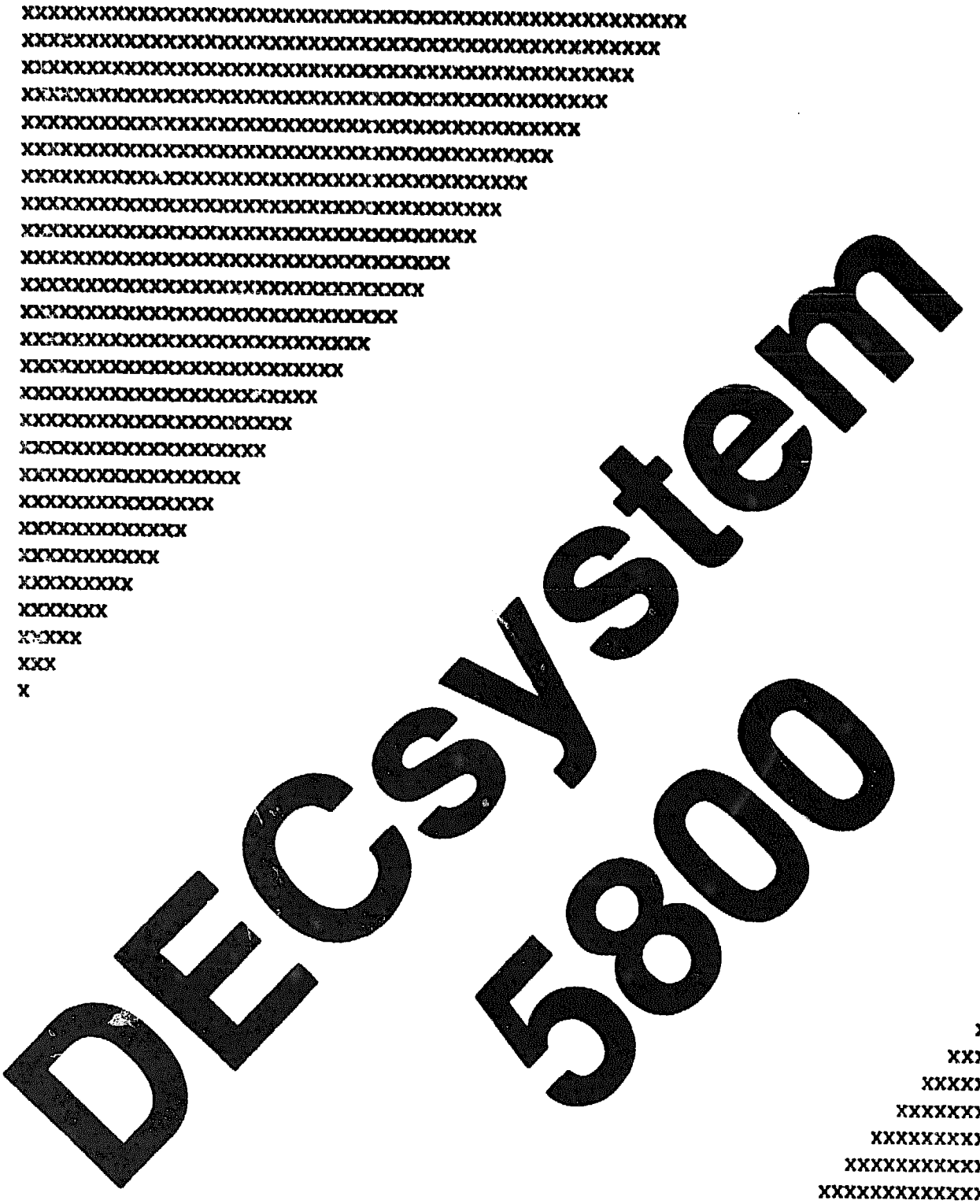

DECsystem
5800

DECsystem 5800 System Technical User's Guide

Order Number EK-580AA-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

First Printing, July 1990

This document was prepared using VAX DOCUMENT, Version 1.1

Contents

TABLES

Preface

Intended Audience

This manual is written for Digital customer 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 seven chapters.

- **Chapter 1** gives you a basic introduction to the DECsystem 5800 system and its parts.
- **Chapter 2** tells you about the XMI bus and protocol.
- **Chapter 3** explains the KN58A/A interface module, and **Chapter 4** explains the KN58A/B CPU module, the two modules that comprise a DECsystem 5800 processor.
- **Chapter 5** explains the MS62A memory module.
- **Chapter 6** tells you about the DWMBA and its DWMBA/A module and DWMBA/B module.
- **Chapter 7** explains the components of the power system and the cooling system.
- The **Index** provides additional reference support.

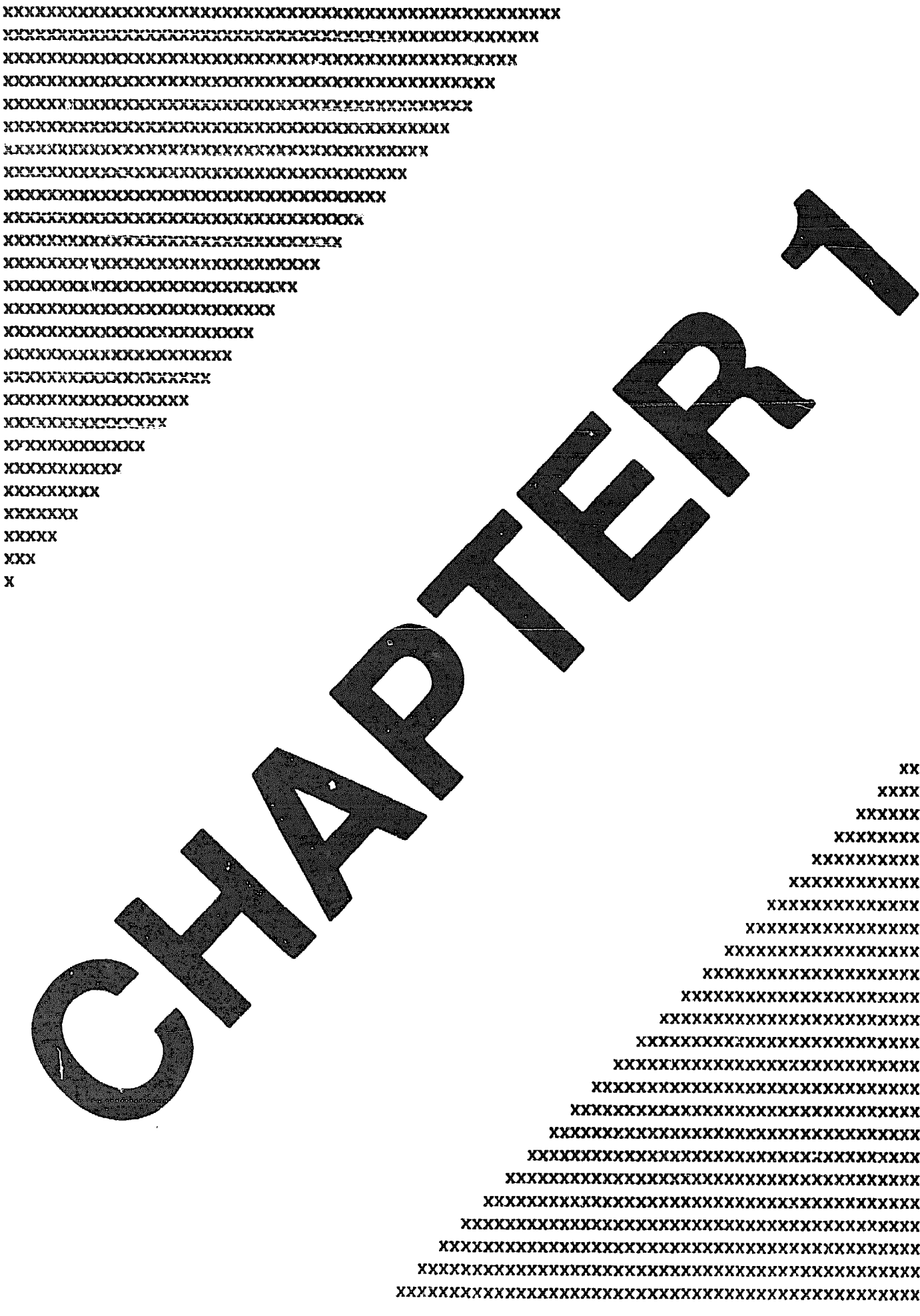
Associated Documents

Other documents in the DECsystem 5800 documentation set include:

Title	Order Number
DECsystem 5800 Installation Guide	EK-580AA-IN
DECsystem 5800 Owner's Manual	EK-580AA-OM
DECsystem 5800 Options and Maintenance	EK-580AA-MG

You may also find the following documents useful:

Title	Order Number
CIBCA User Guide	EK-CIBCA-UG
DEBNI Installation Guide	EK-DEBNI-IN
Guide to Ethernet Communication Services	AA-NL22A-TE
Guide to Languages and Programming for RISC Processors	AA-ML94A-TE
Guide to Networking for RISC Processors	AA-ML88A-TE
Guide to System Environment Setup	AA-NL18A-TE
H4000 DIGITAL Ethernet Transceiver Installation Manual	EK-H4000-IN
H9657-EU Installation Guide	EK-VBIEU-IN
HSC Installation Manual	EK-HSCMN-IN
Introduction to System and Network Management for RISC Processors	AA-ML80A-TE
KDB50 Disk Controller User's Guide	EK-KDB50-UG
RA82 Disk Drive User's Guide	EK-ORA82-UG
RA90 Disk Drive User's Guide	EK-ORA90-UG
SA70 Enclosure User Guide	EK-SA70E-UG
SC008 Star Coupler User's Guide	EK-SC008-UG
Technical Summary for RISC Processors	AA-MM35A-TE
TK70 Streaming Tape Drive Owner's Manual	EK-OTK70-OM
TU81/TA81 and TU81 PLUS Subsystem User's Guide	EK-TUA81-UG
VAXBI Options Handbook	EB-32255-46
VAX Diagnostic Architecture Reference Manual	EY-3459E-DP
VAX Diagnostic Supervisor User's Guide	AA-FK66A-TE
VAX Systems/DECsystems Systems and Options Catalog	EC-I0413-46


CHAPTER 1

1 The DECsystem 5800 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:

- DECsystem 5800 Introduction
- DECsystem 5800 Configurations
- DECsystem 5800 System Architecture
- Typical System
- DECsystem 5800 (Front View)
- DECsystem 5800 (Rear View)
- Supported VAXBI Adapters and Options
- XMI Backplane and Card Cage
- VAXBI Backplane and Card Cage
- VAXBI Expander Cabinet
- TK70 Tape Drive
- I/O Connections
- Power System
- Cooling System

1.1 DECsystem 5800 Introduction

The DECsystem 5800, a general purpose computer system based on a reduced instruction set computer (RISC) CPU, is designed for growth and can be configured for many different applications. The DECsystem 5800 can support many users in a time-sharing environment.

The DECsystem 5800 does the following:

- Supports a full set of ULTRIX-32 applications
- Functions as a stand-alone system or as the node of a network
- Allows for expansion of processors, memory, and I/O
- Implements multiprocessing where all processors have equal access to memory
- Uses the VAXBI bus as the I/O interconnect
- Uses a high-bandwidth internal system bus (XMI) 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 DECsystem 5800 Configurations

The DECsystem 5800 system family has configuration packages that differ in the number of processors and amount of memory.

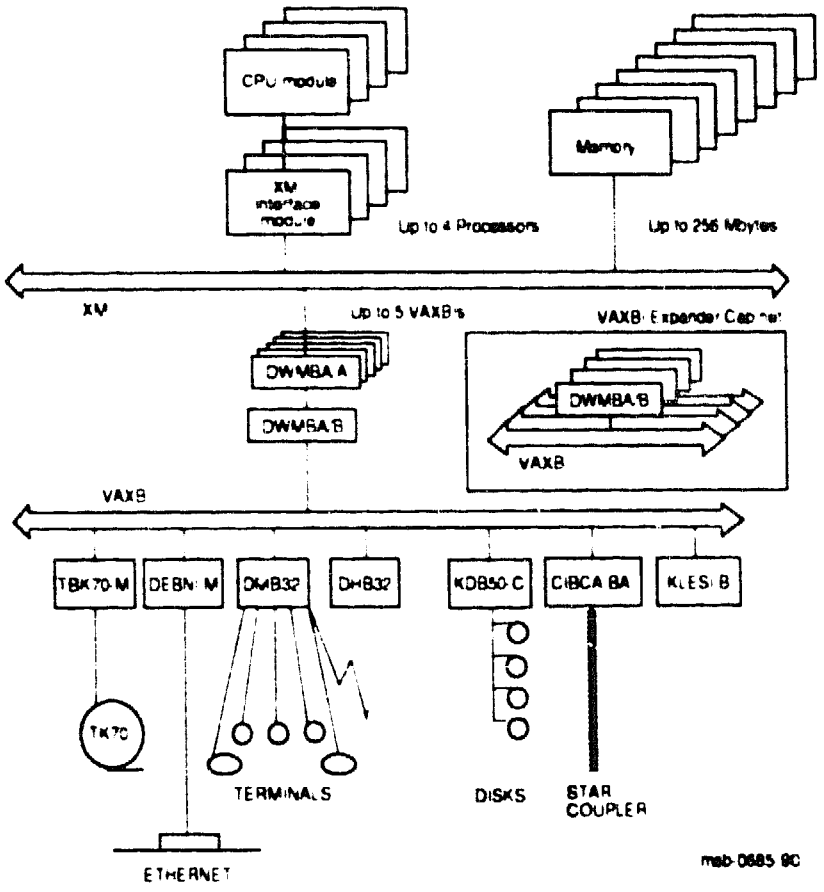
Refer to the *VAX Systems / DECsystems Systems and Options Catalog* for the available configurations.

Each configuration has a 60-inch system cabinet that includes one 14-slot high-bandwidth internal system bus backplane (XMI) and one 12-slot VAXBI backplane.

1.3 DECsystem 5800 System Architecture

The DECsystem 5800 system supports multiprocessing with up to four KN58A processors. The system uses a high-speed system bus called the XMI bus to interconnect its KN58A processors and its MS62A memory modules. All I/O devices connect to the VAXBI bus.

Figure 1–1 DECsystem 5800 System Architecture



The XMI bus is the DECsystem 5800 system bus; the VAXBI bus supports the I/O subsystem. The XMI bus is a 64-bit system bus[1] that interconnects the central processors, memory modules, and VAXBI I/O adapters.

The VAXBI and XMI buses share similar but incompatible connector and module architecture. Both the VAXBI and XMI buses use the concept of a **node**. A node is a single functional unit that consists of one or more modules.

The XMI bus has three types of nodes: processor nodes (KN58A), memory nodes (MS62A memory modules), and the XMI-to-VAXBI I/O adapters (DWMBA).

A processor node, called a KN58A, is a two-board set. It consists of a KN58A/A interface module and a KN58A/B CPU module. The KN58A/B CPU module is the processor's computational engine and contains a MIPS[2] R3000 CPU chip, an R3010 floating-point chip, R3020 [illegible]te buffers, and primary cache. The KN58A/A interface module provides [illegible]e means by which the KN58A/B CPU module accesses the XMI bus a[illegible]d contains a CVAX chip, a second-level cache, a system support chip ([illegible]SC), and XMI interface logic. The module also participates in the system self-test.

Processors communicate with main memory over the XMI bus. The system supports multiprocessing with up to four processors. One processor is designated as the boot processor according to its physical location in the card cage and that processor handles all system communication. The other processors (if any) are called secondary processors. The processor node number corresponds to the number of the XMI slot holding the KN58A/A interface module.

A **memory node** is an MS62A memory module. Memory is a global resource equally accessible by all processors on the XMI bus. Each MS62A memory module has 32 Mbytes of memory, consisting of MOS 1-Mbit dynamic RAMs, ECC logic, and control logic. The system supports up to eight MS62A memory modules (256 Mbytes of memory). The memories are interleaved by the console on power-up. The default can be changed by console command.

An **XMI-to-VAXBI adapter**, called a **DWMBA**, is a two-board adapter that transfers data between these two buses. The DWMBA/A module is installed on the XMI bus; it is cabled to the DWMBA/B module on the VAXBI bus. The system supports up to five VAXBI buses. Every VAXBI bus on this system must have a DWMBA adapter. Therefore, systems with two VAXBI channels have two DWMBA/A modules on the XMI bus, and each VAXBI channel has a DWMBA/B module. Systems with more than one VAXBI require a VAXBI expander cabinet. System error messages and self-test results refer to the pair of DWMBA modules as XBI.

[1] The XMI bus has a 64-nanosecond bus cycle, with a maximum throughput of 100 Mbytes per second.

[2] MIPS is a registered trademark of MIPSCO, Inc

1.4 Typical System

A typical DECsystem 5800 system has a main cabinet with a TK70 tape drive and optional RA disks, a console terminal and printer, an accessories kit, and a set of documentation. The system may have additional tape or disk drives.

Figure 1–2 Typical DECsystem 5800 System

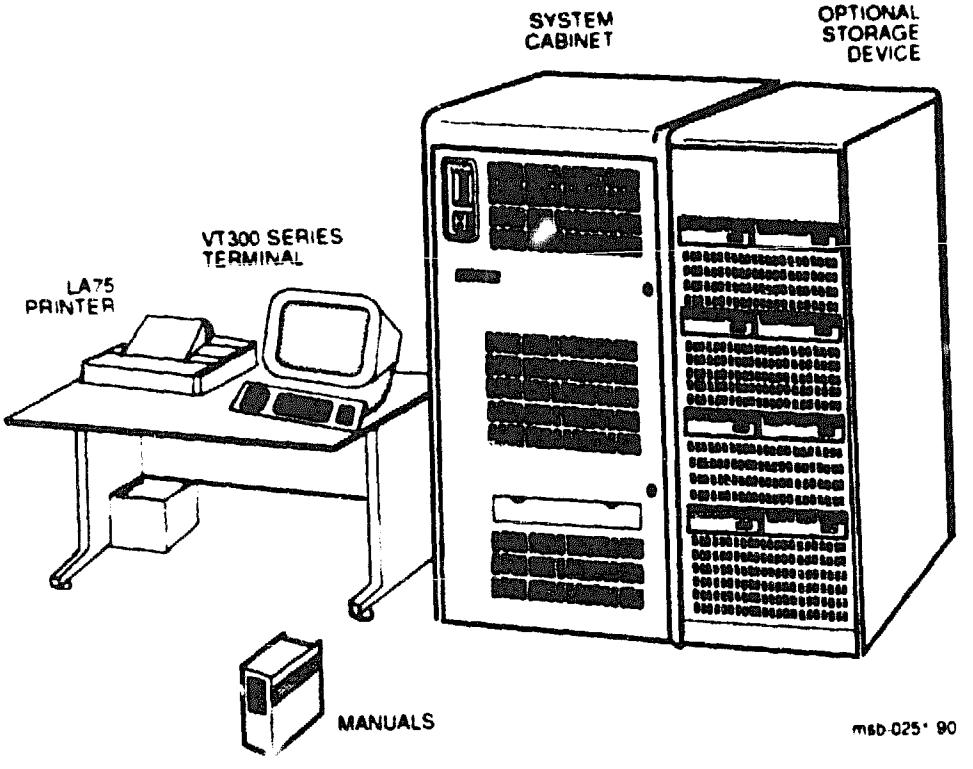


Figure 1–2 shows a typical system.

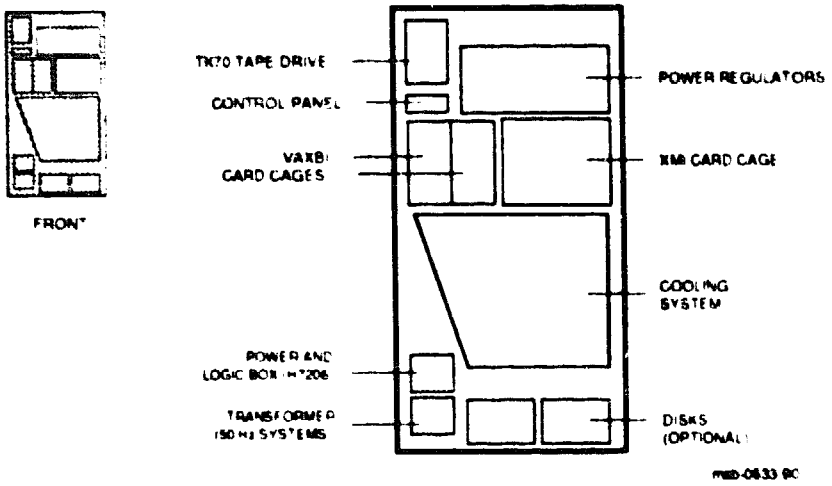
- **The main cabinet** houses a TK70 tape drive, up to eight RA disks (optional), the XMI card cage (which contains the processors and memories), VAXBI card cages, the control panel switches, status indicators, and restart controls.
- **The TK70 tape drive** in the main cabinet is used for installing operating systems, software, and some diagnostics.
- **The optional RA disk drive(s)** in the main cabinet are used for installing operating systems and software and are used for local storage and archiving.
- **The optional disk drive cabinet** provides additional local storage and archiving.
- **The console terminal** is used for console and system management operations.
- **DECsystem 5800 hardware information kit** that ships with the system includes:
  - — *DECsystem 5800 Installation Guide*
  - — *DECsystem 5800 Owner's Manual*
  - — *DECsystem 5800 Console Patch TK50 (tape)*
  - — *DECsystem 5800 Console TK50 (tape)*

  See the Preface for a complete list of system documentation and associated documents.

1.5 DECsystem 5800 (Front View)

The TK70 tape drive and control panel are on the front of the system cabinet, accessible with the doors closed. With the front door open, Digital customer service engineers can access the VAXBI and XMI card cages, the cooling system, the RA disk(s), if present, and power regulators.

Figure 1–3 DECsystem 5800 (Front View)



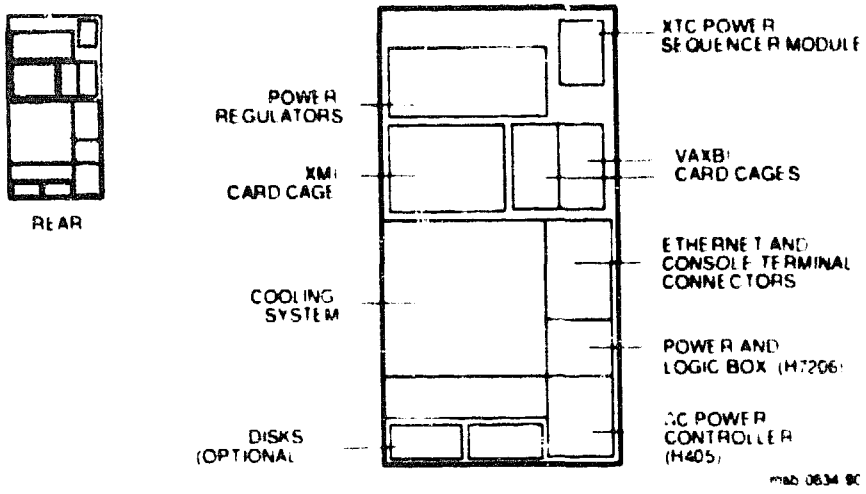
These components are visible from the inside front of the cabinet (see Figure 1–3 for their location):

- TK70 tape drive
- Control panel
- Power regulators
- Two VAXBI card cages hardwired together to form a single VAXBI channel. Only one VAXBI is permitted in the main cabinet. Additional VAXBIs require an expander cabinet.
- XMI card cage
- Cooling system (one of the two blowers is visible)
- Transformer (on 50 Hz systems only)
- Power and logic box (H7206)
- RA disks (if installed)

1.6 DECsystem 5800 (Rear View)

With the rear door open, Digital customer service engineers can access the power regulators; power sequencer module (XTC); cooling system; power and logic box; RA disk(s), if present; AC power controller; Ethernet and console terminal connectors; and the I/O bulkhead space.

Figure 1–4 DECsystem 5800 System (Rear View)



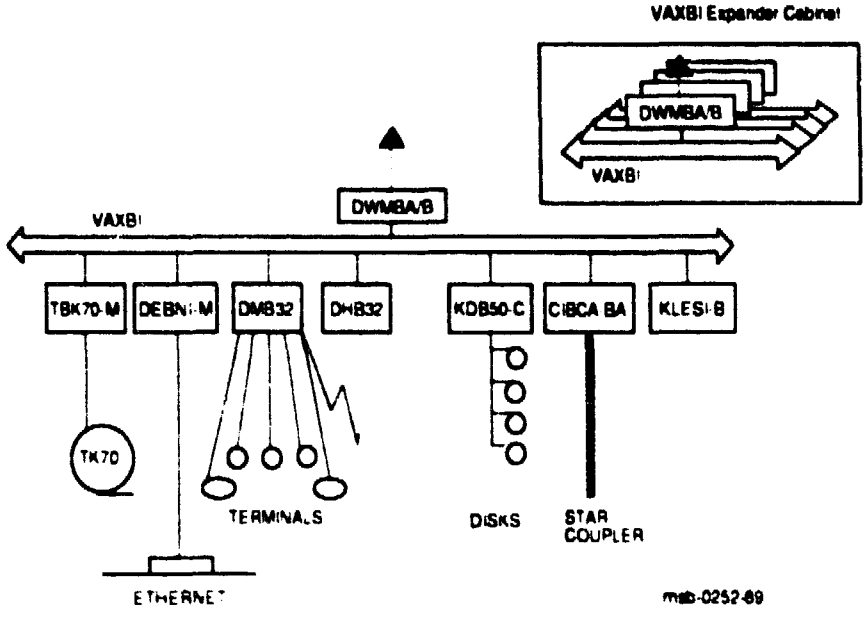
These components are visible from the rear of the cabinet (see Figure 1–4):

- Five field-replaceable power regulators
- Power sequencer module (XTC) located on the back of the TK70 tape drive and control panel unit
- I/O bulkhead space
  The panel covering the XMI and VAXBI areas is the I/O bulkhead panel and provides space for additional I/O connections.
- Cooling system, with open grid over a blower
- VAXBI and XMI adapter bulkhead cables
- Ethernet and console terminal connectors
- Power and logic box (H7206)
- RA disk(s) (if installed)
- AC power controller (H405)

1.7 Supported VAXBI Adapters and Options

The system supports the use of the following VAXBI adapters: CIBCA, DEBNI, DHB32, DMB32, KDB50, TBK70, TU81E, and DWMBA.

Figure 1–5 VAXBI Adapters

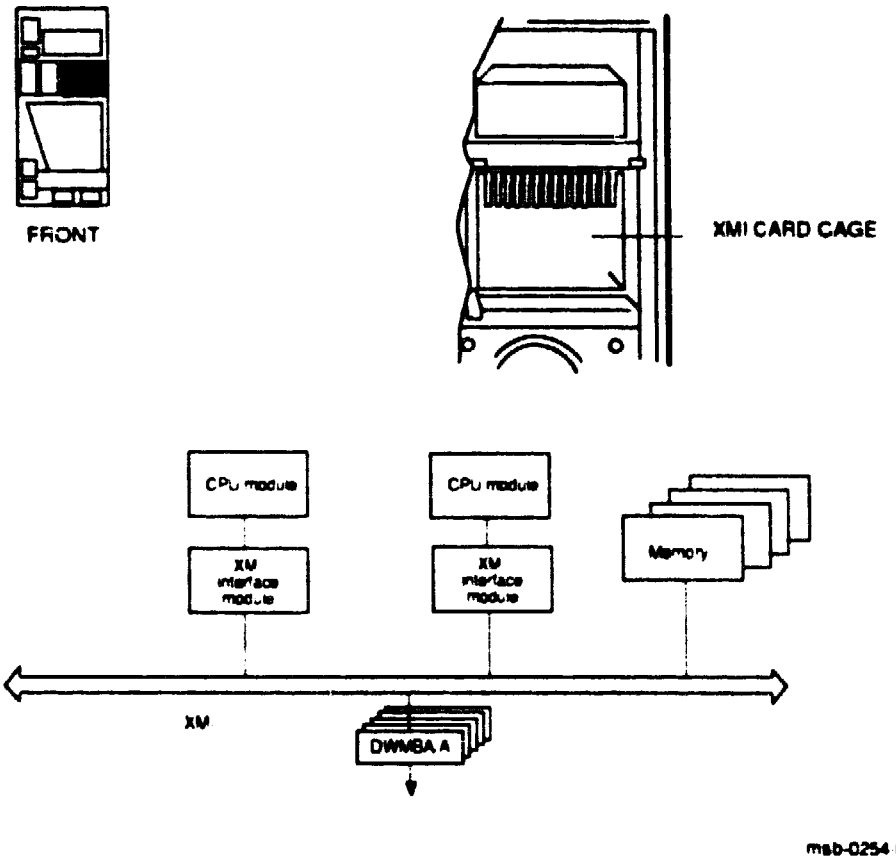


See the *VAX Systems/DECsystems Systems and Options Catalog* for a complete list of VAXBI adapters available for the DECsystem 5800 and the *VAXBI Options Handbook* for detailed information on each VAXBI adapter.

1.8 XMI Backplane and Card Cage

The XMI high-speed system bus interconnects processors and memory modules. It has a maximum bandwidth of 100 Mbytes per second and supports up to four processors. The 14-slot XMI card cage houses XMI-to-VAXBI adapters, processors, and memories.

Figure 1–6 DECsystem 5800'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 hardwired 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 *DECsystem 5800 Options and Maintenance* manual. For in-depth technical information, see the appropriate chapter of this manual.

Table 1–1 XMI Slots

Slot	Node	Permissible Modules[1]
1	1	KN58A/A, I/O, Mem
2	2	KN58A/A, KN58A/B, I/O, Mem
3	3	KN58A/A, KN58A/B, I/O, Mem
4	4	KN58A/A, KN58A/B, I/O, Mem
5	5	KN58A/B, Mem
6	6	Mem
7	7	Mem
8	8	Mem
9	9	Mem
10	A	KN58A/A, Mem
11	B	KN58A/B, I/O, Mem
12	C	KN58A/A, I/O, Mem
13	D	KN58A/B, I/O, Mem
14	E	I/O, Mem

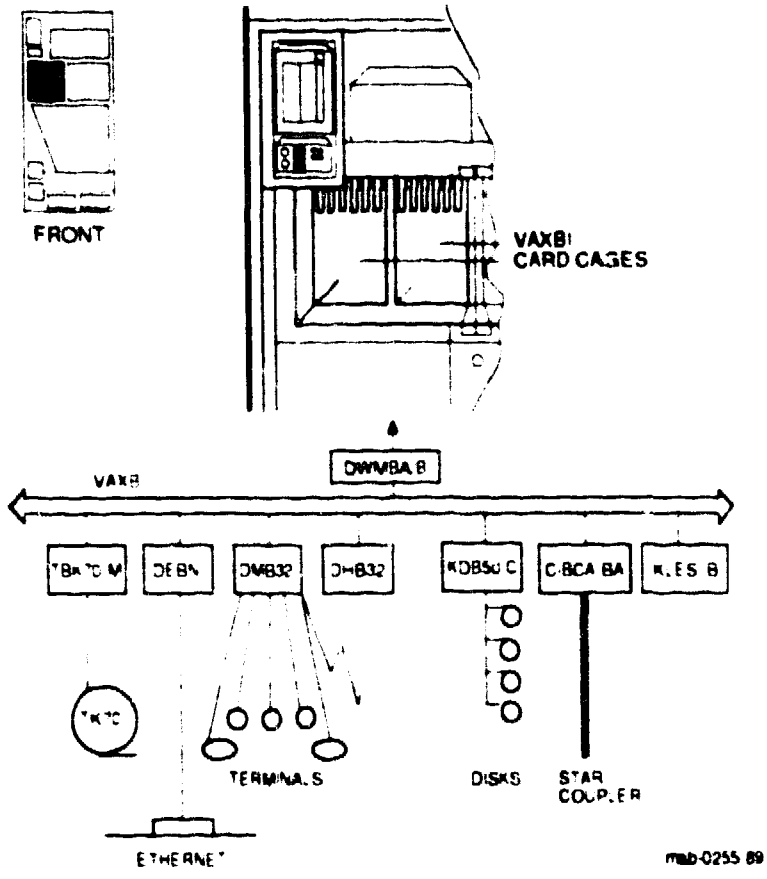
[1]Key to permissible modules:

KN58A/A = Interface module
KN58A/B = 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, multiple terminals, and other peripherals.

Figure 1–7 DECsystem 5800's VAXBI

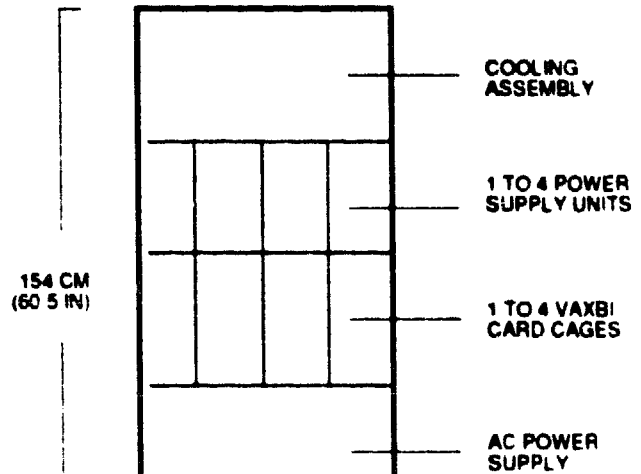


The VAXBI bus is a high-performance 32-bit bus that is the system's I/O interface. The system cabinet contains one logical VAXBI channel, comprised of two 6-slot card cages. The VAXBI card cages are located in the upper third of the cabinet on the left side, as viewed from the front of the cabinet. A clear latched door (closed for normal operation) protects the components housed in the VAXBI card cages and helps to direct the airflow over the modules. A VAXBI expander cabinet can also be added to the system as described in Section 1.10. See the *DECsystem 5800 Options and Maintenance* manual for more information on the VAXBI card cages.

1.10 VAXBI Expander Cabinet

A VAXBI expander cabinet can be ordered to increase the system's VAXBI I/O slots. One to four VAXBI cages can be added to a system.

Figure 1–8 VAXBI Expander Cabinet



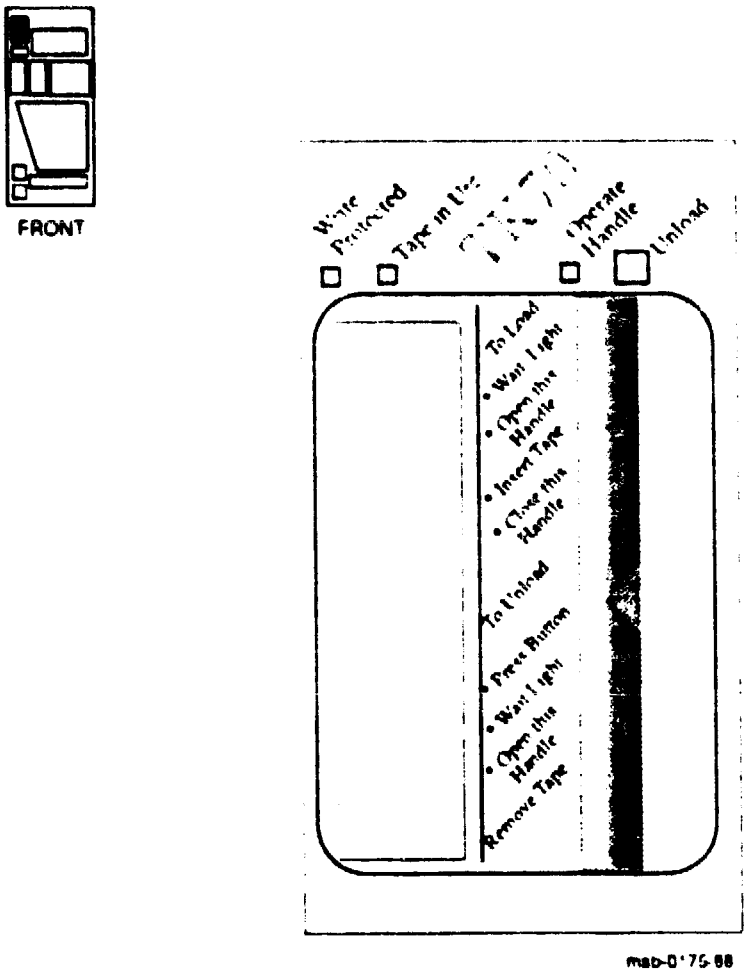
A VAXBI expander cabinet (see Figure 1–8) allows you to attach additional VAXBI channels, each with its required DWMBA/B.

The cabinet holds one to four VAXBI card cages, each with its own power supply. Two blowers cool the cabinet, and an AC power controller completes the power system.

1.11 TK70 Tape Drive

The TK70 tape drive is mounted at the front of the system cabinet in the upper left corner.

Figure 1–9 TK70 Tape Drive



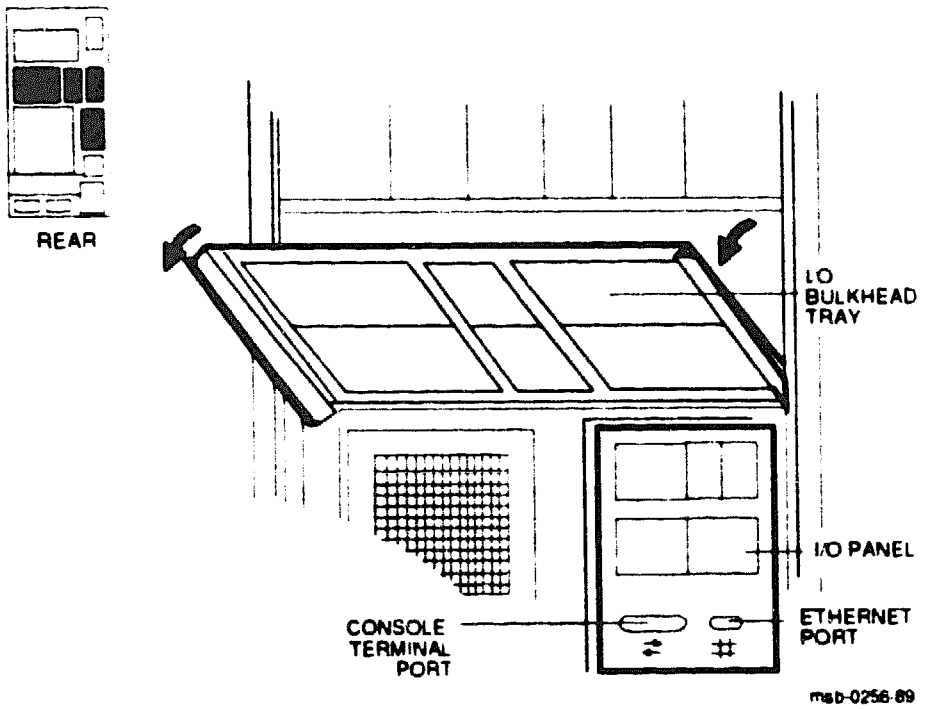
The TK70 tape drive is used for:

- Installing or updating software
- Loading diagnostics
- Interchanging user data
- Updating the contents of the EEPROM

1.12 I/O Connections

I/O connections are installed on the bulkhead connections tray and the I/O connection panel. The I/O tray is located in the rear of the cabinet, above the cooling system and below the power regulators, and covers the XMI and VAXBI backplanes. The I/O panel is just below the right-hand side of the I/O tray and houses the Ethernet and console terminal ports.

Figure 1–10 Console and Terminal Connectors



1.13 Power System

The power system consists of an AC power controller (H405E/F) with circuit breaker, the power and logic box (H7206), and five power regulators for the XMI and VAXBI backplanes.

Figure 1–11 Power System (Rear View)

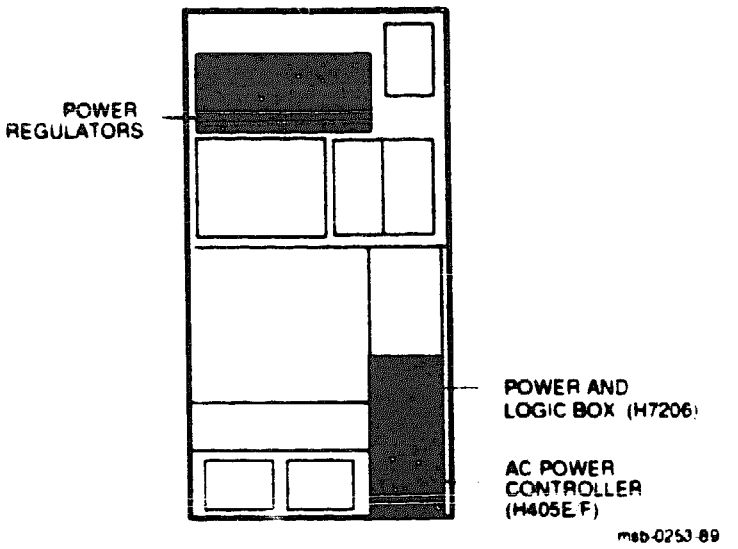


Table 1–2 Input Voltage

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

Most of the power system is visible from the rear of the cabinet. An AC power controller with circuit breaker is in the lower right corner. The power and logic box is just 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 1–3.

Table 1–3 Power Supply Available for VAXBI Options

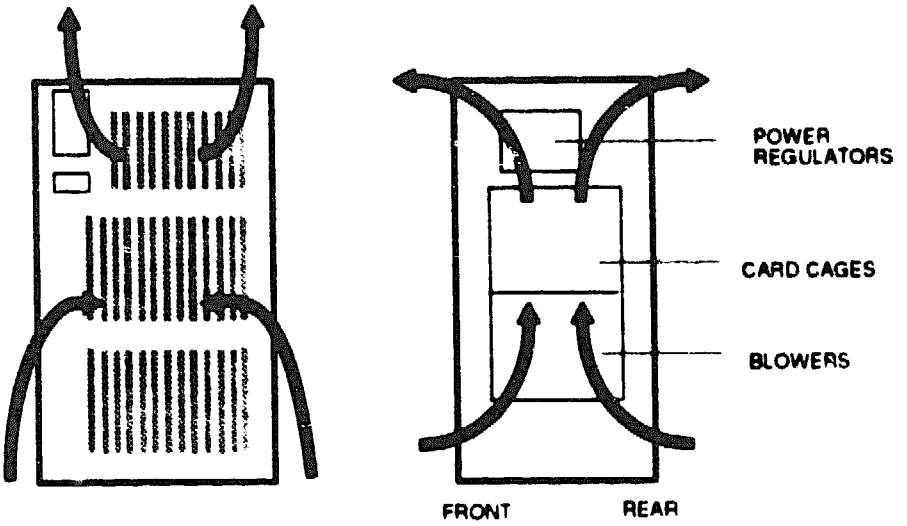
DC Voltage	Available VAXBI Current	Note
+5V	86.0 A	Main logic
+5VBB	Connected to +5V	Not battery backed up
+12V	4.0 A	RS–232
–12V	2.4 A	RS–232
–5.2V	20.0 A	ECL logic
–2V	7.0 A	ECL logic

Two power connections are on the back face of the power controller and are fuse-protected. When the system is powered down, the devices attached at these switches are also powered down. Three neon lights on the H405-E AC power controller (60 Hz systems only) indicate the presence of the 3-phase voltages at the input to the power controller.

1.14 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.

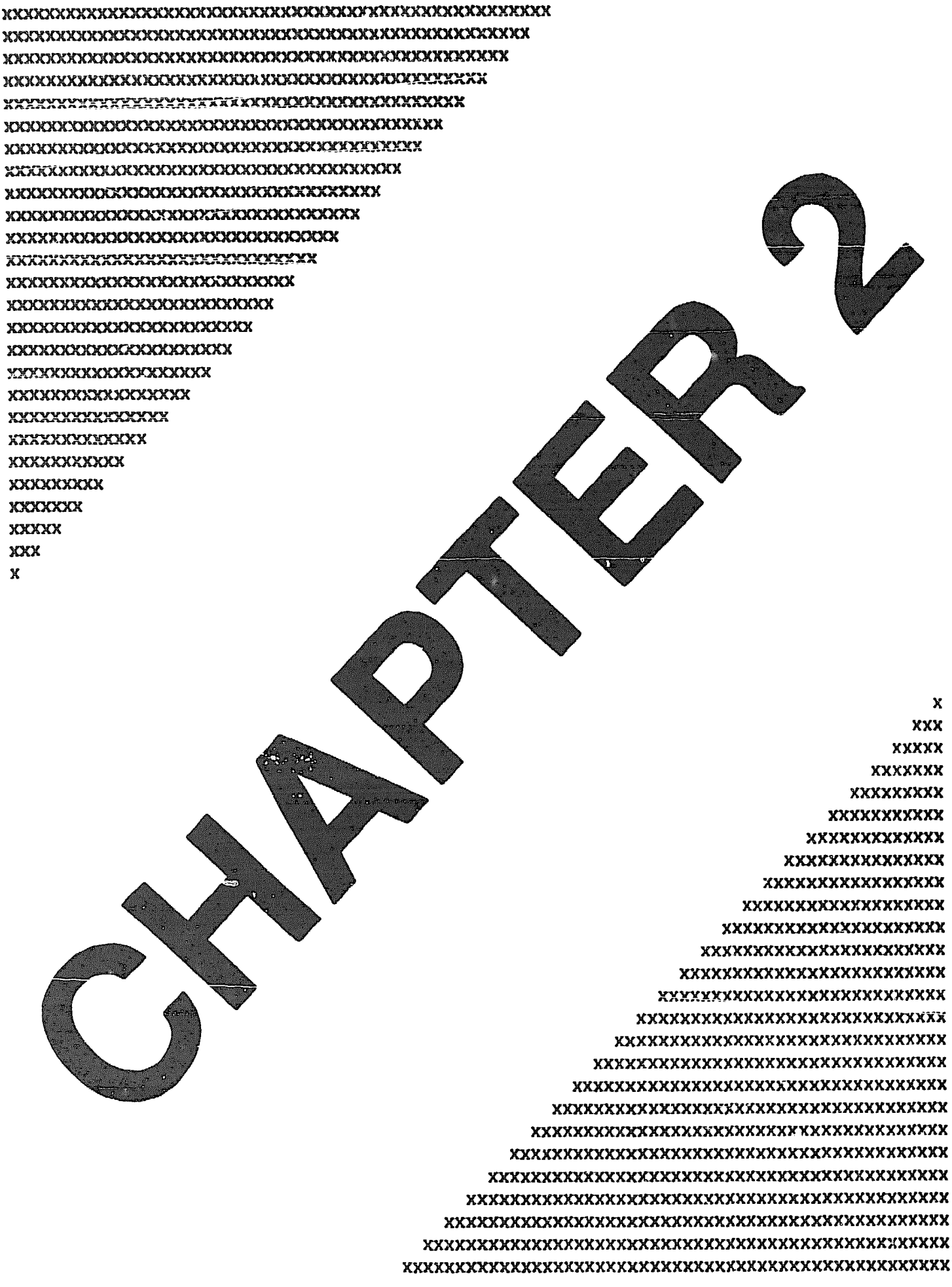
Figure 1-12 Airflow Pattern



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 39.4 inches (1 meter) from the cabinet, to maximize air intake.

The blowers, located in the lower half of the cabinet, draw air in through the doors and 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. A fan cools the power and logic box.

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.


CHAPTER 2

2 The XMI

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

This chapter includes the following sections:

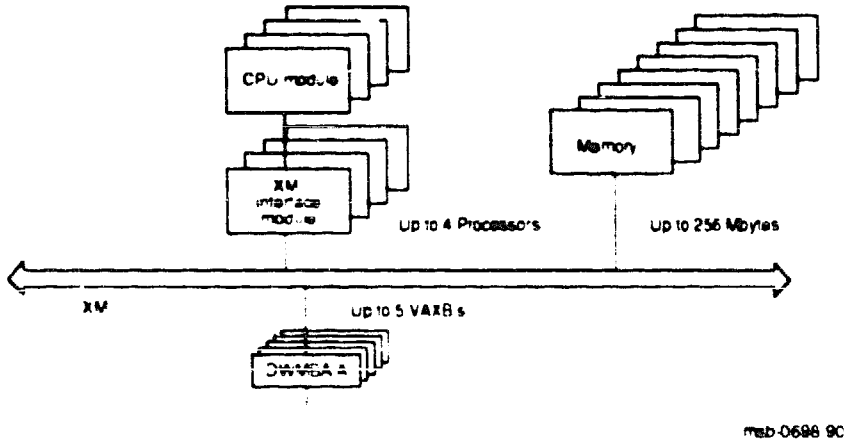
- XMI Overview
- XMI Addressing
- Arbitration Cycles
- XMI Cycles
- XMI Transactions
- XMI Initialization
- XMI Registers
- XMI Errors

2.1 XMI Overview

The XMI is the primary interconnect for the DECsystem 5800 system. The XMI supports multiple processors, multiple memory modules, and multiple I/O adapters. Figure 2–1 shows a four-processor DECsystem 5800 system.

2.1.1 XMI System Block Diagram Description

Figure 2–1 XMI System Block Diagram



msb-0698 90

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).

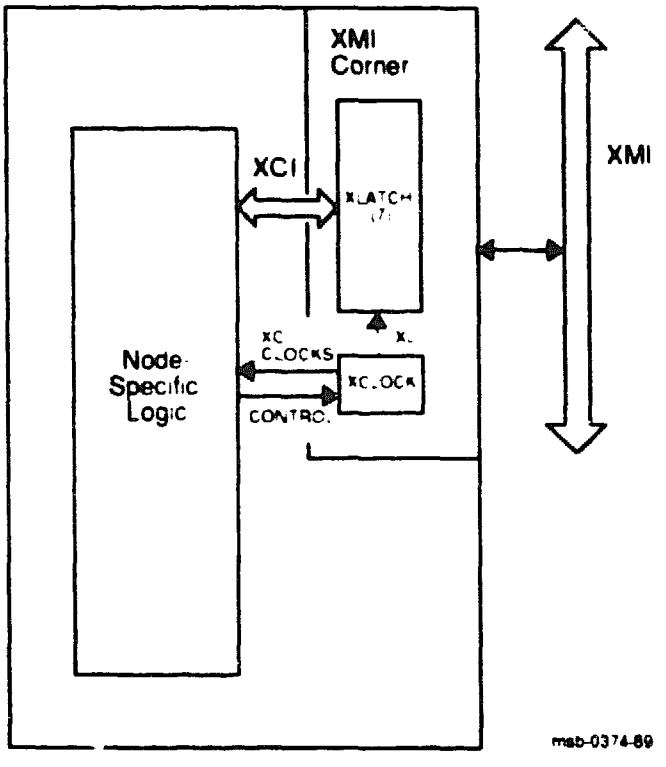
Table 2–1 Usable XMI Bandwidth

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 software 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 interprocessor interrupts (IPL = 14 (hex), vector = 80 (hex)) and write error interrupts (IPL = 1D (hex), vector = 60 (hex)). 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 DECsystem 5800 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.

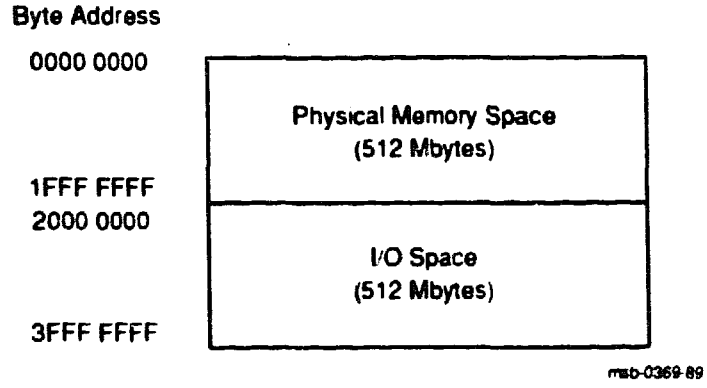
The features of the XMI that enhance bus integrity and reliability are:

- 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

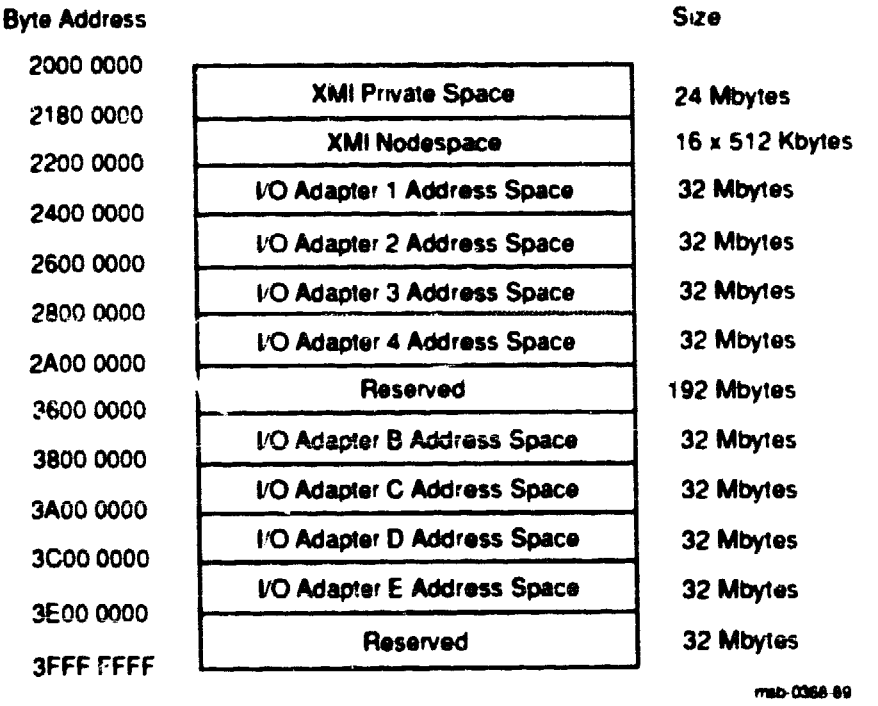
A/D<29> selects between the memory and I/O space. A/D<29> = 0 selects physical memory space, while A/D<29> = 1 selects I/O space.

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



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 DECsystem 5800 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 XMI nodes typically allocated to DWMBA adapters are as follows:

  E - the VAXBI in the system cabinet
  D - the first VAXBI in an expander cabinet; usually leftmost
  C - the second VAXBI in an expander cabinet; usually center-left
  B - the third VAXBI in an expander cabinet; usually center-right
  1 - the fourth VAXBI in an expander cabinet; usually rightmost

- Bits<16:13> correspond to the VAXBI node number. For the VAXBI inside the system cabinet, the nodes are usually numbered 1 to 12. For the VAXBIs in a VAXBI expander cabinet, consult the system-specific configuration chart.

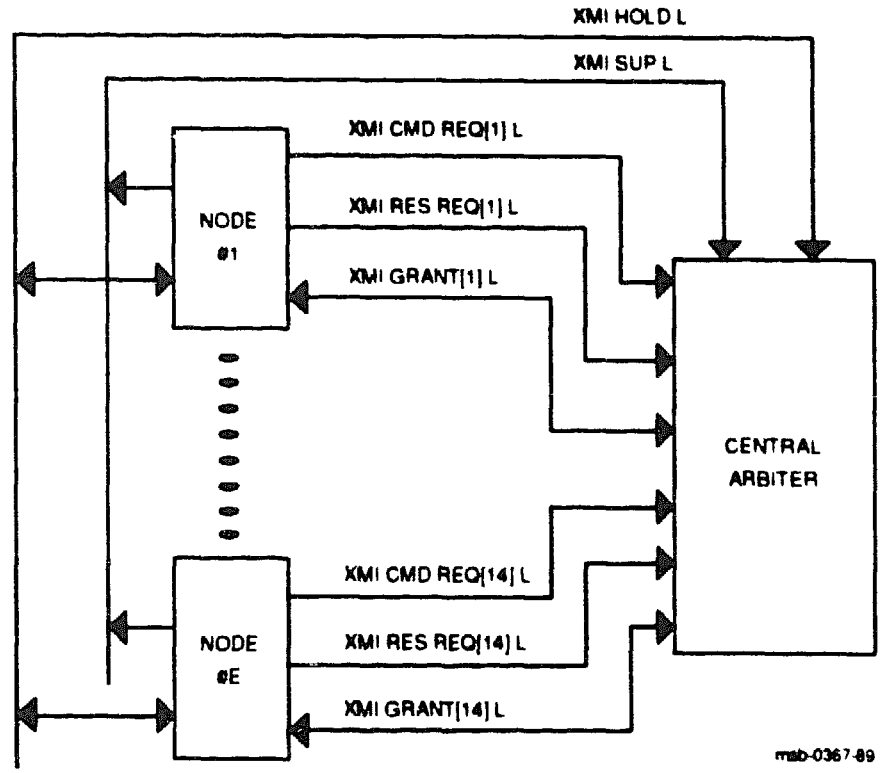
For example, the leftmost slot in the VAXBI in the system cabinet, usually VAXBI node 12 would be connected to XMI node E. The DTYPE register for the VAXBI option in that slot would be addressed as 3C01 8000.

I/O addresses associated with I/O adapters 0, 1, 2, and 3 are accessed via kseg1. I/O addresses associated with I/O adapters 4, 5, 6, and 7 are accessed via kseg2. See Section 4.2.7.2 for more information and an example.

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 DECsystem 5800 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 hard-wired 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

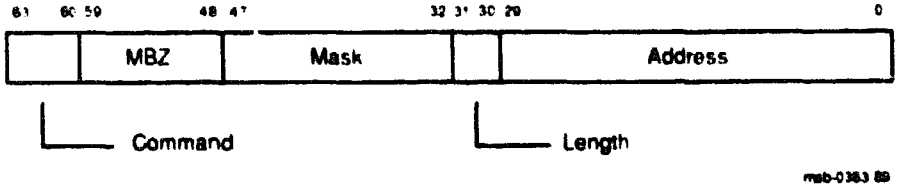
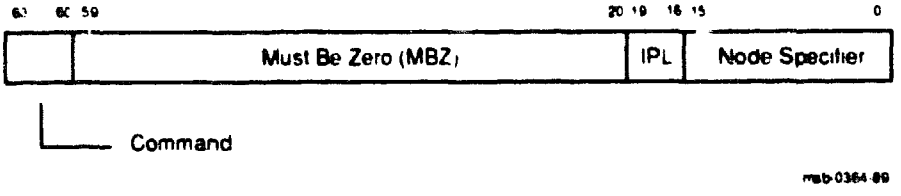


Figure 2–7 Interrupt Transaction Command Cycle Format



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

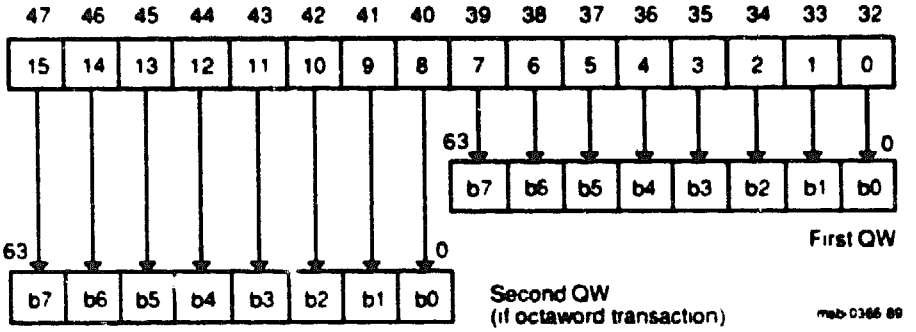
XMI D<63:60> L Logic Levels					
63	62	61	60	Command	Mnemonic
0	0	0	0	Reserved	
0	0	0	1	Read	READ
0	0	1	0	Interlock Read	IREAD
0	0	1	1	Reserved	
0	1	0	0	Reserved	
0	1	0	1	Reserved	
0	1	1	0	Unlock Write Mask	UWMASK
0	1	1	1	Write Mask	WMASK
1	0	0	0	Interrupt	INTR
1	0	0	1	Identify	IDENT
1	0	1	0	Reserved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

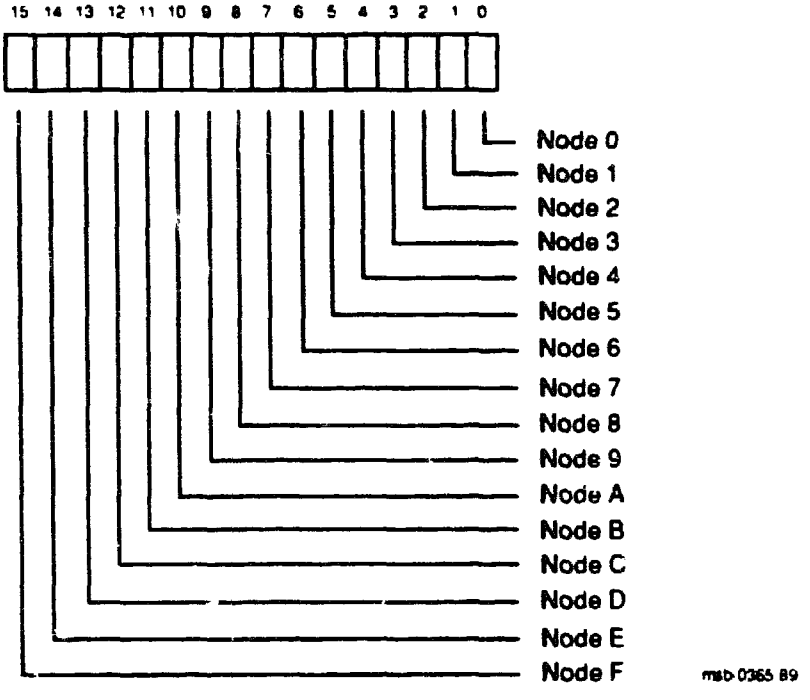
For a longword-oriented device, bit<1> is ignored as an address bit and a full longword of data is returned for a read operation.

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 DECsystem 5800 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 (IREAD) transaction, combined with a corresponding Unlock Write transaction, permits mutually exclusive access to memory space locations.

The IREAD transaction 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 (WMASK) transactions transfer data from the commander to the responder.

WMASK transactions 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 (UWMASK) 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 a UWMASK transaction 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.

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 I/O adapters on the XMI can send out an Interrupt (INTR) transaction to one or more CPU nodes, as designated by a destination mask. One of the processors eventually issues an Identify (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 software passive release.

2.5.6 Implied Vector Interrupt Transactions

The Implied Vector Interrupt (IVINTR) 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 16 (hex) with a vector of 80 (hex). Write error interrupts are issued at IPL 1D (hex) with a vector of 60 (hex). 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

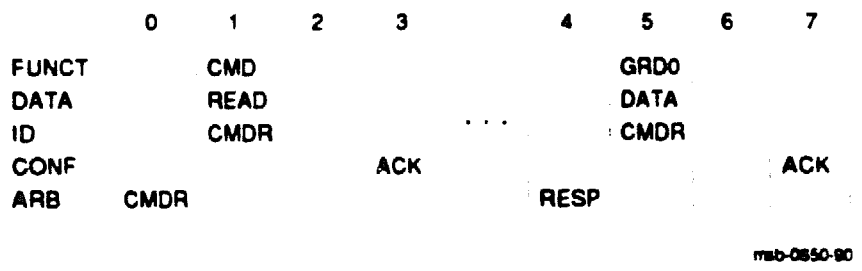
The four types of single data cycle reads are:

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

The following symbol conventions are used in the transaction figures that follow:

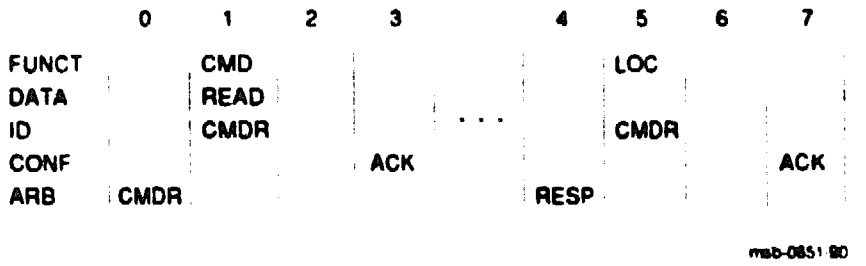
ACK = acknowledge; ARB = arbitration winner; DATN = data n; CMD = command; CMDR = commander; CRDN = corrected read data n; FUNCT = function; GRDN = good read data n; RESP = responder; WDAT = write data; WRTM = write mask.

Figure 2–10 Read Transaction



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

Figure 2–11 Interlock Read Transaction to a Locked Location



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

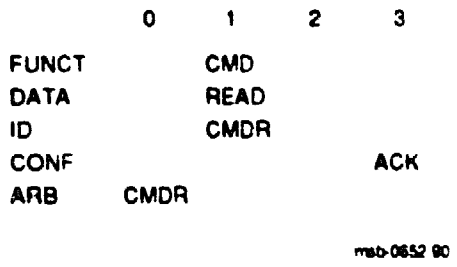
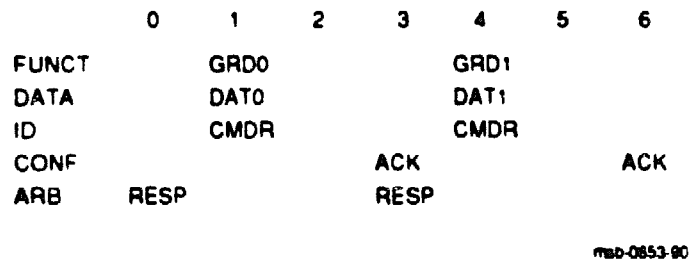


Figure 2–13 Read Data Cycles



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 2–14 Read Data Cycles with HOLD

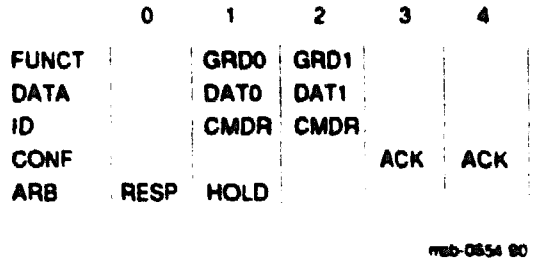


Figure 2–13 is a diagram of the return data transfer applicable to octaword reads, moving four longwords of data. The function field of the bus in cycle 1 indicates "good read data 0" with the ID field identifying the intended receiver (the transaction commander). Cycle 4 is a Good Read Data 1 cycle. 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 cycle one 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 responder 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		GRD0	GRD1	GRD2		GRD3		
DATA		DAT0	DAT1	DAT2		DAT3		
ID		CMDR	CMDR	CMDR		CMDR		
CONF				ACK	ACK	ACK		ACK
ARB	RESP	HOLD	HOLD		RESP			

mab-0555-90

Figure 2–16 Hexword Data Return with Uncorrectable Read Error

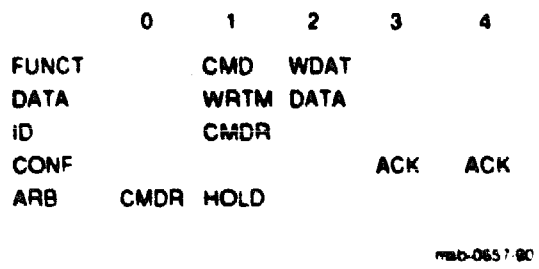
	0	1	2	3	4	5
FUNCT		GRD0	GRD1	RER		
DATA		DAT0	DAT1			
ID		CMDR	CMDR	CMDR		
CONF				ACK	ACK	ACK
ARB	RESP	HOLD	HOLD			

mab-0556-90

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

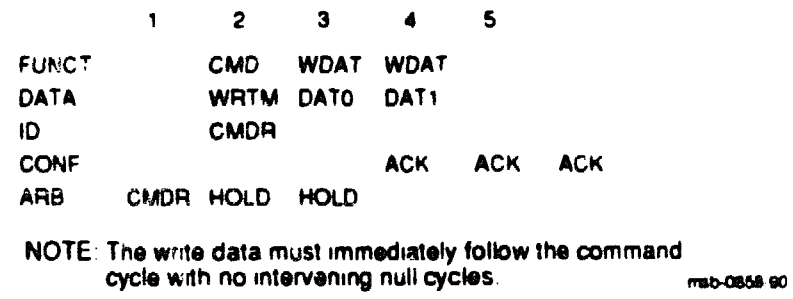


Longword and quadword writes move the number of bytes specified by the Mask field. The commander arbitrates for the XMI bus and, upon winning it, drives the appropriate write command, the intended address, the data mask, its own ID, and asserts HOLD to signal that it will need the next cycle as a Write 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

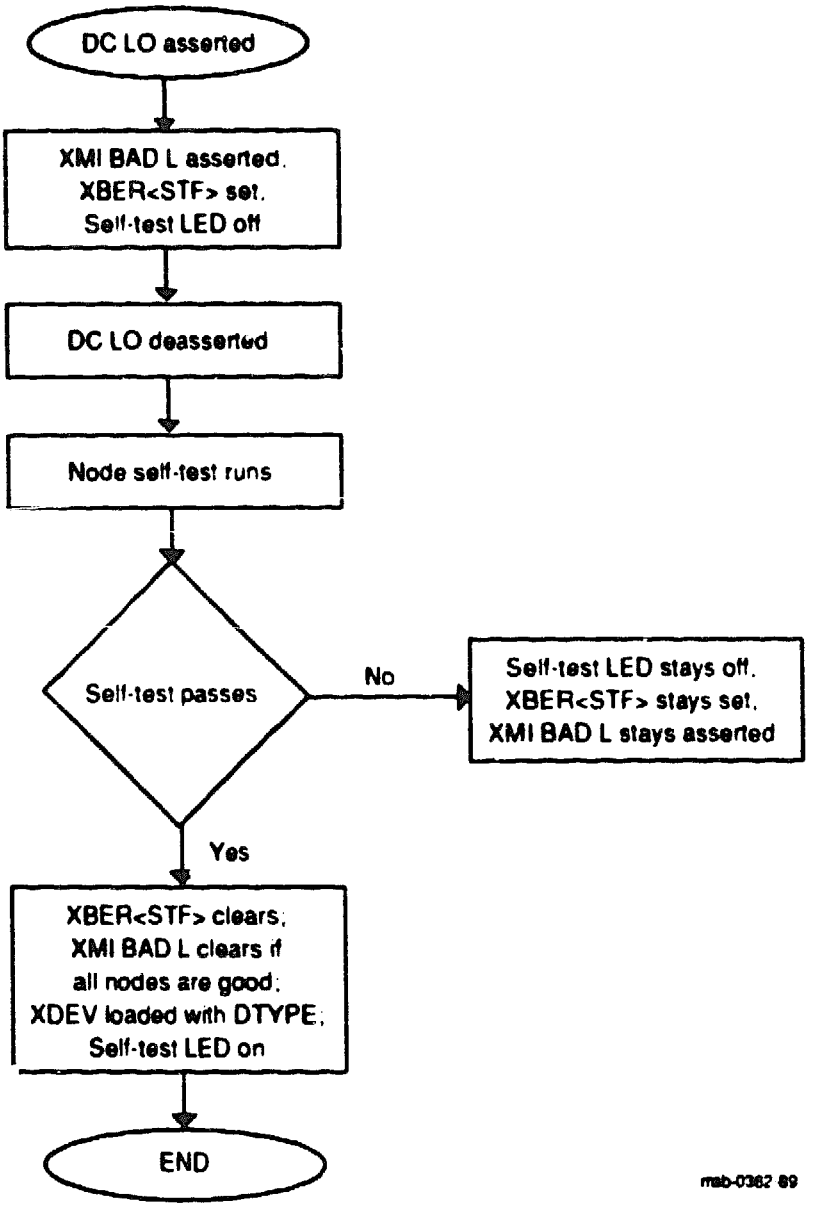


Multiple data cycle writes identify the first cycle of the transfer with the desired write length. HOLD is asserted while successive cyles provide new data.

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 (see Figure 2–19).

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.

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.

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 summarizes 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.

Descriptions of the XMI registers for the CPU are given in Chapter 3, and other module-specific XMI register descriptions 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.

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.

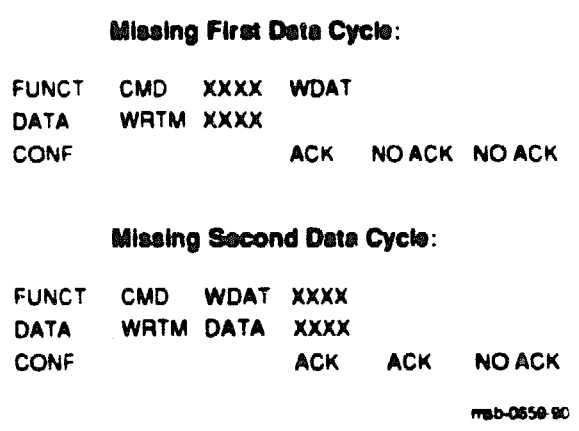
2.8.1.4 Sequence Error

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 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

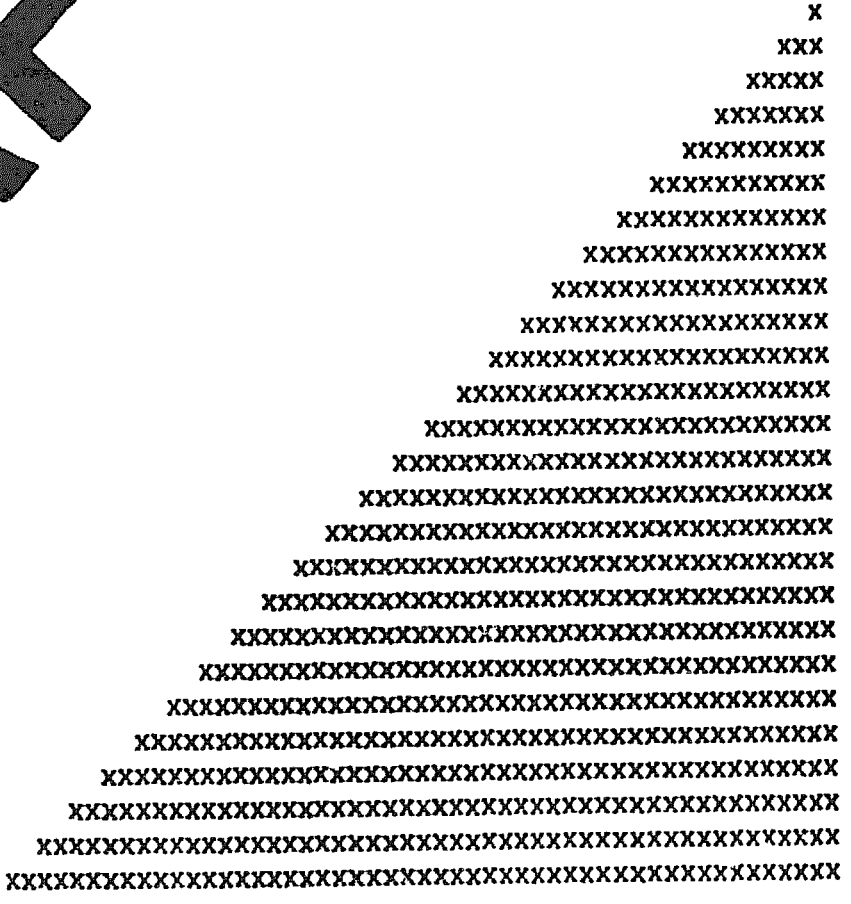
The XMI bus protocol supports two mechanisms that signal error conditions to processors if normal transaction-level error reporting cannot be used.

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 INT 3 to R3000 (XCPGA asserts MEMERR_L) 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 INT 3 to R3000 (XCPGA asserts MEMERR_L) 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.

CHAPTER 3



3 KN58A/A Interface Module

This chapter describes the KN58A/A interface module, the module that provides the interface between the KN58A/B CPU module and the XMI bus. It describes the operation of the major components of the KN58A/A interface module such as the CVAX, the second-level cache, and the XMI interface. Topics on overall KN58A/A interface module operation such as initialization and error handling are also discussed.

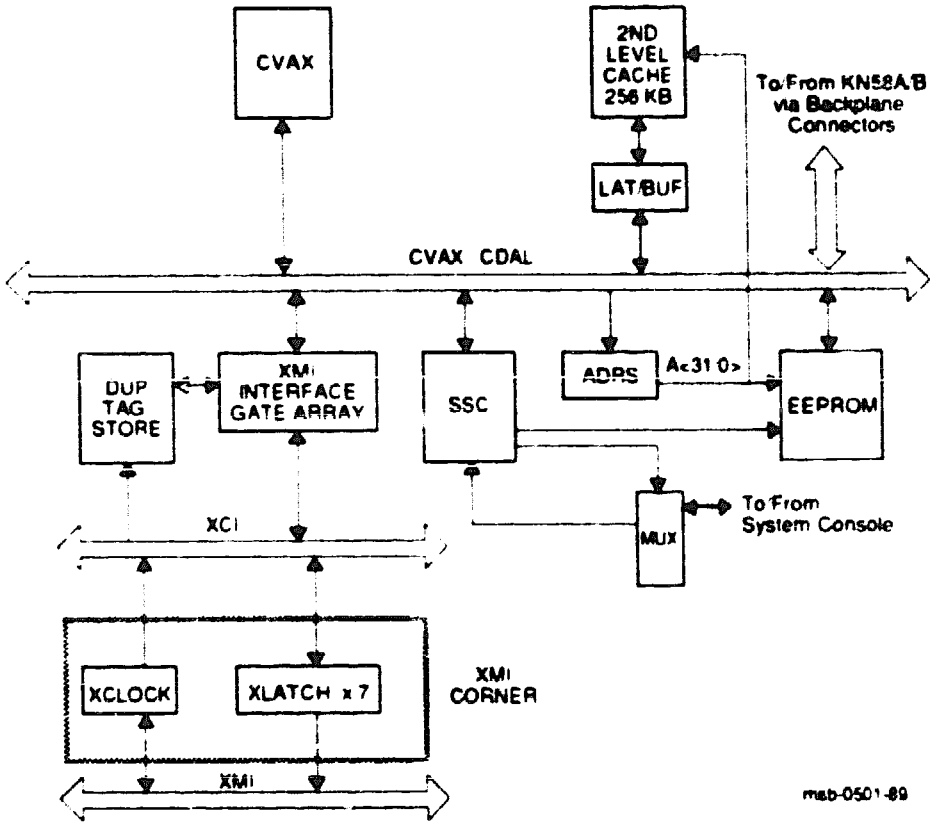
This chapter includes the following sections:

- Module Features
- Private I/O Address Space Map
- Maintenance Processor
- System Support Chip (SSC)
- EEPROM
- Second-Level Cache
- XMI Corner-to-KN58A/A Interface
- Module Registers
- Initialization, Self-Test, and Booting
- Interprocessor Communication through the Console Program
- Error Handling

3.1 KN58A/A Interface Module Features

The KN58A/A interface module has four functional sections (see Figure 3–1): the maintenance processor section, the second-level cache, the XMI interface, and support logic for the KN58A/B CPU module.

Figure 3–1 KN58A/A Interface Module Block Diagram



The maintenance processor section includes:

- A CVAX processor chip that handles the maintenance and diagnostic functions of the KN58A/A interface module. It is disabled whenever the KN58A/B CPU module is running.
- A clock chip that synchronizes the RDY, ERR, and RESET signals for the KN58A/A interface module.

The second-level cache is for instruction and data storage for the KN58A/B CPU module. The second-level cache is 256 Kbytes, organized with 4096 tags. The cache is write-through and direct-mapped. If a processor read misses an entry in the cache, or if the entry is invalid, the XMI interface 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 XMI interface includes:

- Two octaword write buffers that decrease bus and memory controller bandwidth needs by packing writes into larger, more efficient blocks prior to sending them to main memory.
- Cache fill logic that loads the second-level cache with one hexword of data for each memory read that misses cache.
- XMI write monitoring logic that uses a duplicate tag store to detect when another XMI node writes a memory location that is cached on this processor. Then the XMI interface gate array invalidates the corresponding entry in the cache.
- Full set of error recovery and logging capabilities

Support logic for the KN58A/B CPU module includes:

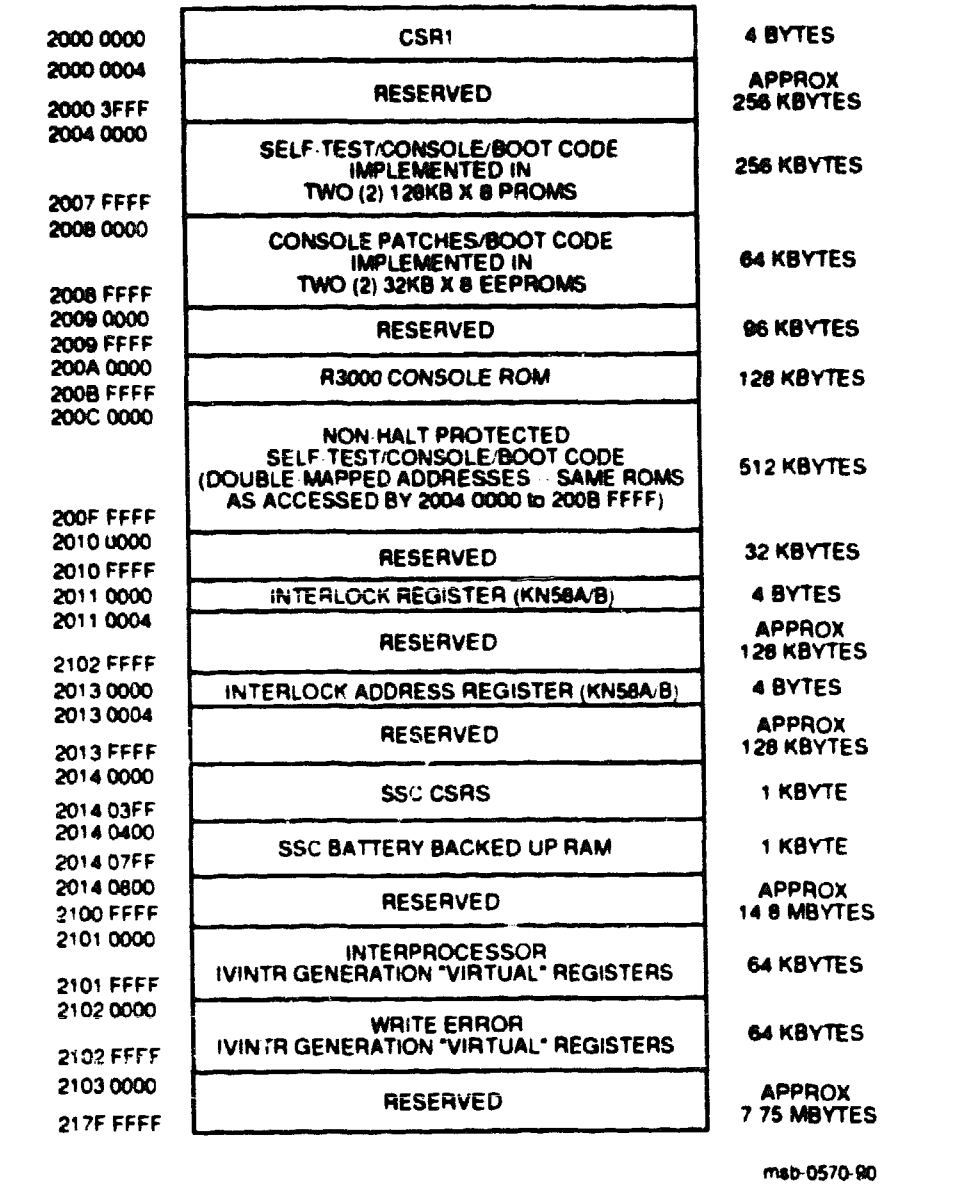
- A maintenance read-only memory (ROM) that contains the code for initialization, executing maintenance mode commands, and bootstrapping the system.
- An electrically-erasable, programmable ROM (EEPROM) that contains system parameters and console patches. You can modify the parameters with the **set nv** console command. Patching console and diagnostic code in the ROMs is accomplished by reading the patches into a special area of the EEPROM.
- A system support chip (SSC) that includes support for external ROM, EEPROM, console terminal UARTs, bus reset logic, interval timer, programmable timers, time-of-year (TOY) clock, bus timeout, and halt arbitration logic.

3.2 Private I/O Address Space Map

Figure 3–2 shows the private I/O address space map for the KN58A/A interface module and KN58A/B CPU module. Addresses shown are IIDAL physical addresses.

To convert IIDAL physical addresses to R3000 virtual addresses, substitute the leading "2" with a "b". For example, address 2000 0000 in IIDAL physical space becomes b000 0000 in R3000 virtual address space.

Figure 3–2 Private I/O Address Space Map



3.3 Maintenance Processor

The maintenance processor section consists of a CVAX chip and a clock chip. The CVAX supports a subset of the VAX instruction set and data types. It also supports VAX memory management.

3.3.1 CVAX Hardware Restart Sequence

The CVAX 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 CVAX cannot continue normal processing.
- When a HALT instruction is executed in kernel mode.

When the hardware restart process begins, the CVAX 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 CVAX. Section 3.9 contains more detailed information on initialization.

3.3.2 Clock Chip

The clock chip handles the synchronization of RDY, ERR, and RESET for the KN58A/A interface module.

3.4 System Support Chip (SSC)

The system support chip (SSC) provides functions to support the DECsystem 5800 system environment.

3.4.1 SSC Functions

The SSC provides the following:

- 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 for use as a console "scratchpad."
- 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 KN58A/A interface 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 an IIDAL bus timeout register.

3.5 EEPROM

The EEPROM stores parameters for initialization of the KN58A/A interface module and patches to the ROM code, which does console emulation, module self-tests, and boot code.

3.5.1 EEPROM Access

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. All the ROM and EEPROM can be accessed by the R3000, but the CVAX cannot access the one 128K x 8 ROM on the KN58A/B CPU module.

Console (initialization) code sets the ROM Size field in the SSC Configuration Register (SSCR<22:20>) to the 1-Mbyte block 2004 0000 to 201? 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).

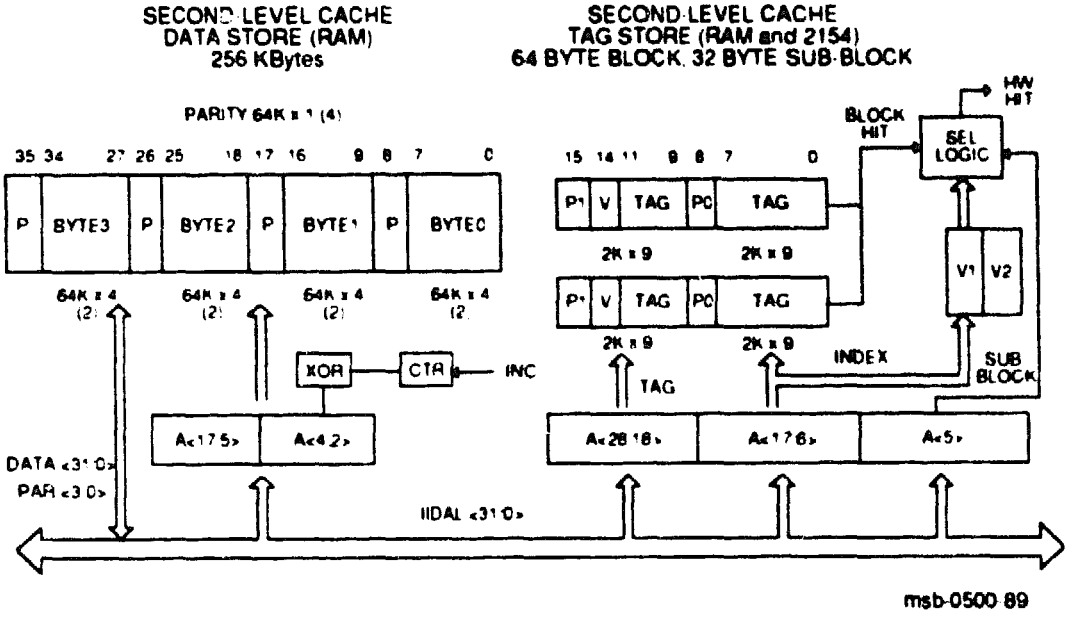
Writes to the remainder of the EEPROM address space must follow these rules:

- Write only a byte of data at a time. The write data must be driven on CDAL<7:0>.
- The two low 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.

3.6 Second-Level Cache

The second-level cache is a 256-Kbyte, direct-mapped, write-through cache with a 160-ns cycle time. All memory space read references made by the R3000 chip except Interlock Reads are stored in the second-level cache.

Figure 3–3 Second-Level Cache Block Diagram



3.6.1 Second-Level Cache Description

The 256-Kbyte, direct-mapped, second-level cache supplements the 128-Kbyte first-level cache on the KN58A/B CPU module. The second-level cache is located on the IIDAL bus and is partitioned into 64-byte blocks that are subdivided into two 32-byte (hexword) sub-blocks. 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 IIDAL bus invalidate traffic.

The second-level cache memory array is a direct mapped 64K x 36-bit array that 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 that 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 is no "allocate on write."

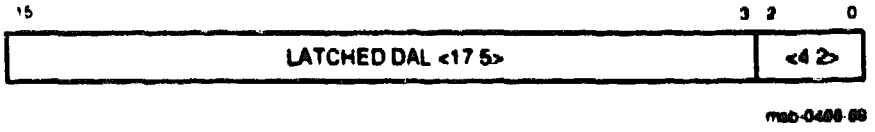
Each of the 4096 tag entries maps two hexwords in the cache. There is one valid bit for the entire 64-byte block and one valid bit for each 32-byte sub-block.

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 DAL lines as shown in Figure 3–4. During cache fill operations, they are driven by latched DAL lines as shown in Figure 3–5.

Figure 3–4 Cache Address Line Contents During a Cache Read

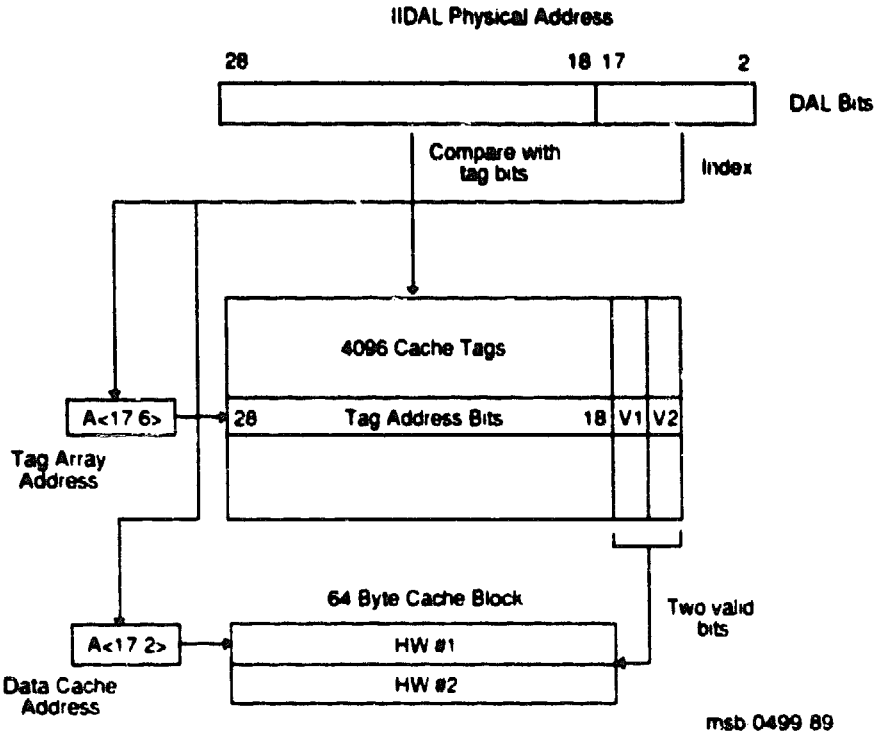
Figure 3–5 Cache Address Line Contents During a Second-Level Cache Fill



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. They are wrapped within a quadword, and the quadwords wrapped within an octaword to fill the hexword sub-block. 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 3–6 shows how the lower bits of the DAL physical address are used to access the cache tag addresses that are compared with the physical address on the DAL. The DAL address bits are also used to drive the data store address lines for addressing the data cache RAMs.

Figure 3–6 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 IIDAL. A comparator generates a hit signal if the data is both resident and valid within the cache data RAMs.

An R3000 read that results in a second-level cache miss will cause the IIDAL/XMI interface to begin a hexword read transaction to update the cache. The first quadword fetched contains the longword requested by the CPU; the remaining seven longwords 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.

3.6.2 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 Read and store the contents of CSR1 in a temporary memory location (TEMP).

2 Perform back-to-back store operations to CSR1 so that the first longword write writes back TEMP to CSR1 with FMISS (CSR1<18>) and FCI (CSR1<20>) set and the second longword write writes back TEMP with FMISS and FCI cleared.

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>).

Example 3–1 Flushing Second-Level Cache

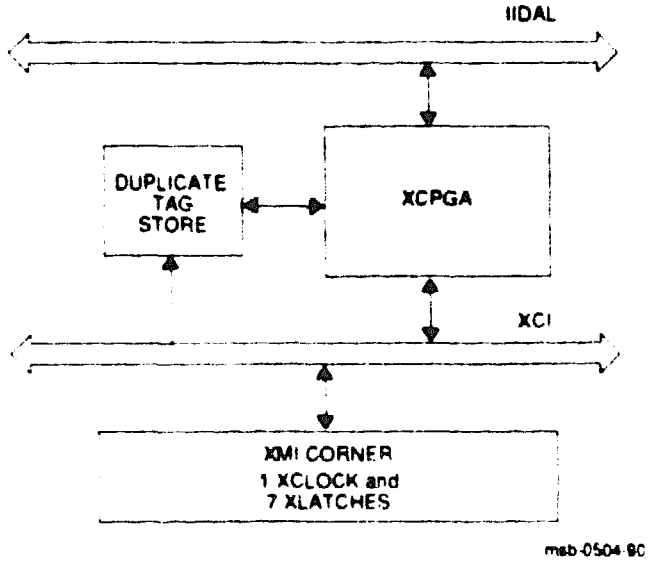
```
#include <regdef.h>
/*
/* define CSR1 address and MASK with CSR1<FCI> and CSR1<FMISS> set
/*
#define CSR1 0xb0000000
#define MASK 0x00140000

flush_SCache:
        .set noreorder
        lw      s0, (CSR1)      # read CSR1
        li      s1, MASK        # load FCI-FMISS mask
        or      s0, s1          # set FCI and FMISS bits
        not     s1, s1
        and     s1, s0          # clear FCI and FMISS bits
        sw      s0, (CSR1)      # flush second-level cache
        lw      zero, (CSR1)    # purges previous 'sw' from R3020 write buffer
        sw      s1, (CSR1)      # reenable second-level cache
        j       ra
        nop
        .set reorder
```

3.7 XMI Corner-to-KN58A/A Interface

The KN58A/A interface module's XMI Corner is a predefined interface to the XMI bus. Refer to Chapter 2 for a discussion of the XMI.

Figure 3–7 XMI Corner-to-KN58A/A Interface



The KN58A/A interface 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 write masks
- Longword I/O unlock write masks
- Write error IVINTRs
- Interprocessor IVINTRs
- IDENTs (in response to R3000 interrupt acknowlege)

The KN58A/A interface module responds to the following XMI transactions:

- Longword nodespace reads
- Longword nodespace write masks
- Interrupts

In addition, the KN58A/A interface module monitors all memory writes for cache invalidates.

The duplicate tag store logic and the XCPGA chip provide the functionality required to interface the R3000 CPU to the XMI Corner.

Table 3–1 Mapping of CPU Operations to XMI Transactions

CPU Operation	Resulting XMI Transaction
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 XCPGA write buffer. If this is an XCPGA write buffer miss, then the "old" XCPGA write buffer data is flushed to main memory with either a Quadword or an Octaword Write Mask.
Unlock Write Mask (forced to miss the XCPGA 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 the XCPGA write buffer)	Longword Write
Unlock Write Mask (forced to miss the XCPGA write buffer)	Longword Unlock Write Mask
Miscellaneous References	
Interrupt Acknowledge	XMI IDENT (assuming that an XMI interrupt is pending and no SSC (only for R3000 IRQ 1) or IP IVINTR (R3000 IRQ 2) interrupts are pending)
I/O Space Write to IVINTR Generation Space	XMI IVINTR

Table 3–2 Detailed CPU Read Operation to XMI Map

Command	IIDAL Length	Second-Level Cache	XMI Transaction	Caches
Read	LW	Hit	None	Fill
Read	LW	Miss	HW Read	Fill
Read-Lock	LW	Force Miss	QW Read Lock	No Fill

Table 3–3 Mapping of XMI Transactions to KN58A/A Interface Module Operations

XMI Operation	Resulting KN58A/A Interface Module Operation
XMI Writes (all types) from other nodes	Perform duplicate tag store lookup. If a hit, load invalidate queue and perform an invalidate on the IIDAL at the next opportunity.
XMI Writes (all types) from the same node	If CSR2<10> is set, perform duplicate tag store lookup. If a hit, load invalidate queue and perform an invalidate on the IIDAL at the next opportunity.
XMI Writes to XMI Private Nodespace	Write the appropriate CSR.
XMI Reads to XMI Private Nodespace	Respond with the appropriate CSR data.
XMI Interrupt	Set the appropriate interrupt pending bit and post interrupt request to the R3000 on IRQ<3:0> lines.
XMI Interprocessor IVINTR	Set the IP IVINTR pending bit and post an R3000 IRQ 2 interrupt request.
XMI Write Error IVINTR	Set XBER<25> and post a hard error INT3 interrupt.
XMI Parity Error Detected	Set XBER<23> and post a soft error interrupt at level 3. If inconsistent, also set XBER<24> and disable the second-level cache.
XMI IDENT	Clear the appropriate INTR pending bit.
XMI Fault Asserted	Set XBER<25> and post a hard error INT3 interrupt.

3.7.1 The XCPGA Chip

The XCPGA is a gate array that serves as the interface between the XMI bus and the KN58A/A interface module's IIDAL bus.

Figure 3–8 XCPGA Block Diagram

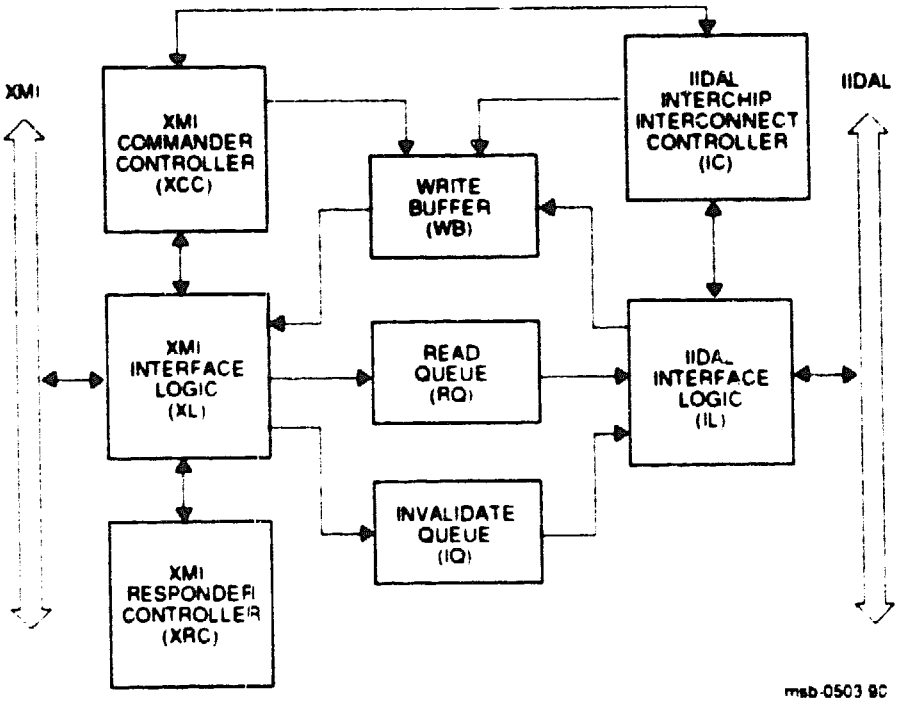


Figure 3–8 is a block diagram of the XCPGA chip. The following is a description of each block:

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

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 IIDAL bus.

The **Invalidate Queue (IQ)** stores up to eight invalidate addresses to be sent to the second-level cache located on the IIDAL bus.

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

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

3.7.2 The XCPGA Write Buffer

The XCPGA Write Buffer (WB) contains two octaword write buffers used to gather writes from the R3000, reducing XMI bus traffic. The XCPGA 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 XCPGA write buffer if the address does not fall within the hexword boundary of the presently active octaword. The XCPGA 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 KN58A/A interface module automatically flushes the XCPGA 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.

3.7.3 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 IIDAL 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 KN58A/A interface module's own writes since the write-through second-level cache always contains the most recent data. A KN58A/A interface 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 IIDAL interface logic arbitrates for the IIDAL 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 IIDAL 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 IIDAL bus.

The KN58A/B CPU module provides the KN58A/A interface module an opportunity to be granted the IIDAL 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 KN58A/A interface 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 both caches.
2. Clear the IQO bit.
3. Enable the second-level cache.

3.7.4 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 KN58A/A interface 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 XCPGA 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 R3000 drops its IPL low enough to recognize the interrupt. The R3000 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 KN58A/A interface 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.

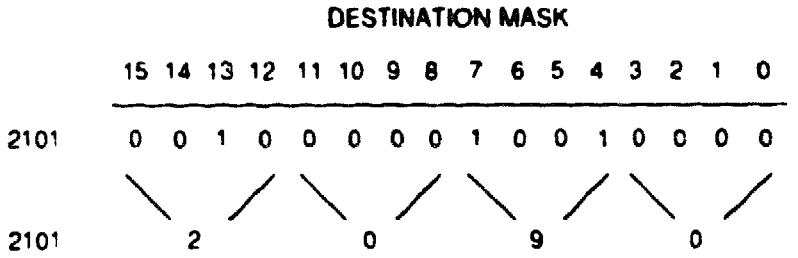
The interrupt-pending bits are controlled as follows:

- All KN58A/A interface modules targeted by an XMI interrupt unconditionally set the corresponding interrupt-pending bits.
- All KN58A/A interface modules unconditonally reset the corresponding interrupt-pending bit whenever an IDENT is transmitted on the XMI. For the KN58A/A interface module generating the IDENT, the interrupt-pending state is cleared before the KN58A/A interface 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 operating system should dismiss the ERR assertion as a passive release. ULTRIX uses a "read nofault" function to read the interrupt vector.
- It is possible that during the time between receipt of an R3000 IRQ 2 XMI interrupt and the generation of the corresponding IDENT that an interprocessor interrupt (IP) IVINTR could be received, as IP IVINTRs interrupt at R3000 IRQ 2. 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 80 (hex) to the CPU. Since no IDENT was transmitted, the interrupt-pending bit at R3000 IRQ 2 is still set, and after servicing the IP IVINTR, the CPU services the XMI device

3.7.5 Implied Vector Interrupts (IVINTR)

The IVINTR is a single-cycle XMI transaction used to implement 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 3-9 Interprocessor IVINTR Generation Address Example



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

msb 0616 90

The KN58A/A interface 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 KN58A/A interface 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 store byte-type instruction. SB (Store Byte) 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 3–9 gives an example of the address needed to send an IP IVINTR to XMI nodes 4, 7, and D.

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 IRQ 2 device interrupt to the CPU. When the CPU acknowledges an IRQ 2 interrupt, the XMI interface checks the IP IVINTR pending bit and, if set, returns a vector of 80 (hex). 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 an INT3 interrupt to the CPU. The CPU vectors directly to 60 (hex) in the SCB for the interrupt. XBER<25> is cleared by an interrupt service routine prior to servicing the write error interrupt. Software then polls all XMI devices to determine which device sent the WE IVINTR.

3.8 KN58A/A Interface Module Registers

The KN58A/A interface module registers consist of registers in XMI private space and XMI required registers.

3.8.1 XMI Registers and Control and Status Register 1 Characteristics

The KN58A/A interface module's XMI registers have the following characteristics:

1 The Mask bits are ignored on writes to the KN58A/A interface module's Control and Status Registers 1 and 2. The CPU always performs a full longword write.

2 Interlocks are supported. The interlock mechanism is explained in Section 4.6.5.

3 The XMI responder queue is only one deep so the KN58A/A interface module will NO ACK subsequent CSR references until the read data for the queued CSR read has been returned.

Table 3–4 XMI Registers for the KN58A/A Interface Module

Register	Mnemonic	Address
XMI Device	XDEV	BB + 00
XMI Bus Error	XBER	BB + 04
XMI Failing Address	XFADR	BB + 08
XMI GPR	XGPR	BB + 0C
Control and Status #2	CSR2	BB + 10

Note: "BB" = base address of a node, which is the address of the first location in nodespace (2180 0000 + (80000 x NODEID)).

Table 3–5 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

Table 3–6 Registers in XMI Private Space

Register	Mnemonic	Address[1]	Location
KN58A/A Interface Module Control/Status #1	CSR1	2000 0000[2]	
R3000 Console ROM		200C 0000 to 2008 FFFF	
KN58A/A ROM		2004 0000 to 200F FFFF	
KN58A/A EEPROM		2008 0000 to 2008 FFFF[2]	
Interlock Register	INTREG	2011 0000	KN58A/B CPU module[3]
Interlock Address	INTADR	2013 0000	KN58A/B CPU module[3]
SSC Base Address	SSCBR	2014 0000	SSC
SSC Configuration	SSCCR	2014 0010	SSC
IIDAL Bus Timeout Control	CBTCR	2014 0020	SSC
Console Select	CONSEL	2014 0030	SSC
Time of Year	TODR	2014 006C	SSC
Console Receiver Control Status	RXCS	2014 0080	SSC
Console Receiver Data Buffer	RXDB	2014 0084	SSC
Console Transmitter Control Status	TXCS	2014 0088	SSC
Console Transmitter Data Buffer	TXDB	2014 008C	SSC
I/O System Reset	IORESET	2014 00DC	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 Internal RAM		2014 0400 to 2014 07FF	
IP IVINTR Generation	IPIVINTRGEN	2101 0000 to 2101 FFFF	
WE IVINTR Generation	WEIVINTRGEN	2102 0000 to 2102 FFFF	

[1]Addresses shown are IIDAL physical addresses. To convert these to R3000 virtual addresses, substitute the leading "2" with a "b". For example, address 2000 0000 in IIDAL physical space becomes b000 0000 in R3000 virtual address space.

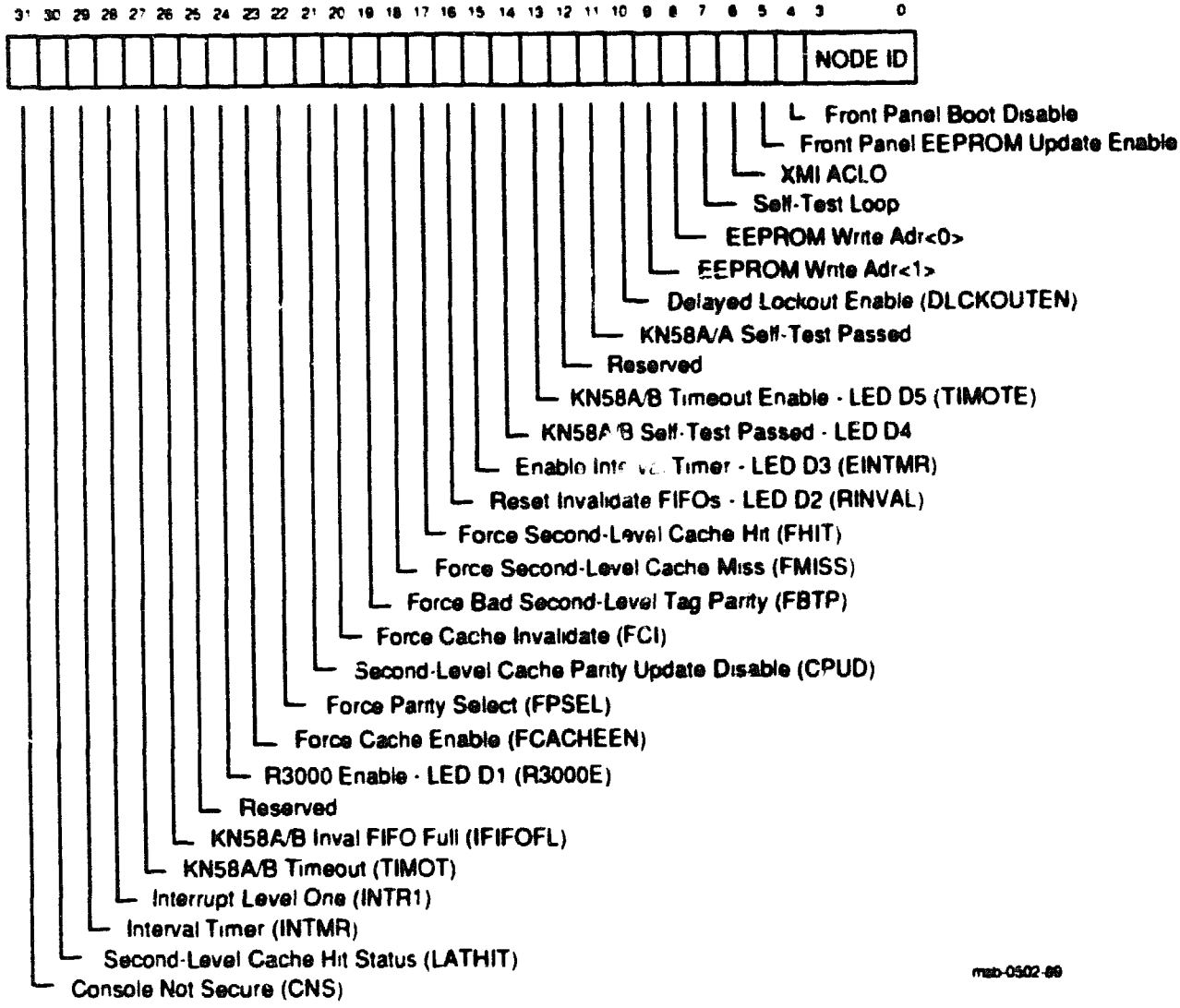
[2]Address and range are determined during processor initialization by using CSR1BADR, CSR1ADMR, EEBADR, and EEADMR.

[3]See Section 4.6.5.

Control and Status Register 1 (CSR1)

CSR1 provides KN58A/A interface module and KN58A/B 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 (reads as a 1), the console is not secure.

bit<30>

Name: Second-Level 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>.

bit<29>

Name: Interval Timer
Mnemonic: INTMR
Type: RO, X

INTMR reflects the state of the SSC's Interval Timer Interrupt pin. When clear, indicates that the interval timer is disabled.

bit<28>

Name: Interrupt Level One
Mnemonic: INTR1
Type: RO, 1

INTR1 reflects the state of Interrupt Level One. When clear, indicates that an interrupt is pending on the II IRQ 1 line.

bit<27>

Name: KN58A/B Timeout
Mnemonic: TIMOT
Type: RO, X

When set, indicates that the KN58A/B CPU module diagnostics have not disabled the timeout logic within the timeout period.

bit<26>

Name: Invalidate FIFO Full
Mnemonic: IFIFOFL
Type: RO, 1

When set, indicates that the first-level cache invalidate FIFO has overflowed.

Error Flag Asserted: R3000 INT4

Additional Status Stored: None

Action: DECsystem 5800 hardware disables the first-level cache invalidate FIFO. To assure cache coherency, the first-level cache is flushed by software. Software then clears the Invalidate FIFO Full status flag.

bit<25>

Name: Read/Write CNTRL P Pending
Mnemonic: CNTRPP
Type: R/W, X

Set when a console CTRL/P command (halt interrrupt) is issued. The KN58A/B CPU module services the interrupt.

bit<24>

Name: R3000 Enable - LED D1
Mnemonic: R3000E
Type: R/W, 0

This bit controls communication between the CVAX and the R3000. When clear, the CVAX is enabled and the R3000 is disabled. When set, control is passed to the R3000, disabling the CVAX. When set, this bit also illuminates status LED D1 on the KN58A/A interface module. See CSR1 <16:12> for descriptions of the bits associated with status LEDs D2 through D6.

bit<23>

Name: Force Cache Enable
Mnemonic: FCACHEEN
Type: R/W, X

Setting FCACHEEN causes the second-level 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 KN58A/A interface 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: Second-Level Cache Parity Update Disable
Mnemonic: CPUD
Type R/W, X

When CPUD is set, the 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, IIDAL 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. See Section 3.6.2 for more information on controlling the second-level cache.

bit<19>

Name: Force Bad Second-Level Tag Parity

Mnemonic: FBTP

Type: R/W, X

When FBTP is set, the parity enable (PE) line on each of the second-level cache 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 Second-Level Cache Miss

Mnemonic: FMISS

Type: R/W, 0

When FMISS is set, the 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 KN58A/A interface 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. See Section 3.6.2 for more information on controlling the second-level cache.

bit<17>

Name: Force Second-Level Cache Hit

Mnemonic: FHIT

Type: R/W, X

When FHIT is set, the 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.

bit<16>

Name: Reset Invalidate FIFOs - LED D2

Mnemonic: RINVAL

Type: R/W, X

Whenever IFIFOFL (CSR1<26>) is set, software resets the KN58A/B CPU module invalidate FIFOs by clearing and then setting RINVAL. RINVAL is also used to hold the invalidate FIFOs reset while a first-level instruction cache flush is in progress. Software sets RINVAL just before a first-level cache flush and clears it just after the flush is complete. When set, this bit also illuminates status LED D2 on the KN58A/A interface module.

bit<15>

Name: Enable Interval Timer - LED D3

Mnemonic: EINTMR

Type: R/W, X

When set, allows the R3000 interval timer to interrupt the R3000. When clear, disables the R3000 interval timer. When set, this bit also illuminates status LED D3 on the KN58A/A interface module.

bit<14>

Name: KN58A/A Self-Test Passed - LED D4

Mnemonic: None

Type: R/W, X

When set, indicates the successful completion of the KN58A/A self-test diagnostics. When set, iliuminates the self-test pass LED on the KN58A/B module and status LED D4 on the KN58A/A module.

bit<13>

Name: KN58A/A Timeout Enable - LED D5

Mnemonic: TIMOTE

Type: R/W, X

When set, enables the KN58A/B CPU module timeout logic. When clear, disables the KN58A/B CPU module timeout logic. TIMOTE must be set for proper operation of the KN58A/B CPU module DMA mechanism. Before control is passed to the KN58A/B CPU module, software enables the timeout logic by first setting EINTMR (CSR1<15>), and then TIMOTE. Diagnostics clear this bit during power-up routines to prevent timeouts. When set, this bit also illuminates status LED D5 on the KN58A/A interface module.

bit<12>

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

bit<11>

Name: Self-Test Pass LED
Mnemonic: STPLED
Type: R/W, 0

STPLED drives the self-test pass LED (D8) on the KN58A/A interface 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 at the same time.

bits<9:8>

Name: EEPROM Write Address <1 0>
Mnemonic: EEWADR
Type: R/W, X

The KN58A/B CPU module provides write data on IIDAL<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>. Ther the write to the EEPROM can be started.

bit<7>

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

When STL is set, the KN58A 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 KN58A 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>

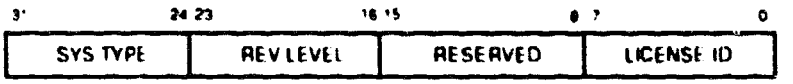
Name: Node ID
Mnemonic: NID
Type: RO

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

System Type (SYSTYPE)

SYSTYPE is a 32-bit register implemented in the KN58A/A interface 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)*



mkb-0551 80

bits<31:24>

Name System Type
Mnemonic SYS TYPE
Type RO

SYS TYPE is 05 (hex) for the KN58A/A interface module.

bits<23:16>

Name Revision Level
Mnemonic REV LEVEL
Type RO

REV LEVEL shows the revision level of the KN58A/A interface module console code. REV LEVEL is encoded 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

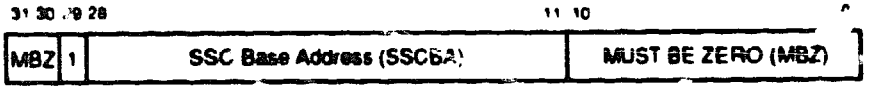
Type: RO

LICENSE ID is set to 01 (hex) to allow the processor to be part of a timesharing system. LICENSE ID is set to 02 (hex) to be part of a fileserver 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 and EEPROM Address Decode Match and Mask Registers, the Diagnostic LED Register, the IIDAL 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>

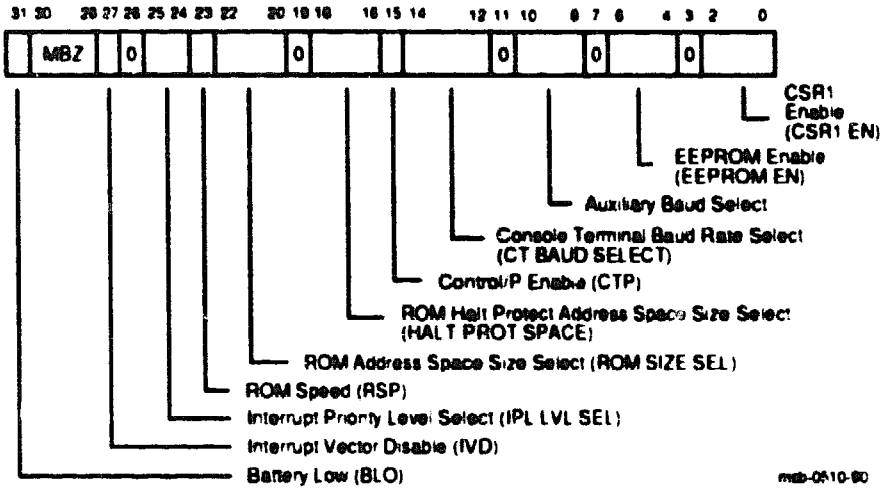
Name: Reserved
Mnemonic: None
Type: R/W, 0

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. Its firmware initialized contents are 0160 A007 (hex).

ADDRESS *2014 0010 (SSC)*



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 KN58A/A interface 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: Interrupt Priority 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 KN58A/A interface module, this field is set to 01 (R3000 IRQ 1) by console code.

bit<23>

Name ROM Speed
Mnemonic RSP
Type R/W, 0

RSP selects the ROM access time. 0=350 ns; 1=250 ns. This bit is normally cleared.

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

During processor initialization, the console code sets this field to 110. This sets the halt protect address space to 512 Kbytes (addresses 2004 0000 to 200B FFFF).

bit<15>

Name: Control/P Enable

Mnemonic: CTP

Type: R/W, 0

When CTP is set and halts are enabled (XMI CON SECURE reset), a CTRL/P typed at the console causes the CVAX to be halted if it is enabled and the R3000 to be interrupted at INT 5 if it is enabled. When CTP is clear and halts are enabled (XMI CON SECURE reset), a BREAK typed at the console causes the CVAX to be halted if it is enabled and the R3000 to be interrupted at INT 5 if it is enabled.

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.

IIDAL Bus Timeout Control Register (CBTCR)

CBTCR controls the amount of time (timeout) allowed to elapse before an IIDAL bus cycle is aborted. This prevents unanswered CVAX 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: IIDAL 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.

bits<30:24>

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.

Time of Year Clock Register (TODR)

TODR is used to measure the duration of power failures.

ADDRESS *2014 0020 (SSC)*

bits<31:0>

Name: Time of Year Since Setting

Mnemonic: TODR

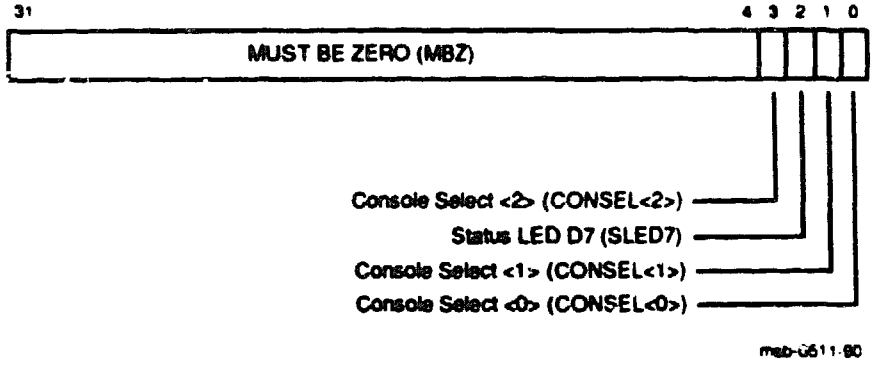
Type: R/W

TODR contains an unsigned 32-bit integer that specifies the number of 10 ms intervals that have elapsed since the last setting. TODR is maintained during a power failure by the XMI TOY BBU PWR line on the XMI backplane.

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; must be zero.

bit<2>

Name Status LED D7
Mnemonic SLED7
Type R/W, 0

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

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:

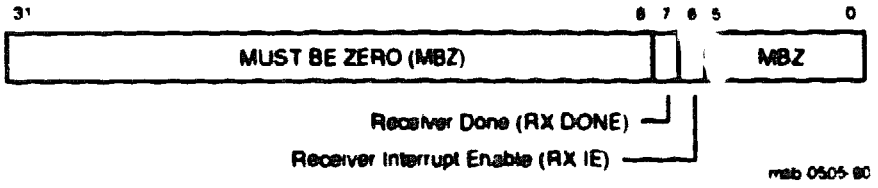
CONSEL<2:0>				RECV		
2	1	0	Drive	XMI	Data	Mode
0	0	0	No	AUX	AUX	Power-up state
0	0	1	No	XMI	AUX	All fail on power-up state
0	1	0	No	LB	AUX	Loopback at XMI XMIT driver input
0	1	1	No	XMI/AUX	Unused	
1	0	0	Yes	AUX	XMI/AUX	Unused
1	0	1	Yes	XMI	XMI/AUX	Boot processor state
1	1	0	Yes	LB	XMI/AUX	Unused
1	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 nodes monitor the XMI console lines for further commands.

Console Receiver Control and Status (RXCS)

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

ADDRESS *2014 0080 (SSC)*



bits<31:8>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be 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>

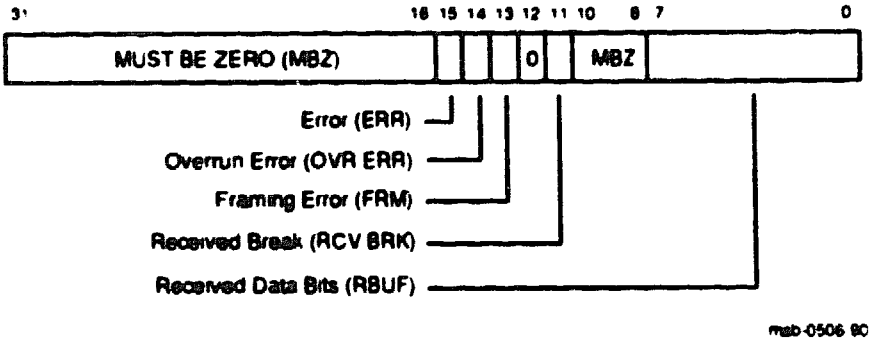
Name: Reserved
Mnemonic: None
Type: –

Reserved; must be 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 *2014 0084 (SSC)*



bits<31:16>

Name Reserved
Mnemonic None
Type -

Reserved; must be 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; must be 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; must be zero.

bits<7:0>

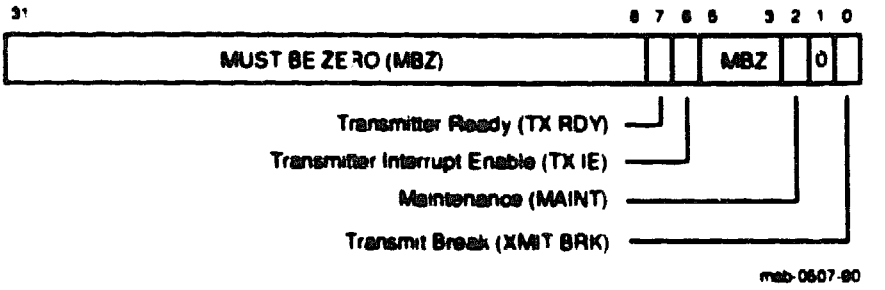
Name: Received Data Bits
Mnemonic: RBUF
Type: RO

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 *2014 0088 (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>

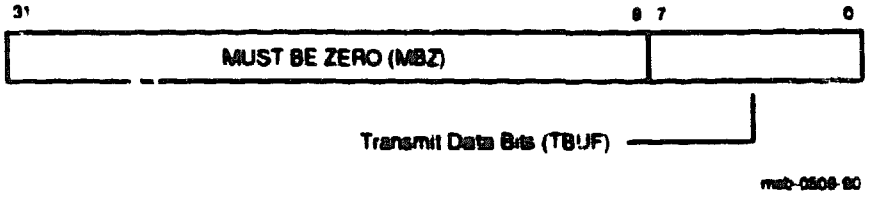
Name: Transmit Break
Mnemonic: XMIT BRK
Type: R/W,0

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 *2014 008C (SSC)*



bits<31:8>

Name: Reserved
Mnemonic: None
Type –

Reserved; must be zero.

bits<7:0>

Name Transmit Data Bits
Mnemonic TBUF
Type WO

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

I/O System Reset Register (IORESET)

IORESET forces a system reset.

ADDRESS *2014 00DC (SSC)*

bits<31:0>

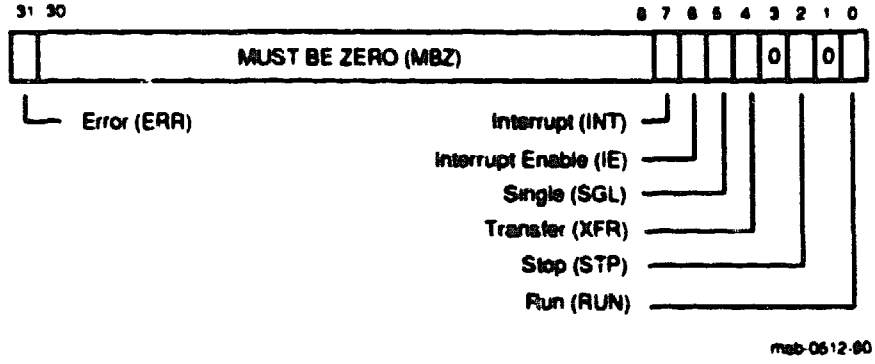
Name: I/O Reset
Mnemonic: IORESET
Type: WO

When IORESET is written, the SSC asserts IORESET L which forces a system hardware reset.

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. Writing a 1 to this bit clears it.

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 R3000 IRQ 0. Writing a 1 to this bit clears it.

bit<6>

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

When IE is set, the timer interrupts at R3000 IRQ 0 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. Always read as zero.

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 R3000 IRQ 0 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

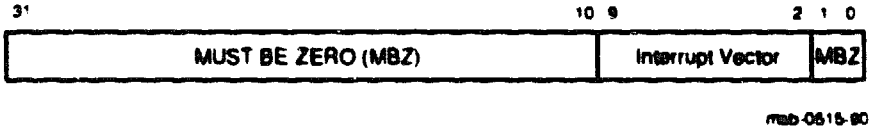
TNIR0 contains the value that is written into TIR0 after an overflow or in response to TCR0<4> (XFR).

Timer Interrupt Vector Register 0 (TIVR0)

TIVR0 is used by timer 0. Although they all occur at the same IPL, interrupts from the console serial line have priority over interrupts from the timers, and timer 0 has priority over timer 1.

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 R300C IRQ 0. When a timer's interrupt is acknowledged, the contents of IV are passed to the CVAX and TCR0<7> is cleared. Interrupt requests are also cleared by clearing TCR0<6> or TCR0<7>.

bits<1:0>

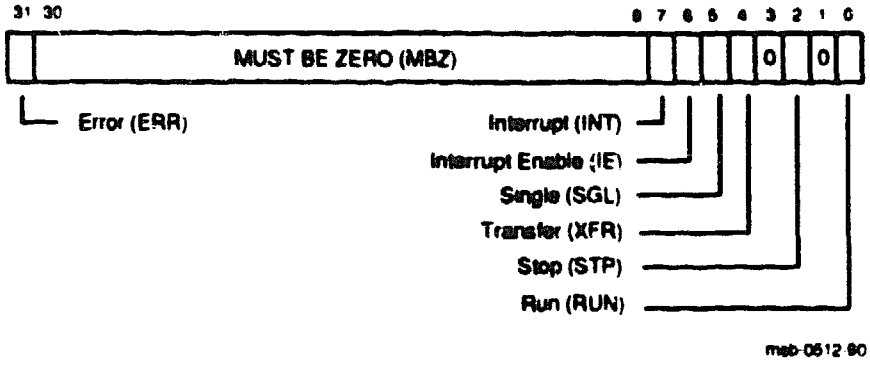
Name: Reserved
Mnemonic: None
Type: R/W, 0

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. Writing a 1 to this bit clears it.

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 R3000 IRQ 0. Writing a 1 to this bit clears it.

bit<6>

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

When IE is set, the timer interrupts at R3000 IRQ 0 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. Always read as zero.

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 R3000 IRQ 1 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

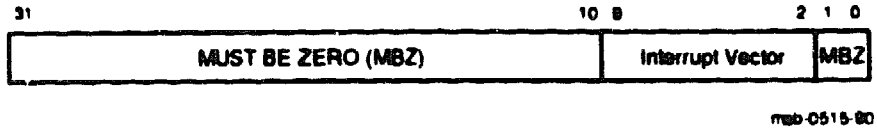
Type: R/W, 0

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 R3000 IRQ 0. When a timer's interrupt is acknowledged, the contents of IV are passed to the CVAX and TCR1<7> is cleared. Interrupt requests are also cleared by clearing TCR1<6> or TCR1<7>.

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 0000 (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)*

mab-0619-90

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>

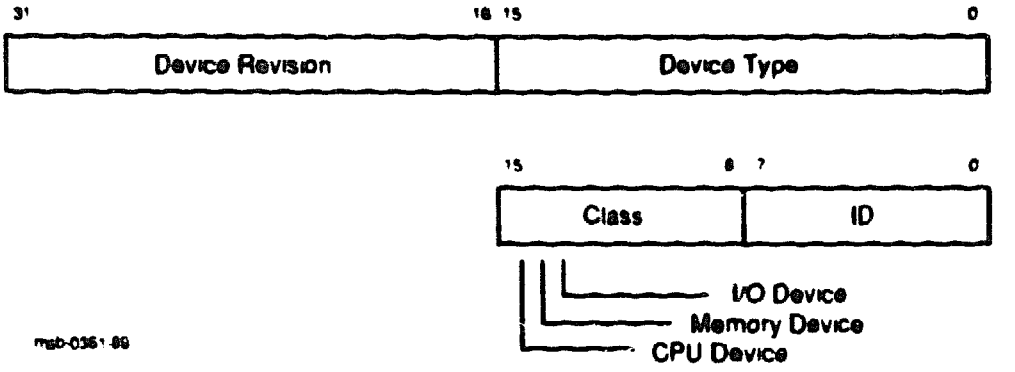
Name: Reserved
Mnemonic: None
Type: R/W, 0

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:

KN58A/A Interface 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

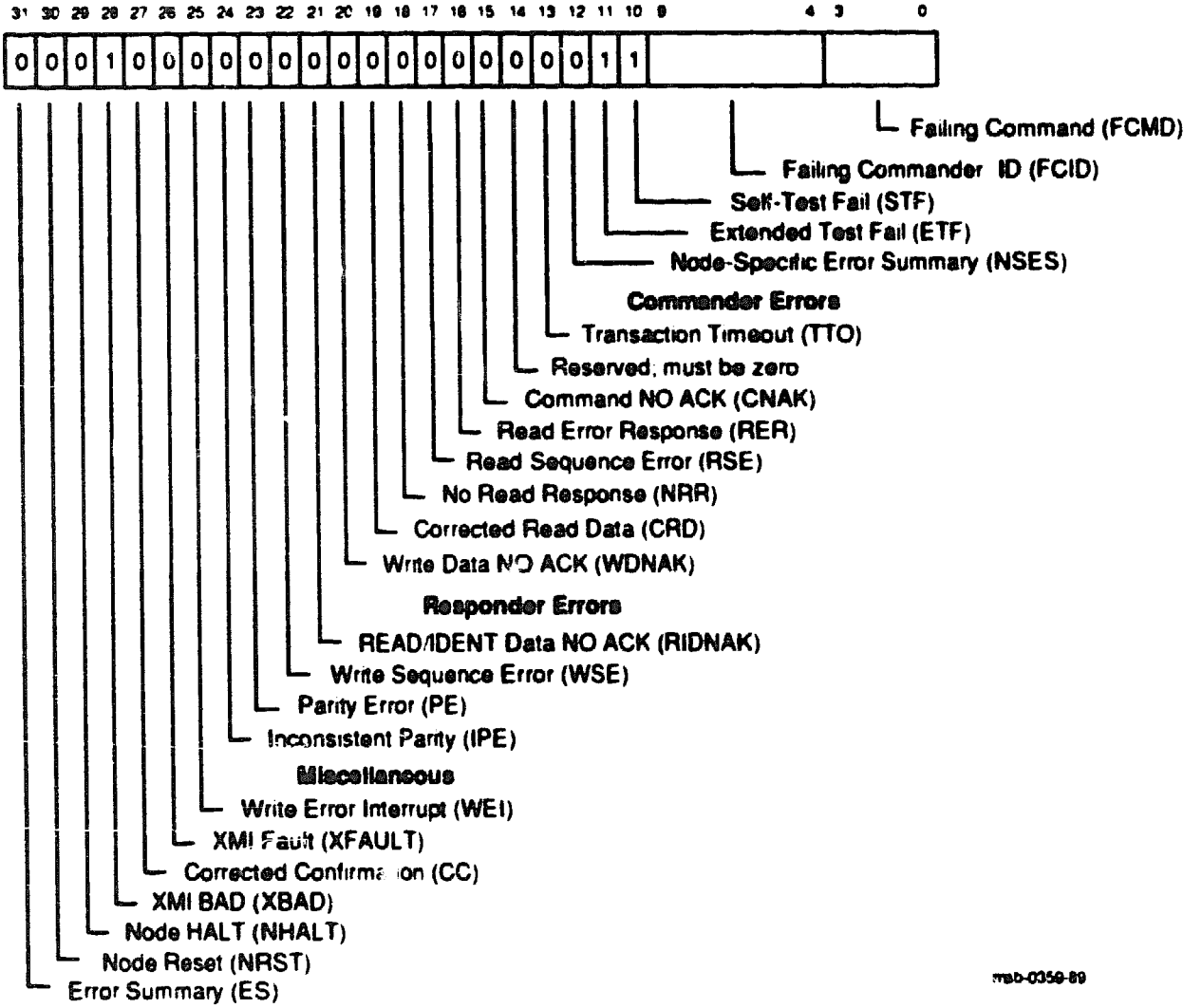
Type: R/W, 0

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 8081 (hex) for the KN58A/A interface 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 KN58A/A interface module will send an INT4 interrupt to the R3000, causing the R3000 to enter console mode.

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: INT3

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: INT3

Additional Status Stored: None

Action: An INT3 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: INT3

Additional Status Stored: None

Action: R3000 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 second-level cache incoherent.

Error Flag Asserted: INT3

Additional Status Stored: None

Action: KN58A/A interface 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: INT3

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 KN58A/A interface 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: INT3

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: INT3

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): BUS ERR if during the first quadword, INT3 (due to CFE) during second-level cache fill.

Error Flag Asserted (READ/IDENT): INT3

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

Action: No retry is attempted. If error flag, the R3000 takes a bus error exception. If CFE, the R3000 takes an INT3 interrupt. 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): BUS ERR if during the first quadword, INT3 (due to CFE) during second-level cache fill.

Error Flag Asserted (READ/IDENT): INT3

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

Action: No retry is attempted. If error flag, the R3000 takes a bus error exception. If second-level cache fill, then CFE is asserted, causing an INT3 interrupt to the R3000 and the sub-block in cache is not validated. Failed address is saved.

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): BUS ERR if during the first quadword, INT3 (due to CFE) during second-level cache fill.

Error Flag Asserted (READ/IDENT): INT3

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

Action: No retry is attempted. If error flag, the R3000 takes a bus error exception. If second-level cache fill, then CFE is asserted, causing an INT3 interrupt to the R3000 and the sub-block in cache is not validated. Failed address is saved.

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): BUS ERR

Error Flag Asserted (WRITE/IDENT): INT3

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>

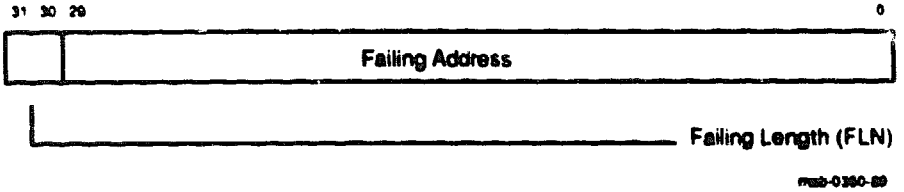
Name: Failing Command
Mnemonic: FCMD
Type: RO

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

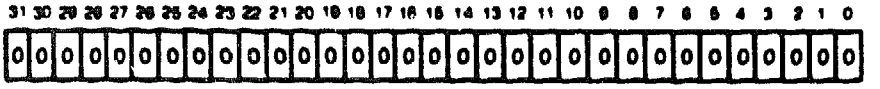
The Failing Address field logs the value of XMI D<29:0> during the command cycle of a failing transaction.

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

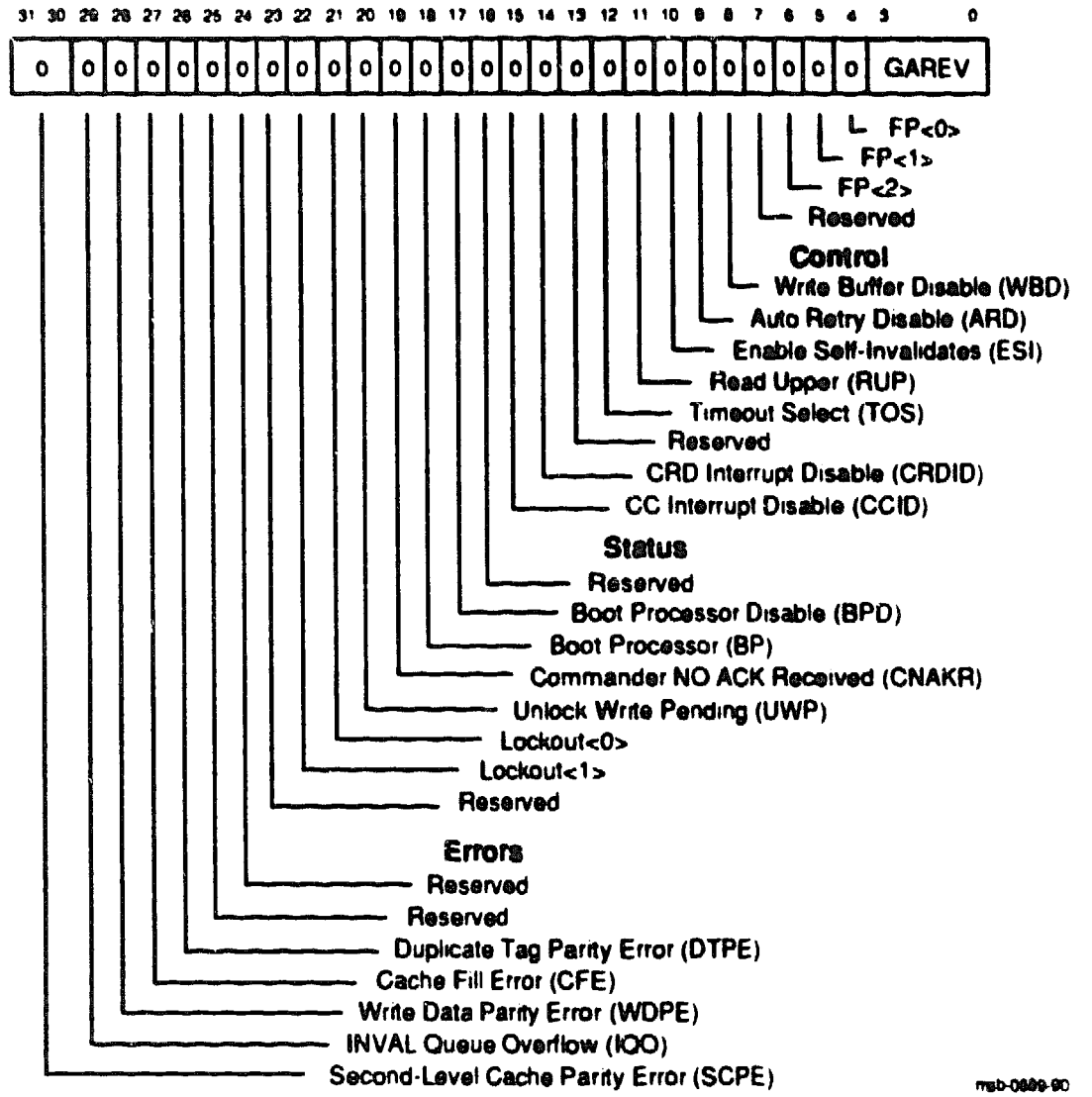
Type: R/W 0

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

Control and Status Register 2 (CSR2)

CSR2 provides KN58A/A interface module control and status to the XMI.

ADDRESS *Nodespace base address + 0000 0010 (SSC)*



bits<31:30>

Name Second-Level Cache Parity Errors
Mnemonic SCPE
Type: R/W1C, 0

These bits indicate second-level cache parity errors as shown:

Bits<31:30>	Error Type
00	None
01	Tag Parity Error (TPE)
10	Valid Bit Parity Error (VPE)
11	Cache Data Parity Error (CDPE)

TPE is a parity error in the Tag Buffer RAMs. VPE is a parity error in the Valid Bit RAMs. CDPE indicates a parity error in the data stored in the second-level cache.

Error Flag Asserted: INT3

Additional Status Stored: None

Action: TPE, VPE, or CDPE cause a cache miss and disable second-level cache by setting FMISS (CSR1<18>). On a write, the occurrence of any of these errors results in a failure to update the cache. Second-level cache should be flushed as described in Section 3.6.2.

bit<29>

Name INVAL Queue Overflow
Mnemonic IQO
Type R/W1C, 0

IQO is set whenever the INVAL queue overflows. The second-level 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: INT3

Additional Status Stored: None

Actions: Second-level cache is flushed as described in Section 3.6.2.

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 IIDAL bus.

Error Flag Asserted: INT3

Additional Status Stored: None

Actions: The write transaction is not allowed to proceed onto the XMI. If a XCPGA write buffer hit, then data is not loaded into the XCPGA 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 CVAX, the second-level cache is disabled because CSR1<FMISS> asserts.

Error Flag Asserted: INT3

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

Action: The Valid bit is not set at the completion of a hexword read. The resulting invalid sub-block 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 KN58A/A interface module hardware disables the second-level cache when this error occurs and posts a soft error interrupt.

Error Flag Asserted: INT3

Additional Status Stored: None

Action: DTPE causes a miss, which if a memory write, results in a potential second-level cache coherency problem. Second-level cache is flushed as described in Section 3.6.2.

---

bits<25:23>

Name: Reserved

Mnemonic: None

Type: -

Reserved.

bits<22:21>

Name Lockout<1:0>

Mnemonic: None

Type: R/W, 01

The KN58A/A interface 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 KN58A/A interface 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 that this KN58A/A is associated with the boot processor. The console code sets this bit after self-test if it determines that this KN58A is associated with 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 this KN58A is diabled from becoming the boot processor. It is loaded by console code on power-up with a state stored in EEPROM.

bit<16>

Name: Reserved
Mnemonic: None
Type: -

Reserved.

bit<15>

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

CCID disables the generation of error interrupts to the KN58A/A interface module 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 INT3 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 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: XCPGA Write Buffer Disable
Mnemonic: WBD
Type: R/W, 0

WBD disables the XCPGA 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 XCPGA 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 XCPGA.

3.9 Initialization, Self-Test, and Booting

This section gives the KN58A/A interface module initialization overview; describes the results of initialization; and then discusses the bootstrapping or restarting of the operating system.

3.9.1 Initialization Overview

The three ways to reset the KN58A/A interface module are:

- Power-Up Sequence—When the DECsystem 5800 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. The XMI does not differentiate between a "real" power-up and a system reset. A system reset is caused by:
  - — Software that asserts XMI RESET L by writing to address 2014 00DC (IORESET). For example, the console **initialize** command generates a system reset if no argument is given by using this mechanism. Note that I/O addresses associated with I/O adapters 0, 1, 2, and 3 are accessed via kseg1. I/O addresses associated with I/O adapters 4, 5, 6, and 7 are accessed via kseg2. See Section 4.2.7.2 for more information and an example.
  - — The XTC power sequencer asserts the XMI RESET L line when the control panel Restart button is pushed.
- Node Reset—Any KN58A/A interface module can be "node reset" by setting its XBER<NRST> bit. The console **initialize** command generates a node reset if a node ID argument is provided. The difference between the node reset and a system reset is that XMI AC LO L is not sequenced during a node reset.

In response to a "cold" power-up or system reset, the KN58A/A interface module(s) participate in the following general initialization sequence:

1 Reset(s) to a known state. (Refer to Table 3–7 for the initialized states of KN58A/A interface module registers on reset and after self-test completes.) The KN58A/B CPU module(s) are held in a reset state during this portion of the initialization sequence.

2 The CVAX(es) start executing the console program at 2004 0000 in ROM. The console program (firmware) initializes the registers and executes a partial ROM-based diagnostic (RBD) self-test.

3 The KN58A/A interface module associated with the boot processor prints the results of the self-test.

4 All KN58A/A interface modules execute a memory interaction test.

5 The CVAX(es) are disabled, and the KN58A/B CPU module(s) are initialized.

6 The R3000(s) start execution at IIDAL physical address 200A 0000. The R3000 portion of the console program executes a self-test for the KN58A/B CPU module. When self-test is complete, the R3000(s) are disabled and the CVAX(es) are enabled.

7 The KN58A/A interface module associated with the boot processor prints the results of the memory interaction test and KN58A/B CPU module self-test.

8 The CVAX(es) run the DWMBA/VAXBI self-tests.

9 The KN58A/A interface module associated with the boot processor prints the results of the DWMBA/VAXBI self-tests.

10 The KN58A/A interface module associated with the boot processor configures memory.

11 If maintenance mode is not selected, the KN58A/A interface module associated with the boot processor is disabled, the R3000 starts execution at 200A 0000, and the KN58A/B CPU module passes control to the operating system.

12 The operating system initialization code performs the final system initialization.

3.9.2 Initialization Details

The following is a flowchart and summary of the initialization process. The sections that follow explain in some detail the process outlined by the flowchart.

Figure 3–10 Initialization Flowchart

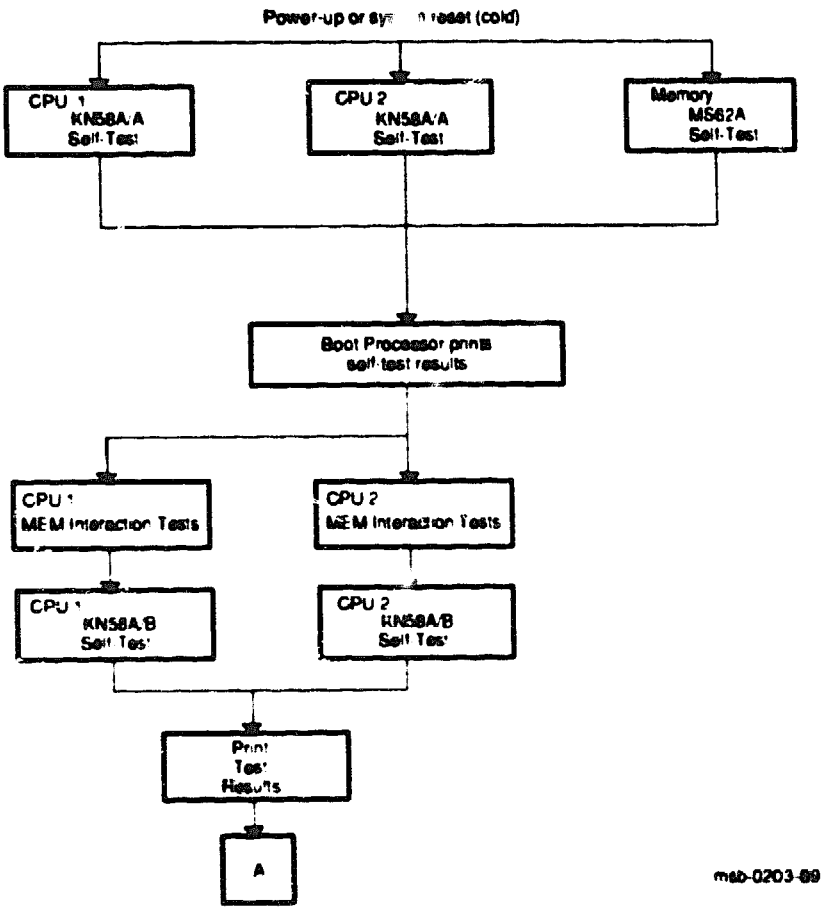
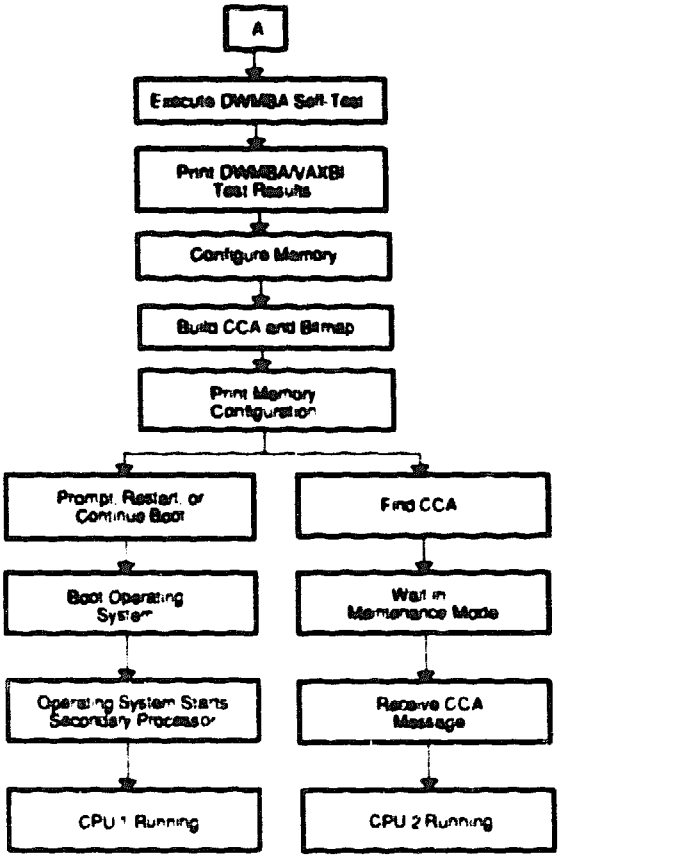


Figure 3–10 Cont'd. on next page

Figure 3–10 (Cont.) Initialization Flowchart



msb-0204-89

3.9.2.1 Restart Sequence

Initialization typically begins with a *CVAX processor restart*. The most common of these is a *03 reset*, which occurs on system power-up or reset. The R3000 is blocked from executing during CVAX initialization.

The first objectives of the console code during a restart sequence are to establish its data area and stack, indicate that it is executing, and save the interrupted machine state, if any. All restarts have the following sequence in common:

1 The SSC Configuration Register (SSCCR) is temporarily set to 0076 0000 to select the ROM size and halt protect region.

2 The T1 interval timer is started.

3 If the self-test ROM is present, the KN58A/A interface module and KN58A/B CPU module self-tests are started.

4 The KN58A/A interface module self-test returns the value of the XBER<NRST> bit, and this value is stored in SSC RAM.

5 The signature longword in SSC RAM is checked and, if valid, its value is changed. This step is then skipped in the future.

6 The SSC address decode registers CSR1BADR and CSR1ADMR are initialized to allow access to CSR1.

7 The SSC address decode registers EEBADR and EEADMR are initialized.

8 The SSC Configuration Register (SSCCR) is loaded with the value 8176 5007, which specifies the following configuration:

Bit(s)	Setting(s)	Description
31	1	Clears any previous battery low condition to prevent the time of year clock from being reinitialized by subsequent resets.
30:28	000	Must be zero.
27	0	Enable interrupt vectors for console lines and programmable timers.
26	0	Must be zero.
25:24	01	Select IPL15 interrupts for console lines and programmable timers.
23	0	Select 350 ns ROM speed.
22:20	111	Select ROM size of 1024K. This maps the ROM and EEPROM address space to appear at both 2004 0000 through 200B FFFF and 200C 0000 through 2012 FFFF.
19	0	Must be zero.
18:16	110	Select halt protect region of 512K. The first image of the ROM/EEPROM address space is halt protected. The second image is halt enabled and used to leave console mode and to run CVAX based boot code.

Bit(s)	Setting(s)	Description
15	1	Select CTRL/P as the console interrupt character.
14:12	101	Select default baud rate of 9600.
11:7	0000	Must be zero.
6:4	000	Disable address decode for EEPROM writes.
3	0	Must be zero.
2:0	111	Enable address decode for CSR1 reads and writes.

9 The IIDAL Bus Timeout Register (CBTCR) is set to 0000 9000.

10 The interrupt vectors for the programmable timers are set to 78 for timer 0 and 7C for timer 1.

11 The internal and external caches are disabled.

12 Additional initialization of the console data area in SSC RAM is performed, as described below.

13 The maintenance mode dispatch routine is called with a CALLG instruction, which causes the remaining CVAX GPRs to be saved in SSC RAM.

The additional initialization of the console data area in SSC RAM consists of the following for a "cold" restart:

1 The Device Revision field (bits<31:16>) of the XDEV Register is set according to the module revision value stored in EEPROM. If EEPROM is unusable, the Device Revision field is left as zero.

2 If self-test passed, XBER<XBAD> is cleared, which deasserts the XMI BAD signal.

3 If this is the KN58A/A interface module associated with the primary processor, the primary sets the console terminal baud rate according to the value stored in EEPROM. If EEPROM is unusable, the primary sets its baud rate to 1200.

4 All processors in the system scan the XMI node space to locate the processor with the lowest XMI number—the primary processor. The primary processor prints initial self-test results on the console terminal and sets CSR2<BP>. If the KN58A/A interface module associated with the primary processor fails self-test, it bypasses further testing and memory configuration and attempts to enter maintenance mode.

5 All KN58A/A interface modules that pass the initial self-test execute an extended test, also referred to as the memory interaction test. If the KN58A/A interface module associated with the primary processor fails the extended test, it bypasses further testing and memory configuration and attempts to enter maintenance mode.

6 Each KN58A/A interface module sets the "R3000 self-test" request code in SSC RAM and passes control to the R3000 portion of the console program, which initiates the KN58A/B CPU module self-test. R3000 code stores the results of the KN58A/B CPU module self-test in SSC RAM and returns control to the CVAX portion of the console. If the

KN58A/B CPU module associated with the primary processor fails self-test, it enters maintenance mode after completing the DWMBA/VAXBI self-test.

7 If a processor passes its extended test and KN58A/B CPU module self-test, it again clears XBER<XBAD> and CSR2<BP> and clears XBER<ETF>.

8 The KN58A/A interface module associated with the primary processor prints the self-test results on the console terminal.

9 The primary processor runs the DWMBA/VAXBI self-test and prints the results. The DWMBA/VAXBI self-test ensures that all DWMBAs have powered up and completed their self-tests before returning to the console.

10 The primary processor configures memory as described in Section 3.9.3. If sufficient memory is found, it builds the console communications area (CCA) in the highest addressed pages of memory.

11 Once the total size of configured memory is known, the primary processor completes initialization of each DWMBA as described in Section 3.9.4.

12 The primary processor once again sets its CSR2<BP> bit to indicate to secondary processors that they should idle in maintenance mode. Secondary processors poll their CCA receive buffers for a command to start R3000 execution.

13 The primary processor checks the ROM and EEPROM revisions posted in the CCA by secondary processors. The primary processor prints a revision banner along with any mismatches that may have been found.

14 The primary processor displays the maintenance mode prompt if maintenance mode is selected. If maintenance mode is not selected, the primary processor loads an "R3000 console mode" request in SSC RAM and passes control to the R3000 portion of the console program.

3.9.2.2 Node Reset

Some of the initialization steps are inappropriate with a node reset since the remainder of the system must continue to function. The sequence is modified as follows:

- Self-test results are not displayed.
- Extended tests are not run. Instead, the XBER<ETF> and XBER<XBAD> bits are cleared.
- DWMBA self-test is not run. No initialization of the DWMBA is performed and, therefore, no test results are printed.
- No memory configuration is performed. The processor searches for the CCA. If the CCA connot be found, the processor hangs.
- The XBER<NRST> bit has its initial value stored in console scratch memory and is then cleared by self-test. The stored value is used by the console program to detect a node reset.

3.9.2.3 Halt Interrupt

A halt interrupt entry to the console program occurs when CTRL/P is entered on the console terminal (and the front panel key switch is not in the *Secure* position) or whenever the XBER<NHALT> bit is set. A halt interrupt produces a halt interrupt signal which is gated to whichever processor chip (CVAX or R3000) is currently controlling the system.

If the CVAX is executing, the halt signal causes a CVAX restart with a code of 02. The console program performs its initialization sequence and displays the maintenance mode prompt.

If the R3000 is executing, the halt signal causes a level 4 interrupt to the R3000. If interrupts are enabled, the operating system handles the interrupt and calls the console program. The console program then displays the console prompt.

3.9.2.4 Errors

An error entry into the console program occurs when the CVAX encounters any restart condition other than reset (03) or halt (02). This occurs when a CVAX HALT instruction is executed or some other catastrophic processor event occurs. The console program performs the common initialization sequence and displays the maintenance prompt.

Errors do not cause entry into the console program while the R3000 is executing. The operating system handles the error and typically calls back to the console program to request a system reboot.

3.9.3 Memory Configuration

The CVAX portion of the console program configures memory by setting the interleave and starting address for each array. The console program controls the memory configuration because the console uses a portion of the main memory to hold the console communications area (CCA) and the R3000 console program state. The console also builds a physical memory bitmap showing all usable and unusable pages. The results of the CPU/memory interaction test are used to determine the defective pages.

The memory configuration process verifies that a minimum of 256 Kbytes of usable memory per processor is available plus the space used by the CCA and bitmap. The location of the CCA is determined and marked as unusable in the bitmap.

If the primary processor is unable to find the minimum required memory, it displays an error message and enters maintenance mode. Some maintenance functions may not be available due to lack of memory. Console mode is unavailable.

3.9.3.1 Selection of Interleave

The interleave is specified by the value of the **interleave** environment variable. There are three types of interleave:

- DEFAULT – The console program makes all interleave decisions.
- EXPLICIT – The user supplies configuration data.
- NONE – No arrays will be interleaved.

If the **interleave** environment variable is set to the string **default**, the console program attempts to form interleave sets. The largest interleave factor is obtained for each group of like-sized arrays. If there are more arrays than can be evenly interleaved, the criteria is repeated for the remaining arrays until only single arrays remain. The array with the lowest XMI node ID is assigned to the lowest physical address. All arrays of the same size are configured before arrays of a different size.

Any array containing hard (unrecoverable) errors is not included in a default interleave set. Instead, it is configured as uninterleaved. The remaining arrays that would have formed the set are freed up for inclusion in another interleave set, if one can be formed. Arrays with hard errors are configured at the top of physical address space, because ULTRIX requires contiguous physical memory.

If the environment variable specifies EXPLICIT interleave sets, the console program interleaves and configures the arrays in the order specified. When an array in the set has unrecoverable errors, all arrays in the set are configured without interleaving. If a set specifies a nonexistent array or is otherwise inconsistent, all arrays in the set are configured without interleaving.

If the environment vaiable contains the string **none**, the arrays are configured uninterleaved, in order by node ID, with the lowest numbered array at the lowest physical address.

If the environment variable is undefined, contains an invalid value, or if EEPROM is corrupt, memory is configured as if the environment variable were **default**.

3.9.3.2 Memory Testing and the Bitmap

Memory self-test indicates that an array has no unrecoverable (hard) errors, one hard error, or multiple hard errors. Self-test executes on all arrays in parallel and is faster than software testing of memory. An attempt is made to use the results of self-test and avoid performing software testing of memory.

A hard (unrecoverable) error is called an RDS error and is defined as one that is an uncorrectable double-bit error by memory hardware. A correctable (CRD) error is not considered a hard error, and pages containing CRD errors are marked as usable.

If self-test indicates that an interleave set contains no hard errors, the set never undergoes software testing. If an array in an interleave set contains one or more hard errors, that set is uninterleaved and the failing array is software tested.

Software testing is performed, one page at a time, beginning with the lowest addressed page in the array. If required, this testing takes about 7 seconds per megabyte. All locations in the page are written with patterns. The locations are then read. If the value read from any location does not match the pattern, or if a machine check occurs reading any location, that page is marked as unusable, and testing resumes with the next page. The testing patterns are in this order:

- All ones
- All zeros
- Alternating one/zero/one
- Alternating one/zero/one with ECC bits complemented
- The address of each longword
- The complement of the address of each longword

The memory bitmap is initially built in the first block of memory large enough to hold it. When the bitmap has been configured, it is then moved to a page-aligned location below the CCA.

If pages must be marked as bad before enough memory has been found to hold the bitmap, some pages are retested after the bitmap has been built. The bitmap shows, in addition to pages marked with hard errors, pages marked as unusable because they are either the bitmap's own pages or are CCA pages. A page is marked as unavailable when its corresponding bitmap bit is cleared.

3.9.4 DWMBA Configuration

The console program performs minimal initialization of the DWMBAs following self-test. The initialization performed for each DWMBA is:

1 The BI Starting Address Register (bb+20) and the BI Ending Address Register (bb+24) are initialized to the starting and ending limits of XMI memory.

2 The BICSR (bb+04) has its BI Broke bit cleared.

3 The DMA-B Transmit Buffer is disabled under these conditions:

- The DWMBA/A module Device Register shows a revision less than 2.
- The system contains five or more XMI commanders and the DWMBA/B module Device Register shows a revision of less than 0A (hex).

If a DWMBA/A module fails its self-test (as indicated by XBER<STF>), the BP asserts its own XBER<XBAD> bit to drive the XMI BAD L line.

3.9.5 Initialized State

In response to a reset, the KN58A/A interface module is initialized to a known *hardware initialized state* by external reset signals. If self-test runs successfully, the state of the KN58A/A interface module is revised to a *firmware initialized state* by the firmware. The firmware initialized state is what the operating system uses as the initial state of the KN58A/A interface module when it starts. The hardware and firmware initial values for the registers of the KN58A/A interface module are summarized in Table 3–7.

Table 3–7 KN58A/A Interface Module Initial Register States

Register	Address	Hardware Init	Firmware Init	Notes
CSR1	2000 0000	*	0000 0000	See footnote[1]
SSCBR	2014 0000	2014 0000	2014 0000	
SSCCR	2014 0010	0000 0000	0160 A007	Bit<31> may be 1
CLKCR	2014 0020	0000 0000	0000 9000	IIDAL timeout = 36.8 ms
CONSEL	2014 0030	0000 0000	0000 000C	If not boot processor
			0000 000D	If boot processor
CSR1BADR	2014 0130	0000 0000	2000 0000	
CSR1ADMR	2014 0134	0000 0000	0000 0000	
EEBADR	2014 0140	0000 0000	2008 0000	
EEADMR	2014 0144	0000 0000	0000 7FFC	
XDEV	2180 0000	0000 0000	0000 8001	
XBER	2180 0004	1000 0C**	0000 00**	Bits<9:0> always contains the last commander ID and command unless an error occurs
XFADR	2180 0008	0000 0000	**** ****	Always contains the last commander address unless an error occurs
XGPR	2180 000C	0000 0000	0000 0000	
CSR2	2180 0010	0000 0000	0000 0000	

[1]The state of CSR1 after power-up but before self-test is (in binary):
11X0 X0X0 XXXX X0XX XXXX 0XXX 1*** ****
An asterisk in a bit position indicates the state is determined by a signal from the backplane, such as the node ID lines.

3.9.6 Restarting or Bootstrapping the Operating System

The console can bootstrap a copy of the operating system for the R3000 from a tape or disk device. In maintenance mode, the console can also bootstrap a copy of the VAX Diagnostic Supervisor (VAX/DS) from a VAX formatted tape or ULTRIX formatted disk.

Only the primary processor can initiate a bootstrap. A secondary processor never attempts to restart the operating system, but rather waits in a halted state until the operating system passes a START command via the console communications area (CCA).

3.9.6.1 Operating System Restart

The primary processor attempts to start/restart the operating system following a system reset, provided:

- The front panel key switches are not set to the combination of *Enable* and *Halt*.
- The CVAX chip was not running when a power failure occurred.

If the "last chip" flag in SSC RAM indicates that the CVAX was running when a power failure occurred, maintenance mode is entered. Otherwise, the console places an "operating system restart" request code into SSC RAM and passes control to the R3000 portion of the console program.

The R3000 portion of the console program controls the restart of the operating system via a memory data structure called the restart parameter block (RPB). The RPB, located at IIDAL physical address 0000 0400 (R3000 virtual address a000 0400), has the format shown in Figure 3–11.

Figure 3–11 Restart Parameter Block Format

SIGNATURE PATTERN (FEED FACE)	0000 0400
ADDRESS OF RESTART ROUTINE	0000 0404
RESTART OCCURRED FLAG	0000 0408
CHECKSUM OF 1ST 32 WORDS OF RESTART	0000 040C
START ADDRESS OF CONSOLE DATA	0000 0410
END ADDRESS OF CONSOLE DATA	0000 0414

msb-0567-90

The restart attempt will fail if the RPB is not valid or if the RPB's "restart occurred" flag is already set. If a restart attempt fails, the system is rebooted as specified by the default boot path.

3.9.6.2 Operating System Bootstrap

The R3000 portion of the console code attempts to bootstrap the operating system whenever one of the following events occurs:

- The front panel key switch is in the *Enabled* position and the **boot** command is typed on the console terminal.
- The system is reset and the front panel key switch is in the *Autostart* position.
- A restart is attempted and fails.

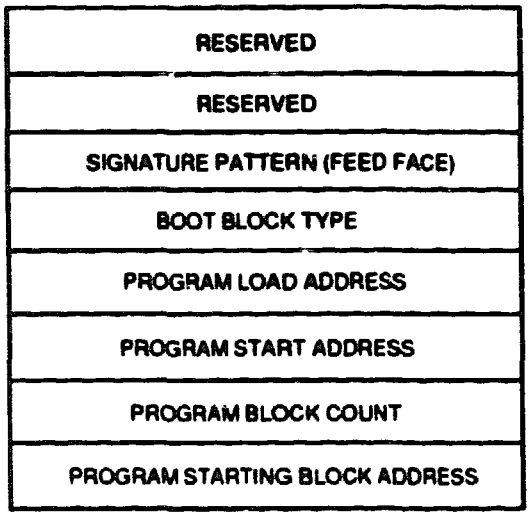
The console brings an operating system loader into memory using a bootblock scheme and then assists the loader through use of the support routines described in Section 3.9.6.2.1 and Appendix A.

The boot devices supported are the KDB50 VAXBI disk adapter, the TBK70 tape adapter, and the CIBCA-B VAXBI CI adapters.

The operating system boot process is as follows:

1. Determine the device to be booted from the **boot** command or take the default boot device from the **bootpath** environment variable. Store the information in SSC RAM.
2. If the boot was initiated by a **boot** command, set a "reset due to bootstrap" flag in SSC RAM and reset the system. This causes an initialization and self-test of all modules.
3. Open the boot device and read block zero. This is the bootblock and it specifies the location of the operating system loader. The format of the bootblock is shown in Figure 3–12.
4. Read the sequence of blocks described by the bootblock into memory.
5. Call the loaded code at the start address found in the bootblock. Any arguments supplied by the **boot** command are passed using the C-language argc/argv conventions. A pointer to the current environment is also passed.
6. If any step in this process cannot be completed, the bootstrap fails and the console prompt is displayed.

Figure 3–12 Bootblock Format



msb-0568-90

3.9.6.2.1 Bootstrap Support Routines in the Console

The ULTRIX bootstrap loader contains no code for performing I/O or machine dependent operations (such as flushing cache). Instead, the loader calls a set of routines provided by the console. The addresses of these routines are obtained in a transfer vector located in console ROM at address B00A 0000. All routines are called as C-language routines using standard R3000 calling conventions.

The routines are of several types:

- console - Invokes console programs for functions such as restarts and reboots.
- saio - Provides basic I/O support for stand-alone programs. Does not support file structure processing or I/O to buffers in kuseg or kseg2. Only one I/O adapter of each type can be active at a time
- machine - Provides machine-specific functions such as interlocked memory access and cache flushes.
- libc - Provides a subset of standard C-language library functions (getenv, setjump, etc.)
- parser - Provides access to console command parser.
- command - Provides access to a subset of console commands.

The list of supported routines and their calling sequences is provided in Appendix A.

3.9.7 Console Use of Address Space

The R3000 portion of the console program reserves memory from physical address 0000 0500 through 0000 FFFF (visible via kseg1 as addresses A000 0500 through A000 FFFF) for its own use. This space holds the console program stack and static data structures.

Memory from physical address 0000 0000 through 0000 04FF is shared with the operating system and holds the restart parameter block and the current exception handling code.

The console generally uses kseg2 addresses starting from address 0000 0000 to access physical addresses not visible from kseg1. These include all addresses beyond 256 Mbytes of memory and all XMI I/O adapter address spaces. The console establishes translation lookaside buffer (TLB) entries for the addresses it needs to map and restores the original contents of the TLB entries upon console exit.

3.9.8 Bootstrap of the VAX Diagnostic Supervisor (VAX/DS)

The VAX Diagnostic Supervisor (VAX/DS) is booted by typing the **BOOT** command at the maintenance mode prompt. The boot device must be in VMS format or have a VMS format bootblock. The CVAX portion of the console code handles all aspects of booting VAX/DS. Only the primary processor can boot VAX/DS.

The console's first goal in booting VAX/DS is to load the primary bootstrap program, VMB, into memory and begin its execution. VMB is loaded from the device specified by the **BOOT** command. A set of minimal device handler routines, called boot primitives, are used 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 stored in SSC RAM. Then the console forces a system reset. The system reset causes all processors, memories, and I/O adapters to perform self-test.

The second phase of bootstrap then begins. The console program is reentered and determines (from values in SSC RAM) that the reentry is part of a bootstrap operation. Boot parameters are passed from SSC RAM to the CVAX GPRs, the boot primitive is again loaded, and control passes to the boot primitive.

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 TBK70 tape adapter, and the CIBCA-B VAXBI CI adapters are supported. The table in EEPROM allows new primitives to be added as new devices are developed.

If the target device is a disk, the boot primitive loads logical block zero (also called the *bootblock*) into memory and transfers control to it. The bootblock contains code that specifies the location and size of the VMB image on disk. If the target device is tape, the boot primitive skips any tape labels and reads the first file from the tape into memory. Once VMB has been loaded, the bootblock passes control to it. The boot primitive must preserve the boot parameters stored in the GPRs.

The register conventions used by the bootblock program and the conventions for passing parameters to VMB are described in the paragraphs that follow.

3.9.8.1 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 to be used.

Table 3–8 show how the registers are used.

Table 3–8 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 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

3.9.8.2 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.

3.9.8.3 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 3–9 and the output parameters shown in Table 3–10.

Table 3–9 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 3–10 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

3.9.8.4 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. GPR8 returns the length of the block just read.

3.10 Interprocessor Communication through the Console Program

Each CPU of a multiprocessor system must communicate with the other CPUs and the operating system. In the DECsystem 5800, interprocessor communication is handled by the KN58A/A interface module. This section describes interprocessor communication.

The CVAX portion of the console program runs on each processor of a multiprocessor DECsystem 5800. These copies of the console program 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).

There is no requirement for communication between multiple copies of the R3000 portion of the console program, since the R3000 portion of the console program runs on the primary processor only. The R3000 portion of the console program, however, uses the CCA to communicate with the CVAX portion of the console program running on a secondary processor. It also uses the CCA to communicate with the operating system.

The primary processor controls the console terminal and, therefore, most of the communication in the DECsystem 5800. There is no communication between secondary processors.

3.10.1 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 primary console to/from secondary console.

The operating system on the primary processor must send complete console commands to the CVAX portion of 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.

2 Primary console to/from secondary console requires two different types of communication.

The CVAX or R3000 portion of the primary console sends complete commands to the CVAX portion of the secondary, allowing the primary console to update the copy of a parameter stored on a secondary processor. An example of this type of communication is to synchronize the console terminal baud rate whenever it is changed on the primary.

The CVAX portion of the secondary console sends complete responses to the CVAX or R3000 portions of 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.

For intelligent I/O adapters only, 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 Console mode primary processor to/from running secondary processor.

This path exists as a side effect of supporting responses from a secondary console, but its use is reserved.

3.10.2 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 CVAX code at bootstrap time through VAX GPR7. The location of the CCA is passed to R3000 code at bootstrap time via the cca environment variable.

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 3–13 and Figure 3–14. The contents of the fields are described in Table 3–11.

Figure 3–13 CCA Layout, Part 1

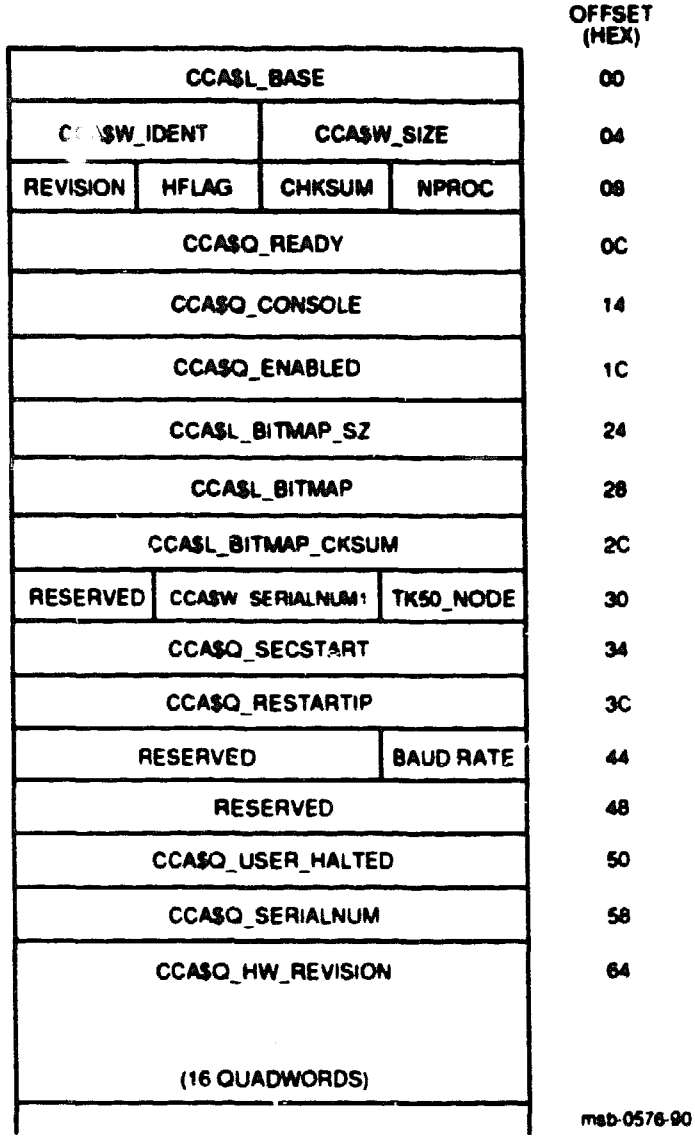


Figure 3–14 CCA Layout, Part 2

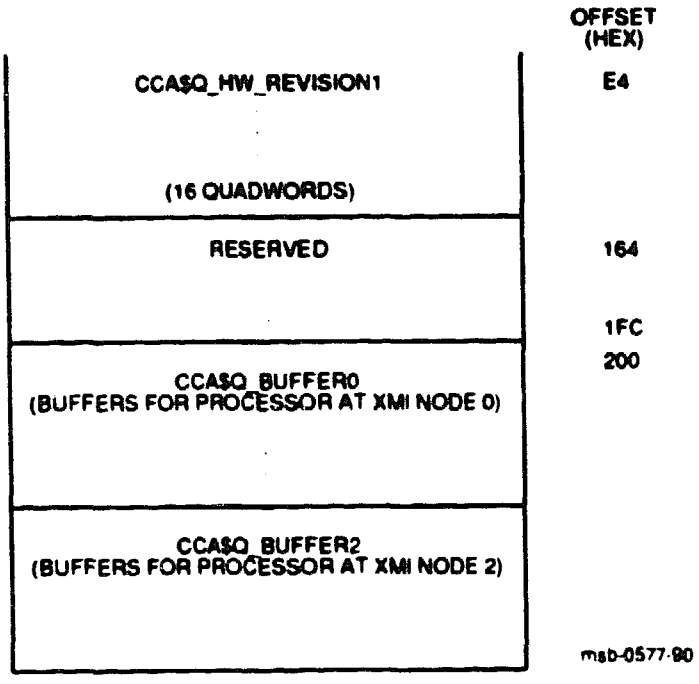
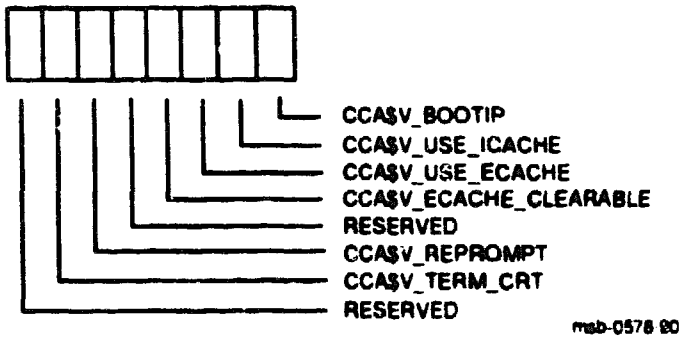


Table 3–11 CCA Fields

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, usually 16.
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:

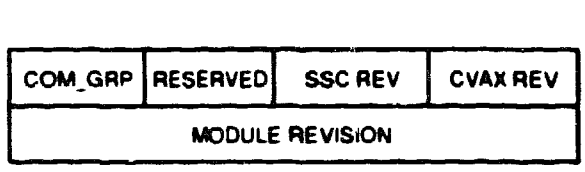


CCA$V_BOOTIP	When set, a bootstrap is being attempted. This prevents repeated attempts to bootstrap after a failure.
CCA$V_USE_ICACHE	When set, the CVAX chip internal cache is to be enabled by the operating system.
CCA$V_USE_ECACHE	When set, the second-level cache is to be enabled by the operating system.
CCA$V_ECACHE_CLEARABLE	When set, the second-level cache clear operation can be used successfully. Some operating system error recovery is needed to clear the cache.
CCA$V_REPROMPT	This bit is used internally by the console to support the SET CPU command.
CCA$V_TERM_CRT	When set, the console terminal is a CRT.

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. 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.
CCA$L_BITMAP_SZ	The size, in bytes, of the physical memory bitmap. The bitmap is always an even number of longwords in length.

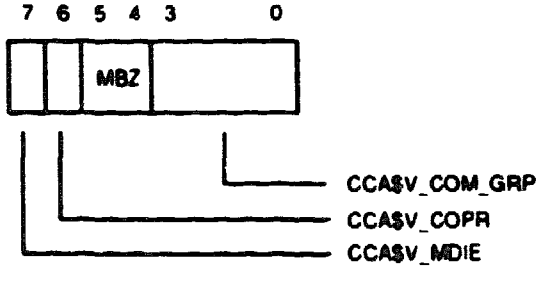
Table 3–11 (Cont.) CCA Fields

Field	Description
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$W_SERIALNUM1	First two characters of the system serial number. Concatenated with CCA$Q_SERIALNUM.
CCA$B_TK50_NODE	Reserved; not used.
CCA$Q_SECSTART	Reserved; not used.
CCA$Q_RESTARTIP	Reserved; not used.
CCA$B_BAUD_RATE	The SSC bit value for the current console baud rate.
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 last eight characters of the system serial number. Concatenated with CCA$W_SERIALNUM1.
CCA$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 layout of this quadword is:



msb-0582-90

The layout of the COM_GRP byte is:



msb-0579-90

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 3–11 (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.
CCA$Q_HW_REVISION1	Consists of an array of 16 quadwords, one for each system node. For nodes with processors, the first longword contains the contents of the R3000 Processor Revision ID register. The second longword contains the console ROM version (right word) and console EEPROM patch level (left word).	

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 3–15, and the contents of the field are described in Table 3–12.

Figure 3–15 Layout of XMI Node Buffers

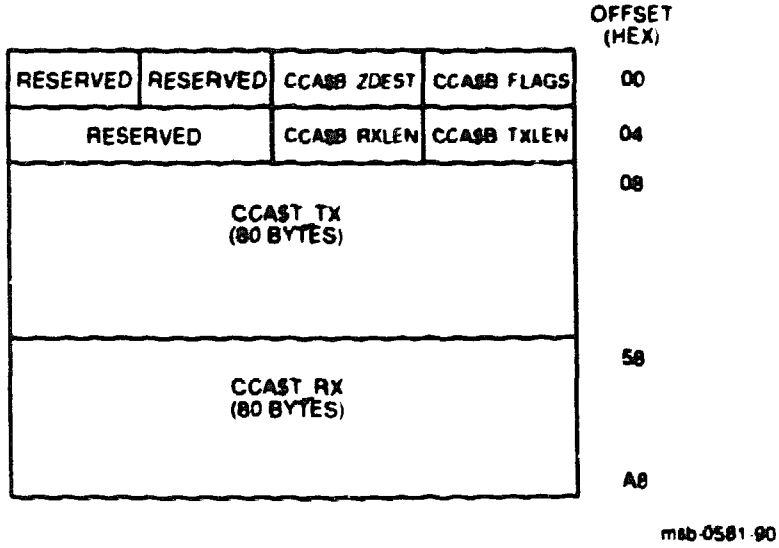
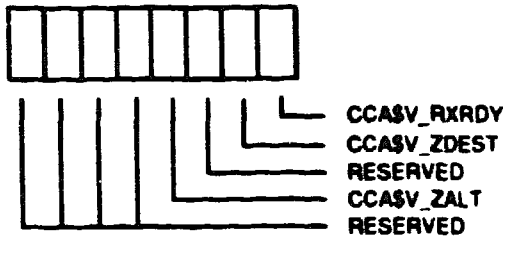


Table 3–12 Buffer Fields

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$V_ZALT	When set, the target of the current "Z" command cannot communicate through the CCA. The target is either a non-processor XMI node or a VAXBI node and must be accessed using alternate RXCD protocol, as described in the *VAXBI System Reference Manual*.
CCA$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_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$T_TX	This buffer is used by the node to transmit a response to the primary processor. Only response data is passed through this buffer since a secondary processor does not send commands to the primary processor.	
CCA$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.	

3.10.3 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 3.

1 Node 1 examines the CCA$V_RXRDY bit in the CCA buffer area for node 3. If the bit is clear, then go to 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 3.

4 Node 1 sets the length of the command into the CCA$B_RXLEN field for node 3.

5 Node 1 sets the CCA$V_RXRDY bit for node 3 to indicate that a command is waiting.

6 Whenever node 3 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 3 will now process the command contained in its CCA$T_RX buffer.

8 After reading the command, node 3 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 3, halts, enters console mode, and sends a "halted" message to the primary processor, located at XMI node 1.

1 Node 3 examines bit 3 of the CCA$Q_READY field. If the bit is clear, then go to Step 3.

2 Node 3 polls this bit until it clears.

3 Node 3 moves the text of its response into its CCA$T_TX buffer.

4 Node 3 sets the length of the response in its CCA$B_TXLEN field.

5 Node 3 sets bit 3 in CCA$Q_READY to indicates that a response is waiting.

6 Node 3 issues an IVINTR interrupt to node 1. If node 1 is running, this alerts the operating system that a response is waiting. Node 3 polls CCA$Q_READY until bit 3 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. If node 1 was in console mode, it would be polling CCA$Q_READY and discover bit 3 set.

8 Node 1 (either the operating system or the console code) processes the response from the CCA$T_TX buffer for node 3. If the console code is running, it displays the response on the console terminal.

9 Node 1 clears bit 3 in CCA$Q_READY, indicating that the buffer is again available.

3.11 KN58A/A Interface Module Error Handling

This section describes the error handling features of the KN58A/A interface module.

The KN58A/A interface 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 are "soft" and are signaled by asserting INT3 to the R3000. KN58A/A interface 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. Any errors that leave the second-level cache incoherent also leave the first-level cache incoherent.

All XMI memory reads are "connected"; the R3000 waits for the data to be returned and, if it cannot be delivered from the XMI, an error flag is returned to the R3000 resulting in a bus error exception. A memory read "hit" in the active XCPGA write buffer causes the write buffer to be purged. If the purge results in an XMI memory write failure, the read is suppressed and a bus error flag is returned to the R3000.

All XMI memory writes are "disconnects." They are acknowledged by the XMI interface and data is placed in the XCPGA write buffer to be written later. If a subsequent write buffer unload or purge results in an XMI write failure, it is signaled to the CPU by posting an INT3 interrupt. These errors are considered hard.

All XMI I/O reads and writes are "connected"; they cause purging of the XCPGA write buffer prior to their initiation on the XMI and they are not acknowledged until all XMI transactions are successfully completed. If the XCPGA write buffer purge results in an XMI memory write failure, the I/O transaction is suppressed and a bus error flag is returned to the R3000. If the XCPGA write buffer purge is successful but the subsequent I/O transaction fails, the R3000 is released with a bus error flag for reads or an INT3 interrupt for writes.

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 an INT3 interrupt, since a bus error flag during an 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).

IIDAL parity errors cause bad data and parity to be stored in second-level cache on cache fills. On writes, the XMI write is suppressed and the data is discarded if a parity error is detected. An INT3 interrupt to the R3000 signals the error.

3.11.1 Parity Generation and Checking for Error Detection

Parity generation and check characteristics of the KN58A/A interface module follows:

- The KN58A/B CPU module generates parity on write data. The KN58A/B CPU module does not generate parity on command/address data.
- The first-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 processor on reads.
- The second-level cache supports parity on the tag bits, valid bits, and data store. On second-level cache fills and writes, parity is stored and then checked by the KN58A/A interface module on reads.
- The XCPGA detects IIDAL parity errors on writes.
- The XMI supports three parity bits covering both data and command information. The KN58A/A interface module generates and checks XMI parity.
- The R3010 FPA 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.
- CSR1 and CSR2 are not parity protected.

3.11.2 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 as described in Section 3.6.2.

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 INT3 interrupt line is edge-sensitive. If this read indicates that no error bits are set, then exit the interrupt service routine; else loop to step 1.

3.11.3 KN58A/A Interface Module Error Matrix

Table 3–13 Second-Level Cache Data Parity Errors

Reference Type	Effect on R3000 Execution	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	Error Indication	Notes
Read	INT3	-	Disabled	-	CSR2<31:30> CSR1<FMISS> set	CDPE
Write	-	-	-	-	-	
Read-Lock	-	-	-	-	-	Always misses 2nd-Level Cache
Unlock Write	-	-	-	-	-	Not Applicable
IDENT	-	-	-	-	-	Always misses 2nd-Level Cache
2nd-Level Cache Fill	-	-	-	-	-	Parity not checked during 2nd-Level Cache fill
1st-Level Cache Fill	INT3	-	Disabled	-	CSR2<31:30> CSR1<FMISS> set	CDPE

Table 3–14 Second-Level Cache Tag/Valid Bit Parity Errors

Reference Type	Effect on R3000 Execution	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	Error Indication	Notes
Read	INT3	-	Disabled	-	CSR2<31:30> CSR1<FMISS> set	TPE/VPE
Write	INT3	-	Disabled	-	CSR2<31:30> CSR1<FMISS> set	TPE/VPE
Read-Lock	-	-	-	-	-	Always misses 2nd-Level Cache
Unlock Write	INT3	-	Disabled	-	CSR2<31:30> CSR1<FMISS> set	TPE/VPE
IDENT	-	-	-	-	-	Always misses 2nd-Level Cache
2nd-Level Cache Fill	-	-	-	-	-	Parity checked on Read
1st-Level Cache Fill	INT3	-	Disabled	-	CSR2<31:30> CSR1<FMISS> set	TPE/VPE

Table 3–15 XMI Bus Timeout Errors

Reference Type	Effect on R3000 Execution	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	Error Indication	Notes
Read	BUS ERR	-	-	-	XBER<NRR> set XBER<TTO> set XFADR<31:0>	16.7ms Timer - NXM Reattempt T/O XMI Failing Adr/Len
Write	INT3	-	-	-	XBER<TTO> set XFADR<31:0>	16.7ms Timer - NXM Reattempt T/O XMI Failing Adr/Len
Read-Lock	BUS ERR	-	-	-	XBER<NRR> set XBER<TTO> set XFADR<31:0>	16.7ms Timer - NXM Reattempt T/O XMI Failing Adr/Len
Unlock Write	INT3	-	-	-	XBER<TTO> set XFADR<31:0>	16.7ms Timer - NXM Reattempt T/O XMI Failing Adr/Len
IDENT	INT3	-	-	-	XBER<NRR> set XBER<TTO> set XFADR<31:0>	16.7ms Timer - NXM Reattempt T/O XMI Failing Adr/Len
2nd Level Cache Fill	INT3	-	Sub-Block Not Validated	-	CSR2<CFE> set XBER<NRR> set XBER<TTO> set XFADR<31:0>	Cache Fill Error 16.7ms Timer - NXM Reattempt T/O XMI Failing Adr/Len
1st-Level Cache Fill	——— No XMI Transaction Generated ———					

Table 3–16 XMI Bus Parity Errors

Reference Type	Effect on R3000 Execution	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	Error Indication	Notes
Read	INT3	-	Sub-Block Not Validated	-	XBER<PE> set XFADR<31:0>	NRR Seq Err could also set XMI Failing Adr/Len
Write	-	-	-	-	-	Parity not checked[1]
Read-Lock	INT3	-	-	-	XBER<PE> set XFADR<31:0>	NRR Seq Err could also set XMI Failing Adr/Len
Unlock Write	-	-	-	-	-	Parity not checked[1]
IDENT	INT3	-	-	-	XBER<PE> set XFADR<31:0>	NRR Seq Err could also set XMI Failing Adr/Len
2nd Level Cache Fill	INT3	-	Sub-Block Not Validated	-	XBER<PE> set XFADR<31:0>	NRR Seq Err could also set XMI Failing Adr/Len
1st-Level Cache Fill	——— No XMI Transaction Generated ———					

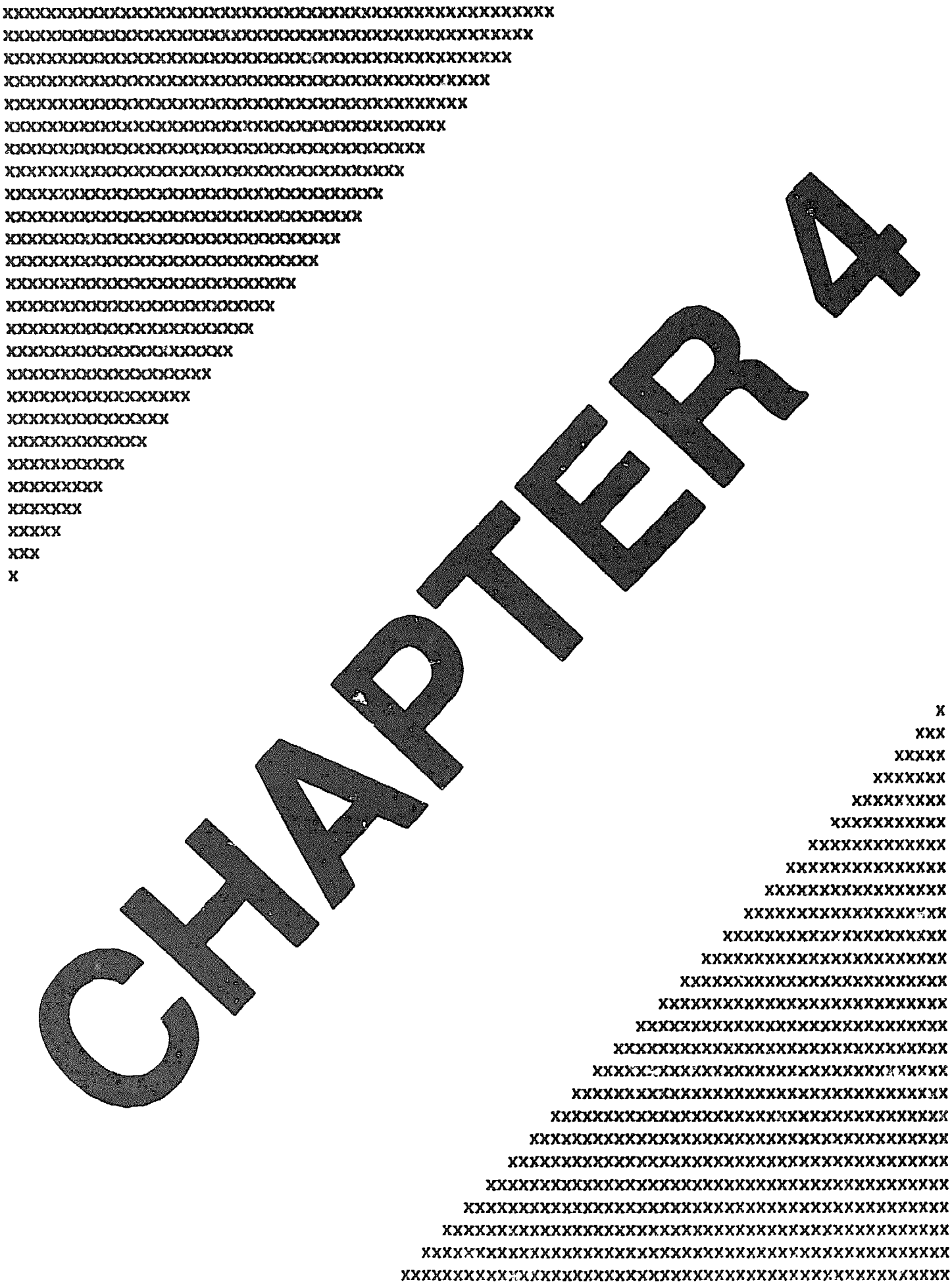
[1] If a bus parity error occurs on a write data cycle, the responder NOACKs all subsequent data cycles and the commander reattempts the transaction.

Table 3–17 Main Memory Correctable Errors

Reference Type	Effect on R3000 Execution	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	Error Indication	Notes
Read	INT3	-	Corrected Data Written	Supplies Corrected Data	XBER<CRD> set	Corrected Read Data
Write	-	-	-	-	-	Not Applicable
Read-Lock	INT3	-	-	Supplies Corrected Data	XBER<CRD> set	Corrected Read Data
Unlock Write	-	-	-	-	-	Not Applicable
IDENT	-	-	-	-	-	Not Applicable
2nd Level Cache Fill	INT3	-	Corrected Data Written	Supplies Corrected Data	XBER<CRD> set	Corrected Read Data
1st-Level Cache Fill	——— No XMI Transaction Generated ———					

Table 3–18 Main Memory Uncorrectable Errors

Reference Type	Effect on R3000 Execution	Effect on 1st-Level Cache	Effect on 2nd-Level Cache	Effect on Main Memory	Error Indication	Notes
Read	BUS ERR	-	-	Supplies Uncorrected Data	XBER<RER> set XBER<RSE> set	Read Error Response Read Sequence Error
Write	-	-	-	-	-	Not Applicable
Read-Lock	BUS ERR	-	-	Supplies Uncorrected Data	XBER<RER> set XBER<RSE> set	Read Error Response Read Sequence Error
Unlock Write	-	-	-	-	-	Not Applicable
IDENT	INT3	-	-	Supplies Uncorrected Data	XBER<RER> set XBER<RSE> set	Read Error Response Read Sequence Error
2nd-Level Cache Fill	INT3	-	-	Supplies Uncorrected Data	XBER<RER> set XBER<RSE> set CSR2<CFE> set	Read Error Response Read Sequence Error Cache Fill Error
1st-Level Cache Fill	——— No XMI Transaction Generated ———					


CHAPTER 4

4 KN58A/B CPU Module

This chapter describes the KN58A/B CPU module, the module that provides the computational power of the KN58A processor. The KN58A/B CPU module is based on the MIPS[1] R3000: a 32-bit, virtual memory Reduced Instruction Set Computer (RISC) microprocessor.

This chapter includes the following sections:

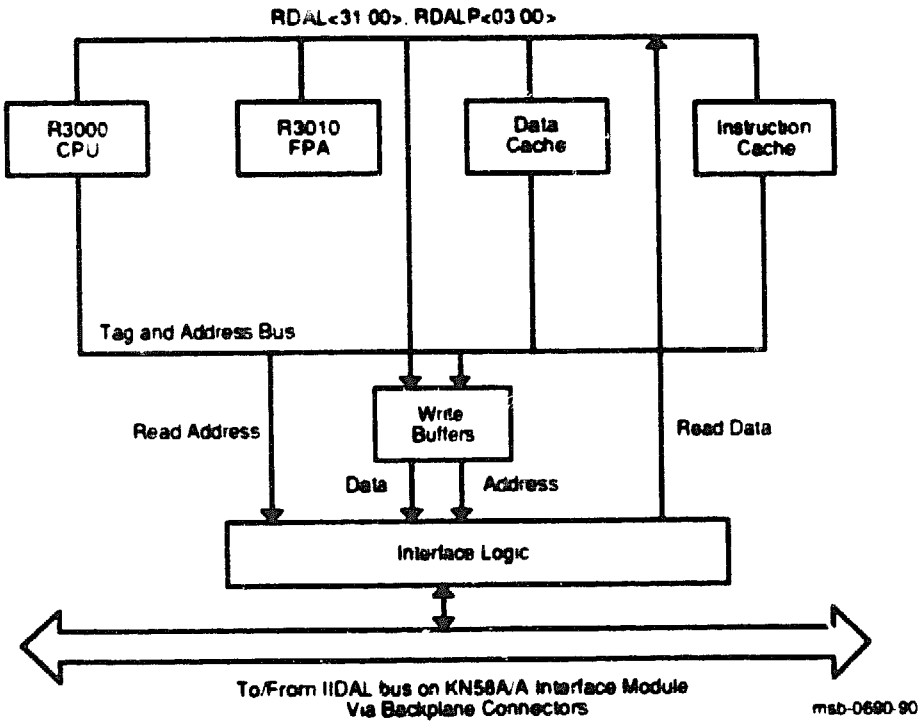
- Module Features
- R3000 CPU
- R3010 FPA
- R3020 Write Buffers
- First-Level Cache Memory
- Interface Logic

[1] MIPS is a registered trademark of MIPSCO, Inc

4.1 KN58A/B CPU Module Features

The KN58A/B CPU module is based on the 32-bit, virtual memory MIPS R3000 RISC microprocessor and its R3010 floating-point coprocessor. Up to four KN58A/B CPU modules can be installed in a DECsystem 5800, each of which must be accompanied by a KN58A/A interface module.

Figure 4–1 KN58A/B CPU Module Block Diagram



The KN58A/B CPU module includes the following:

- **The MIPS R3000 CPU chip,** which features:
  - Full 32-bit operation. The R3000 contains thirty-two 32-bit registers. All instructions and addresses are 32 bits in length.
  - Efficient pipelining. The R3000 5-stage pipeline helps achieve an execution rate of close to one instruction per cycle. Pipeline stalls and exceptional events are handled efficiently.
  - On-chip cache control. The R3000 contains a high bandwidth cache memory interface that handles transactions with the external Instruction and Data caches. Both Instruction and Data caches are accessed during a single CPU cycle.
  - On-chip memory management. The R3000 contains a memory management unit (MMU) that supports a 4-Gbyte virtual address space. The MMU has a fully associative 64-entry Translation Lookaside Buffer (TLB) designed for multitasking operating system environments and provides fast virtual-to-physical address translation.
  - Coprocessor interface. The R3000 generates all addresses and handles memory interface control for the R3010 FPA coprocessor.
- **The MIPS R3010 FPA coprocessor,** which adds floating-point instructions to the CPU's base instruction set. It has sixteen 64-bit registers dedicated to floating-point operations and conforms to the IEEE 754-1985 standard. It supports single- (32-bit) and double- (64-bit) precision floating-point operations.
- **A first-level cache,** which consists of a 64-Kbyte instruction cache and a 64-Kbyte data cache, implemented as an array of 28 16-Kbyte x 4 data RAMs. Both caches are direct-mapped. The data cache is "allocate on write." Both caches are filled in increments of 8-word blocks and have as a line size one 32-bit word. Both caches contain tag and parity information. An invalidate FIFO for the data cache aids in the maintenance of first-level cache coherency.
- **Write buffers,** which enhance the performance of the R3000 CPU chip by allowing the chip to perform write operations during CPU run cycles. There is one write buffer implemented in four R3020 write buffer chips. This provides four-deep buffering of 32 bits of address and 36 bits of data and parity.
- **Interface logic** (including a read buffer for data from the module), which provides the means by which the KN58A/B CPU module communicates with the KN58A/A interface module. The interface logic provides the KN58A/B CPU module access to the secondary cache, the system support chip (SSC), and the XMI corner on the KN58A/A interface module.

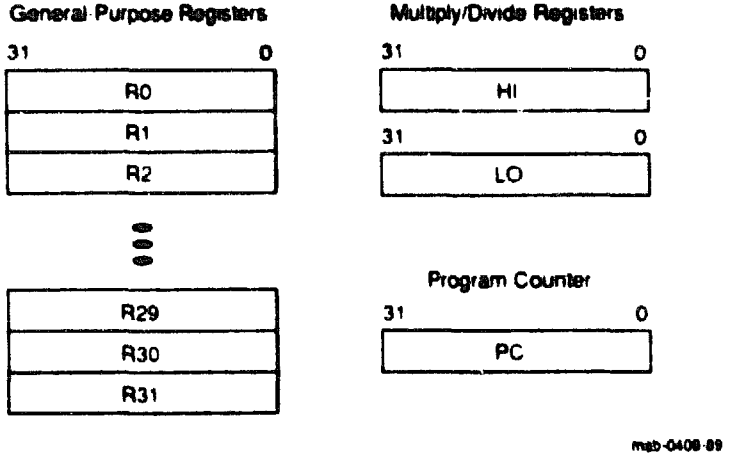
Each of these major portions of the KN58A/B CPU module is explained in the sections that follow.

4.2 R3000 CPU

The KN58A/B CPU module is based on the R3000 CPU chip. The module implements the entire R3000 instruction set, data types, and memory management. These are outlined in the sections that follow. Detailed information can be found in *MIPS R2000 RISC Architecture*.

4.2.1 R3000 Registers

Figure 4–2 R3000 Registers



As shown in Figure 4–2, the R3000 registers consist of 32 32-bit general-purpose registers, a 32-bit program counter, and two 32-bit registers that hold the results of multiply and divide operations. The contents of these registers define the state of the CPU.

The functions traditionally associated with a processor status word (PSW) are provided by the Status Register and Cause Register. These registers are part of Coprocessor 0 and are explained in Section 4.2.2.

4.2.2 Coprocessor 0 (CP0) Registers

Coprocessor 0 (CP0) is contained on the R3000 chip. It is tightly coupled with the R3000 and performs memory management and exception handling. Virtual memory is implemented with a Translation Lookaside Buffer and the group of programmable registers listed in Table 4–1 and detailed in the pages that follow.

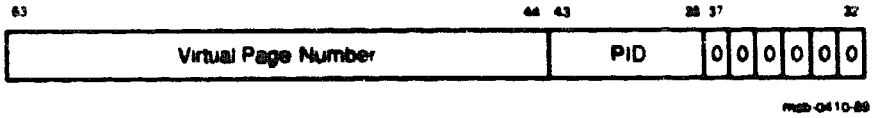
Table 4–1 Coprocessor 0 Registers

Register	Description
EntryHi	High half of a TLB entry
EntryLo	Low half of a TLB entry
Index	Programmable pointer into the TLB array
Random	Pseudo-random pointer into the TLB array
Status	Mode, interrupt enables, and diagnostic status information
Cause	Indicates nature of last exception
EPC	Exception Program Counter
Context	Pointer into kernel's virtual Page Table Entry array
BadVAddr	Most recent bad virtual address
PRId	Processor Revision Identifier Register

TLB EntryHi Register (EntryHi)

The EntryHi and EntryLo registers provide the data pathway through which the TLB is accessed. When address translation exceptions occur, these registers are loaded with relevant information about the address that caused the exception. The registers are also used to build a new TLB entry using the tlbwi and tlbwr instructions. The format of an EntryHi/EntryLo register pair is the same as that of a TLB entry. EntryHi contains the upper half of a TLB entry.

ADDRESS *EntryHi (R3000)*



bits<63:44>

Name Virtual Page Number
Mnemonic VPN
Type R/W

This field contains bits <31:12> of virtual address.

bits<43:38>

Name Process ID
Mnemonic PID
Type R/W

This is an 8-bit field that lets multiple processes share the TLB while each process has a distinct mapping of otherwise identical virtual page numbers.

bits<37:32>

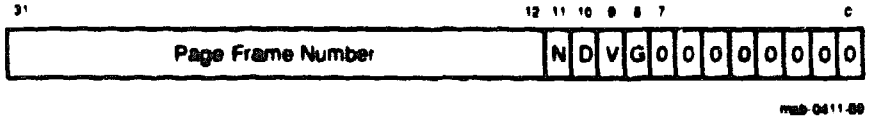
Name Reserved
Mnemonic None
Type R/W, 0

Reserved; must be zero.

TLB EntryLo Register (EntryLo)

The EntryHi and EntryLo registers provide the data pathway through which the TLB is accessed. When address translation exceptions occur, these registers are loaded with relevant information about the address that caused the exception. The registers are also used to build a new TLB entry using the tlbwi and tlbwr instructions. The format of an EntryHi/EntryLo register pair is the same as that of a TLB entry. EntryLo contains the lower half of a TLB entry.

ADDRESS *EntryLo (R3000)*



mab-0411-89

bits<31:12>

Name: Page Frame Number
Mnemonic: PFN
Type: R/W

This field contains bits <31:12> of the physical address. The R3000 maps a virtual page to the PFN.

bit<11>

Name: Non-cachable
Mnemonic: N
Type: R/W

If this bit is set, the page is marked as non-cachable and the R3000 directly accesses main memory instead of first accessing cache.

bit<10>

Name: Dirty
Mnemonic: D
Type: R/W

If this bit is set, the page is marked as writable. This is a "write protect" bit that software can use to prevent alteration of data. If a write is attempted on an entry with the D bit cleared, the R3000 causes a TLB Mod trap and the TLB entry is not modified.

bit<9>

Name: Valid
Mnemonic: V
Type: R/W

If this bit is set, the TLB entry is valid. Otherwise, a TLBL or TLBS Miss occurs.

bit<8>

Name: Global
Mnemonic: G
Type: R/W

If this bit is set, the R3000 ignores the PID match requirement for valid translation. In kseg2, the Global bit lets the kernel access all mapped data without requiring it to save or restore Process ID values.

bits<7:0>

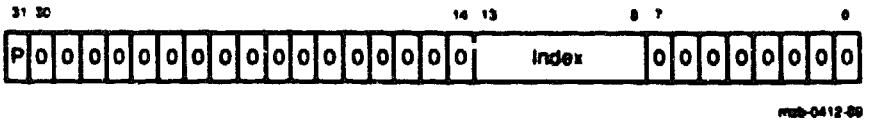
Name: Reserved
Mnemonic: None
Type R/W, 0

Reserved; must be zero.

TLB Index Register (Index)

The TLB Index Register is a 32-bit, read/write register. It contains 6 bits that index an entry in the TLB. The high-order bit of the register shows the success or failure of a TLB Probe (tlbp) instruction. The register also specifies the TLB entry that will be affected by the TLB Read (tlbr) and TLB Write Index (tlbwi) instructions.

ADDRESS *Index (R3000)*



bit<31>

Name: Probe failure
Mnemonic: P
Type: R/W

Set to one if the last TLB Probe instruction was unsuccessful.

bits<30:14>

Name: Reserved
Mnemonic: None
Type: R/W

Reserved; must be zero.

bits<13:8>

Name: Index
Mnemonic: Index
Type: R/W

Index to the TLB entry that will be affected by the TLB Read and TLB Write instructions.

bits<7:0>

Name: Reserved
Mnemonic: None
Type: R/W

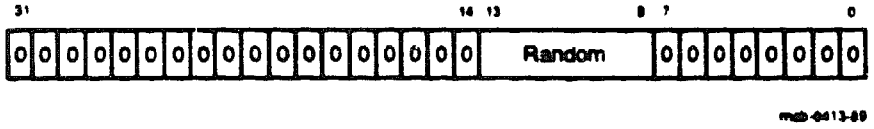
Reserved; must be zero.

TLB Random Register (Random)

The TLB Random Register contains a 6-bit Random field that indexes a random entry in the TLB. The value of the field ranges from 8 to 63. The field is initialized to 63 when the R3000 is reset and is decremented every machine cycle. The value of the field wraps back around to 63 when 8 is decremented.

ADDRESS

Random (R3000)



bits<31:14>

Name: Reserved
Mnemonic: None
Type: R/W

Reserved; must be zero.

bits<13:8>

Name: Random
Mnemonic: Random
Type: R/W

A random index (with a value ranging from 8 to 63) to a TLB entry.

bits<7:0>

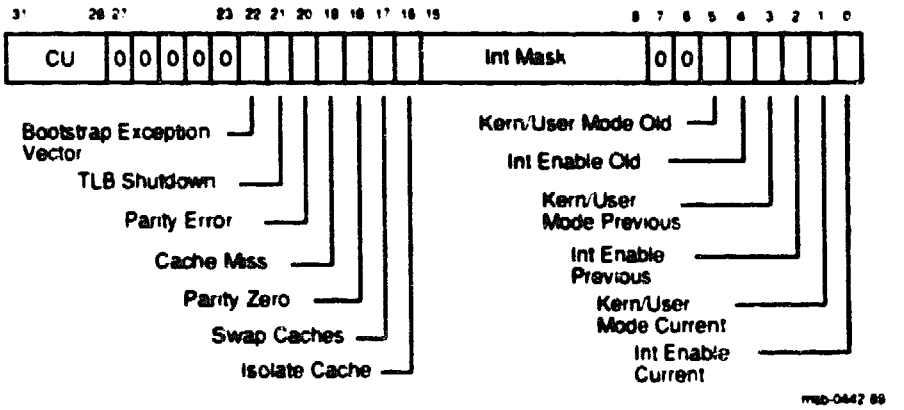
Name: Reserved
Mnemonic: None
Type: R/W

Reserved; must be zero.

R3000 Status Register (Status)

The R3000 Status Register contains all major status bits for the R3000.

ADDRESS *Status (R3000)*



bits<31:28>

Name Coprocessor Usability
Mnemonic CU
Type R/W

These bits control the usability of the four possible coprocessors. Each bit corresponds to a coprocessor and when set indicates that a coprocessor is usable. For example, if CU<0> is set, Coprocessor 0 is usable.

bits<27:23>

Name Reserved
Mnemonic None
Type R/W

Reserved; must be zero.

bit<22>

Name Bootstrap Exception Vector
Mnemonic BEV
Type R/W

If set to one, causes the R3000 to use alternate, bootstrap vectors for ULTB Miss and general exceptions.

bit<21>

Name: TLB Shutdown
Mnemonic: TS
Type: RO

Set to one if the R3000 has disabled TLB due to catastrophic error. Cleared only by R3000 reset.

bit<20>

Name: Parity Error
Mnemonic: PE
Type: R/W

Set to one if a parity error occurs. Reset by writing a one to this bit.

bit<19>

Name: Cache Miss
Mnemonic: CM
Type: R/W

Set to one if the most recent D-Cache load resulted in a miss (only when the D-Cache is isolated).

bit<18>

Name: Parity Zero
Mnemonic: PZ
Type: R/W

When set to one, causes zero to replace normal outgoing parity bits.

bit<17>

Name: Swap Caches
Mnemonic: SwC
Type: R/W

Controls swapping of I-Cache and D-Cache usage. When set to one, the the I-cache is swapped with the D-cache.

bit<16>

Name: Isolate Cache
Mnemonic: IsC
Type: R/W

When set to one, isolates D-Cache from main memory system.

bits<15:8>

Name Interrupt Mask
Mnemonic IntMask
Type R/W

When a bit is set to one, the corresponding hardware or software interrupt is enabled. Bits <15:10> correspond to hardware interrupts 5 through 0, and bits <9:8> correspond to software interrupts 1 and 0.

bits<7:6>

Name Reserved
Mnemonic None
Type R/W

Reserved; must be zero.

bit<5>

Name Kern/User Mode Old
Mnemonic KUo
Type R/W

Set to zero if kernel, one if user.

bit<4>

Name Int Enable Old
Mnemonic IEo
Type R/W

Set to one to enable, zero to disable.

bit<3>

Name Kern/User Mode Previous
Mnemonic KUp
Type R/W

Set to zero if kernel, one if user.

bit<2>

Name Int Enable Previous
Mnemonic IEp
Type R/W

Set to one to enable, zero to disable.

bit<1>

Name: Kern/User Mode Current
Mnemonic: KUc
Type: R/W

Set to zero if kernel, one if user.

bit<0>

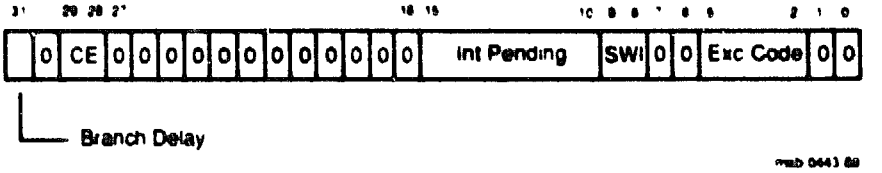
Name: Int Enable Current
Mnemonic: IEc
Type: R/W

Set to one to enable, zero to disable.

Cause Register (Cause)

The Cause Register contains information about the last exception

ADDRESS *Cause (R3000)*



bit<31>

Name Branch Delay
Mnemonic BD
Type RO

Set to one if the last exception was taken while executing in a branch delay slot.

bit<30>

Name Reserved
Mnemonic None
Type RO

Reserved; must be zero.

bit<29:28>

Name Coprocessor Error
Mnemonic CE
Type RO

Indicates the unit number referenced when a Coprocessor Unusable exception is taken.

bits<27:16>

Name Reserved
Mnemonic None
Type RO

Reserved; must be zero.

bit<15:10>

Name: Interrupts Pending
Mnemonic: IP
Type: RO

Indicates the external interrupts that are pending. Bits <15:10> correspond to interrupts <5:0>.

bits<9:8>

Name: Software Interrupts
Mnemonic: Sw
Type: R/W

Indicates which of the two software interrupts is pending. This field is used to set or reset software interrupts.

bits<7:6>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<5:2>

Name: Exception Code
Mnemonic: ExcCode
Type: RO

Exception code as described in the following table:

Number	Mnemonic	Description
0	Int	External Interrupt
1	MOD	TLB modification exception
2	TLBL	TLB miss exception (Load or instruction fetch)
3	TLBS	TLB miss exception (Store)
4	AdEL	Address error exception (Load or instruction fetch)
5	AdES	Address error exception (Store)
6	IBE	Bus error exception (for an instruction fetch)
7	DBE	Bus error exception (for a data load or store)
8	Sys	Syscall exception
9	Bp	Breakpoint exception
10	RI	Reserved Instruction exception
11	CpU	Coprocessor Unusable exception
12	Ovf	Arithmetic Overflow exception
13–15	RSVD	Reserved

bits<1:0>

Name Reserved
Mnemonic None
Type RO

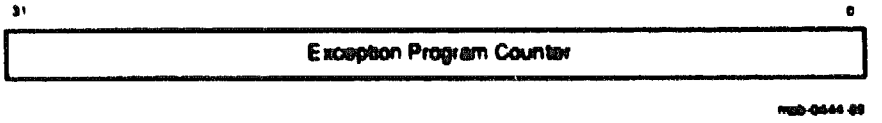
Reserved, must be zero.

Exception Program Counter Register (EPC)

The Exception Program Counter Register contains the address where processing can resume after an exception has been serviced.

ADDRESS

EPC (R3000)



bits<31:0>

Name: Exception Program Counter Register
Mnemonic: EPC
Type: RO

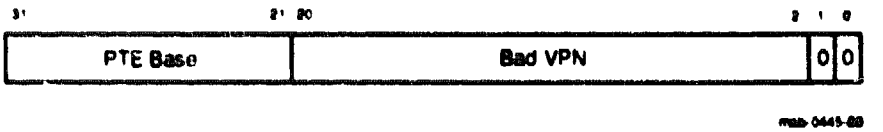
The Exception Program Counter Register contains the virtual address of the instruction that caused the last exception. When that instruction resides in a branch delay slot, the register contains the virtual address of the immediately preceding Branch or Jump instruction. The R3000 also sets the Cause Register's BD bit if the exception occurred in the branch delay slot.

Context Register (Context)

The Context Register duplicates some of the information provided in the BadVAddr Register, but provides the information in a form that may be more useful for a software TLB exception handler. It is designed for use in a UTLB miss handler, which loads TLB entries for normal user-mode references.

ADDRESS

Context (R3000)



bits<31:21>

Name Page Table Entry Base
Mnemonic PTEBase
Type R/W

Holds the base for the Page Table Entry (set by software).

bits<20:2>

Name Bad Virtual Page Number
Mnemonic BadVPN
Type RO

Holds the failing Virtual Page Number (set by hardware). Contains bits <30:12> of the BadVAddr register.

bits<1:0>

Name Reserved
Mnemonic None
Type RO

Reserved, must be zero.

Bad Virtual Address Register (BadVAddr)

The Bad Virtual Address Register contains the entire bad virtual address for any address exception: AdEL or AdEs.

ADDRESS *BadVAddr (R3000)*

bits<31:0>

Name: Bad Virtual Address Register
Mnemonic: BadVAddr
Type: RO

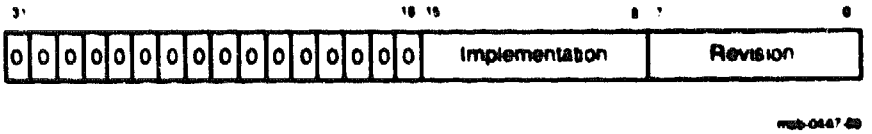
Note that this register does not save any information for bus errors since these are not addressing errors.

Processor Revision Identifier Register (PRId)

The Process Revision Identifier Register contains data that specifies the implementation and revision level of the R3000

ADDRESS

PRId (R3000)



mab-0447-89

bits<31:16>

Name Reserved
Mnemonic None
Type RO

Reserved, must be zero

bits<15:8>

Name Implementation Identifier
Mnemonic Imp
Type RO

Identifies the implementation number of the R3000

bits<7:0>

Name Revision Identifier
Mnemonic Rev
Type RO

Identifies the revision level of the R3000.

4.2.3 R3000 Pipeline Architecture

The R3000 instruction pipeline consists of five stages:.

1 **IF** - Instruction Fetch. Fetch the instruction. The R3000 calculates the instruction address required to read an instruction from the I-cache.

2 **RD** - Read any required operands from the R3000 registers while decoding the instruction.

3 **ALU** - Perform the required operation on instruction operands.

4 **MEM** - Access memory (D-Cache).

5 **WB** - Write back results to register file.

Each of these stages requires approximately one cycle. When the pipeline is full, the R3000 can execute instructions at a rate of approximately one instruction per cycle. The pipeline operates efficiently because different CPU resources (address and data bus accesses, ALU operations, register accesses, and so on) are utilized simultaneously without interfering with one another.

4.2.4 Data Types

The R3000 defines a 32-bit word, a 16-bit half word, and an 8-bit byte. Byte ordering on the KN58A is compatible with VAX architecture and is called *little endian*, which means that byte 0 is the least significant and rightmost byte.

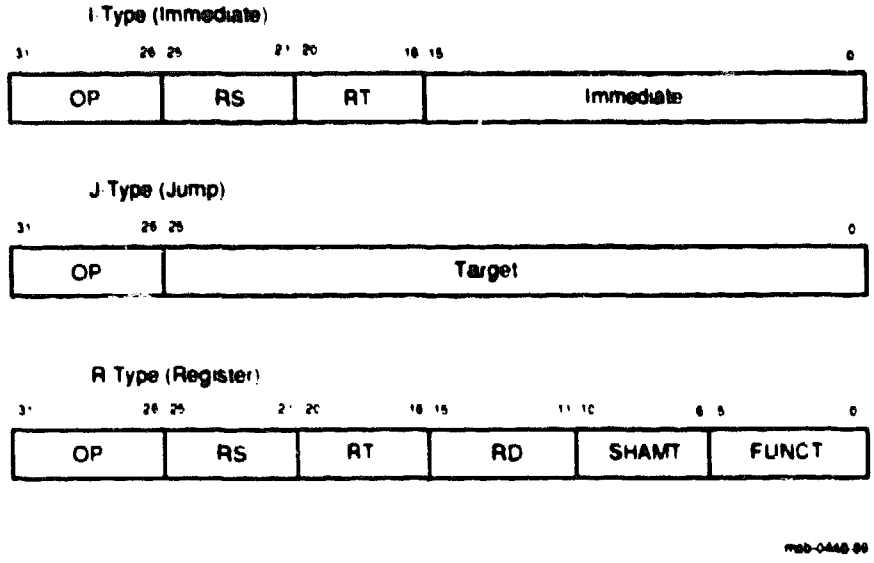
The R3000 addresses aligned bytes for half word and word accesses; half word accesses must be aligned on an even byte boundary, and word accesses must be aligned on a byte boundary divisible by 4.

Special instructions are provided for addressing words not aligned on a 4-byte (word) boundary. These instructions (load word left, load word right, store word left, and store word right) when paired appropriately allow unaligned words to be accessed in one cycle more than would be required for an aligned word.

4.2.5 Instruction Set

Every R3000 instruction consists of a single word (32 bits) aligned on a word boundary. For simplicity in instruction decoding, there are only three instruction formats: I-Type, J-Type, and R-Type. These are illustrated in Figure 4–3.

Figure 4–3 Instruction Formats



Where:

OP	is a 6-bit operation code
RS	is a 5-bit source register specifier
RT	is a 5-bit target (source/destination) register or branch condition
Immediate	is a 16-bit immediate data, branch displacement, or address displacement
Target	is a 26-bit jump target address
RD	is a 5-bit destination register specifier
SHAMT	is a 5-bit shift amount
FUNCT	is a 6-bit function field

4.2.5.1 Load and Store Instructions

Load and store instructions move data between memory and general registers. They are all I-type instructions, since the only addressing mode supported is *base register plus 16-bit signed immediate offset*.

Most load operations have a latency of one instruction. That is, the instruction immediately following a load usually cannot use the contents of the loaded register. An exception is that the target register for the load word left and load word right instructions may be specified as the same register as the destination of a load instruction that immediately precedes it.

The load or store instruction opcode specifies the access type, which also specifies the size of the data item to be loaded or stored. The address specifies the least significant byte. The bytes within the addressed word that are used are determined by the access type and the two low-order bits of the address, as shown in Table 4–2. Note that certain combinations of access type and low-order address bits never occur (word/01/10/11, triple-byte/10/11, and halfword/01/11).

Table 4–2 Byte Specifications for Load and Store Instructions

Access Type	Low-Order Address Bits	Bytes Accessed Little Endian Format
1 1 (word)	0 0	3,2,1,0
1 0 (triple-byte)	0 0	2,1,0
1 0 (triple-byte)	0 1	3,2,1
0 1 (halfword)	0 0	1,0
0 1 (halfword)	1 0	3,2
0 0 (byte)	0 0	0
0 0 (byte)	0 1	1
0 0 (byte)	1 0	2
0 0 (byte)	1 1	3

4.2.5.2 Computational Instructions

Computational instructions perform arithmetic, logical, and shift operations on values in registers. They occur in both R-type (both operands are registers) and I-type (one operand is a 16-bit immediate) formats. There are four categories of computational instructions:

- **ALU Immediate** instructions such as ADD Immediate and OR Immediate.
- **3-Operand Register-Type** instructions such as Add, Subtract, and NOR.
- **Shift** instructions such as Shift Left Logical and Shift Right Logical.
- **Multiply/Divide** instructions such as Multiply Unsigned, Divide, and Move from LO.

4.2.5.3 Jump and Branch Instructions

Jump and branch instructions change the control flow of a program. Jumps are always to absolute 26-bit word addresses (J-type format) or 32-bit register addresses (R-type). Branches have 16-bit offsets relative to the program counter (I-type). Jump and Link instructions save a return address in Register 31.

All jump and branch instructions have a delay of one instruction. That is, the instruction immediately following a jump or branch executes while the target instruction is fetched from storage.

Since instructions must be word aligned, a jump register or jump and link register instruction must use a register whose two low-order bits are zero. If these low-order bits are not zero, an address exception will occur when the target instruction is subsequently fetched.

4.2.5.4 Coprocessor Instructions

Coprocessor instructions perform operations in the coprocessors. Coprocessor loads and stores are I-type. Coprocessor computational instructions have coprocessor-dependent formats.

Because of the protected status of coprocessor 0, the move to/from coprocessor instructions are the only valid mechanism for reading from and writing to the CP0 registers (see Section 4.2.2). The coprocessor instructions allow coprocessor 0 to directly read, write, and probe TLB entries and to modify the operating modes in preparation for returning to user-mode or interrupt-enabled states.

4.2.5.5 Special Instructions

There are only two special instructions, system call and breakpoint. These instructions allow software to initiate traps. They are always R-type.

4.2.6 Memory Management

The R3000 has an addressing range of 4 Gbytes. However, since most systems implement a physical memory smaller than 4 Gbytes, the R3000 translates addresses composed in a large virtual address space into available physical memory addresses. Virtual memory is always enabled, and the 4 Gbyte virtual address space is divided into 2 Gbytes for users and 2 Gbytes for the kernel. The physical memory space available on the XMI to the KN58A is 256 Mbytes for memory and 512 Mbytes for I/O.

4.2.6.1 Translation Lookaside Buffer

The Translation Lookaside Buffer (TLB) reduces the overhead associated with translating virtual addresses to physical addresses. The on-chip TLB allows very fast virtual memory access to meet the requirements of multitasking operating systems. The fully associative TLB contains 64 entries, each of which maps a 4-Kbyte page, with controls for read/write access, cachability, and process identification. The TLB allows each user to access up to 2 Gbytes of virtual address space.

4.2.6.2 R3000 Operating Modes

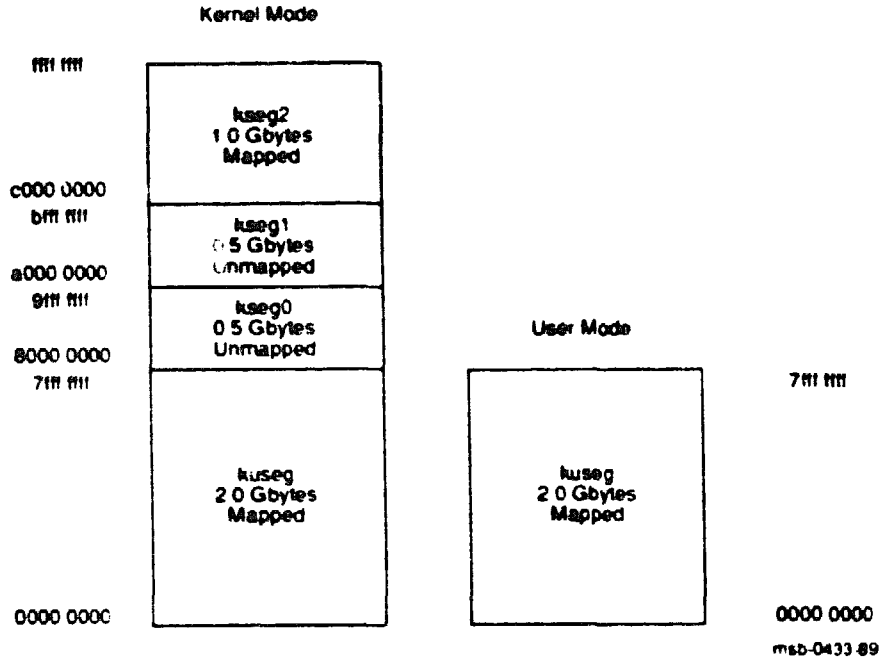
The R3000 has two operating modes: user and kernel. The R3000 normally operates in user mode until an exception forces it into kernel mode. The R3000 remains in kernel mode until a Restore From Exception (RFE) instruction is executed. The manner in which memory addresses are mapped depends on the operating mode of the R3000. Figure 4–4 illustrates the virtual address space for the two modes.

As shown in Figure 4–4, user mode defines a single uniform virtual address space (kuseg) of 2 Gbytes. Each virtual address is extended by a 6-bit process identifier to form unique virtual addresses for up to 64 user processes. All references are mapped through the TLB. Use of first-level cache is determined by bit settings for each page within the TLB entries.

As shown in Figure 4–4, kernel mode defines four virtual address segments: kuseg, kseg0, kseg1, and kseg2.

- **kuseg** - Kernel mode references to kuseg are identical to those in user mode.
- **kseg0** - References to this 512-Mbyte segment use cache memory but are not mapped through the TLB. Instead, they always map to the first .5 Gbytes of physical memory.
- **kseg1** - References to this 512-Mbyte segment do not use cache memory and are not mapped through the TLB. Instead, they always map to the first .5 Gbytes of physical memory.
- **kseg2** - References to this 1-Gbyte segment are always mapped through the TLB and use of the cache is determined by the bit settings of the TLB entry.

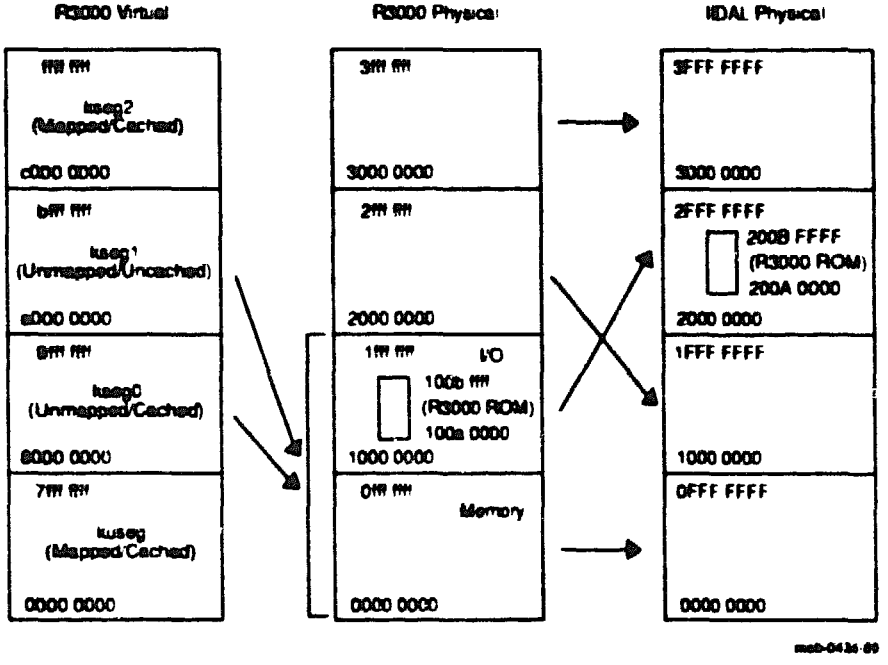
Figure 4–4 Virtual Memory for Kernel and User Modes



4.2.7 Memory Mapping

Memory mapping is necessary for the R3000 CPU to communicate with VAX-based hardware on the IIDAL. Figure 4–5 illustrates the relationship between R3000 virtual addresses, R3000 physical addresses, and IIDAL (VAX) physical addresses.

Figure 4–5 R3000 Memory Mapping



The R3000 does not have the same concept of I/O space as a VAX-based CPU, but since the R3000 must operate on a VAX bus, the design of the KN58A/B CPU module implements a 256-Mbyte I/O space separate from the memory space. Physical address bits 28 and 29 are swapped between the KN58A/A interface module and KN58A/B CPU module. This effectively divides the R3000 physical address range into 256 Mbytes of I/O space and 256 Mbytes of memory space and allows kseg0 and kseg1 to map the first 256 Mbytes of XMI I/O space without the use of translation buffers.

kseg0 and kseg1 both map to the first 512 Mbytes of R3000 physical memory. kseg0 maps through the cache, and kseg1 maps around the cache. When these address ranges are accessed, the R3000 clears physical address bits 31, 30, and 29 and uses the remaining address as the physical

address. kuseg and kseg2 require that translation buffer entries be valid for the R3000.

4.2.7.1 R3000 Boot PROM Mapping

The R3000 boot PROM is located in kseg1 (unmapped, uncached space) at R3000 virtual address b00a 0000 through b00b ffff. When the RESET line is deasserted, the first address that the R3000 accesses is 1fc0 0000. Hardware translates that address directly to 200A 0000, locating the PROM space on the XMI from 200A 0000 to 200B FFFF.

4.2.7.2 I/O Mapping Example: Reading a Register Associated with I/O Adapter 7

All XMI node spaces and XMI I/O adapters 0 through 3 exist in XMI physical address space 2000 0000 through 2FFF FFFF. This range is typically accessed directly via kseg1 using R3000 virtual addresses 0xb000 0000 through 0xbfff ffff. I/O adapters 4 through 7 exist at XMI physical address 3600 0000 through 3DFF FFFF. The TLB must be used to map R3000 virtual addresses to XMI addresses in this range. kseg2 is used to map these addresses in system space.

Example 4–1 maps the VAXBI base address of the DWMBA/B module at XMI node E BI node 1, then reads the DWMBA/B Bus Error Register, and then unmaps the address.

Example 4–1 I/O Mapping

```
li      t0, 0x800       # use 8th TLB entry
mtc0    t0, INDEX       # load index register
li      t0, 0x3c002f00  # load DTYPE reg physical address + NDVG bits
mtc0    t0, ENTRYLO     # load LO half of PTE
li      t0, 0xfc002000  # map DTYPE reg virtual address via kseg2
mtc0    t0, ENTRYHI     # load HI half of PTE
nop
tlbwi                   # write PTE into TLB
nop

lw      v0, 8(t0)       # read BER register of DWMBA/B
nop

mtc0    zero, ENTRYLO   # invalidate PTE (clearing V bit)
nop
tlbwi                   # write invalid PTE to TLB
```

4.2.8 Interrupts

The R3000 has six external hardware interrupt levels and two software interrupt levels. Table 4–3 summarizes the types and levels of hardware interrupts.

Table 4–3 R3000 External Hardware Interrupts

Level	Condition
5	R3010 floating-point interrupt
4	CTRL/P generated by console
	Node Halt bit (XBER<29>) written
3	KN58A Hard Errors:
	XMI AC LO (CSR1<6>)
	IIDAL Write Data Parity Error (CSR2<28>)
	XMI Write Error IVINTR (XBER<25>)
	XMI FAULT (XBER<26>)
	XMI Write Errors (XBER<20>, XBER<15>)
	XMI IDENT Errors (XBER<18:15>)
	KN58A Soft Errors:
	Invalidate FIFO Full (CSR1<26>)
	Second-Level Cache Parity Errors (CSR2<31:30>) (disables second-level cache)
	Invalidate Queue Overflow (CSR2<29>) (disables second-level cache)
	Second-Level Cache Fill Error (CSR2<27>)
	Duplicate Tag Parity Error (CSR2<26>) (disables second-level cache)
	XMI Corrected Confirmation (XBER<27>) (interrupt can be disabled)
	XMI Inconsistent Parity Error (XBER<24>) (disables second-level cache)
	XMI Parity Error (XBER<23>)
	XMI Corrected Read Data (XBER<19>) (interrupt can be disabled)
2	XMI level 7 interrupt transaction
1	XMI interprocessor IVINTR
	XMI level 6 interrupt transaction
	Interval timer interrupt
0	XMI level 5 interrupt transaction
	Console terminal interrupt

Table 4–3 (Cont.) R3000 External Hardware Interrupts

Level	Condition
	Programmable timer interrupt
	XMI level 4 interrupt transaction

When one of the external hardware interrupts from Table 4–3 is recognized by the R3000, the R3000 branches to the general exception vector (0x8000 0080). The R3000 sets bits <5:2> of the Cause Register to zero and sets bits <15:10> with the level of the interrupt. Software obtains the appropriate interrupt vector by reading the addresses shown in Table 4–4.

Table 4–4 Interrupt Acknowledge Vectors

R3000 Interrupt Line	XMI Interrupt Level	IIDAL Interrupt Level	Address
0	XMI 4	0	4000 0050
0	XMI 5	1	4000 0054
1	IVINTR XMI 6	2	4000 0058
2	XMI 7	3	4000 005C

No vector is supplied for R3000 interrupt levels 3, 4, or 5. These interrupts are used as shown in Table 4–5.

Table 4–5 R3000 Interrupt Levels 3, 4, and 5

R3000 Interrupt Level	Purpose
3	Corrected read data
	First-level invalidate FIFO overflow
	Memory error
	Power fail
4	Halt
5	Floating-point unit

4.2.9 Exceptions

When the R3000 detects an exception, the normal sequence of instruction execution is suspended. The processor exits user mode and is forced into kernel mode where it can respond to the abnormal or asynchronous event. Events that initiate exception processing are described in detail in the *R3000 RISC Architecture* manual. In summary, they are:

- **Reset** - Assertion and deassertion of the R3000's RESET signal causes an exception that transfers control to the vector at virtual address 0xbfc0 0000.
- **UTLB miss** - User TLB miss. A reference is made (in either user or kernel mode) to a page in kuseg that has no matching TLB entry.
- **TLB miss** - A referenced TLB entry's Valid bit is not set, or there is a reference to a kseg2 page that has no matching TLB entry.
- **TLB modified** - During a store instruction, the Valid bit is set but the Dirty bit is not set.
- **Bus error** - Assertion of the R3000's BUS ERR signal, which occurs as a result of a nonexistent memory read. IIERR on the IIDAL bus is asserted by the SSC and in response to that, BUS ERR is asserted to the R3000.
- **Address error** - Attempt to load, fetch, or store an unaligned word; that is, a word or halfword at an address not evenly divisible by 4 or 2 respectively. Also caused by reference to a virtual address with most significant bit set while in user mode.
- **Overflow** - Two's complement overflow during add or subtract.
- **System call** - Execution of the SYSCALL instruction.
- **Breakpoint** - Execution of the BREAK instruction.
- **Reserved instruction** - Execution of an instruction with an undefined or reserved major operation code (bits <31:26>) or a special instruction whose minor opcode (bits <5:0>) is undefined.
- **Coprocessor unusable** - Execution of a coprocessor instruction when the CU (Coprocessor Unusable) bit is not set for the target coprocessor.
- **Interrupt** - Assertion of one of the R3000's six hardware interrupt inputs or setting of one of the two software interrupt bits in the Cause Register.

4.3 R3010 FPA

The R3010 floating-point accelerator (FPA) operates in conjunction with the R3000 CPU and extends the R3000's instruction set to perform arithmetic operations on values in floating-point representations. The R3010 FPA fully conforms to the requirements of ANSI/IEEE Standard 754-1985, "IEEE Standard for Binary Floating-Point Arithmetic."

Features of the FPA include:

- **Full 64-bit operation.** The FPA provides 16 64-bit registers that can be used to hold single-precision or double-precision values. The FPA also includes a 32-bit control/status register that provides access to all IEEE-Standard exception handling capabilities.
- **Load/store instruction set.** Like the R3000 processor, the FPA uses load/store instructions with single-cycle loads and stores. Floating-point operations are started in a single cycle and overlapped with other fixed-point or floating-point operations.
- **Tightly coupled coprocessor interface.** The FPA connects with the R3000 to form a tightly coupled unit with a seamless integration of floating-point and fixed-point operations. Since the FPA can receive and execute instructions in parallel, some floating-point instructions can execute at the same single-cycle rate as integer instructions.

4.3.1 FPA Registers

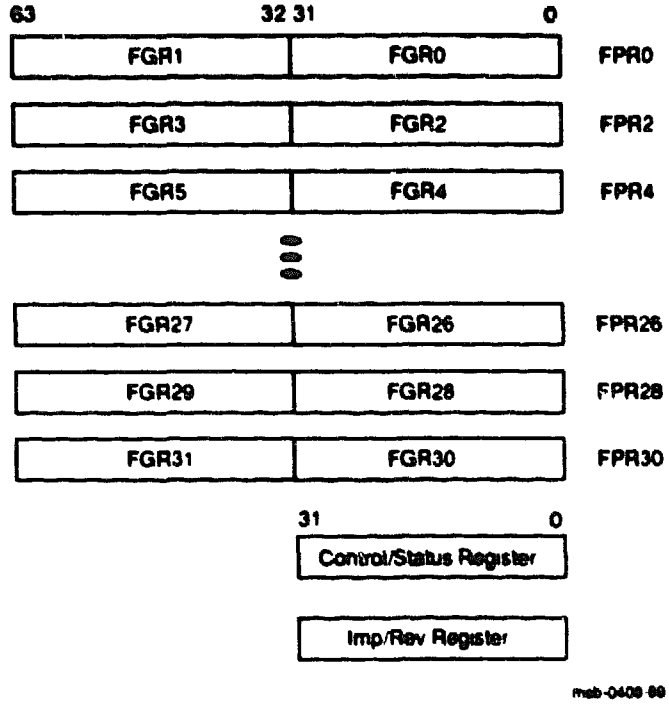
As shown in Figure 4–6, the FPA has 32 general-purpose 32-bit registers, a control/status register, and an implementation/revision register.

Floating-point coprocessor operations (that is, operations involving Coprocessor 1) reference three types of registers:

- Floating-point general registers (FGRs)
- Floating-point registers (FPRs)
- Floating-point control registers (FCRs)

The sections that follow provide a brief description of these types of registers.

Figure 4–6 FPA General-Purpose Registers



4.3.1.1 Floating-Point General Registers (FGRs)

The FPA contains 32 32-bit floating-point general registers (FGRs). They are directly addressable and are accessed by load, store, or move operations.

4.3.1.2 Floating-Point Registers (FPRs)

As shown in Figure 4–6, the 32 FGRs are logically configured as 16 64-bit floating-point registers (FPRs). FGR1 and FGR0, for example, are the upper and lower halves (respectively) of FPR0.

Only even-numbered addresses are used to access FPRs: odd-numbered addresses are invalid. The FPRs contain data in either single- or double-precision floating-point format. During single-precision operations only the even-numbered FGRs are used. Double-precision operations access FGRs in pairs.

4.3.1.3 Floating-Point Control Registers (FCRs)

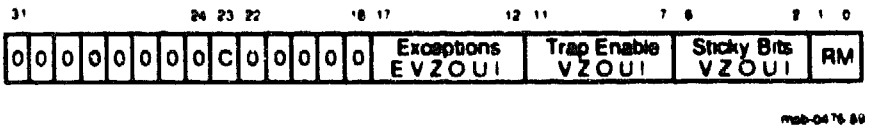
The FPA has two floating-point control registers (FCRs), which are accessed only by move operations. These are the control/status register (FCR31) and the implementation/revision register (FCR0).

FPA Control/Status Register (FCR31)

The Control/Status Register is used to control and monitor exceptions, operating modes, and rounding modes. It is written with a Move Control to Coprocessor 1 (CTC1) instruction.

ADDRESS

FCR31 (R3010)



bits<31:24>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<23>

Name: Condition
Mnemonic: C
Type: R/W

Set/cleared to reflect the result of a Compare instruction and drives the FPA's CpCond output signal.

bits<22:18>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<17:12>

Name: Exceptions
Mnemonics: E, V, Z, O, U, I
Type: R/W

These bits are set to indicate any exceptions that occurred during the most recent instruction. E indicates an unimplemented operation, V indicates an invalid operation, Z indicates division by zero, O indicates overflow, U indicates underflow, and I indicates an inexact operation.

bits<11:7>

Name: Trap Enable

Mnemonics: V, Z, O, U, I

Type: R/W

These bits enable assertion of the FPA's CpInt signal if the corresponding Exception bit is set during a floating-point operation. V is for an invalid operation, Z is for division by zero, O is for overflow, U is for underflow, and I is for an inexact operation.

bits<6:2>

Name: Sticky bits

Mnemonics: V, Z, O, U, I

Type: R/W

These bits are set if an exception occurs and are reset only by explicitly loading new settings into this register with a Move instruction. V is for an invalid operation, Z is for division by zero, O is for overflow, U is for underflow, and I is for an inexact operation.

bits<2:0>

Name: Rounding Mode

Mnemonic: RM

Type: R/W

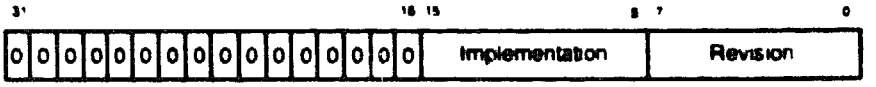
Specify which of the four rounding modes is to be used by the FPA.

FPA Implementation/Revision Register (FCR0)

The Implementation/Revision Register is used by diagnostic software to determine the FPA revision level.

ADDRESS

FCR0 (R3010)



mab-044' 89

bits<31:16>

Name Reserved
Mnemonic None
Type RO

Reserved; must be zero

bits<15:8>

Name Implementation Identifier
Mnemonic Imp
Type RO

Identifies the implementation number of the R3010.

bits<7:0>

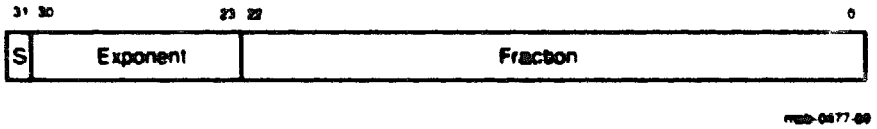
Name Revision Identifier
Mnemonic Rev
Type RO

Identifies the revision level of the R3010.

4.3.2 FPA Formats

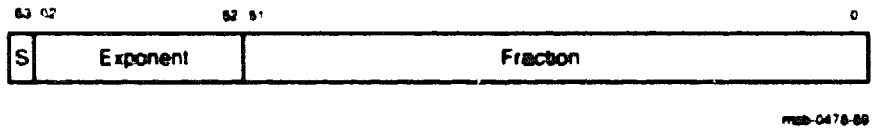
The R3010 FPA performs both 32-bit (single-precision) and 64-bit (double-precision) IEEE standard floating-point operations. The 32-bit format has a 24-bit signed-magnitude fraction field and an 8-bit exponent as shown in Figure 4–7.

Figure 4–7 Single-Precision Floating-Point Format



The 64-bit format has a 53-bit signed-magnitude fraction field and an 11-bit exponent as shown in Figure 4–8.

Figure 4–8 Double-Precision Floating-Point Format



4.3.3 Coprocessor Operation

As an R3000 coprocessor, the FPA continually monitors the R3000's instruction stream, ignoring non-FPA instructions. When it detects an FPA instruction, the FPA executes it and transfers the results and necessary exception data synchronously to the R3000. The FPA can perform the following types of operations:

- Load and store operations
- Moves
- Two and three register floating-point operations

4.3.3.1 Load, Store, and Move Operations

Load, store, and move operations move data between memory or the R3000 registers and the FPA registers. These operations perform no format conversions and cause no floating-point exceptions. Load, store, and move operations reference a single 32-bit word of either the FGRs or the FCRs.

4.3.3.2 Floating-Point Operations

The FPA supports the following single- and double-precision format floating-point operations:

- Add
- Subtract
- Multiply
- Divide
- Absolute Value
- Move
- Negate
- Compare

In addition, the FPA supports conversion between single- and double-precision floating-point formats and fixed-point formats.

4.3.3.3 Exceptions

The FPA supports all five IEEE standard exceptions:

- Invalid Operation
- Inexact Operation
- Division by Zero
- Overflow
- Underflow

4.3.4 Instruction Set Overview

The R3010 FPA instructions are 32 bits long and can be divided into the following groups:

- **Load, store, and move** instructions move data between memory, the R3000, and the FPA general registers.
- **Computational** instructions perform arithmetic operations on floating-point values in the FPA registers.
- **Conversion** instructions perform conversion operations between the various data formats.

- **Compare** instructions perform comparisons of the contents of registers and set a condition bit based on the results.

4.3.5 R3010 Pipeline Architecture

The R3010 FPA provides an instruction pipeline that parallels that of the R3000 processor. The FPA, however, has a 6-stage pipeline instead of the 5-stage pipeline of the R3000: the additional FPA pipe stage is used to provide efficient coordination of exception responses between the FPA and the R3000. The pipeline for the FPA has the following stages:

- **IF** - Instruction Fetch. The R3000 calculates the instruction address required to read an instruction from the I-cache. No action is required of the FPA during this stage since the R3000 is responsible for address generation.
- **RD** - The instruction is present on the data bus during phase 1 of this stage, and the FPA decodes the data on the bus to determine if it is an instruction for the FPA.
- **ALU** - If the instruction is an FPA instruction, instruction execution begins during this stage.
- **MEM** - If this is a coprocessor load or store instruction, the FPA presents or captures the data during phase 2 of this stage.
- **WB** - The FPA uses this stage solely to deal with exceptions.
- **FWB** - The FPA uses this stage to write back arithmetic results to its register file. This stage is the equivalent of the WB stage in the R3000 processor.

4.4 R3020 Write Buffers

All R3000 write references are simultaneously written into the first-level cache (see Section 4.5) and the R3020 write buffers. The paragraphs that follow describe the operation of the write buffers.

The KN58A/B CPU module has four R3020 write buffer chips that buffer R3000 write references before they appear on the IIDAL bus. The write buffers enhance R3000 performance because they allow the R3000 to perform write operations during run cycles, which would otherwise stall the pipeline. Each write buffer handles an 8-bit slice of address and an 8-bit slice of data As a unit, the four buffers allow four-deep buffering of 32 bits of address and 32 bits of data and parity.

When the R3000 performs a write operation, the write buffers capture the output data and its address (including the bits that indicate the access type). The write buffers can hold up to four such data/address sets (or pairs) while waiting to drive the IIDAL bus. Transfers from the R3000 to the write buffers occur synchronously at a cycle rate of 25 MHz, and the write buffers stall the R3000 if they are unable to accept write data. The write buffers communicate asynchronously with the IIDAL control logic to coordinate the transfer of write data over the IIDAL bus

4.4.1 Write Buffer Flush

The write buffers are flushed by the IIDAL control logic under any of the following conditions:

- I/O Read - An R3000 read reference with address bit <28> set. This corresponds to an IIDAL reference with address bit <29> set.
- Interrupt Acknowledge Transaction - An R3000 read reference with address bit <30> set.
- Read Lock Reference - See Section 4.6.5.
- Main Memory Read Reference

In general, the CPU gives write operations priority over read operations. Refer to Section 4.6 for more information on the IIDAL control logic.

4.4.2 Write Buffer Byte Gathering

The write buffers perform byte, half-byte, tri-byte, and word gathering to decrease the number of write transfers to the same longword location. Byte gathering allows bytes or half-words to be collected and written to main memory. Without byte gathering, the write buffers present address/data sets individually to the IIDAL control logic in the sequence in which they were received from the R3000.

During byte gathering, sequential writes to the same longword address have their data gathered into the same address/data set in the write buffers. Writes to the same byte location are overwritten in the write buffers.

Gathering does not occur for the address/data set that is currently available to the IIDAL control logic. The first write into empty write buffers will not have subsequent writes gathered. Subsequent writes are placed in the next available buffer. Also, byte gathering does not occur for nonsequential writes to the same address.

One result of the gathering scheme is that in some cases two IIDAL bus write references are required to empty a single write buffer entry. For example, if bytes 0 and 3 of a word are sequentially written, two references are required to empty the write buffer entry.

Another result of the gathering scheme is that where order is important in writing (such as for I/O controllers), software should avoid sequential accesses to the same word. In cases where write-read access order is important but the reading of the written location is not desired (such as in I/O), a write followed by an I/O read to a dummy location will ensure that the first write has occurred before continuing.

The byte gathering mechanism is transparent and inaccessible to the user. Its state at any particular time is not readily determined. Software should not rely on the operation of the byte gathering mechanism.

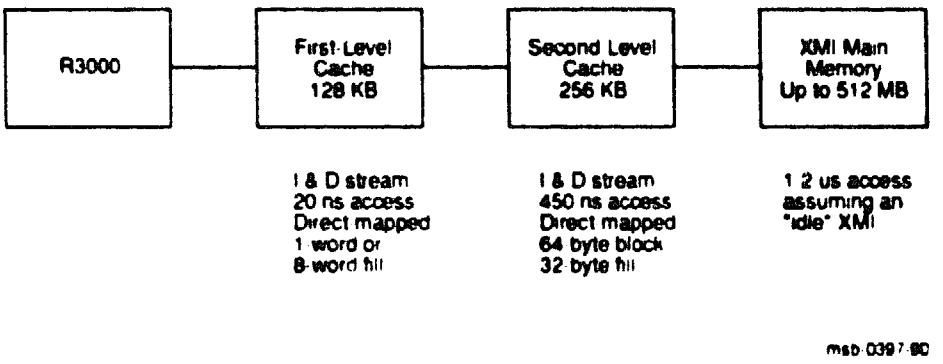
4.4.3 Write Buffer Parity

The KN58A/B CPU module contains logic that generates parity for the output of the R3020 write buffers before the output is driven onto the IIDAL bus.

4.5 First-Level Cache Memory

As shown in Figure 4–9, the R3000 interfaces directly with a 128-Kbyte first-level cache. The cache is direct-mapped and write-through with a 20 ns cycle time. The first-level cache is organized as a 64-Kbyte instruction cache (I-cache) and a 64-Kbyte data cache (D-cache). First-level D-cache coherency is maintained by hardware. First-level I-cache coherency must be maintained by software.

Figure 4–9 Cache Organization



4.5.1 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 must be initialized properly, as explained in Section 4.5.3.

The R3000 generates IIDAL references, depending on the reference type, as follows:

- Whenever the R3000 generates a non-FL cachable reference, a single longword reference of the same type is generated on the IIDAL bus.
- Whenever the R3000 generates an FL cachable read reference that is already stored in the first-level cache, no reference is generated on the IIDAL bus.

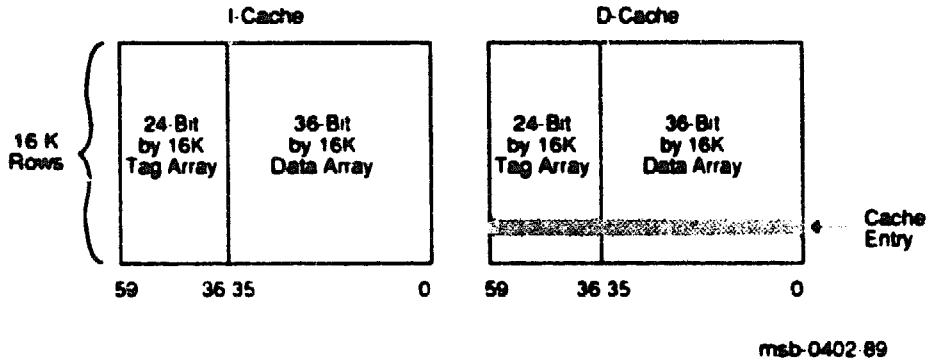
- Whenever the R3000 generates an FL cachable write reference, a write reference is generated for both the FL cache and the IIDAL bus (see Section 4.5.7).

All R3000 writes first pass through the R3020 write buffers (see Section 4.4) before they appear on the IIDAL bus.

4.5.2 First-Level Cache Organization

The first-level cache is divided into two independent storage arrays called the I-cache and the D-cache (see Figure 4–10). Each one contains a 64K-row x 24-bit tag array and a 64K-row 36-bit data array.

Figure 4–10 First-Level Cache Organization



A row within the I-cache or D-cache corresponds to a cache entry. Each cache entry contains a 24-bit tag portion (including valid and parity bits) and a 36-bit data portion (including parity bits). There are 16K entries in the I-cache and 16K entries in the D-cache. Figure 4–11 shows the format of a cache entry. Table 4–6 describes the component parts of an entry.

Figure 4–11 Cache Entry

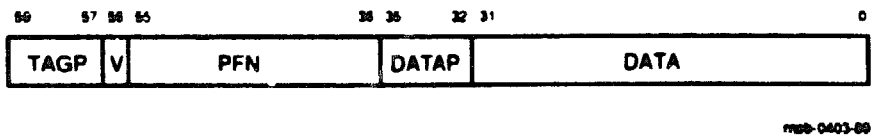


Table 4–6 Cache Entry Fields

Field	Description
TAGP	Tag parity. Parity over the V and PFN fields. TAGP0 (bit <57>) contains parity over the low byte of the PFN (bits <43:36>). TAGP1 (bit <58>) contains parity over the next lowest byte of the PFN (bits <51:44>). TAGP2 (bit <59>) contains parity over the upper bits of the PFN (bits <55:52>) and the V bit (bit <56>).
V	Valid bit. When set, indicates a valid cache entry.
PFN	Page frame number. Specifies the page frame number.
DATAP	Data parity. Parity over the DATA field. There is one parity bit for each byte of the DATA field. DATAP0 (bit <32>) contains parity over bits <7:0>, DATAP1 contains parity over bits <15:8>, DATAP2 contains parity over bits <23:16>, and DATAP3 contains parity over bits <31:24>.

4.5.3 Initializing the First-Level Cache

When the first-level cache is first powered up, the Valid bit (bit <56>) of each cache entry has an unpredictable value. The first-level cache must therefore be completely initialized or invalidated by the operating system upon power-up. D-cache and I-cache are initialized differently.

D-cache is initialized by isolating and flushing it. Isolation is accomplished by setting the ISC bit (bit <16>) in the Status Register and the RINVAL bit (bit <16>) in CSR1. Software must then read the RINVAL bit to ensure that it is set. All writes then hit the cache. Software flushes D-cache by performing partial word stores, each of which clears the Valid bit of a cache entry. When the flush is complete, software must clear the RINVAL bit.

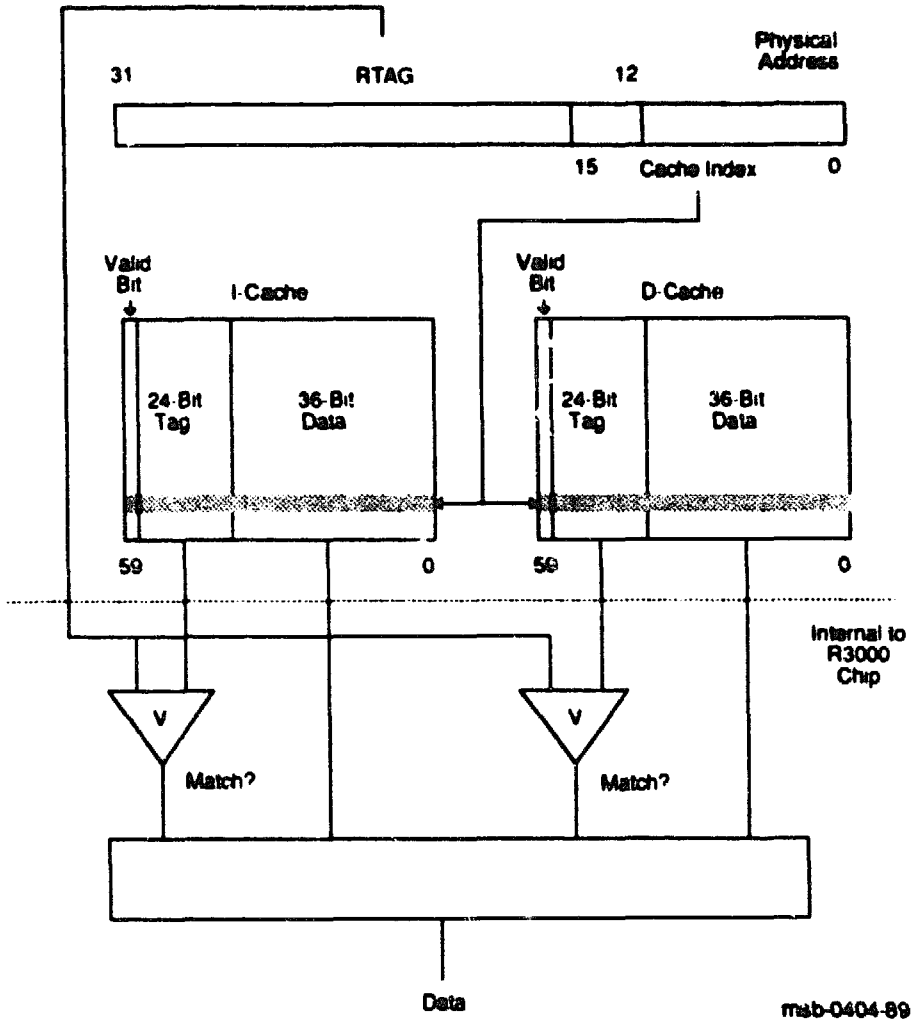
I-cache is initialized by isolating, swapping, and flushing it. Swapping (or exchanging I-cache and D-cache control signals) is required because I-cache cannot ordinarily be written directly. Swapping is accomplished by setting the SWC bit (bit <17>) in the Status Register.

Before initializing I-cache, the R3000 must be executing from uncached space. To initialize the cache, it is isolated by the software as previously described and then swapped. The R3000 must not execute any load/store instructions immediately before the swap operation. Software completes the initialization by flushing the cache with partial word stores, each of which clears the Valid bit of a cache entry.

4.5.4 First-Level Cache Address Translation

Whenever the R3000 requires I-stream or D-stream data, the first-level cache is checked to determine if the referenced location is stored there.

Figure 4–12 Cache Address Translation



The first-level (FL) cache is checked by translating the physical address (see Figure 4–12) 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 IIDAL 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 <15:2> of the address, is used to select one of the 16K rows of the cache, with each row containing a single entry of data and tag. The RTAG field, bits <31:12> of the physical address, is then compared to the PFN of the entry in the selected row.
- If a match occurs with the PFN of the entry, and the Valid bit within the entry is set, and no parity errors are encountered, the contents of the referenced location is contained in the cache and a cache "hit" occurs. No IIDAL bus transfers are initiated on R3000 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. In either case, a single longword transfer is initiated on the IIDAL bus.

4.5.5 First-Level Cache Data Block Allocation

FL cachable references that miss the first-level cache cause a longword read to be initiated on the IIDAL bus. When the requested longword is in the second-level cache, the hexword containing the requested longword is transmitted on the IIDAL to the R3000 and stored in the FL cache.

If the read reference misses the second-level cache, a 16-word block is deallocated in the first-level D-cache and the requested longword is supplied by main memory. The longword is then transmitted on the IIDAL to the R3000 and stored at the addressed location in the previously invalidated block.

4.5.6 First-Level Cache Behavior on Writes

The first-level cache is "allocate on write." All R3000 cached write references are written into the first-level cache regardless of whether they hit or miss the FL cache. Write references are also latched by the R3020 write buffer, which stores the write until it is gated onto the IIDAL bus.

4.5.7 First-Level Cache Coherency

First-level I-cache coherency must be maintained by the operating system software. There is no hardware on the KN58A/B CPU module to implement this function.

First-level D-cache coherency is maintained as a subset of the second-level cache. A read miss in the first-level cache (either I-cache or D-cache) and the second-level cache will force the 16-longword block specified by the tag address to be invalidated in the first-level D-cache.

The XCPGA on the KN58A/A interface module maintains an 8-deep invalidate queue. When there is an entry in the invalidate queue, the XCPGA arbitrates for the IIDAL to send an invalidate address to the second-level cache, invalidating a 16-longword block.

The KN58A/B CPU module contains an invalidate FIFO buffer that stores the invalidate addresses sent to the second-level cache. Each address in the invalidate FIFO causes the invalidation of a 16-longword block in the first-level D-cache. If the invalidate FIFO buffer is full and the XCPGA generates another invalidate, an overflow is indicated in CSR1 and the R3000 is interrupted.

Since the first-level cache is "allocate on write" and the second-level cache is not, writes that miss the second-level cache must be marked invalid in the first-level cache. This is necessary to maintain the first-level cache as a proper subset of the second-level cache. The invalidate FIFOs are used to accomplish this write invalidate. If a write misses the second-level cache, one word in the first-level D-cache is invalidated. If a read misses the second-level cache, 16 words in the first-level D-cache are invalidated.

Device drivers not using an interlock instruction when reading a semaphore must perform a "dummy" read of CSR1 after reading a semaphore and before reading the data. This is required because of the latency involved between an XMI write operation and its associated invalidation of the first-level cache.

4.5.8 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, each of the lower two bytes of the PFN, and the Valid bit plus the upper four bits of the PFN are stored with an associated parity bit. An even parity scheme is used. Tag and data parity (on the entire longword) are checked on read and partial store references that hit the cache.

Upon detection of a parity error, the PE bit (bit <20>) in the Status Register is set. No interrupt is generated. The R3000 transparently recovers from parity errors by taking a cache miss and accessing main memory for a good copy of the cache entry.

4.6 Interface Logic

The interface logic controls communication between the KN58A/B CPU module and the KN58A/A interface module via the IIDAL bus. The interface logic is particularly important in performing control functions (such as lock transactions) that cannot be performed by the R3000 chip itself. This section discusses the IIDAL and the types of IIDAL transactions initiated and coordinated by the interface logic.

4.6.1 The IIDAL Bus

The protocol for the IIDAL bus was derived from that of the CVAX CPBUS. It is nearly identical in its timing From the perspective of the IIDAL, the KN58A/B CPU module appears as a CVAX when it is the IIDAL bus master.

4.6.2 Read Operation

An IIDAL read operation is initiated whenever the R3000 misses the first-level cache or makes an uncached read reference. The IIDAL interface logic can only issue longword reads on the IIDAL bus. At the beginning of the operation, IIDAL<31:30> specifies a longword transaction and CSDP<3:0> indicates an I-stream read request, even if data is being fetched. Read operations typically require two cycles to complete.

A read stall is defined as any additional cycles occurring between the address and data portion of the read operation. This can occur whenever read data is not readily available. IIDAL control logic is stalled by withholding the IIRDY signal. Stalls occur in increments of one cycle.

4.6.3 Write Operation

During an IIDAL write cycle, the R3020 write buffers output write information onto the IIDAL bus. A write operation takes a minimum of 2 cycles and can occur every 4 cycles.

A write stall is defined as any additional cycles occurring between the address and data portion of the write operation. This can occur any time write data cannot be readily accepted by the target device. IIDAL control logic is stalled by withholding the IIRDY signal. Stalls occur in increments of one cycle.

4.6.4 Interrupt Acknowledge Operation

The IIDAL control logic initiates an interrupt acknowledge in response to an R3000 interrupt acknowledge read reference. When the R3000 is interrupted, the interrupt is vectored to a single interrupt handler. The software performs an uncached read reference with address bit <30> set and bit <31> clear. Address bits <6:2> indicate the IPL (in hex). The other address bits are zero. This address is then driven onto the IIDAL bus and CSDP<3:0> indicate an interrupt acknowledge operation. The read data driven onto the bus in response is the desired interrupt acknowledge vector. The software uses the vector to jump to the correct interrupt service routine.

4.6.5 Lock Transactions

The IIDAL control logic initiates a lock transaction in response to an R3000 read lock reference and an R3000 write unlock reference. The transaction is initiated on the IIDAL by asserting the correct code on CSDP<3:0>.

Software generates an R3000 read lock reference by writing the Interlock Address Register (2013 0000) with the address of the interlock variable. The software must then read the Interlock Register (2011 0000). This causes the IIDAL control logic to perform a read lock transaction on the IIDAL bus using the address in the Interlock Address Register. The read data is returned to the R3000.

Software generates a write unlock reference by writing the address of the interlock variable to the Interlock Address Register (2013 0000). The software must then write the Interlock Register (2011 0000). This causes the IIDAL control logic to perform a Write Unlock transaction on the IIDAL bus using the address in the Interlock Address Register. To maintain consistency between the first-level and second-level caches, the unlock write transaction must be preceded by a cached write to the interlock variable.

Note that only the lock transaction occurs on the IIDAL bus. The writing of the Interlock Address Register and any reads or writes of the Interlock Register will not appear on the IIDAL bus.

4.6.6 DMA on the IIDAL Bus

The KN58A/B CPU module contains DMA logic that controls the granting of the IIDAL bus. There are three possible masters of the IIDAL bus:

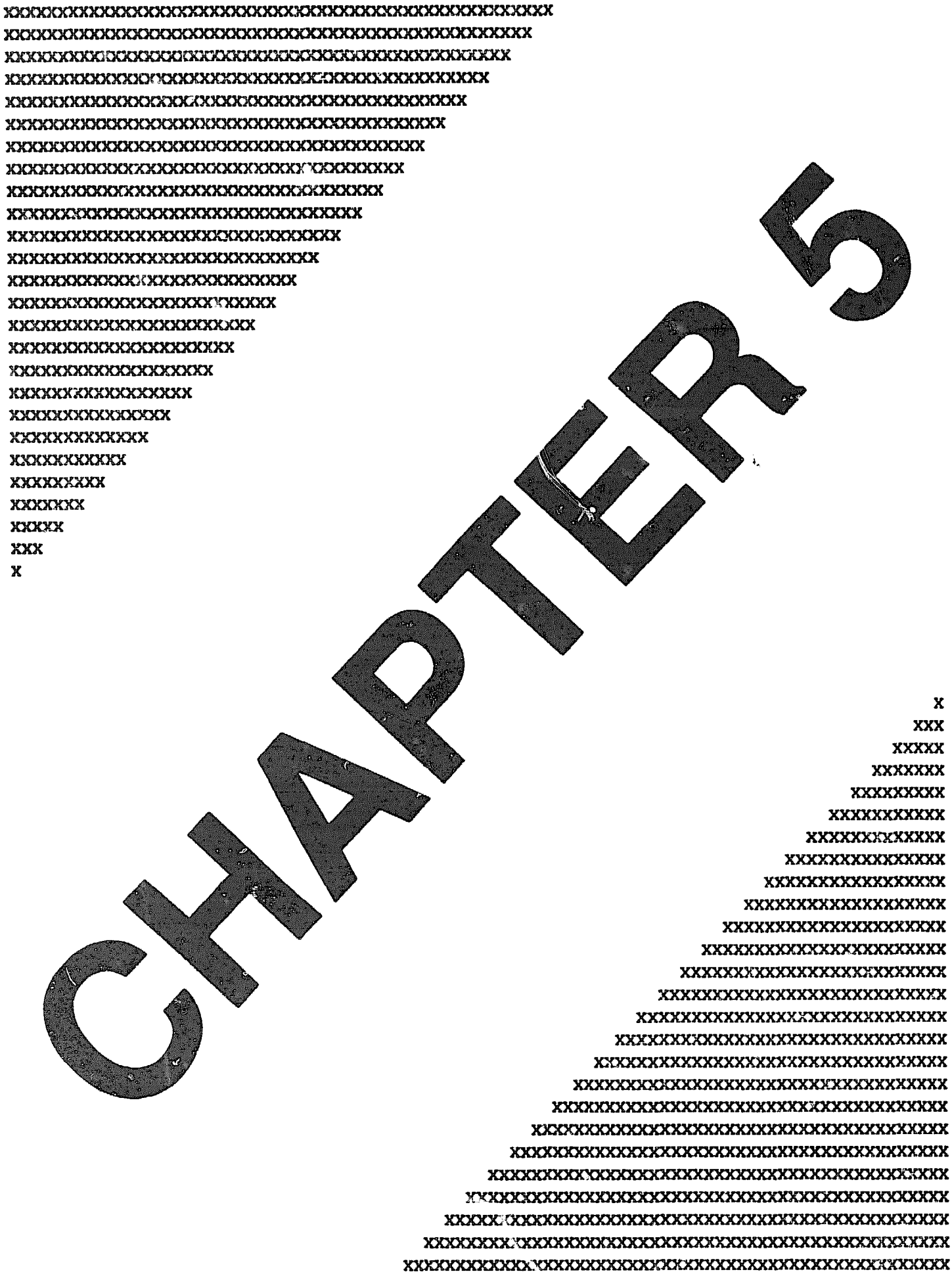
- The CVAX chip (on the KN58A/A interface module)
- The IIDAL control logic

- The XCPGA (on the KN58A/A interface module)

The CVAX or XCPGA is bus master only at power-up or after reset. At power-up or after reset, bus ownership defaults to the CVAX and the XCPGA must request the bus to gain ownership. Setting bit <24> of CSR1 allows the IIDAL control logic to gain ownership of the bus. When this bit is set, bus ownership defaults to the IIDAL control logic and the XCPGA must request the bus to gain ownership.

4.6.7 Idle

The IIDAL bus is idle when no activity is taking place on it. During an idle, the IIDAL lines are undefined and the control signals are deasserted.


CHAPTER 5

5 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 DECsystem 5800 system and communicates over the XMI bus.

This chapter contains the following sections:

- Module Features
- Technical Description
- Self-Test and Initialization
- Starting Address and Interleaving
- Control and Status Registers
- Error Handling and Command Responses

5.1 Module Features

The MS62A memory module is a dynamic random access memory (DRAM) that communicates through the XMI bus to provide DECsystem 5800 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.

5.2 Technical Description

The MS62A memory module uses XMA logic, DRAM arrays, and a PROM to provide 32-Mbytes of memory to the DECsystem 5800 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 KN58A processor.

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 +5V. If power to the system is lost, memory is lost as well.

5.3 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 loses 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 of the *DECsystem 5800 Owner's Manual*.

5.4 Starting Address and Interleaving

On power-up the DECsystem 5800 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 5.5 gives a description of the SEADR.

5.4.1 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.

5.4.2 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.

5.5 Control and Status Registers

The CSR names and their relative addresses are shown in Table 5–1. Descriptions of the CSRs are also included in this section.

Table 5–1 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.

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.

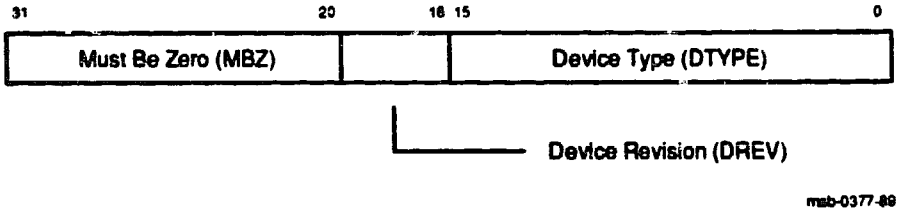
R/W1C, 1 – Indicates a read and write register, set on power-up.

W/O, 0 – Indicates a write only register, cleared on power-up.

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>

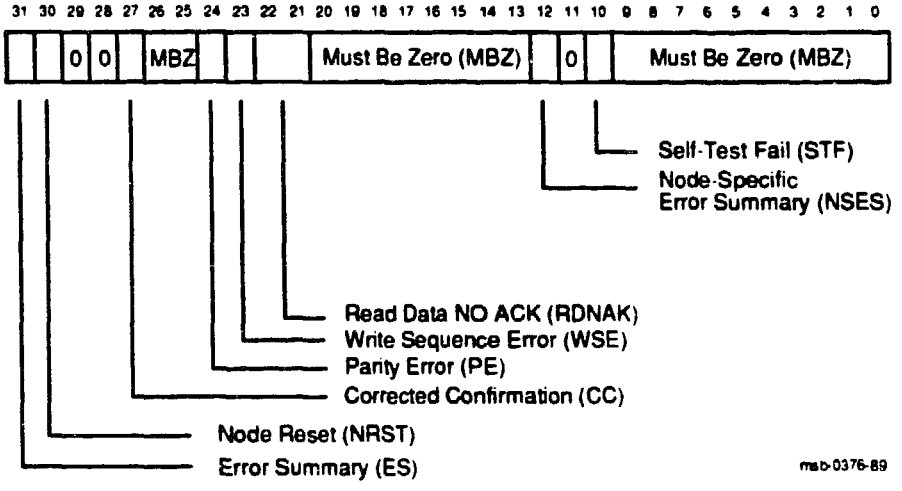
Name: Device Type
Mnemonic: DTYPE
Type: RO

Identifies the type of node. The device type for an MS62A memory module is 4001 (hex). This value is set in the Device Register.

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>

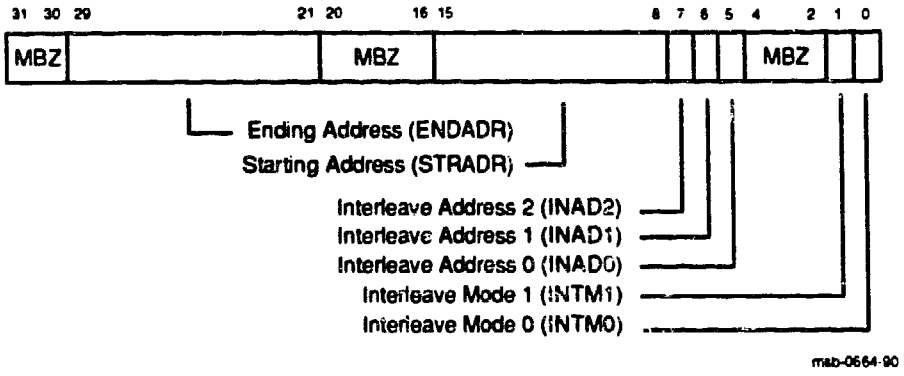
Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

Starting and Ending Address Register (SEADR)

The Starting and Ending Address Register contains the memory starting and ending addresses. See Section 5.4.1 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

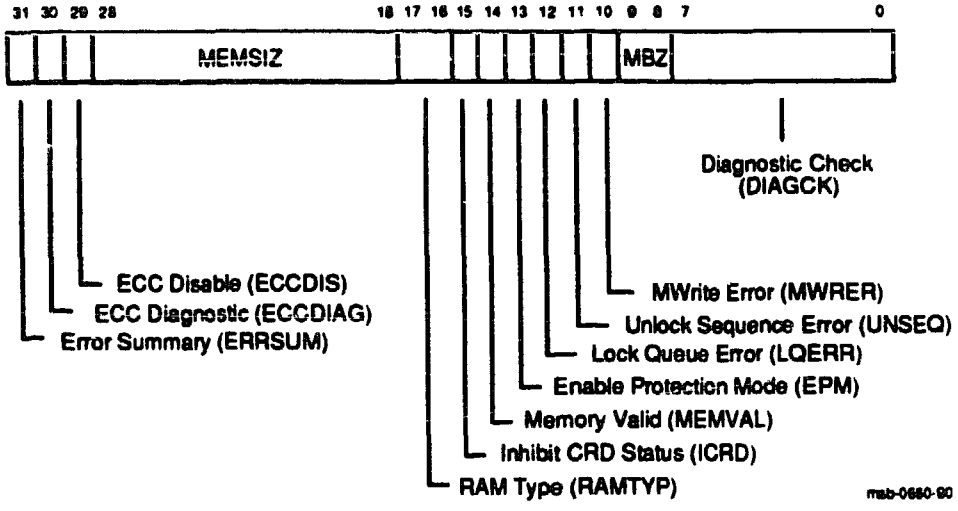
These bits show how many ways the module is being interleaved and are used to determine the addresses that the module will respond to.

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 ECC Error Register (MECER)

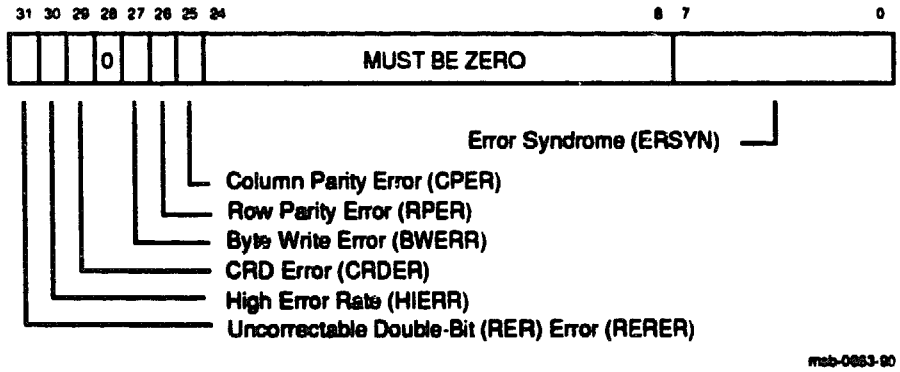
The Memory ECC Error Register logs ECC error status. The MECER also logs uncorrectable error codes for row parity error, column parity errors, and byte write errors. The MECER logs ECC error information during read cycles only. If an RER error occurs during a Write Mask cycle, the MWRITE error bit in 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.

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*

msb-0662-90

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>

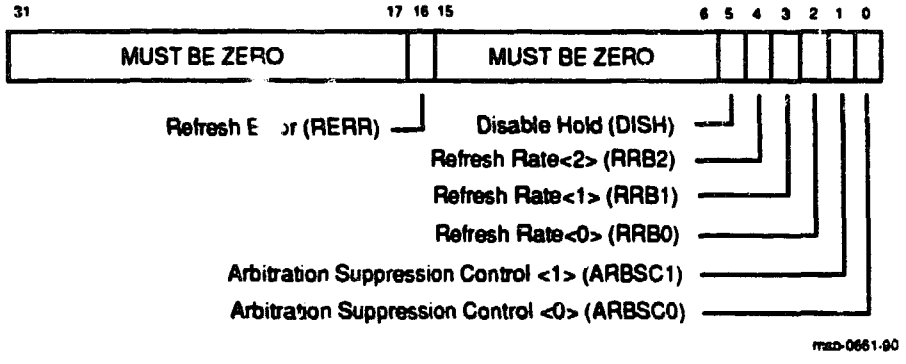
Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

Memory 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

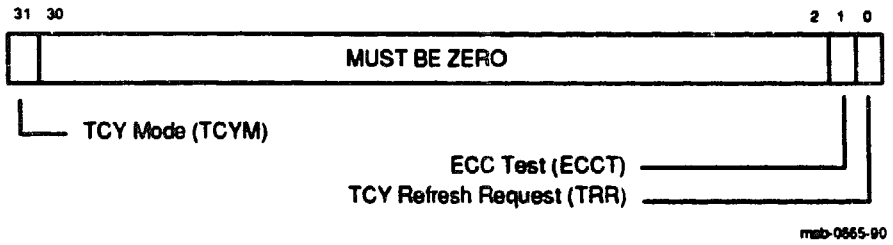
These bits control the Arbitration Supression mode.

TCY Tester Register (TCY)

The TCY Tester Register contains control bits to implement manufacturing tests.

ADDRESS

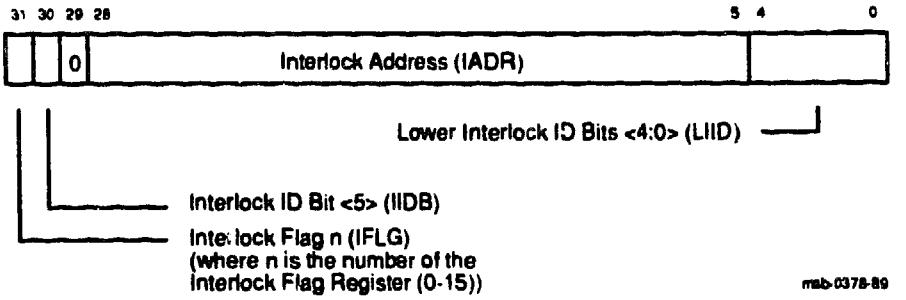
Nodespace base address + 0000 0034



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)*



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.

5.6 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.

5.6.1 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.

5.6.2 Full Write Errors

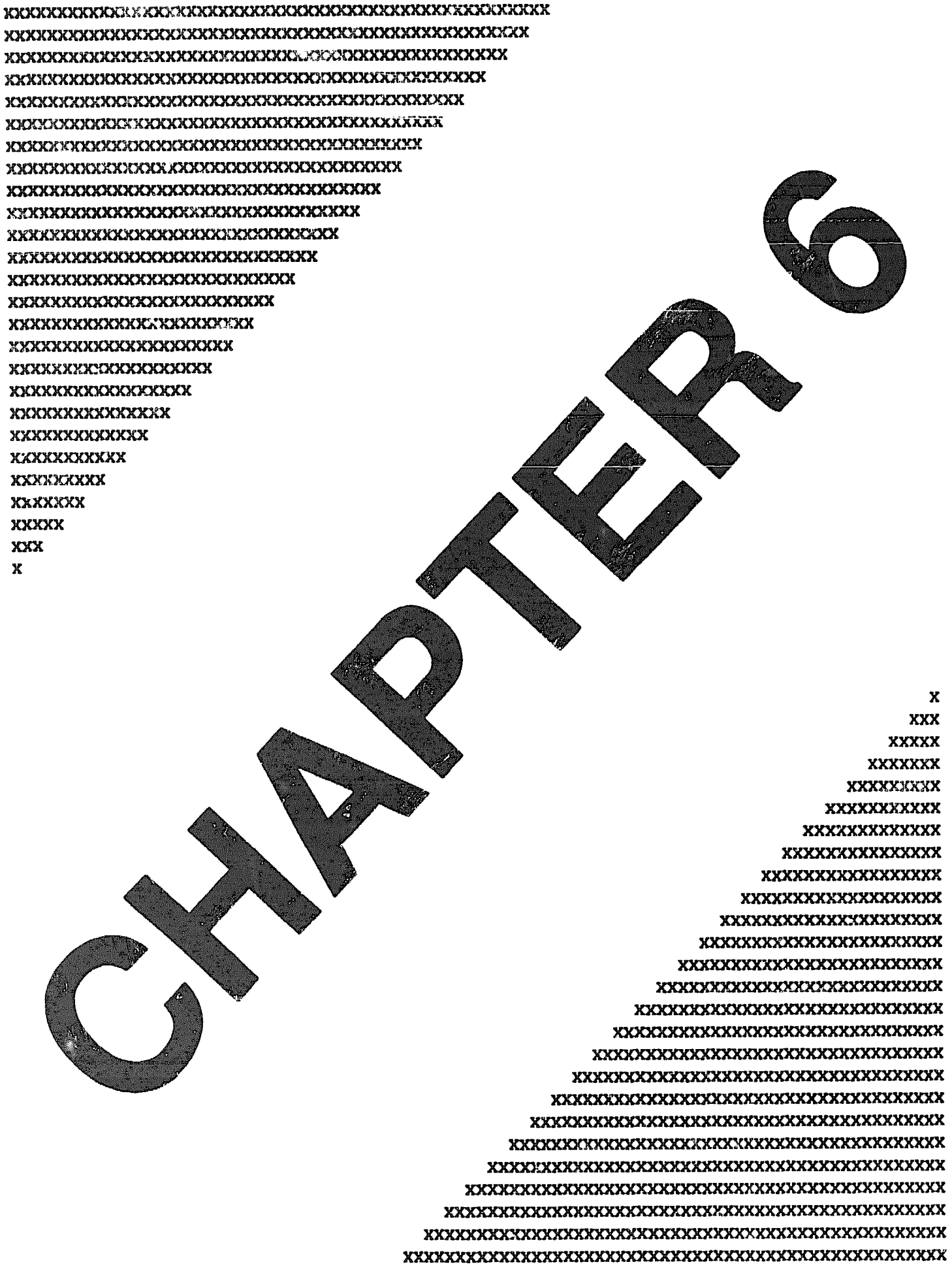
A full write is performed on a quadword or octaword, dependent on the number of mask bits that are set. If mask bits <47:32> are set, an octaword write transaction takes place. If mask bits <39:32> are set, a quadword write transaction takes place. Write data is written into memory with the generated 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.

5.6.3 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.


CHAPTER 6

6 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.

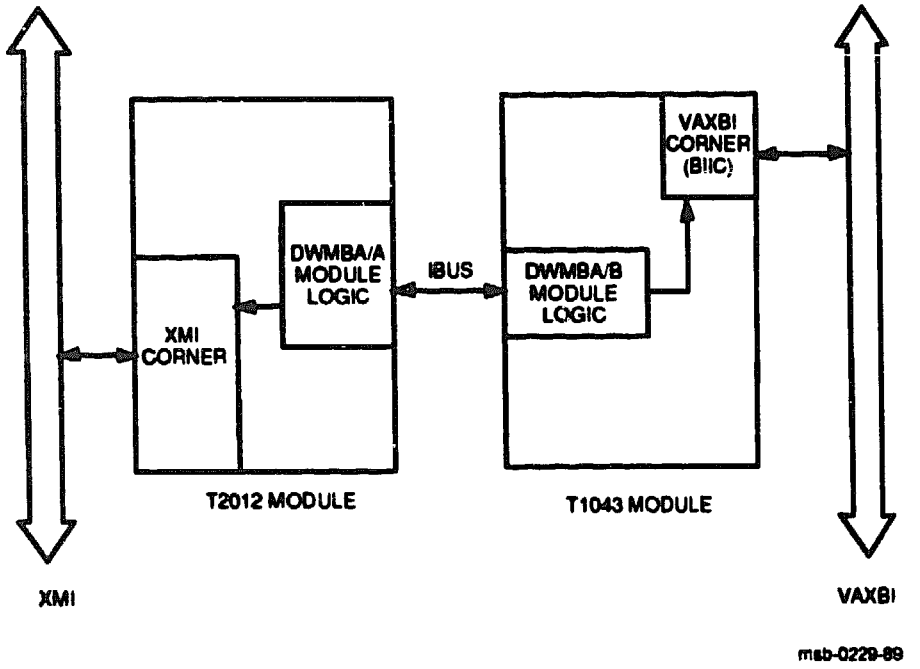
This chapter contains the following sections:

- DWMBA Overview
- CPU Transactions
- DMA Transactions
- DWMBA Registers
- Interrupts
- Error Reporting
- DWMBA Initialization, Self-Test, and Booting

6.1 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 6–1 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 KN58A processor(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 KN58A processor. 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 DECsystem 5800 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.

6.2 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 6–1 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

6.2.1 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.

6.2.2 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.

6.2.3 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 KN58A processor never issues interlocked instructions to I/O space.

6.2.4 Interrupts

6.2.4.1 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>).

6.2.4.2 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.

6.2.4.3 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>).

6.3 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 6–2 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

Table 6–2 (Cont.) VAXBI-to-XMI Command Translations

VAXBI	XMI
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 6.3.4.

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.

6.3.1 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.

6.3.2 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. In that case, the memory returns a Locked Response (LOC) to the DWMBA. The DWMBA issues a RETRY confirmation code to the VAXBI commander, releasing the VAXBI. The DWMBA returns to idle and awaits the next VAXBI request.

6.3.3 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.

6.3.4 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.

6.4 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 6–3 lists the DWMBA/A module XMI module registers. Table 6–4 lists the DWMBA/B module VAXBI module registers. Table 6–5 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 6.4 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 more information on I/O addressing.

Table 6–3 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 6–4 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 6–5 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

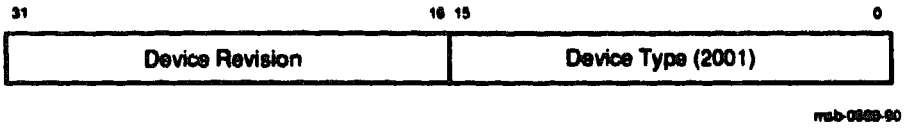
[1]The abbreviation "bb" refers to the base address of a VAXBI node (the address of the first location of the nodespace).

[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>

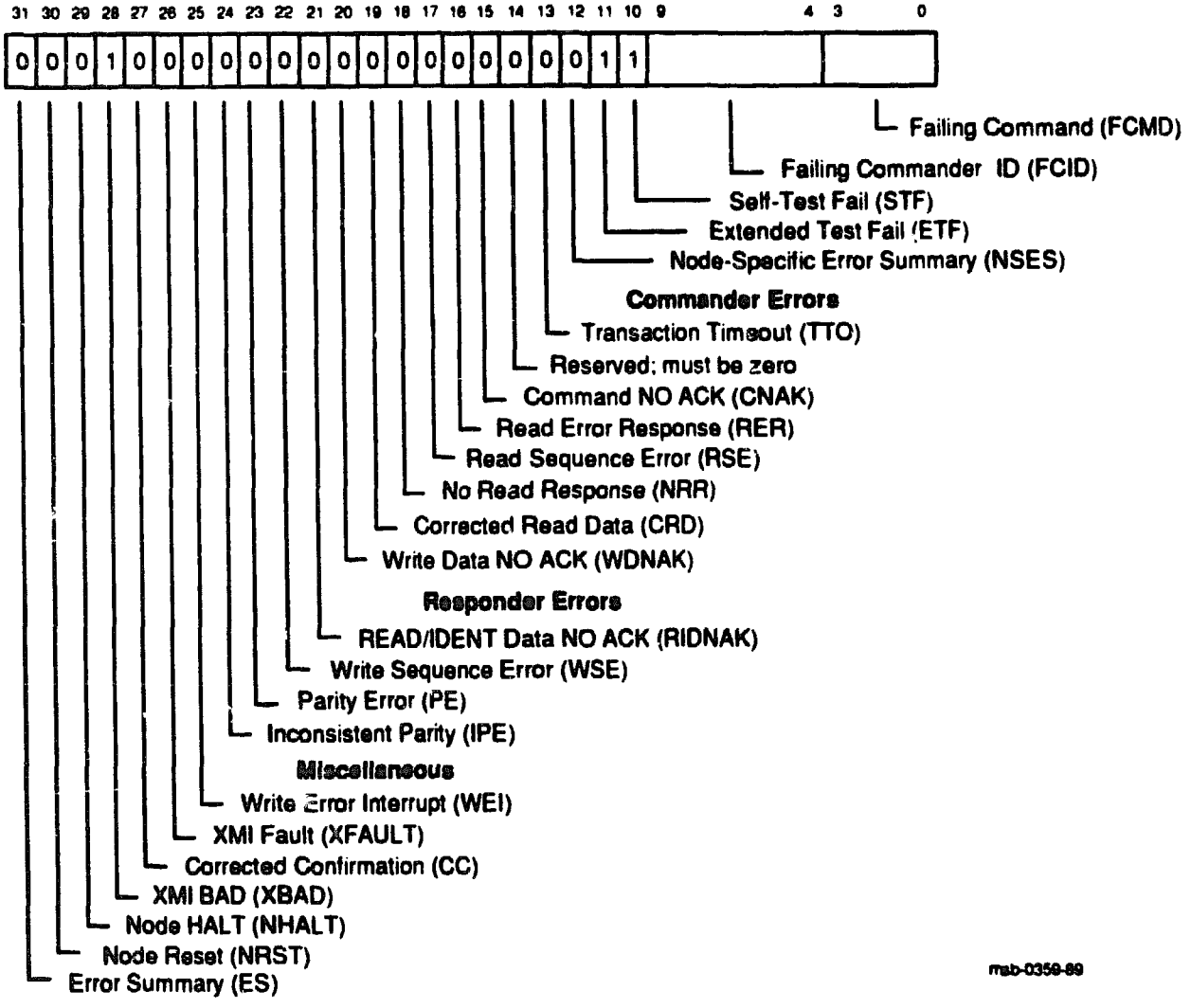
Name: Device Type
Mnemonic: DTYPE
Type: RO, 0

Identifies the type of node. 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, 0

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: Self-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>

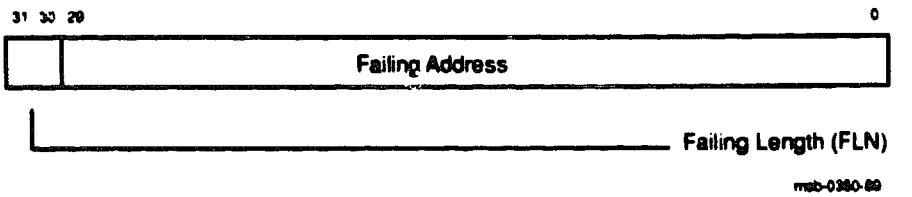
Name: Failing Command
Mnemonic: FCMD
Type: RO

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 *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

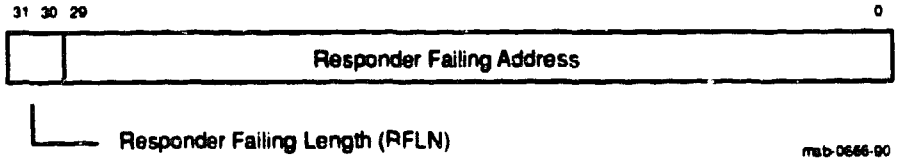
The Failing Address field logs the value of XMI D<29:0> during the command cycle of a failing transaction.

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>

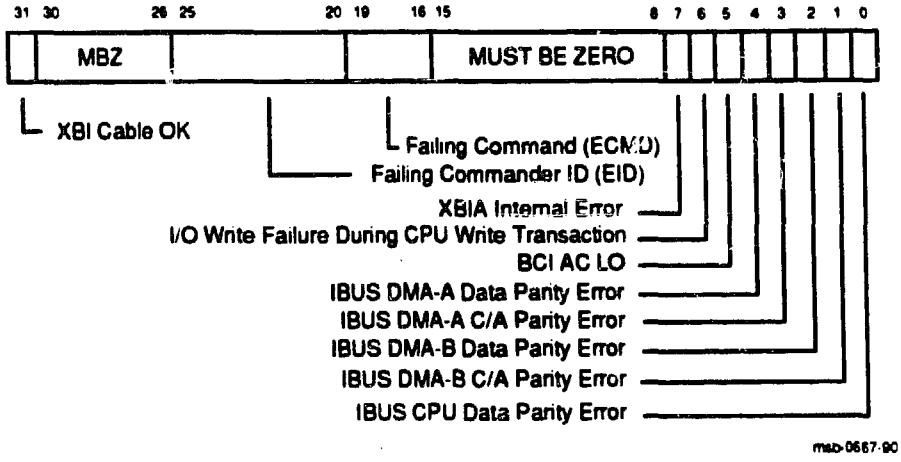
Name: Responder Failing Address
Mnemonic: None
Type: RO

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 "mem 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

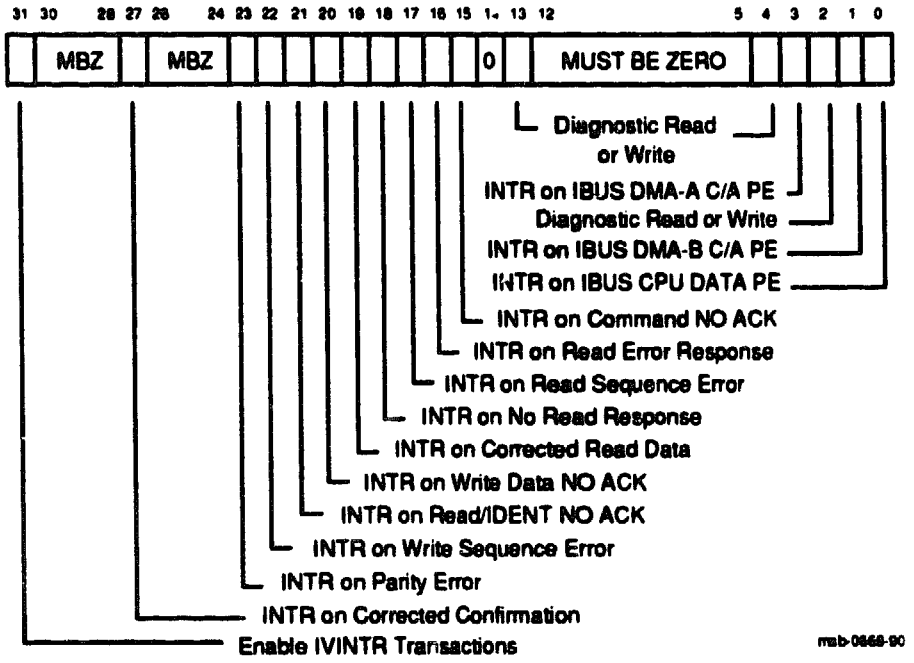
Type: R/W1C, 0

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

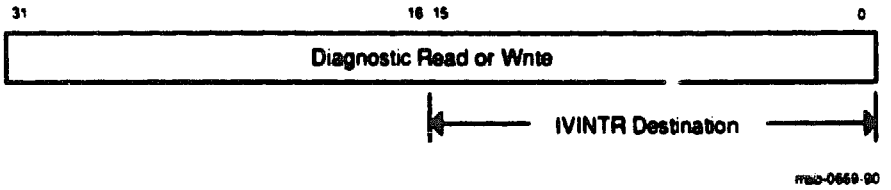
When the INTR on IBUS CPU DATA PE bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if a parity error was detected on the IBUS when the DWMBA/B module was loading the CPU data location.

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

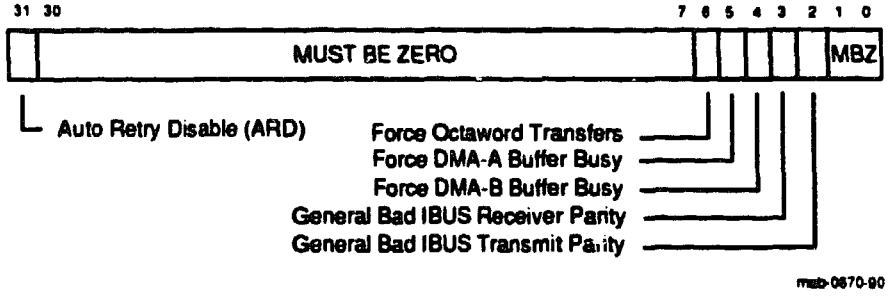
The IVINTR Destination bits determine which nodes on the XMI will be targeted by the DWMBA when it issues an Implied Vector Interrupt transaction. Each of the 16 bits corresponds to one of the 16 XMI nodes (only 14 nodes are used in the DECsystem 5800). When a bit is set, the selected node will be the 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: RC, 0

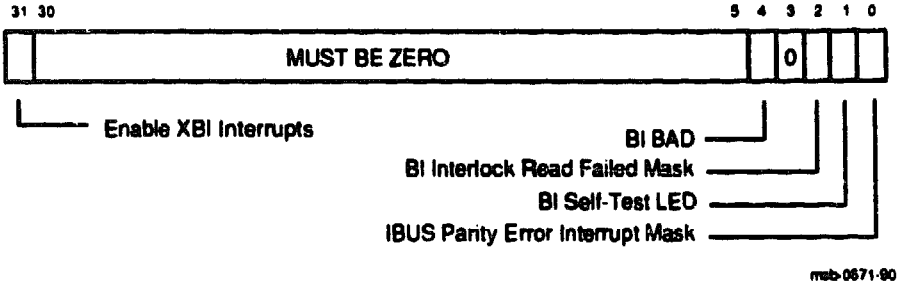
Reserved; must be zero.

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<30:5>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

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.

The BI BAD bit sets to logic level one when the VAXBI BI BAD L deasserts. When the BI BAD bit sets, it indicates that all VAXBI nodes have passed self-test, except for the DWMBA/B module, which does not assert BI BAD L.

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: BI Self-Test LED
Mnemonic: None
Type: R/W, 0

The BI Self-Test LED bit is set by the XMI boot processor node when the XBI self-test completes without error. If any portion of the XBI self-test fails, this bit does not set. When the BI Self-Test LED bit sets, the VAXBI Self-Test LED lights on the DWMBA/B module.

NOTE: The BI Self-Test LED bit has NO EFFECT on the operation of the XMI Self-Test LED on the DWMBA/A module. The XMI Self-Test LED is controlled by XBER<10>, Self-Test Fail.

bit<0>

Name: IBUS Parity Error Interrupt Mask

Mnemonic: None

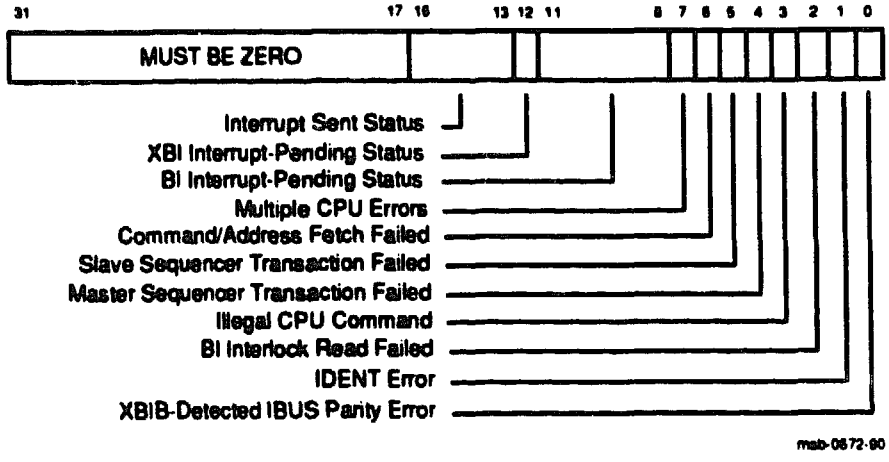
Type: R/W, 0

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.

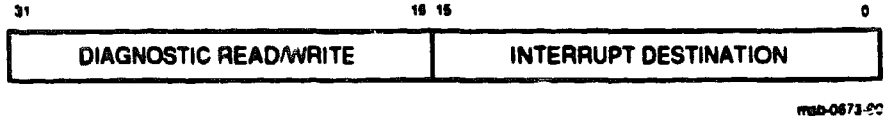
When the XBIB-Detected IBUS Parity Error bit is set with its corresponding mask bit, an error interrupt request is generated.

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

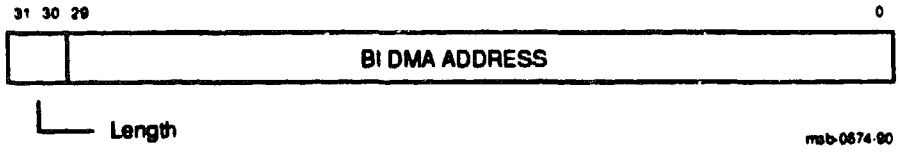
The Interrupt Destination bits determine the nodes on the XMI that are targeted by the DWMBA when it issues an interrupt transaction. Each bit in the 16-bit field corresponds to one of the 16 XMI nodes (only 14 nodes are used in the DECsystem 5800). When a bit is set to one, the selected node is the 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

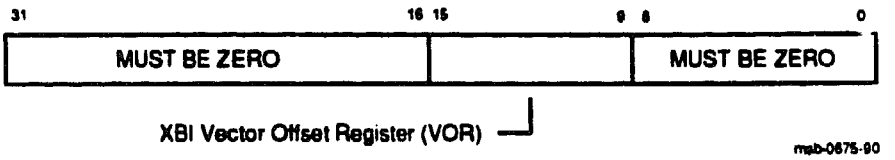
The BI DMA Failing Address contains the longword physical address (bits<29:0>) and length of the received VAXBI-to-XMI transaction (bits<31:30>.) If no errors are detected, the register reads back the last VAXBI transaction. The register logically locks upon error and unlocks when that error clears.

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

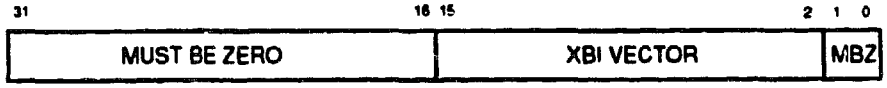
Reserved; must be zero.

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



mab-0676-90

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>

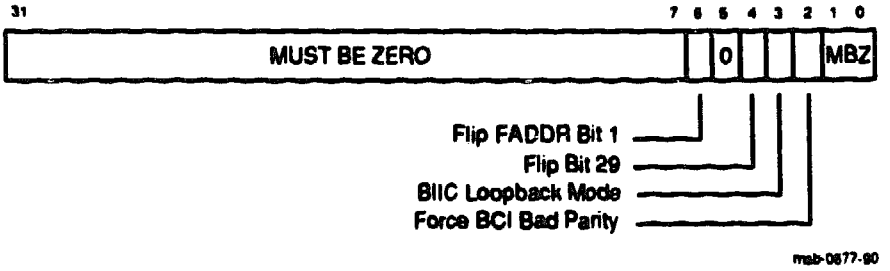
Name: Reserved
Mnemonic: None
Type: RO, 0

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

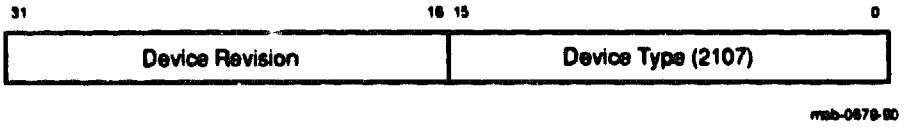
Type: Undefined

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.

6.5 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.

6.5.1 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 6–2 XMI Bus Vector Format

Figure 6–3 UNIBUS Vector Format

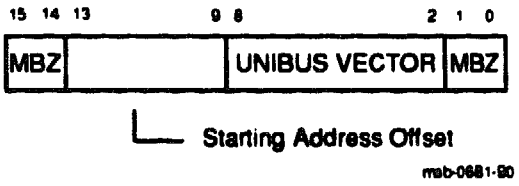
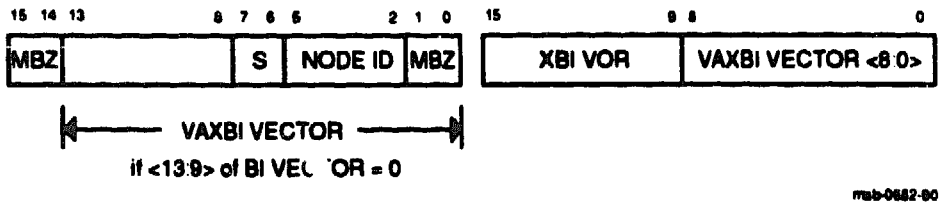


Figure 6–4 VAXBI Node Bus Vector Format



6.5.1.1 XMI Bus Vector Format

XMI device-initiated interrupts return vectors in the format shown in Figure 6–2 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.

6.5.1.2 Offsettable Bus Vectors

There are several interrupt vectors returned by offsettable devices, including the BUA (VAXBI-to-UNIBUS Adapter) and the KLESI-B (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 6–3 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.

6.5.1.3 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 6–4 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.

6.5.2 Interrupt Levels and Vectors

Table 6–6 lists the interrupt conditions used by the DWMBA adapter.

Table 6–6 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.

6.5.3 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.

6.5.3.1 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.

6.5.3.2 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.

6.5.4 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.

6.5.4.1 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.

6.5.4.2 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.

6.6 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.

6.6.1 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.

6.6.2 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.

6.6.3 DWMBA XMI-to-VAXBI Adapter Error Response Matrix

Table 6–7 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 6–8 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 6–9 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 6–10 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 6–11 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 6–12 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 6–12 (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	–	–	–	–

6.7 DWMBA Initialization, Self-Test, and Booting

This section discusses the DWMBA adapter initialization and diagnostics.

6.7.1 DWMBA Initialization

The three ways to reset the DWMBA are:

- Power-Up Sequence—When the DECsystem 5800 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 KN58A processor 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 DECsystem 5800 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.

6.7.2 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.

6.7.2.1 Loopback

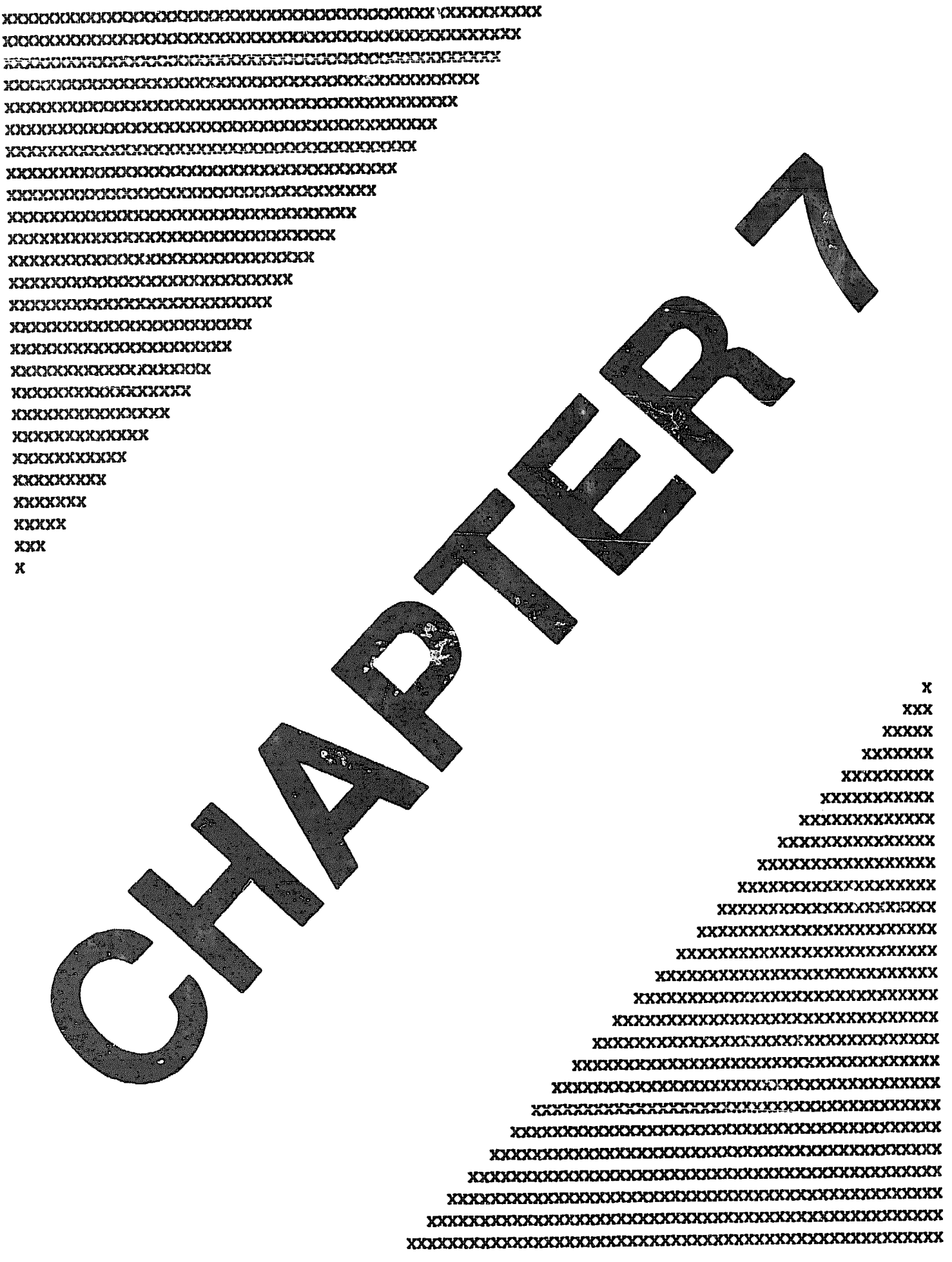
Two diagnostic loopbacks are implemented on the DWMBA: the BIIC loopback of the VAXBI and a transformation of a CPU transaction into a DMA transaction.

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.

6.7.2.2 Self-Test

DWMBA self-test is executed by the boot processor on the XMI using the processor's resident ROM.


CHAPTER 7

7 Power and Cooling Systems

The DECsystem 5800 power system consists of an AC power controller, the power and logic unit, five power regulators, 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 12 of the *DECsystem 5800 Options and Maintenance* manual for more on power components.

7.1 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

7.1.1 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.

7.1.2 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 DECsystem 5800 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.

7.1.3 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.

7.1.4 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.

7.1.5 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

7.1.5.1 XMI Reset Timing Control Logic

The XMI reset timing control logic handles these sequences:

- Cold 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

7.1.5.2 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 battery voltage.

7.1.5.3 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.

7.1.6 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 7–1.

Table 7–1 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 DECsystem 5800.
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 DECsystem 5800 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.
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.

Table 7–1 (Cont.) Power System Signals

Name	Origin	Destination	Description
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.

7.2 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.


APPENDIX A

A Console Entry Points

The following entry points are defined by the R3000 portion of the console program for the use of the operating system loader, operating system, and other stand-alone programs.

All routines are called as normal C-language routines, using the normal R3000 calling conventions. The entry points are located in a table beginning at address b00a 0000. The beginning of a routine is at address:

b00a 0000 + (8 * *entry_nr*)

where *entry_nr* is the entry point number shown in the routine description.

It is the responsibility of the caller to handle exceptions. The console does not enable its own exception handlers.

A.1 reset - Power-up console entry - Entry 0

This entry point receives control when the R3000 chip is reset. It is not normally invoked by software.

A.2 promexec - Exec new program - Entry 1

Not currently supported.

A.3 exit - Reenter console - Entry 2

This entry point returns control of the processor to the console program, without reinitializing the console state. The console will display a message indicating an exit was performed with the return value *status*. Control will then pass to the console prompt.

The state of the program that called exit is not preserved.

```
void
exit(status)
int status;
```

A.4 reinit_console - Reinitialize the console - Entry 3

This entry returns control of the processor to the console program and reinitializes the console's state. It is normally called only by the console program itself to recover from unexpected error conditions.

A.5 conditional_boot - Invoke power-up action - Entry 4

This entry returns control of the processor to the console program, and takes the action selected by the front panel key switches. If the key switches are set to Halt and Enabled, the console prompt will be displayed. If the key switches are set otherwise, the system will be rebooted as if a system power-up had occurred.

```
void
conditional_boot()
```

A.6 reboot - Reboot the system - Entry 5

This entry unconditionally attempts to reboot the system, from the device specified in the **bootpath** environment variable. If the bootpath is not set or is invalid, an error message is displayed and control remains with the console program.

```
void
reboot()
```

A.7 open - Open a file - Entry 6

Open provides I/O access to a file or device specified by *filename* in a fashion similar to the Ultrix system call. The *flags* parameter indicate the type of access desired. Returns an integer file descriptor that must be supplied on calls to routines that manipulate the file.

If an error is encountered, a message is displayed on the console terminal, and open returns -1.

The pointer used to mark the current position within the file is set to the beginning of the file.

A maximum of 7 files can be open simultaneously. On the first open for a given device controller, the console may reinitialize the controller.

```
int
open(filename, flags)
char *filename;
int flags;
```

A.7.1 filename

The string supplied for the filename parameter has the general form:

dev(controller,unit,partition,path)

The parentheses and the dev and controller components are always required. All the numeric portions of the filename can be given either in decimal or in hexadecimal, by using the *0x* prefix. The components of the filename string are:

- dev—Identifies the type of device being opened. Recognized devices are discussed in Section A.45.

- *controller*—Identifies the physical path through the I/O adapters to the device being opened. The *controller* string is of the form "*/xl/bm/cn*" where *l*, *m*, and *n* are the XMI node number, VAXBI node number, and CI node numbers needed to locate the boot device. The /b and /c components are omitted if they are not applicable.
- *unit*—Identifies the unit number of the device. If the unit number is omitted, it defaults to zero.
- *partition*—Identifies the starting logical block of the software managed partition of the device. Partition is only meaningful for disk devices. If omitted, partition defaults to zero.
- *path*—Supplies the Ultrix path name of the file to be opened. For disk devices, the path may be specified but is ignored. For Ethernet boots, the specified path is requested from the booting system. Specifying the path parameter is illegal on boots from tape.

A.7.2 flags

The *flags* argument indicates the type of access requested. Legal values are:

O_RDONLY = 0000
O_WRONLY = 0001
O_RDWR = 0002

A.8 read - Read from a file - Entry 7

This function attempts to read *cnt* bytes from a file into a buffer, *buf*. The data is read from the current position of the file, and the number of bytes read is returned. *fd* is a file descriptor returned by an open( ) call.

A returned value of 0 indicates the end of the file. If an error is encountered, a message is displayed on the console terminal and -1 is returned.

```
int
read(fd, buf, cnt);
int fd;
char *buf;
int cnt;
```

A.9 write - Write to a file - Entry 8

The *write* entry attempts to write *cnt* bytes of data from the buffer *buf* to a file. The file is specified by *fd*, a file descriptor returned from an open( ) call. The data is written at the current file position.

The number of bytes written is returned if the write was successful. If an error occurs, a message is displayed on the console terminal, and -1 is returned.

```
int
write(fd, buf, cnt)
int fd;
char *buf;
int cnt;
```

A.10 ioctl - Device-specific I/O operation - Entry 9

This entry performs a device-specific operation, as specified by the value supplied in *cmd*. The device is specified by *fd*, a file descriptor returned from an open( ) call. The supported operations are discussed in Section A.45. If an error occurs, a message is issued on the console terminal, and -1 is returned.

```
int
ioctl(fd, cmd, arg)
int fd;
int cmd;
int arg;
```

A.11 close - Close an open file - Entry 10

The **close** function terminates access to the file associated with the file descriptor *fd*. The console requires all files associated with a given device or controller to be closed before other software attempts to gain control of the device or controller.

If an error is encountered, a message is displayed on the console terminal and a value of -1 is returned.

```
int
close(fd)
int fd;
```

A.12 lseek - position within a file - Entry 11

The **lseek** function moves the pointer associated with a file open for reading or writing. The file descriptor *fd* indicates the file to be positioned. If *how* is set to zero, the file is positioned to the specified *offset*. If *how* is set to one, *offset* is added to the current file file position.

If an error occurs, an error message is written on the console terminal, and a value of -1 is returned.

```
int
lseek(fd, offset, how)
int fd;
unsigned offset;
int how;
```

A.13 getchar - Input a single character - Entry 12

This function reads a single character from the current console input device. Control does not return until a character is available.

```
int
getchar();
```

A.14 putchar - Output a single character - Entry 13

This function writes a single character to the current console output device.

A newline character is preceded with a carriage return. A tab character is converted to sufficient blanks to position to the next "tab stop," where "tab stops" occur every 8 characters.

```
void
putchar(c)
char c;
```

A.15 showchar - Output a single character - Entry 14

This function writes a single character to the current console output device.

All printing characters are displayed normally. The function displays backspace as "\b", form feed as "\f", newline as "\n", carriage return as "\r", and tab as "\t". All other nonprinting characters are displayed as "\xxx", where xxx is the octal code for the character.

```
void
showchar(c)
char c;
```

A.16 gets - Get line of input - Entry 15

The gets entry reads a line of input for the console terminal and places the results into *buf*. Control returns when a carriage return or newline character is received, terminating the line of input. The terminating character is not buffered. The address of *buf* is returned.

Line editing characters can be used when data is being read by gets.

```
char *
gets(buf)
char *buf;
```

A.17 puts - Display a line of output - Entry 16

This entry writes a line of output pointed to by *line* to the console terminal. Characters are handled as described for the putc function.

```
void
puts(line)
char *line;
```

A.18 printf - Print formatted values - Entry 17

The **printf** entry formats and displays one or more value arguments on the console terminal. The formatting is controlled by the string passed as *fmt*.

This function implements a subset of the standard C library printf function. Consult C language documentation for details. The following format items are supported: %x, %d, %u, %o, %c, %b, %s, %%, numeric field widths with optional leading minus, and numeric field width with leading zero.

```
void
printf(fmt, va_alist)
char *fmt;
va_dcl
```

A.19 flush_cache - Flush processor cache - Entry 28

This function causes all entries in the processor's primary instruction and data caches to be invalidated.

```
void
flush_cache()
```

A.20 clear_cache - Clear part of the processor cache - Entry 29

This function causes any cache entries associated with a specific range of addresses to be invalidated. The range begins at address *base* and extends for *cnt* bytes. Entries are cleared from both the instruction and data caches.

```
void
clear_cache(base, cnt)
unsigned base;
int cnt;
```

A.21 setjmp - Save program context - Entry 30

The **setjmp** entry point is used for dealing with exceptions and error conditions encountered by low-level console routines. When **setjmp** is called, it saves the current stack position and register contents of the program, and returns zero. A later call to the **longjmp** entry point will restore this saved context and return control to the point following the call to **setjmp**.

```
typedef jmp_buf[11];
int
setjmp(jmp_buf);
jmp_buf jmp_buf;
```

A.22 longjmp - Restore program context - Entry 31

This entry terminates execution in the current context and restores the program context stored by a previous call to **setjmp**. The context is specified by supplying the *jmp_buf* used on the call to **setjmp**. When context is restored, the **setjmp** call returns the value supplied in *rval*.

```
longjmp(jmp_buf, rval);
struct jmp_buf *jmp_buf;
int rval;
```

A.23 utlbmiss_except - Console UTLB miss vector - Entry 32

This entry corresponds to the console's UTLB miss exception handler. The position of this entry in the entry point table is dictated by the MIPS processor architecture. Calls to this entry point are not supported.

A.24 getenv - Get value of an environment variable - Entry 33

This entry returns the value of the console environment variable specified by *name*. If the specified name is not found, a null pointer is returned.

```
char *
getenv(name)
char *name;
```

A.25 setenv - Set value of an environment variable - Entry 34

This entry sets the value of the console environment variable specified by *name* to the string supplied in *value*. The function always returns zero.

Certain environment variables are known to the console. An error message is issued if **setenv** is called for a variable marked read only. Variables not marked as "volatile" are also stored in EEPROM, after the memory copy of the variable has been set. An error messages is issued if the EEPROM copy cannot be written (corrupted EEPROM, key switch not set to "Update," etc.) Various side effects also apply (for example, changing the variable *baud* changes the console terminal baud rate).

```
int
setenv(name, value)
char *name;
char *value;
```

A.26 atob - Convert ASCII to binary - Entry 35

Convert the ASCII string *str* to a binary value that is placed in the location pointed to by *intp*. Conversion continues until a nonnumeric character (or the null string terminator) is encountered. The return value is a pointer to the unprocessed portion of *str*.

The string must be integer, with an optional minus sign. Leading blanks are ignored. If the first digit seen is not "0", the number is considered to be in decimal notation. A prefix of "0x" indicates hexadecimal notation, "0b" indicates binary notation, and "0" followed any other digit indicates octal notation.

If the converted value generates an arithmetic overflow, a warning message is printed on the console terminal.

```
char *
atob(str, intp)
char *str;
int **intp;
```

A.27 strcmp - Compare two strings - Entry 36

The **strcmp** function compares two null-terminated strings. The return value is zero if the two strings are the same, a negative value if *str1* is less than *str2*, or a positive value if *str1* is greater than *str2*.

```
int
strcmp(str1, str2)
char *str1;
char *str2;
```

A.28 strlen - Find string length - Entry 37

This function returns the length in bytes of a null-terminated string.

```
int
strlen(str)
char *str;
```

A.29 strcpy - Copy a string - Entry 38

This function copies the null-terminated string *str2* to *str1* and returns a pointer to the next available character position in *str1*. *str1* must be long enough to contain *str2*.

```
char *
strcpy(str1, str2)
char *str1;
char *str2;
```

A.30 strcat - Concatenate two strings - Entry 39

This function appends a copy of the null-terminated string *str2* to the end of the null-terminated string *str1*. *str1* must be long enough to hold the additional characters. A pointer to *str1* is returned.

```
char *
strcat(str1, str2)
char *str1;
char *str2;
```

A.31 parse - Parse a simple command - Entry 40

This entry provides access to the command parser used by the console. When this routine is called, it enters an endless loop executing the following steps:

1. The string *prompt* is displayed on the console terminal.
2. A line of input is read from the console terminal using gets.
3. The **argvize** function is used to build an argc/argv argument list.
4. The *cmd_table* is searched for a command that matches the first entry in the argv list.
5. If a command is found, the corresponding routine is called with the argc/argv arguments.
6. If the routine returns a non-zero value, the usage string is displayed.
7. If the command is not found, an error message is printed.

```
struct cmd_table{
char *name;  /* Command name */
int (*routine)(); /* Command routine */
char *usage;  /* Help or usage string */
};

int
parser(cmd_table, prompt, reserved)
struct cmd_table *cmd_table;
char *prompt;
char *reserved;
```

A.32 parse_range - Parse an address range - Entry 41

This entry provides access to the console routine that parses the address range syntax used in console commands. The string *str* is parsed. The first component of the range is stored using *basep*, and the second component of the range is stored using *cntp*. The function returns zero if the range is of the form *address:address*, one if the range is of the form *address#count*, and -1 if the range cannot be parsed. A range consisting of a single number is treated as *address#count*, with a count of one.

```
int
parse_range(str, basep, cntp)
char *str;
unsigned *basep;
unsigned *cntp;
```

A.33 argvize - Parse string into tokens - Entry 42

This entry breaks the string *str* into tokens and fills in a structure *tlp* with a copy of each of the tokens. Tokens are delimited by spaces, which are otherwise ignored. Text enclosed in single or double quotes is considered as one token. The function returns the number of tokens found.

```
struct token_list{
     char *strptrs[MAXSTRINGS];  /* Vector of token pointers */
     char *strbuf[STRINGBYTES];  /* The token strings */
     char *strp;   /* Ptr to next free byte in strbuf */
     char *strcnt;   /* Number of tokens in structure */
};

int
argvize(str, tlp)
char *str;
struct token_list *tlp;
```

A.34 help - Print help from a command table - Entry 43

This entry displays on the console terminal the help text associated with one or more commands. The commands and help text are defined in *cmd_table*, as described for the command_parser entry point. An error message is printed for any command not found in the table.

If no commands are supplied (*argc*=1), all help text is displayed.

```
int
help(argc, argv, cmd_table)
int argc;
char **argv;
struct cmd_table *cmd_table;
```

A.35 dumpcmd - Invoke console dump command - Entry 44

This entry point allows access to the console's **dump** command, for producing formatted dumps of memory. *argv* points to a sequence of character string pointers. The first string is ignored. The remaining strings are interpreted as the tokens that would be typed on the **dump** command line. Tokens are the sequences of characters delimited by spaces.

If an error is encountered, a message is displayed on the console terminal.

```
void
dumpcmd(argc, argv)
int argc;
char **argv;
```

A.36 setenvcmd - Invoke console setenv command - Entry 45

This entry point allows access to the console's **setenv** command. *argv* points to three character strings, the first of which is ignored. The second string specifies the variable to be set. The third string specifies the value to be stored in the variable.

If an error is encountered, a message is displayed on the console terminal.

```
void
setenvcmd(argc, argv)
int argc;
char **argv;
```

A.37 unsetenvcmd - Invoke console setenv command - Entry 46

This entry point allows access to the console's **unsetenv** command. *argv* points to two character strings, the first of which is ignored. The second string specifies the variable to be removed.

If an error is encountered, a message is displayed on the console terminal.

```
void
unsetenvcmd(argc, argv)
int argc;
char **argv;
```

A.38 printenvcmd - Invoke console printenv command - Entry 47

This entry point allows access to the console's **printenv** command. *argv* points to a sequence of character string pointers, the first of which is ignored. The remaining strings specify the variables for which the values should be displayed. If no additional strings are supplied (*argc*=1), then all known environment variables are displayed along with their values.

If an error is encountered, a message is displayed on the console terminal.

```
void
printenvcmd(argc, argv)
int argc;
char **argv;
```

A.39 general_except - Console general exception vector - Entry 48

This entry corresponds to the console's general exception handler. The position of this entry in the entry point table is dictated by the MIPS processor architecture. Calls to this entry point are not supported.

A.40 clear_nofault - Clear console fault handlers - Entry 51

Not supported.

A.41 not_implemented - Unimplemented function - Entry 52

This position in the entry point table does not correspond to a routine. Instead, it is guaranteed to contain the value used in the table to represent an unimplemented function. To determine if a particular entry point is implemented, compare the 32-bit value in the desired entry with the 32-bit value in entry 52. If they are equal, the desired function is not implemented.

A.42 halt_interrupt - Service halt interrupt - Entry 54

This is the entry point where the console expects to receive control on a halt interrupt. Halt interrupts are caused when the console hardware detects a Control-P typed on the console terminal, or when the processors XBE:NHALT bit is set. The R3000 is interrupted (if enabled) using interrupt 4. Any program that establishes a general exception handler should avoid masking this interrupt and should pass control to this control entry point whenever the interrupt is received. The interrupt handler should avoid modifying any machine state except for the kt0 general-purpose register. If this is not possible, then the register contents visible with the console **examine** will reflect the changes made by the interrupt handler.

On entry to this routine, the console will save all processor registers that can then be examined with the console e (examine) command.

The console **continue** command restores all saved states and returns from this routine.

```
void
halt_interrupt()
```

A.43 enter_maintmode - Enter maintenance mode - Entry 96

This entry point causes control of the system to be returned to the maintenance mode command parser, executing on the CVAX portion of the processor. The contents of memory are not altered, but all state internal to the R3000 processor is lost.

```
void
enter_maintmode()
```

A.44 start_maint - Start code on the maintenance processor - Entry 97

This entry point causes control of the system to be returned to code executing on the CVAX portion of the processor. The call must include a parameter giving the CVAX physical address at which execution should begin. All R3000 internal processor state is lost.

```
void
start_maint(address)
unsigned address;
```

A.45 Prom device drivers

The console standard I/O entry points are layered on a set of device drivers. The following sections describe some details of the supported device drivers:

A.45.1 bootp - BOOTP protocol Ethernet driver

Not supported.

A.45.2 ra - MSCP disk driver

The ra driver supports MSCP disks connected via a KDB50 or via a CIBCA-B and HSC disk controller.

- Up to three KDB50 controllers can be active. A controller is active until all the file descriptors open for it are closed. Any number of units may be accessed, up to the open file limit.
- Only one CIBCA-B controller can be active at a time. Only one unit can be accessed.
- lseek operations must be on a 512-byte block boundary.
- There are no supported *ioctl* operations commands.

A.45.3 mop - MOP protocol Ethernet driver

Not supported.

A.45.4 tms - MSCP tape driver

The tms driver supports TMSCP tapes connected via a TBK50 or TBK70 controller.

- Only one TBKxx controller can be active.
- *lseek* is not supported. The tape is always accessed sequentially.
- The following *ioctl* commands are supported:

  MTWEOF - Write a tape mark
  MTREW - Rewind the tape
  MTOFFL - Rewind the tape and unload (if unload supported by the drive).

A.45.5 tty - console terminal port

The tty driver controls the console terminal line. Only one unit, unit zero is supported. The following *ioctl* commands are supported:

FIOCSCAN - Poll device for input
TIOCRAW - Disable recognition of control characters
TIOCFLUSH - Flush all pending input
TIOCREOPEN - Reopen the device, applying the current parameters (for example, the baud rate).


INDEX

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X