address

Mnemonic: None

Type: RO

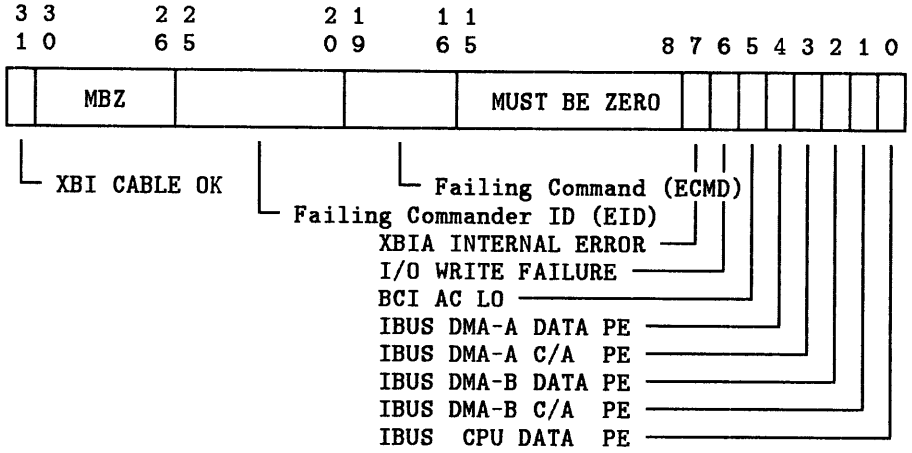
The Responder Failing Address bits log the value of XMI D<29:0> during the cycle that the DWMBA accepts the C/A cycle from the XMI commander.

Error Summary Register (AESR)

AESR is used to capture DWMBA/A module-related error conditions.

ADDRESS

XMI nodespace base address + 0000 0010



bit<31>

Name: XBI Cable OK

Mnemonic: None

Type: RO

XBI Cable OK sets to one on initialization if the four IBUS cables are correctly connected and if the DWMBA/B module has DC power from the VAXBI backplane. If XBI Cable OK clears and the DWMBA/B module has VAXBI DC power, then one or more of the cables is not connected or is incorrectly installed.

bits<30:26>

Name: Reserved

Mnemonic: None

Type: RO, O

Reserved; must be zero.

bits<25:20>

Name: Failing Commander ID

Mnemonic: EID

Type: RO

EID logs the XMI commander ID of a failed DWMBA I/O write, I/O read, or XMI IDENT transaction. The DWMBA will load this register after it ACKs the XMI commander's C/A cycle. EID locks if the DWMBA is unable to complete the requested operation as follows:

1 A failing CPU write transaction that sets the I/O Write Failure bit in the DWMBA/A module's Error Summary Register.

2 A CPU read or IDENT transaction that sets the Data NO ACK bit in the DWMBA/A module's Bus Error Register (XBER).

The lock on EID clears when both of the locking error conditions clear.

bits<19:16>

Name: Failing Command

Mnemonic: ECMD

Type: RO

ECMD logs the XMI commander command of a failed DWMBA I/O write, I/O read, or XMI IDENT transaction. The DWMBA loads this register after it ACKs the XMI commander's C/A cycle. ECMD locks if the DWMBA is unable to complete the requested operation as follows:

1 A failing CPU write transaction that sets the I/O Write Failure bit in the DWMBA/A module's Error Summary Register.

2 A CPU read or IDENT transaction that sets the Data NO ACK bit in the DWMBA/A module's Bus Error Register (XBER).

The lock on EID clears when the locking error conditions clear for both ECMD and EID.

bits<15:8>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bit<7>

Name: XBIA Internal Error

Mnemonic: None

Type: R/W1C, 0

The XBIA Internal Error bit sets to indicate that an UNEXPLAINED internal error to the DWMBA/A module gate array was detected, generally a hardware problem where control logic encountered UNDEFINED conditions. The DWMBA/A module issues an IVINTR transaction with "memory write error" set in the Type field when XBIA Internal Error sets.

bit<6>

Name: I/O Write Failure During CPU Write Transaction

Mnemonic: I/O Write Failure

Type: R/W1C, 0

I/O Write Failure During CPU Write transaction sets if the DWMBA/B module is unable to complete a CPU write transaction to either its register space or to VAXBI address space. Its assertion coincides with the generation of an IVINTR transaction due to this error condition. The DWMBA issues an IVINTR with "mem write error" set in the Type field when I/O Write Failure During CPU Write Transaction is asserted. Software uses this bit and other error bits to determine the cause of a DWMBA-generated IVINTR transaction.

When I/O Write Failure During CPU Write Transaction sets, the contents of the DWMBA/A module Responder Error Address Register, the Failing Commander ID bits, and the Failing Command bits lock.

bit<5>

Name: BCI AC LO

Mnemonic: None

Type: R/W1C, 1

The BCI AC LO bit sets when VAXBI power falls below specifications, as indicated by an asserted BCI AC LO L signal (asserted = one). The DWMBA issues an IVINTR with "mem write error" set in the Type field when BCI AC LO is asserted so that software can determine the cause of this IVINTR transaction. Software then clears BCI AC LO as part of the interrupt service routine that executes as a result of the IVINTR.

The DWMBA self-test program clears BCI AC LO.

bit<4>

Name: IBUS DMA-A Data Parity Error

Mnemonic: None

Type: R/W1C, 0

IBUS DMA-A Data Parity Error sets when the DWMBA/A module detects a parity error on the IBUS when the DWMBA/B module was loading a DMA-A data buffer location. The DWMBA issues an IVINTR with "mem write error" set in the Type field when IBUS DMA-A Data Parity Error asserts.

bit<3>

Name: IBUS DMA-A C/A Parity Error

Mnemonic: None

Type: R/W1C, 0

IBUS DMA-A C/A Parity Error sets when the DWMBA/A module detects a parity error on the IBUS when the DWMBA/B module was loading a DMA-A C/A location. The DWMBA issues an IVINTR with "mem write error" set in the Type field when IBUS DMA-A C/A Parity Error asserts and the failing DMA transaction is a write or interrupt. The DWMBA issues an error interrupt if this error bit is set and the appropriate mask bit is also set.

bit<2>

Name: IBUS DMA-B Data Parity Error

Mnemonic: None

Type: R/W1C, 0

IBUS DMA-B Data Parity Error sets when the DWMBA/A module detects a parity error on the IBUS when the DWMBA/B module was loading a DMA-B data buffer location. The DWMBA issues an IVINTR with "mem write error" set in the Type field when IBUS DMA-B Data Parity Error asserts.

bit<1>

Name: IBUS DMA-B C/A Parity Error

Mnemonic: None

Type: R/W1C, 0

IBUS DMA-B C/A Parity Error sets when the DWMBA/A module detects a parity error on the IBUS when the DWMBA/B module was loading a DMA-B C/A location. The DWMBA issues an IVINTR with "mem write error" set in the Type field when IBUS DMA-B C/A Parity Error asserts and the failing DMA transaction is a write. The DWMBA issues an error interrupt if this error bit is set and the appropriate mask bit is also set.

bit<0>

Name: IBUS CPU DATA Parity Error

Mnemonic: None

Type: R/W1C, 0

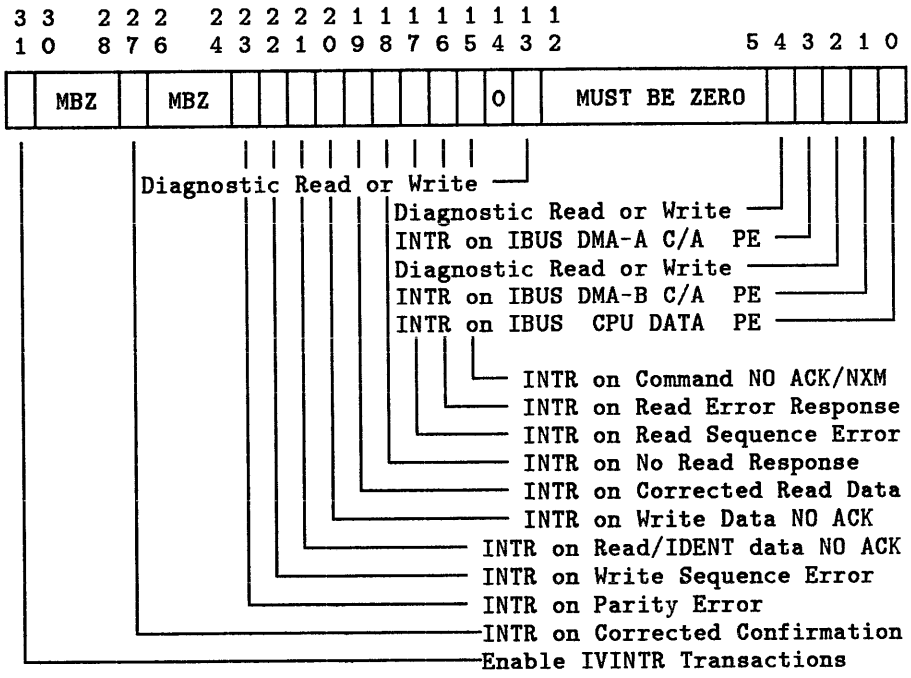
IBUS CPU DATA Parity Error sets when the DWMBA/A module detects a parity error on the IBUS when the DWMBA/B module was loading CPU DATA location during a CPU-initiated I/O read or IDENT. The DWMBA issues a Read Error Response (RER) to the commander when IBUS CPU DATA Parity Error asserts. The DWMBA issues an error interrupt to the XMI if this error bit is set and the appropriate mask bit is also set.

Interrupt Mask Register (AIMR)

AIMR enables/disables the generation of an error interrupt transaction when the corresponding error bit in both the DWMBA/A module's XMI Bus Error Register (XBER) and the DWMBA/A module's Error Summary Register (AESR) is set.

ADDRESS

XMI nodespace base address + 0000 0014



bit<31>

Name: Enable IVINTR Transactions

Mnemonic: None

Type: R/W, 0

When Enable IVINTR Transactions is set and the IVINTR Destination Register is properly configured, IVINTRs are enabled and can be issued on the XMI bus.

CAUTION: **The Enable IVINTR Transactions bit MUST be set to ensure proper error reporting in the case of asynchronous write failures and to report the occurrence of a pending VAXBI power-fail not initiated by XMI AC LO, XMI DC LO, or XBI Node Reset.**

bits<30:28>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero

bit<27>

Name: INTR on Corrected Confirmation

Mnemonic: None

Type: R/W, 0

When INTR on Corrected Confirmation sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<23> (PE) is set.

bits<26:24>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bit<23>

Name: INTR on Parity Error

Mnemonic: None

Type: R/W, 0

When the INTR on Parity Error bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<23> (PE) is set.

bit<22>

Name: INTR on Write Sequence Error
Mnemonic: None
Type: R/W, 0

When the INTR on Write Sequence Error bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<22> (WSE) is set.

bit<21>

Name: INTR on Read/IDENT NO ACK
Mnemonic: None
Type: R/W, 0

When the INTR on Read/IDENT NO ACK sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<21> (RIDNAK) is set.

bit<20>

Name: INTR on Write Data NO ACK
Mnemonic: None
Type: R/W, 0

When the INTR on Write Data NO ACK sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<20> (WDNAK) is set.

bit<19>

Name: INTR on Corrected Read Data
Mnemonic: None
Type: R/W, 0

When the INTR on Corrected Read Data bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<19> (CRD) is set.

bit<18>

Name: INTR on No Read Response
Mnemonic: None
Type: R/W, 0

When the INTR on No Read Response bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<18> (NRR) is set.

bit<17>

Name: INTR on Read Sequence Error
Mnemonic: None
Type: R/W, 0

When the INTR on Read Sequence Error bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<17> (RSE) is set.

bit<16>

Name: INTR on Read Error Response
Mnemonic: None
Type: R/W, 0

When the INTR on Read Error Response bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<16> (RER) is set.

bit<15>

Name: INTR on Command NO ACK
Mnemonic: None
Type: R/W, 0

When the INTR on Command NO ACK bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<15> (CNAK) is set.

bit<14>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<13>

Name: Diagnostic Read or Write
Mnemonic: None
Type: RO, X

Diagnostic Read or Write is used by diagnostic tests.

bits<12:5>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<4>

Name: Diagnostic Read or Write
Mnemonic: None
Type: RO, X

Diagnostic Read or Write is used by diagnostic tests.

bit<3>

Name: INTR on IBUS DMA-A C/A PE
Mnemonic: None
Type: R/W, 0

When the INTR on IBUS DMA-A CA PE bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if a parity error was detected on the IBUS when the DWMBA/B module was loading a DMA-A C/A location.

bit<2>

Name: Diagnostic Read or Write
Mnemonic: None
Type: RO, X

Diagnostic Read or Write is used by diagnostic tests.

bit<1>

Name: INTR on IBUS DMA-B C/A PE
Mnemonic: None
Type: R/W, 0

When the INTR on IBUS DMA-B C/A PE bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if a parity error was detected on the IBUS when the DWMBA/B module was loading a DMA-B C/A location.

bit<0>

Name: INTR on IBUS CPU DATA PE

Mnemonic: None

Type: R/W, 0

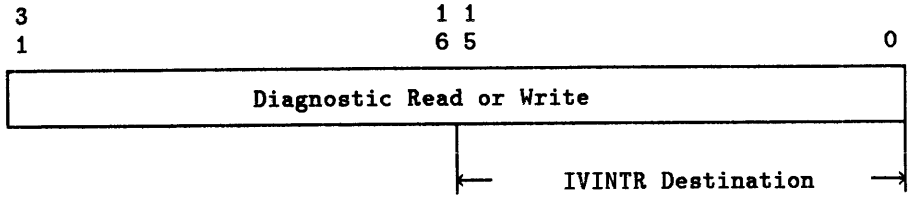
When the INTR on IBUS CPU DATA PE bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if a parity error was detected on the IBUS when the DWMBA/B module was loading the CPU data location.

Implied Vector Interrupt Destination/Diagnostic Register (AIVINTR)

The AIVINTR is used during diagnostics and DWMBA-initiated IVINTR transactions.

ADDRESS

XMI nodespace base address + 0000 0018



bits<31:0>

Name: Diagnostic Read or Write

Mnemonic: None

Type: R/W

The Diagnostic Read or Write bits are used by diagnostic routines to verify the integrity of the DWMBA/A module's main data path inside the DWMBA/A module gate array. When used in this manner, diagnostics need to raise the processor's IPL level above IPL 30 so that, should an error occur causing the DWMBA/A module to issue an IVINTR transaction, an unexpected interrupt will not occur.

During DWMBA-initiated IVINTR transactions, bits<15:0> are used as IVINTR Destination bits.

bits<15:0>

Name: IVINTR Destination

Mnemonic: None

Type: R/W, 0

The IVINTR Destination bits determine which nodes on the XMI will be targeted by the DWMBA when it issues an Implied Vector Interrupt transaction. Each of the 16 bits corresponds to one of the 16 XMI nodes (only 14 nodes are used in the VAX 6200). When a bit is set, the selected node will be the target. For example, if bit<12> becomes set, then XMI node 12 is the node that the DWMBA selects to participate in the IVINTR transaction. Any number of bits can be set.

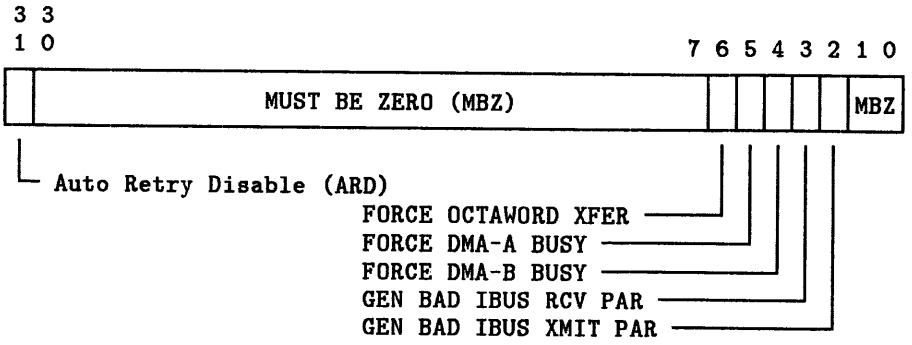
A second use for the IVINTR Destination bits is by diagnostics.

Diag 1 Register (ADG1)

ADG1 is used by diagnostics to test parity and other features in the DWMBA/A module and the IBUS.

ADDRESS

XMI nodespace base address + 0000 001C



bit<31>

Name: Auto Retry Disable

Mnemonic: ARD

Type: R/W, 0

Setting Auto Retry Disable disables reattempts of failed XMI commander transfers. XMI error indications (NO ACKs) are immediately logged in the XMI Bus Error Register, and the appropriate action is taken.

CAUTION: **A NO ACK confirmation is a legal response that an XMI node may issue if it is currently unable to respond to the requested transaction because it is busy. If the user sets Auto Retry Disable, the user must ensure that either a "busy" NO ACK cannot be issued by the targeted node on the XMI or the DWMBA has the capability to handle a transaction that may not complete.**

bits<30:7>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bit<6>

Name: Force Octaword Transfers

Mnemonic: None

Type: R/W, 0

When Force Octaword Transfers is set, the DWMBA/A module generates octaword DMA transactions regardless of the length code that the DWMBA/B module loaded into the DMA buffer. The Force Octaword Transfers bit is used with Force DMA-A/B Busy (ADG1<5:4>), Flip FADR bit 1 (BDCR1<6>), and Flip Bit 29 (BDCR1<4>) to allow diagnostics to test the DWMBA's DMA buffer memory using CPU loopback transactions to XMI memory.

CAUTION: When Flip Bit 29 (BDCR1<4>) has been set to use the diagnostic feature "DMA loopback mode," only LEGAL addresses are permitted. ILLEGAL addresses result in UNDEFINED data. The CPU-generated address must be either 2xxx xxx0 or 2xxx xxx4 to be legal. The following are ILLEGAL addresses: 2xxx xxx8 and 2xxx xxxC.

bit<5>

Name: Force DMA-A Buffer Busy

Mnemonic: None

Type: R/W, 0

When set, the Force DMA-A Buffer Busy bit forces the DMA buffer control logic to place the DMA-A buffer into the BUSY state, forcing all DMA traffic through the DMA-B buffer.

CAUTION: If both ADG1<5> and ADG1<4> are set, all DMA transactions (VAXBI transactions that select the DWMBA as the slave and whose address falls within the bounds of the Starting and Ending Address Registers) will stall.

bit<4>

Name: Force DMA-B Buffer Busy

Mnemonic: None

Type: R/W, 0

When set, the Force DMA-B Buffer Busy bit forces the DMA buffer control logic to place the DMA-B buffer into the BUSY state, forcing all DMA traffic through the DMA-A buffer.

CAUTION: If both ADG1<5> and ADG1<4> are set, all DMA transactions (VAXBI transactions that select the DWMBA as the slave and whose address falls within the bounds of the Starting and Ending Address Registers) will stall.

bit<3>

Name: General Bad IBUS Receiver Parity

Mnemonic: GEN BAD IBUS RCV PAR

Type: R/W, 0

Setting GEN BAD IBUS RCV PAR causes the parity check bit on the DWMBA/A module for IBUS parity to be a one, regardless of the data that is loaded onto the buffer. Diagnostic routines use this bit and specific data patterns to force IBUS parity check errors on the DWMBA/A module when the DWMBA/B module loads the contents of the C/A or data buffers contained in the DWMBA/A module gate array.

bit<2>

Name: General Bad IBUS Transmit Parity

Mnemonic: GEN BAD IBUS XMIT PAR

Type: R/W, 0

Setting GEN BAD IBUS XMIT PAR causes the parity bit sent to the DWMBA/B module for IBUS parity to be a one, regardless of the data that resides in the buffer. Diagnostic routines use this bit and specific data patterns to force IBUS parity errors on the DWMBA/B module when the DWMBA/B module fetches the contents of the C/A or data buffers contained in the DWMBA/A module gate array.

bits<1:0>

Name: Reserved

Mnemonic: None

Type: RO, 0

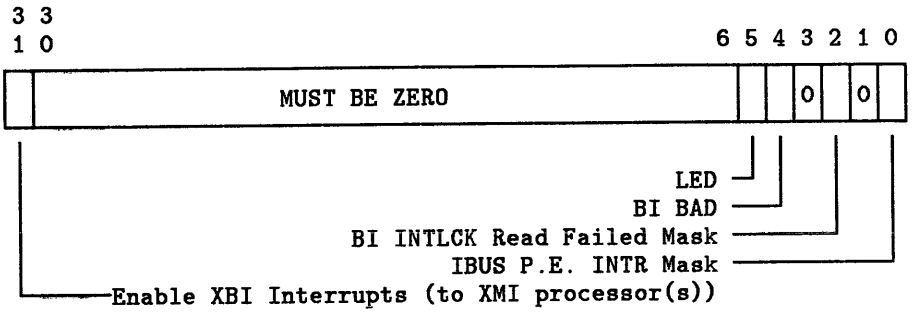
Reserved; must be zero.

Control and Status Register (BCSR)

BCSR contains DWMBA/B module operational control and status bits.

ADDRESS

XMI nodespace base address + 0000 0040



bit<31>

Name: Enable XBI Interrupts

Mnemonic: None

Type: R/W, 0

Setting Enable XBI Interrupts enables the DWMBA to generate XMI interrupt requests in response to DWMBA-generated or VAXBI-generated interrupts. The appropriate interrupt mask bits must also be set for interrupts to be generated.

bits<30:6>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bit<5>

Name: LED

Mnemonic: None

Type: R/W, 0

LED powers up cleared, causing LED D1 to be off. Writing a one to this bit turns LED D1 on.

bit<4>

Name: BI BAD
Mnemonic: None
Type: RO

The initial state of the BI BAD bit on power-up or reset reflects the state of the BI BAD L line on the VAXBI by monitoring the line. It is used by console initialization software and error handling software to detect faulty VAXBI nodes. The assertion of BI BAD L on a VAXBI node results in the assertion of the XMI BAD line.

bit<3>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<2>

Name: BI Interlock Read Failed Mask
Mnemonic: None
Type: R/W, 0

Setting BI Interlock Read Failed Mask to a one causes the DWMBA to generate an error interrupt request if BESR<2> (BI Interlock Read Failed) is set.

bit<1>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<0>

Name: IBUS Parity Error Interrupt Mask
Mnemonic: None
Type: R/W, 0

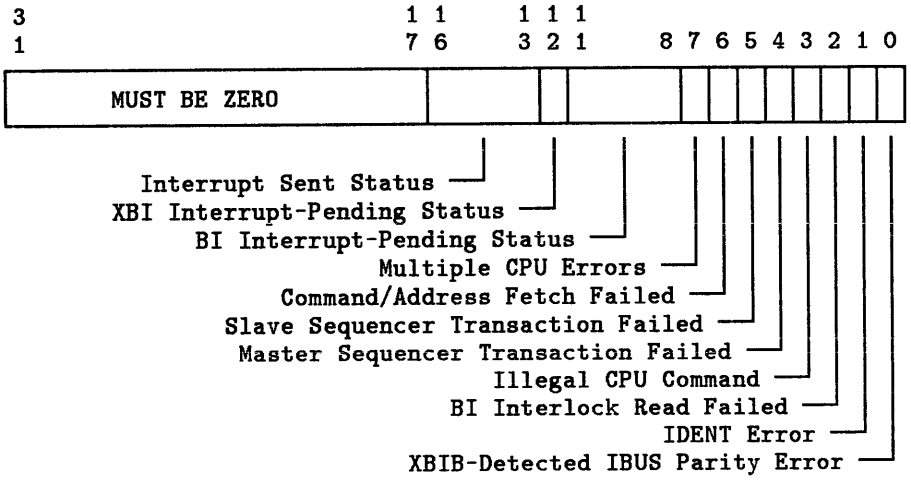
Setting IBUS Parity Error Interrupt Mask to one causes the DWMBA to generate an error interrupt request if BESR<0> (XBIB-Detected IBUS Parity Error) is set.

Error Summary Register (BESR)

The BESR contains status bits for errors detected by the DWMBA/B module.

ADDRESS

XMI nodespace base address + 0000 0044



bits<31:17>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bits<16:13>

Name: Interrupt Sent Status

Mnemonic: None

Type: RO, 0

The Interrupt Sent Status bits correspond to the 4-bit interrupt sent flops internal to the gate array, with BESR<16> corresponding to IPL<17>, BESR<15> corresponding to ILP<16>, etc. The interrupt sent status flops and BSER<12:8> determine the current interrupt-pending status.

bit<12>

Name: XBI Interrupt-Pending Status

Mnemonic: None

Type: RO, 0

The XBI Interrupt-Pending Status bit is a direct read of the XBI interrupt-pending flip-flop. A one indicates that a DWMBA interrupt is pending.

bits<11:8>

Name: BI Interrupt-Pending Status

Mnemonic: None

Type: RO, 0

The BI Interrupt-Pending Status bits set to indicate that one or more of the VAXBI interrupt-pending flip-flops is set. When asserted, they indicate that a VAXBI-generated interrupt targeting the DWMBA was successfully received and that a CPU IDENT at the correct IPL has not yet been received. These bits are a direct read of the VAXBI interrupt-pending flip-flops, with BESR<11> corresponding to IPL<17> and BESR<8> corresponding to IPL<14>.

bit<7>

Name: Multiple CPU Errors

Mnemonic: None

Type: R/W1C, 0

Multiple CPU Errors sets when BESR<4> and BESR<0> have previously set due to a CPU transaction IBUS parity error when C/A or data is removed from the CPU buffer. This indicates that an error occurred on a subsequent CPU transaction before software had acknowledged a previously failed CPU transaction. This bit does not set on a parity error on write data accompanying the command/address on which an error was detected since the transaction has already been recorded as having failed.

bit<6>

Name: Command/Address Fetch Failed

Mnemonic: C/A Fetch Failed

Type: RO, 0

C/A Fetch Failed, when set with BESR<0> set, indicates that the DWMBA/B module detected an IBUS parity error on the C/A fetch from the CPU C/A buffer. C/A Fetch Failed will NOT set on a DWMBA/B module detected IBUS parity error when write data is fetched from the CPU Write Data buffer.

bit<5>

Name: Slave Sequencer Transaction Failed

Mnemonic: None

Type: RO, 0

Slave Sequencer Transaction Failed sets with BESR<0> to indicate that an IBUS parity error occurred while the slave sequencer had control of the IBUS during a read data fetch from the DMA read buffer.

bit<4>

Name: Master Sequencer Transaction Failed

Mnemonic: None

Type: RO, 0

Master Sequencer Transaction Failed sets with BESR<0> to indicate that an IBUS parity error occurred while the master sequencer had control of the IBUS during a C/A or write data fetch from the CPU buffer.

NOTE: This bit will be set but NOT VALID unless bit<0> in this register is also set.

bit<3>

Name: Illegal CPU Command

Mnemonic: None

Type: RO

Illegal CPU Command sets to indicate that an illegal CPU command was decoded by the DWMBA/B module. This error occurs only if an undetected multi-bit parity error happened during the time when the DWMBA/B module fetches the commmand/address from the CPU buffer. The error results in the master sequencer terminating the transaction and signaling the DWMBA/A module that the transaction failed.

The Illegal CPU Command bit does NOT generate an error interrupt.

bit<2>

Name: BI Interlock Read Failed

Mnemonic: None

Type: R/W1C, 0

BI Interlock Read Failed sets to indicate that a VAXBI-to-XMI memory Interlock Read operation failed to successfully complete on the VAXBI. When this error occurs, it is highly probable that the lock set in XMI memory will not be unlocked by the VAXBI device that issued the Interlock Read. The contents of the Timeout Address Register and the setting of BI Interlock Read Failed can be used to determine the locked address in XMI memory. The operating system can clear the lock in XMI memory by writing to a specific CSR in XMI memory.

BI Interlock Read Failed sets whenever a VAXBI Interlock Read command has been decoded and the summary EV code Illegal CNF Received for Slave Data (ICRSD) is decoded during a VAXBI Interlock Read transaction. Setting BI Interlock Read Failed locks the contents of the Timeout Address Register. Writing a one to BI Interlock Read Failed clears both the bit and its lock on the register.

When BI Interlock Read Failed is set with its corresponding mask bit, an error interrupt request is generated.

bit<1>

Name: IDENT Error

Mnemonic: None

Type: R/W1C, 0

IDENT Error sets to indicate that the DWMBA received an XMI IDENT transaction and no VAXBI nor DWMBA interrupt requests were pending at the IDENTed IPL. A set IDENT Error indicates an error condition on the XMI bus with multiple IDENTs being issued on the XMI for the same interrupt transaction. (Only one XMI IDENT is issued on the XMI if a single interrupt targets multiple CPUs.) All other CPUs that are waiting for an XMI bus grant to issue their XMI IDENTs will cancel their IDENT transactions if they see an IDENT transaction that matches the node ID and IPL of the IDENT that they are waiting to issue.

IDENT Error sets if a CPU IDENT command is decoded and no interrupts are pending in the DWMBA/B module gate array.

The setting of IDENT Error does NOT generate a DWMBA error interrupt.

bit<0>

Name: XBIB-Detected IBUS Parity Error

Mnemonic: None

Type: R/W1C, 0

XBIB-Detected IBUS Parity Error sets if the DWMBA/B module detects an IBUS parity error on a CPU transaction's C/A cycle, on a write data cycle when the data is removed from the CPU buffer by the master sequencer, or on a DMA transaction read data cycle when the read data is removed from the DMA read buffer by the slave sequencer. When XBIB-Detected IBUS Parity Error sets, the appropriate bit of BESR<6:4> sets.

The Timeout Address Register also locks on IBUS parity errors detected during DMA read data fetches from the buffer.

Writing a one to XBIB-Detected IBUS Parity Error also clears BESR<6:4> and the lock on the Timeout Address Register.

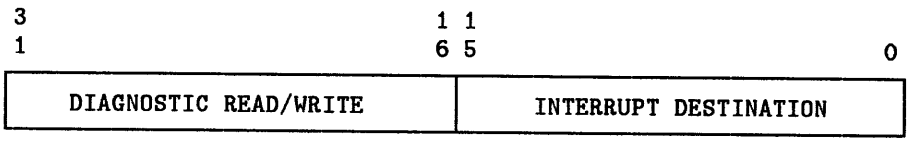
When the XBIB-Detected IBUS Parity Error bit is set with its corresponding mask bit, an error interrupt request is generated.

Interrupt Destination Register (BIDR)

BIDR is used by the DWMBA module to determine the targeted nodes on the XMI for an interrupt transaction. BIDR is used by both VAXBI-initiated and DWMBA error/status-initiated interrupts.

ADDRESS

XMI nodespace base address + 0000 0048



bits<31:0>

Name: Diagnostic Read/Write

Mnemonic: None

Type: R/W

Diagnostic R/W bits are used by diagnostics to verify much of the data path integrity of the DWMBA/B module gate array.

bits<15:0>

Name: Interrupt Destination

Mnemonic: None

Type: R/W, 0

The Interrupt Destination bits determine the nodes on the XMI that are targeted by the DWMBA when it issues an interrupt transaction. Each bit in the 16-bit field corresponds to one of the 16 XMI nodes (only 14 nodes are used in the VAX 6200). When a bit is set to one, the selected node is the targeted node that the DWMBA will interrupt. Multiple bits can be set to interrupt as many XMI nodes as the user desires.

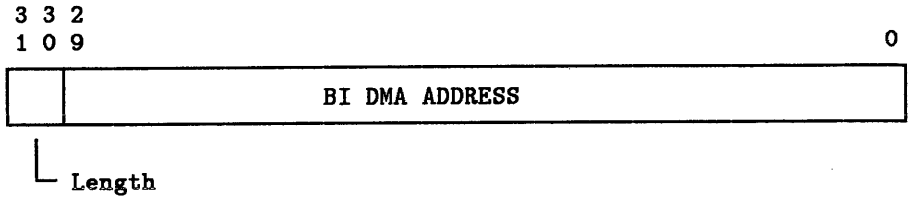
During diagnostics, bits<15:0> are used as part of the Diagnostic Read/Write bits<31:0>, as described above.

Timeout Address Register (BTIM)

The Timeout Address Register is loaded each time a VAXBI command/address is latched off the VAXBI. BTIM locks when (1) a VAXBI-to-XMI memory Interlock Read fails, causing the BI Interlock Read Failed bit (BESR<2>) to set, or (2) a VAXBi-to-XMI memory read-type fails, causing the XBIB-Detected IBUS Parity Error bit (BESR<0>) to set.

ADDRESS

XMI nodespace base address + 0000 004C



bits<31:0>

Name: BI DMA Failing Address

Mnemonic: None

Type: RO

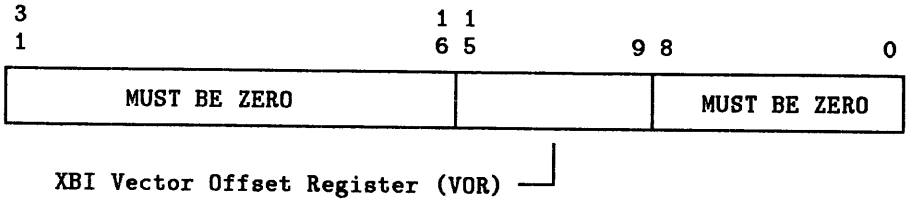
The BI DMA Failing Address contains the longword physical address (bits<29:0>) and length of the received VAXBI-to-XMI transaction (bits<31:30>.) If no errors are detected, the register reads back the last VAXBI transaction. The register logically locks upon error and unlocks when that error clears.

Vector Offset Register (BVOR)

BVOR contains a value that is concatenated with the VAXBI device-supplied vector, if bits<13:9> of the VAXBI-supplied vector are equal to zero.

ADDRESS

XMI nodespace base address + 0000 0050



bits<31:16>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bits<15:9>

Name: XBI Vector Offset Register

Mnemonic: VOR

Type: R/W, 0

BVOR is a 7-bit register loaded by software upon system initialization. BVOR contains a value that is concatenated with the VAXBI device-supplied vector, providing that bits<13:9> of the VAXBI-supplied vector are equal to zero, ensuring that multiple DWMBA/VAXBIs with the same devices on each bus will have a unique entry point into the SCB.

bits<8:0>

Name: Reserved

Mnemonic: None

Type: RO, 0

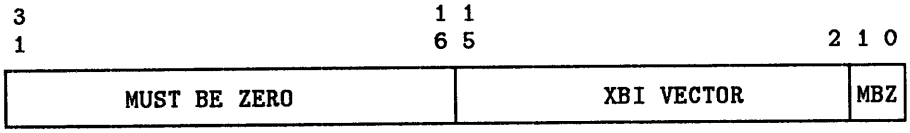
Reserved; must be zero.

Vector Register (BVR)

BVR is loaded by software upon system initialization. BVR contains the DWMBA vector that will be transmitted to the IDENTing XMI node when the DWMBA has a pending interrupt request that matches the interrupt source and IPL sent during the XMI IDENT transaction.

ADDRESS

XMI nodespace base address + 0000 0054



bits<31:16>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bits<15:2>

Name: XBI Vector

Mnemonic: None

Type: R/W, 0

The XBI vector is transmitted to the IDENTing XMI node when the DWMBA has a pending interrupt request that matches the interrupt source and IPL sent during the XMI IDENT transaction. This vector is NOT sent for any VAXBI-generated interrupts or BIIC interrupts due to error conditions.

bits<1:0>

Name: Reserved

Mnemonic: None

Type: RO, 0

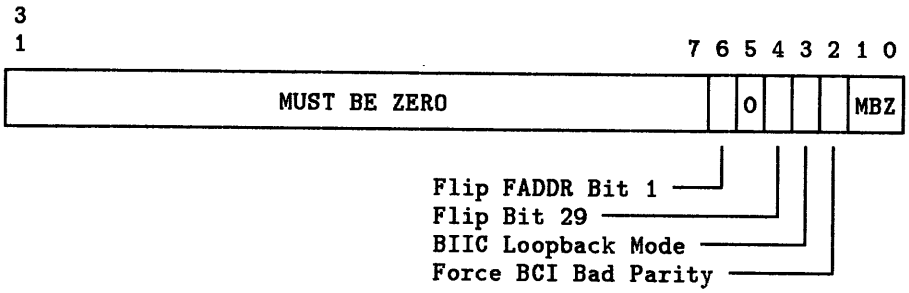
Reserved; must be zero.

Diagnostic Control Register 1 (BDCR1)

The BDCR1 is used by diagnostics to perform various diagnostic functions on the DWMBA/B module, ensuring that its hardware operates properly.

ADDRESS

XMI nodespace base address + 0000 0058



bits<31:7>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bit<6>

Name: Flip FADDR Address Bit 1

Mnemonic: None

Type: R/W, 0

The Flip FADDR Address Bit 1, used with Force DMA-A/B Busy bits (ADG1<5:4>) and Flip Bit 29, enables diagnostics to test the DWMBA's DMA buffer memory using CPU loopback transactions to XMI memory. When Flip FADDR Address Bit 1 is set, the invert state of FADDR Address Bit 1 is used to address the data words in the buffer, allowing diagnostics to use the buffer locations that normally would only be used for transfers greater than a quadword.

Setting Flip FADDR Address Bit 1 only affects FADDR address bit 1 when the DWMBA/B module logic accesses data locations in the buffer. During the cycle when the C/A is addressed in the buffer, the setting of Flip FADDR Address Bit 1 has no effect on the buffer address.

bit<5>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<4>

Name: Flip Bit 29
Mnemonic: None
Type: R/W, 0

Setting Flip Bit 29 inverts the state of bit 29 and BCI parity after the CPU C/A has been fetched and decoded by the master sequencer. The new address, which now resides in XMI memory space, is issued to the VAXBI. The DWMBA is the selected slave for the transaction, which processes this transaction like any other VAXBI-initiated DMA longword transaction, allowing diagnostic programs executing on the XMI to issue a CPU transaction to the DWMBA, which then converts it into a DMA transaction.

bit<3>

Name: BIIC Loopback Mode
Mnemonic: None
Type: R/W, 0

All requests to the master port of the BIIC become loopback requests whenever BIIC loopback mode is set, allowing the master sequencer to make loopback requests to access BIIC registers. The loopback mode prevents the BIIC from initiating VAXBI cycles to access the BIIC registers. When the BIIC is in loopback mode, it ignores the node ID portion of the address presented to it.

bit<2>

Name: Force BCI Bad Parity
Mnemonic: None
Type: R/W, 0

When Force BCI Bad Parity is set, bad parity is forced onto the BCI bus to the VAXBI during CPU C/A, CPU data cycles, and DMA read data cycles.

bits<1:0>

Name: Reserved
Mnemonic: None
Type: RO, 0

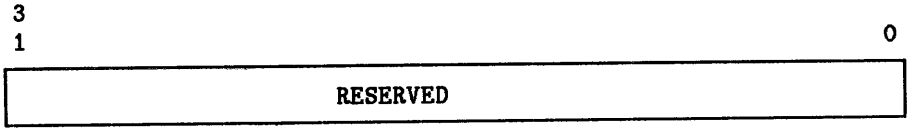
Reserved; must be zero.

Reserved Register

The Reserved Register is an undefined register that is reserved for future use. Reads to this register return UNDEFINED data with correct parity. Writes to this register appear to complete successfully.

ADDRESS

XMI nodespace base address + 0000 005C



bits<31:0>

Name: Reserved Register

Mnemonic: None

Type: Undefined

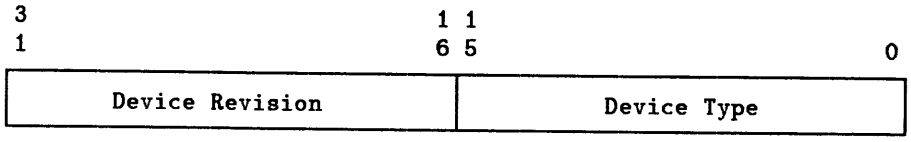
The reserved register bits are reserved for future use.

Device Register (DTYPE)

The VAXBI Device Register is loaded during self-test by console code with the DWMBA VAXBI device type and by the revision select logic with the revision level.

ADDRESS

VAXBI nodespace base address + 0000 0000



bits<31:16>

Name: Device Revision

Mnemonic: DREV

Type: R/W, 0

Identifies the revision level of the device. The revision level is loaded by hardware during BCI DC LO. For revision H, the DREV field contains 7 (hex). There is no revision I. Starting with revision J, the DREV field reflects the letter revision of the module as follows:

DWMBA/B Revision	DREV (decimal)	DREV (hex)
J0	10	000A
J1	10	000A
K0	11	000B
K1	11	000B
.		
.		
.		
Z0	26	001A

bits<15:0>

Name: Device Type

Mnemonic: DTYPE

Type: R/W, 0

Identifies the type of VAXBI node. The processor's console code loads DTYPE with 2107 (hex) after successful completion of self-test.

5.5 Interrupts

The DWMBA XMI-to-VAXBI adapter implements two mechanisms for generating interrupts to XMI CPUs. One is in response to interrupts from the VAXBI bus and one in response to errors detected on the XMI bus. The BIIC also generates error interrupts on the VAXBI in response to errors on the VAXBI.

5.5.1 DWMBA XMI-to-VAXBI Adapter Vector Formats and Requirements

Interrupt vectors returned by VAXBI nodes, as seen by the XMI IDENT transactions, fall into three categories:

- XMI bus device interrupt vectors
- UNIBUS device interrupt vectors
- VAXBI bus device interrupt vectors

Figure 5–2 XMI Bus Vector Format

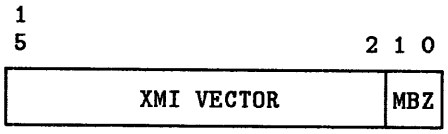


Figure 5–3 UNIBUS Vector Format

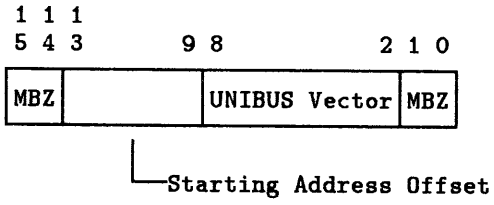
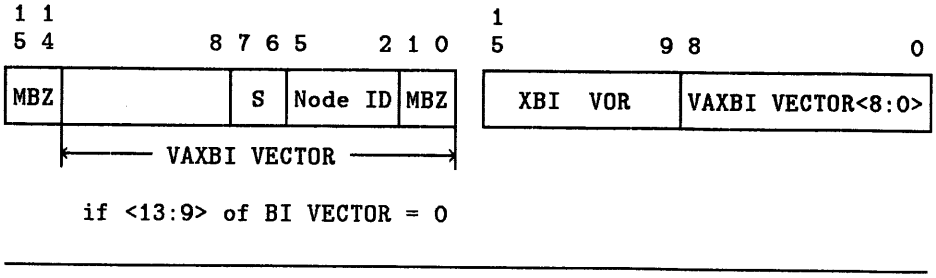


Figure 5–4 VAXBI Node Bus Vector Format



5.5.1.1 XMI Bus Vector Format

XMI device-initiated interrupts return vectors in the format shown in Figure 5–2 as a response to an XMI IDENT transaction. It is the responsibility of the operating system software to assign vector values to any vector register(s) that may exist on XMI devices that are capable of generating interrupt requests.

5.5.1.2 Offsettable Bus Vectors

There are several interrupt vectors returned by offsettable devices, including the BUA (VAXBI to UNIBUS Adapter) and the BI-LESI (VAXBI to Low-End Storage Interconnect). These other buses support devices that generate interrupts that must be differentiated from vectors generated by VAXBI devices. Figure 5–3 shows an example of the UNIBUS vector.

The UNIBUS vector field is an architecturally fixed vector returned by UNIBUS devices. Bits<8:0> cannot be modified by software. The SAO field must be a non-zero software assemble offset value to be used to index into the SCB with a unique vector.

5.5.1.3 VAXBI Node Vectors

The VAXBI node vector format has bits<15:9> as non-zero and are assigned a value by the operating system during initialization. The offset value, contained in XBI VOR (Vector Offset Register or BVOR) on the DWMBA/B module is concatenated with the vector value returned by a VAXBI node, bits<8:2>, providing that bits<13:9> of the VAXBI vector are zero. This new value is returned to the XMI commander during XMI IDENT cycles when a VAXBI node generates the interrupt request. If bits<13:9> of the VAXBI vector are non-zero, the vector will not be concatenated with the BVOR and will be passed to the XMI commander unchanged.

VAXBI device-initiated interrupts return vectors in the format shown in Figure 5–4 as a response to an XMI IDENT transaction. *Node ID* is the VAXBI node ID of the interrupt node. *S* is the interrupt vector number, which can be one of four possible interrupt vectors per node. *BVOR* must be a non-zero software assemble offset value to be used to index into the SCB with a unique vector for multiple VAXBI devices. BVOR bits<15:9> may be supplied by the DWMBA. The BVOR is necessary as the XMI is capable of supporting multiple DWMBA nodes, where the same device may exist on multiple VAXBIs. Since some VAXBI nodes might have fixed vectors that are unchangeable by software, the BVOR is used to ensure that multiple VAXBI devices with fixed vectors have a unique entry point into the SCB.

5.5.2 Interrupt Levels and Vectors

Table 5–6 lists the interrupt conditions used by the DWMBA adapter.

Table 5–6 DWMBA Adapter Interrupt Levels and Vectors

IPL (hex)	Name	Vector (hex)
17	DWMBA VAXBI Error/Status Change	XMI-7
17	VAXBI Level 7 Interrupt	VAXBI-7
16	VAXBI IPINTR 6 Interrupt	BIIC UINTRCSR REG-6[1]
16	VAXBI Level 6 Interrupt	VAXBI-6
15	VAXBI Level 5 Interrupt	VAXBI-5
14	VAXBI Level 4 Interrupt	VAXBI-4

[1]The DWMBA treats IPINTR as an error. The IPINTR value is written in the UINTRCSR as a generic VAXBI interrupt. For example, if bits<13:0> of the vector value equals zero, then the DWMBA will logically "OR" the contents of the BVOR (Vector Offset Register) with the value contained in bits<8:0> of the vector.

5.5.3 Types of Interrupts

Two types of interrupts are generated or passed through the DWMBA to the XMI bus. They are the interrupts generated by the DWMBA due to a status change or error condition and those interrupts generated on the VAXBI bus by I/O devices. The VAXBI interrupts are translated into XMI interrupt transactions.

5.5.3.1 DWMBA-Generated Interrupts

The DWMBA generates two types of interrupts: error interrupts and power-fail interrupts.

Errors detected by the DWMBA logic set bits in the DWMBA/A module and DWMBA/B module error summary registers. If the corresponding interrupt mask bit is enabled, an interrupt at level 7 (IPL 17) is requested by the DWMBA. A DWMBA error interrupt request is cleared when an XMI IDENT transaction is received at IPL 17.

The DWMBA generates an IVINTR transaction when it detects that a power failure is about to take place on the VAXBI. When BCI AC LO is asserted, the DWMBA/A module generates an IVINTR transaction with "mem write error" set in the Type field that targets the XMI node(s) specified in the Destination field of the command. During power-up and initialization, the DWMBA does not issue IVINTR transactions.

5.5.3.2 VAXBI-Generated Interrupts

Interrupts directed at the DWMBA node are passed on to the XMI bus. The BIIC handles INTR transactions directed at the DWMBA node and sets one of four interrupt level flip-flops, which store the acceptance of an INTR transaction at the given level. The INTR transaction causes the DWMBA/B module to issue an XMI interrupt command, at the corresponding IPL, to be posted on the XMI.

The BIIC generates INTR transactions on the VAXBI in response to errors detected on the VAXBI. The user has control of this mechanism via the BIIC Error Interrupt Control Register. The DWMBA's BIIC is configured to select itself as a destination node for INTR transactions, thereby informing an XMI CPU of VAXBI-related errors.

Interprocessor interrupts generated by VAXBI nodes targeting the DWMBA are supported. For the DWMBA to receive interprocessor interrupts, the software must set the DWMBA/B module's IPINTR Mask Register and enable the IPINTREN bit in the DWMBA/B module's BCI Control and Status Register.

The DWMBA handles interprocessor interrupts by asserting the BCI INT 6 signal on the DWMBA/B module's BIIC, causing the BIIC to generate an IPL 16 interrupt. The DWMBA/B module's BIIC Interrupt Destination Register configures to select itself as the destination of the interrupt transaction, thus causing this interrupt to be received by the DWMBA/B module as a generic VAXBI IPL 16 interrupt. When the DWMBA/B module receives an IDENT transaction from the XMI, it issues the IDENT onto the VAXBI. If no other interrupts are pending on the VAXBI, the DWMBA/B module's BIIC issues the vector that had been previously written by software during initialization onto the BIIC's UINTRCSR register.

The interprocessor interrupt vector value written in the UINTRCSR is treated by the DWMBA hardware as a generic VAXBI interrupt. If bits<13:9> of the vector value are zero, then the DWMBA logically ORs the contents of the BVOR with the value contained in bits<8:0> of the vector.

5.5.4 XMI IDENT to VAXBI IDENT

There are two XMI to VAXBI IDENT transactions for the DWMBA: one when the DWMBA has no interrupts pending and one when the DWMBA has an interrupt pending.

5.5.4.1 XMI to VAXBI IDENT

The DWMBA issues a VAXBI IDENT when an XMI CPU issues an XMI IDENT unless the DWMBA has a pending interrupt at the IDENTed level.

The DWMBA issues an IDENT response cycle on the XMI (Good Read Data response—function code = 1000 with the vector in bits<15:2> of the data field) upon receiving a vector from the VAXBI.

The VAXBI interrupt-pending flip-flop(s) and the INTR Sent Flip-Flop(s) that correspond to the IDENTed IPL are cleared when BCI RAK L is asserted, after the DWMBA/B module makes a VAXBI request.

If the requesting VAXBI node aborts its interrupt request before the XMI CPU generates an IDENT transaction at that level, the resulting IDENT on the VAXBI gets NOACKed. The DWMBA then issues a Read Error Response (RER) to the XMI commander and sets the IDENT Error bit in the DWMBA/B module's Error Summary Register.

5.5.4.2 XMI to VAXBI IDENT (DWMBA Interrupt Pending)

If the DWMBA has its interrupt-pending flip-flop set and it decodes an XMI IDENT transaction with IPL 17 set in D<19:16> of the IDENT command, it responds by issuing the DWMBA's vector that is located in BVOR. When the vector has been written into the DWMBA/A module's register file by the DWMBA/B module's master sequencer (state machine controller), the DWMBA's interrupt-pending and sent flip-flops clear.

If an XMI CPU issues an IDENT to the DWMBA and the DWMBA has no interrupt-pending flip-flops set, the DWMBA issues the IDENT on the VAXBI. There is a direct mapping of the XMI IDENT IPL (D<19:16> to that of the VAXBI D<19:16>). No remapping is required.

5.6 Error Reporting

The DWMBA adapter uses two mechanisms for detecting and reporting errors. One mechanism is the BIIC for VAXBI-related errors and the other mechanism deals with DWMBA-internal and XMI-related errors.

5.6.1 VAXBI Errors

The BIIC implements error checking and reporting features that deal with the VAXBI. These errors are reported to an XMI CPU via BIIC registers where bus errors are reported: the Bus Error Register, Error Interrupt Control Register, and the Interrupt Destination Register.

5.6.2 DWMBA Errors

Error generation and checking is performed on the DWMBA, both ports of the CPU, DMA-A and DMA-B register files, and the IBUS data path between the modules.

A specific error is flagged in one of the two Error Summary Registers (AESR and BESR) so that errors can be traced by software and diagnostics. When an error occurs, the DWMBA locks its error and address registers to ensure that a subsequent transaction will not change any states in the DWMBA until software services the error condition(s).

Even though an error causes the DWMBA/A module to assert IVINTR, any pending DMA or CPU transactions that are error free are processed to completion, even if a previous transaction was halted due to an error.

5.6.3 DWMBA XMI-to-VAXBI Adapter Error Response Matrix

Table 5–7 XMI Errors During DMA Transactions (VAXBI to XMI Memory)

XMI Error	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
XMI Fault	–	–	–	–
Corrected Read Data	–	DWMBA generates interrupt	–	–
Corrected Confirmation	DWMBA generates interrupt	–	DWMBA generates interrupt	DWMBA generates interrupt
Read Error Response	–	DWMBA generates interrupt (NO ACK to VAXBI)	–	–
Inconsistent Parity	–	–	–	–
Parity Error	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt
Write Data NO ACK	–	–	–	DWMBA generates interrupt
Command NO ACK	DWMBA generates interrupt	–	DWMBA generates interrupt	–
Write Sequence Error	–	–	–	–
Read Sequence Error	–	DWMBA generates interrupt (NO ACK to VAXBI)	–	–
Transaction Timeout	DWMBA/A module generates IVINTR	–	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
No Read Response	–	DWMBA generates interrupt (NO ACK to VAXBI)	–	–
Write Error Interrupt	–	–	–	–
Read/IDENT Data NO ACK	–	–	–	–

Table 5–8 XMI Errors During CPU I/O Transactions (XMI to VAXBI)

XMI Error	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
XMI Fault	–	–	–	–
Corrected Read Data	–	–	–	–
Corrected Confirmation	–	DWMBA generates interrupt	–	–
Read Error Response	–	–	–	–
Inconsistent Parity	–	–	–	–
Parity Error	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt
Write Data NO ACK	–	–	–	–
Command NO ACK	–	–	–	–
Write Sequence Error	–	–	–	DWMBA generates interrupt
Read Sequence Error	–	–	–	–
Transaction Timeout	–	–	–	–
No Read Response	–	–	–	–
Write Error Interrupt	–	–	–	–
Read/IDENT Data NO ACK	–	DWMBA generates interrupt	–	–

Table 5–9 DWMBA Errors During DMA Transactions (VAXBI to XMI Memory)

DWMBA Error	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
		DWMBA/A XMI Module		
I/O Write Failure	–	–	–	–
BCI AC LO	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
IBUS DMA-A Data Parity Error	–	–	–	DWMBA/A module generates IVINTR
IBUS DMA-A C/A Parity Error	DWMBA/A module generates interrupt (NO ACK to VAXBI)	–	DWMBA/A module generates IVINTR	–
IBUS DMA-B Data Parity Error	–	–	–	DWMBA/A module generates IVINTR
IBUS DMA-B C/A Parity Error	DWMBA/A module generates interrupt (NO ACK to VAXBI)	–	DWMBA/A module generates IVINTR	–
IBUS CPU Data Parity Error	–	–	–	–
		DWMBA/B VAXBI Module		
Multi-CPU Errors	–	–	–	–
Interlock Read Error	–	DWMBA generates interrupt Lock Time Register	–	–
IDENT Error	–	–	–	–
IBUS Data Parity Error	–	DWMBA generates interrupt. Bad Data/Parity to VAXBI.	–	–
Illegal CPU Command	–	–	–	–

Table 5–10 DWMBA Errors During CPU I/O Transactions (XMI to VAXBI)

DWMBA Error	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
		DWMBA/A XMI Module		
I/O Write Failure	–	–	DWMBA/A module generates iVINTR	DWMBA/A module generates iVINTR
BCI AC LO	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
IBUS DMA-A Data Parity Error	–	–	–	–
IBUS DMA-A C/A Parity Error	–	–	–	–
IBUS DMA-B Data Parity Error	–	–	–	–
IBUS DMA-B C/A Parity Error	–	–	–	–
iBUS CPU Data Parity Error	–	DWMBA generates interrupt. RER to XMI.	–	–
		DWMBA/B VAXBI Module		
Multi-CPU Errors	DWMBA generates interrupt. RER to XMI	–	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
Interlock Read Error	–	–	–	–
IDENT Error	–	RER to XMI	–	–
IBUS Data Parity Error	DWMBA generates interrupt. RER to XMI.	–	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
Illegal CPU Command	DWMBA generates interrupt. RER to XMI	–	DWMBA/A module generates IVINTR	–

Table 5–11 VAXBI Errors During DMA Transactions (VAXBI to XMI Memory)

VAXBI Error (DWMBA/B module's BIIC)	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
NO ACK to Multi-responses	–	–	–	
Master Xmit Error	–	–	–	–
Control Xmit Error	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt
Master Parity Error	–	–	–	–
Interlock Sequence Error	–	–	DWMBA generates interrupt	–
Transmitter During Fault	–	DWMBA generates interrupt	–	–
IDENT Vector Error	–	–	–	–
Command Parity Error	DWMBA generates interrupt	–	DWMBA generates interrupt	–
Slave Parity Error	–	–	–	DWMBA generates interrupt
Read Data Substitute	–	–	–	–
Retry Timeout	–	–	–	–
Stall Timeout	–	–	–	–
Bus Timeout	–	–	–	–
Nonexistent Address	–	–	–	–
Illegal Confirmation Error	–	DWMBA generates interrupt.	–	–
ID Parity Error	DWMBA generates interrupt.	–	DWMBA generates interrupt.	–
Corrected Read Data	–	–	–	–
Null Bus Parity Error	–	–	–	–

Table 5–12 VAXBI Errors During CPU I/O Transactions (XMI TO VAXBI)

VAXBI Error (DWMBA/B module's BIIC)	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
NO ACK to Multi-responses	DWMBA generates interrupt	–	–	–
Master Xmit Error	DWMBA generates interrupt. RER to XMI	–	DWMBA generates interrupt. DWMBA/A module generates IVINTR.	DWMBA generates interrupt. DWMBA/A module generates IVINTR.
Control Xmit Error	DWMBA generates interrupt	–	DWMBA generates interrupt	–
Master Parity Error	–	DWMBA generates interrupt. RER to XMI	–	–
Interlock Sequence Error	–	–	DWMBA generates interrupt	DWMBA generates interrupt
Transmitter During Fault	DWMBA generates interrupt	–	DWMBA generates interrupt	DWMBA generates interrupt
IDENT Vector Error	–	DWMBA generates interrupt	–	–
Command Parity Error	DWMBA generates interrupt	–	DWMBA generates interrupt	–
Slave Parity Error	–	–	–	DWMBA generates interrupt
Read Data Substitute	–	DWMBA generates interrupt. RER to XMI	–	–
Retry Timeout	DWMBA generates interrupt. RER to XMI	–	DWMBA generates interrupt. DWMBA/A module generates IVINTR.	–
Stall Timeout	–	–	–	–
Bus Timeout	DWMBA generates interrupt. RER to XMI	–	DWMBA generates interrupt. DWMBA/A module generates IVINTR.	–
Nonexistent Address	DWMBA generates interrupt. RER to XMI	–	DWMBA generates interrupt. DWMBA/A module generates IVINTR	–
Illegal Confirmation Error	DWMBA generates interrupt. RER to XMI	DWMBA generates interrupt. RER to XMI	DWMBA generates interrupt. DWMBA/A module generates IVINTR	DWMBA generates interrupt. DWMBA/A module generates IVINTR
ID Parity Error	DWMBA generates interrupt	–	DWMBA generates interrupt	–

Table 5–12 (Cont.) VAXBI Errors During CPU I/O Transactions (XMI TO VAXBI)

VAXBI Error (DWMBA/B module's BIIC)	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
Corrected Read Data	–	DWMBA generates interrupt	–	–
Null Bus Parity Error	–	–	–	–

5.7 DWMBA Initialization, Self-Test, and Booting

This section discusses the DWMBA adapter initialization and diagnostics.

5.7.1 DWMBA Initialization

The three ways to reset the DWMBA are:

- Power-Up Sequence—When the VAX 6200 is powered up, XMI AC LO L and XMI DC LO L are sequenced so that all XMI nodes are reset.
- System Reset—The XMI emulates a power-up sequence by asserting the XMI RESET L line, causing the power supply to sequence XMI AC LO L and XMI DC LO L as in a "real" power-up. Software asserts XMI RESET L by writing to IPR55. The XMI does not differentiate between a "real" power-up and a system reset. The console INITIALIZE command generates a system reset if no argument is given.
- Node Reset—A DWMBA is "node reset" by setting its XBER<30> (NRST) bit. The console INITIALIZE command generates a node reset if a node ID argument is provided. For the KA62A CPU module the differences between the node reset and a system reset are as follows:
  - XMI AC LO L is not sequenced during node reset.
  - VAXBI "self-test" is not run during node reset.

When initalized, the DWMBA performs as follows:

- All DWMBA logic resets to a known state.
- The DWMBA asserts XMI STF L until self-test completes successfully.
- The DWMBA registers are initialized to a known value by self-test.

The VAXBI subsystem of the DWMBA resets as would any VAXBI system whenever the XMI resets. Each VAXBI backplane in a VAX 6200 is connected to power, and each DWMBA/B module has logic that controls the VAXBI backplane.

Setting XBER<30> (NRST) initiates a node reset, which resets both the DWMBA/A module and DWMBA/B module as well as the corresponding VAXBI subsystem. When NRST is written to a one, the DWMBA/B module sequences the BI AC LO and BI DC LO signals, causing each VAXBI node to reset its logic.

A DWMBA/B module and its VAXBI subsystem, when powered down, has no effect on the DWMBA/A module and the XMI bus.

5.7.2 DWMBA Self-Test and Diagnostics

The two diagnostic control registers are used to force bad parity internal to the DWMBA, for performing a loopback in the BIIC, and to act as temporary storage registers for diagnostic routines.

5.7.2.1 Loopback

Two diagnostic loopbacks are implemented on the DWMBA: the BIIC loopback of the VAXBI and a transformation of a CPU transaction into a DMA transaction.

The BIIC loopback of the VAXBI occurs when the DWMBA/B module's BDCR1<3> (Force BIIC Loopback Mode) sets to a one.

When BDCR1<4> (Flip Bit 29) sets to a one, the DWMBA/B module inverts the state of address bit<29> and BCI parity when they are sent to the BIIC, allowing a VAXBI I/O space request to be converted into a DMA request that targets the DWMBA as the selected slave. This causes a CPU transaction to be transformed into a DMA transaction (longword only) that accesses XMI memory.

5.7.2.2 Self-Test

DWMBA self-test is executed by the boot processor on the XMI using the processor's resident ROM.

6 Power and Cooling Systems

The VAX 6200 power system consists of an AC power controller, the power and logic unit, five power regulators, an optional battery backup unit, and a temperature sensor. The cooling system consists of two blower units and an airflow sensor, with the airflow path through the XMI and VAXBI card cages. See Chapter 2 of the *VAX 6200 Options and Maintenance* manual for more on power components.

6.1 Power System

The power system contains the following components:

- An H405-E AC power controller for 60 Hz systems; for 50 Hz, an H405-F and a high-voltage autotransformer
- An H7206 power and logic unit (PAL)
- Two H7215 power regulators, one for the XMI card cage and one for the VAXBI card cages
- Three H7214 power regulators, two for the XMI card cage and one for the VAXBI card cages
- An XTC power sequencer
- A temperature sensor and an airflow sensor
- An optional H7231 battery backup unit (BBU)

6.1.1 Input Power

The input power is five-wire (three-phase AC, neutral, and ground). 208V 60 Hz AC enters the H405-E AC power controller. Either 380 or 416V 50 Hz AC inputs the H405-F AC power controller and then enters the high-voltage autotransformer, which reduces the voltage to 208.

The H405 AC power controllers suppress conducted emissions. The AC power controller has a contactor that closes when the control panel upper key switch is in any position except "0," allowing AC power to the H7206, and opens if the cabinet's temperature sensor detects an excessive temperature.

6.1.2 H7206 Power and Logic Unit

The H7206 PAL:

- Rectifies the three-phase power into 300V DC for the DC-to-DC power regulators
- Develops regulated +14V DC for both internal use and the DC-to-DC power regulators
- Develops 110 watts of 24V DC for the cooling system blowers and its own internal fan
- Controls the interface between power regulators
- Controls the interface between the power regulators and the rest of the VAX 6200 system

A red LED on the front face of the PAL lights to indicate that an inhibit (shutdown) latch has been set.

Two green LEDs light to indicate the presence of the +14V DC for internal use and the presence of 300V DC.

6.1.3 H7214 Power Regulator

The H7214 inputs 300V DC and +14V bias. A 30 kHz clock synchronizes this to all other power components. Outputs are 120 A of +5V DC and 0.5 A of +13.5V DC for Ethernet transceivers. A green LED lights to indicate that the +5V output is present.

6.1.4 H7215 Power Regulator

The H7215 inputs 300V DC and outputs 20 A of -5V DC, 7 A of -2V DC, 4 A of +12V DC, and 2.5 A of -12V DC. A green LED lights to indicate that the outputs are present. An internal overtemperature switch asserts the OVERTEMP signal when necessary.

6.1.5 XTC Power Sequencer

The XTC power sequencer contains:

- XMI reset timing control logic
- Time-of-year (TOY) clock power circuits
- EIA RS-232/RS-423-compatible console line driver and receiver

6.1.5.1 XMI Reset Timing Control Logic

The XMI reset timing control logic handles these sequences:

- Cold start power-up
- Warm start power-up
- Loss of AC power followed by a cold start power-up
- Reset, which mimics a power-down and then a cold start power-up

6.1.5.2 TOY Circuits

The TOY circuits consist of a battery charger circuit that trickle charges the TOY clock battery and a voltage-level detection circuit that monitors the TOY BBU battery voltage.

6.1.5.3 Console Line Driver and Receiver

The XTC power sequencer contains the system console line driver and receiver, which are EIA RS-232/RS-423 compatible.

6.1.6 Power System Signals

Power system signals are partitioned so that a failure of power supply 1 shuts down only the XMI side or a failure of power supply 2 shuts down only the VAXBI side.

The power system signals are described in Table 6–1.

Table 6–1 Power System Signals

Name	Origin	Destination	Description
ON SENSE L	Control panel	XTC	Asserts when the control panel upper key switch is in any position except "0."
PNL RESET L	Control panel	XTC	Asserts while the control panel Restart button is pressed. Causes the XTC to start the reset sequence.
STANDBY CMD L	Control panel	H7206	Asserts when the control panel upper key switch is in any position except "0."
ON CMD L	Control panel	H7206	Asserts when the control panel upper key switch is in either the Enable or Secure position. Applies DC power to entire VAX 6200.
PB REQ L	Control panel	H7206, then from H7206 to DEC power bus and AC power controller	Asserts when STANDBY CMD L asserts to close a contactor in the AC power controller, applying AC power to H7206 and DC power to cooling system and memory. Controls all peripherals tied to the DEC power bus.
DEC Power Bus	Control panel	H405	Safety Extra Low Voltage (SELV) circuit that allows the VAX 6200 to turn other equipment on and off.
DCOK H	H7206	XTC	Asserts to indicate that the DC outputs from the power regulators are OK. Used by the XTC power sequencer to start the power-up/power-down sequence.
ACOK H	H7206	XTC	Asserts to indicate that the AC input voltage is adequate. It deasserts when the H7206's 300V DC output level reaches a level that guarantees 4.2 milliseconds of acceptable 300V DC prior to the deassertion of DCOK H. Used by the XTC power sequencer during the power-up/power-down sequence.
BBU STATUS	BBU	Control panel	Controls the green Battery LED.
MODULE ENABLE L	BBU	H7206	Asserts to indicate that the BBU is supplying 300V DC to the H7206, which causes only the memory modules to receive low voltage DC power.
BATTERY BACKUP ENABLE H (BBUE H)	H7206	BBU	Asserts to indicate that the memory module's power regulators are operational.

Table 6–1 (Cont.) Power System Signals

Name	Origin	Destination	Description
BATTERY BACKUP REQUEST H (BBUR H)	H7206	BBU	Asserts when ACOK deasserts to tell the BBU to start supplying 300V DC.
CHANNEL *n* OK (CH *n* OK)	Power regulator *n*	H7206	Asserts to tell the H7206 that the power regulator specified by the number *n* is OK.
OVER TEMPERATURE *n*	H7215	H7206	Asserts to tell the H7206 that the H7215 temperature is above specification, causing an orderly system shutdown followed by a latched inhibit of the appropriate outputs.
INTERLOCK *n* INHIBIT H	Cabinet interlock switch	H7206	Asserts to tell the H7206 that an interlock switch has been thrown, causing an orderly system shutdown followed by a latched inhibit of the appropriate outputs.
BLOWER FAULT H	Cooling system	H7206	Asserts to indicate an airflow sensor has detected a loss of airflow. When asserted for more than 30 seconds, an orderly system shutdown occurs followed by a latched inhibit of the outputs.
CHANNEL *n* INHIBIT	H7206	Power regulator *n*	Asserts to command the respective power regulator to turn off and reset to a ready state so that output power restores as the signal deasserts.
SYNC	H7206	Power regulator	A pulse train used to synchronize dependent power regulators.

6.2 Cooling System

The cooling system consists of two identical blowers, one for the front of the cabinet, the other for the back. An airflow sensor signals a loss of airflow.

The H7206 PAL unit has an internal fan.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X