

# DEC 4000 AXP Model 600 Series

---

## Technical Manual

Order Number: EK-KN430-TM. A01

**Digital Equipment Corporation  
Maynard, Massachusetts**

---

**First Printing, December 1992**

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

© Digital Equipment Corporation 1992.

All rights reserved.

The following are trademarks of Digital Equipment Corporation: AXP, DEC, DEC VET, DECchip, OpenVMS, ULTRIX, VAX, VMS, the AXP logo, and the DIGITAL logo.

OSF/1 is a registered trademark of Open Software Foundation, Inc. All other trademarks and registered trademarks are the property of their respective holders.

S1757

This document was prepared with VAX DOCUMENT, Version 1.2.

---

# Contents

Preface .....	xxxi
---------------	------

## Part I DEC 4000 System Description

### 1 DEC 4000 System Features

1.1	BA640 Cabinet .....	1-5
1.1.1	DC on/off Switch .....	1-8
1.1.2	RESET Switch .....	1-8
1.1.3	The Halt Switch .....	1-9
1.1.4	Power-up Console Baud rate switch .....	1-9
1.1.5	Status LEDs .....	1-10
1.2	The Backplane Assembly .....	1-10
1.2.1	The Storage Backplane Module .....	1-11
1.2.2	System Backplane Module .....	1-13
1.2.3	System Bus Routing .....	1-13
1.2.4	Futurebus+ Routing .....	1-13
1.2.5	Local I/O Bus Routing .....	1-13
1.2.6	Serial Control Bus Routing .....	1-13
1.3	KN430 CPU Subsystem .....	1-14
1.3.1	Processor Module Differences .....	1-14
1.3.2	Module order .....	1-17
1.3.3	KN430 Processor Module .....	1-17
1.3.3.1	Processor module components .....	1-19
1.3.3.2	DECchip 21064™ Alpha CPU .....	1-20
1.3.3.3	Backup Cache Memory Devices .....	1-21
1.3.3.4	System bus interface .....	1-21
1.3.3.5	System bus clock generator/distributor circuitry .....	1-21
1.3.3.6	Clock/ power detect circuit .....	1-21
1.3.3.7	Serial Control Bus Interface .....	1-21
1.3.4	KFA40 I/O Module .....	1-22
1.3.4.1	EPROM .....	1-24
1.3.4.2	FEPROMS .....	1-24
1.3.4.3	Ethernet Controllers .....	1-24
1.3.4.4	Serial Line Units .....	1-25
1.3.4.5	SCSI Controllers .....	1-25
1.3.4.6	IONIC gate arrays .....	1-25
1.3.4.7	Time of Year Clock (TOY) .....	1-25
1.3.5	MS430 Memory Module .....	1-26
1.3.5.1	DRAMS .....	1-28
1.3.5.2	Driver chips .....	1-29
1.3.5.3	CMIC chips .....	1-29
1.4	The System Bus .....	1-29
1.4.1	Supported transactions .....	1-29

1.4.2	Physical Address Space Layout .....	1-32
1.4.3	System clock signals .....	1-33
1.4.4	Subsystem communications with secondary buses .....	1-33
1.5	The Mass Storage Area .....	1-34
1.6	I/O Expansion Area .....	1-36
1.7	The Power Supply Subsystem .....	1-37
1.7.1	H7853 Front End Unit (FEU) .....	1-41
1.7.2	H7851 Power System Controller (PSC) .....	1-41
1.7.3	H7179 5 Volt Converter (DC5) .....	1-42
1.7.4	H7178 3 Volt Converter (DC3) .....	1-42
1.7.5	Disk Power Supplies .....	1-42
1.7.6	System Voltages .....	1-42
1.8	The Serial Control Bus .....	1-43
1.9	System Firmware .....	1-43
1.10	System Ports .....	1-47
1.10.1	The serial line ports (SLU) .....	1-47
1.10.2	The Ethernet Port .....	1-48
1.10.3	SCSI Storage Expansion Ports .....	1-48
1.10.4	AC Power Connector Port .....	1-48

## Part II The Processor Module

### 2 The KN430 Processor Module

#### 3 Alpha Architecture

3.1	Addressing .....	3-1
3.2	Data Types .....	3-1
3.3	Alpha Registers .....	3-11
3.3.1	Program Counter Register .....	3-11
3.3.2	Processor Status Register .....	3-11
3.3.3	Integer Registers .....	3-11
3.3.4	Floating-point Registers .....	3-12
3.3.5	Lock Registers .....	3-12
3.3.6	Alpha Internal Processor Registers .....	3-12
3.3.7	Optional Registers .....	3-12
3.4	Instruction Formats .....	3-12

#### 4 The DECchip™ 21064 CPU chip

4.1	I-box Internal Processor Registers .....	4-2
4.1.1	Translation Buffer Tag Register (TB_TAG) .....	4-2
4.1.2	Instruction Translation Buffer Page Table Entry Register (ITB_PTE) .....	4-3
4.1.3	Instruction Cache Control and Status Register (ICCSR) .....	4-4
4.1.4	Instruction Translation Buffer Page Table Entry Temporary Register (ITB_PTE_TEMP) .....	4-7
4.1.5	Exception Address Register (EXC_ADDR) .....	4-8
4.1.6	Serial Line Clear Register (SL_CLR) .....	4-9
4.1.7	Serial Line Receive Register (SL_RCV) .....	4-10
4.1.8	Instruction Translation Buffer Zap Register (ITBZAP) .....	4-10
4.1.9	Instruction Translation Buffer ASM (ITBASM) .....	4-10
4.1.10	Instruction Translation Buffer IS Register (ITBIS) .....	4-10

4.1.11	Processor Status (PS)	4-11
4.1.12	Exception Summary Register (EXC_SUM)	4-11
4.1.13	Privileged Architecture Library Base Register (PAL_BASE)	4-13
4.1.14	Hardware Interrupt Request Register (HIRR)	4-13
4.1.15	Software Interrupt Request Register (SIRR)	4-15
4.1.16	Asynchronous Trap Request Register (ASTRR)	4-15
4.1.17	Hardware Interrupt Enable Register (HIER)	4-16
4.1.18	Software Interrupt Enable Register (SIER)	4-17
4.1.19	Asynchronous System Trap Enable Register (ASTER)	4-17
4.1.20	Serial Line Transmit Register (SL_XMIT)	4-18
4.2	A-box Internal Processor Registers	4-19
4.2.1	Translation Buffer Control Register (TB_CTL)	4-19
4.2.2	Data Translation Buffer Page Table Entry Register (TB_PTE)	4-19
4.2.3	Data Translation Buffer Page Table Entry Temporary Register (DTB_PTE_TEMP)	4-20
4.2.4	Memory Management Control and Status Register (MM_CSR)	4-21
4.2.5	Virtual Address Register(VA)	4-22
4.2.6	Data Translation Buffer Zap Register (DTBZAP)	4-22
4.2.7	Data Translation Buffer ASM Register (DTBASM)	4-22
4.2.8	Data Translation Buffer Invalidate Signal Register (DTBIS)	4-22
4.2.9	Flush Instruction Cache Register (FLUSH_IC)	4-22
4.2.10	Flush Instruction Cache ASM Register (FLUSH_IC_ASM)	4-22
4.2.11	A-box Control Register (A-BOX_CTL)	4-23
4.2.12	Alternate Procressor Mode Register (ALT_MODE)	4-24
4.2.13	Cycle Counter Register	4-25
4.2.14	Cycle Counter Control Register (CC_CTL)	4-25
4.2.15	Bus Interface Control Unit Register (BIU_CTL)	4-26
4.3	Privileged Architecture Library Temporary Registers (PAL_TEMP)	4-28
4.3.1	Data Cache Status Register (DC_STAT)	4-29
4.3.2	Bus Interface Unit Status Register (BIU_STAT)	4-31
4.3.3	Bus Interface Unit Address Register (BIU_ADDR)	4-32
4.3.4	Fill Address Register (FILL_ADDR)	4-33
4.3.5	Fill Syndrome Register (FILL_SYNDROME)	4-34
4.3.6	Backup Cache Tag Register (BC_TAG)	4-36
4.4	Error Detection and Correction (EDC)	4-37

## 5 Backup Cache (B-cache)

5.1	Control Store	5-1
5.2	Tag Store	5-2
5.3	Data Store	5-2
5.4	Backup Cache Control Register Definitions	5-3
5.4.1	Backup Cache Control Register (CSR0)	5-3
5.4.2	Backup Cache Correctable Error Register (CSR1)	5-8
5.4.3	Backup Cache Correctable Error Address Register (BCCEA) CSR2	5-11
5.4.4	Backup Cache Uncorrectable Error Register (CSR3)	5-12
5.4.5	Backup Cache Uncorrectable Error Address Register (CSR4)	5-15
5.4.6	System Bus Cycles	5-17
5.5	Cache Block Merge Buffer	5-17
5.6	Duplicate Primary Data Cache Tag Store	5-18
5.6.1	Duplicate Tag Error Register (CSR5)	5-20
5.7	Lack of Duplicate Primary Instruction Cache Tag Store	5-22
5.8	Lack of Cache Block Prefetch	5-22
5.9	Data Integrity	5-22

## 6 System Bus Interface

6.1	CPU System Bus Register Definitions . . . . .	6-1
6.1.1	System Bus Control Register (CSR6) . . . . .	6-1
6.1.2	System Bus Error Register (CSR7) . . . . .	6-4
6.1.3	System Bus Error Address Low Register (CSR8) . . . . .	6-10
6.1.4	System Bus Error Address High Register (CSR9) . . . . .	6-11
6.2	Multiprocessor Configuration CSR Definitions . . . . .	6-13
6.2.1	Processor Mailbox Register (CSR10) . . . . .	6-13
6.2.2	Interprocessor Interrupt Request Register (CSR11) . . . . .	6-14
6.3	System Interrupt Clear Register (CSR12) . . . . .	6-14
6.4	Address Lock Register (CSR13) . . . . .	6-15
6.5	Miss Address Register (CSR14) . . . . .	6-17
6.6	C <sup>3</sup> Revision Register (CSR15) . . . . .	6-18
6.7	D-bus . . . . .	6-19
6.8	System Bus Arbiter . . . . .	6-22
6.9	System Bus Clock Generator/Distributor . . . . .	6-22
6.10	System Bus CRESET L Generation . . . . .	6-22
6.11	OCF Halt Request Buffer . . . . .	6-23
6.12	Nonvolatile EEPROM . . . . .	6-23

## 7 CPU Module Transactions

7.1	Processor Transactions . . . . .	7-1
7.1.1	21064 CPU Chip Transactions . . . . .	7-2
7.1.1.1	Fast External Cache Read Hit . . . . .	7-3
7.1.1.2	Fast External Cache Write Hit . . . . .	7-3
7.1.1.3	READ_BLOCK . . . . .	7-4
7.1.1.4	Write_block . . . . .	7-5
7.1.1.5	LDxL . . . . .	7-6
7.1.1.6	STxC . . . . .	7-6
7.1.1.7	BARRIER . . . . .	7-6
7.1.1.8	FETCH . . . . .	7-7
7.1.1.9	FETCHM . . . . .	7-7
7.1.2	Cacheable versus Non-Cacheable, and Allocate-Invalid . . . . .	7-7
7.2	System Bus Transactions . . . . .	7-9
7.2.1	CPU as Commander . . . . .	7-10
7.2.2	CPU as Bystander . . . . .	7-10
7.2.3	CPU as Responder . . . . .	7-10
7.3	Control Flow of CPU Module Transactions . . . . .	7-10
7.3.1	Processor Initiated Transactions . . . . .	7-10
7.3.2	System Bus Initiated Transactions . . . . .	7-15

## 8 Cache Invalidate Management

8.1	Processor Caused Invalidates . . . . .	8-1
8.2	System Bus Caused Invalidates . . . . .	8-1

## 9 Exceptions and Interrupts

9.1	Processor-Generated Interrupts	9-1
9.1.1	Exception Handling	9-2
9.1.1.1	PAL Priority Level	9-3
9.1.1.2	PALcode 0020 Entry Characteristics	9-3
9.1.1.3	Parse Tree PALcode Entry 0020 <sub>16</sub>	9-3
9.1.1.4	PAL Routine Behavior	9-7
9.1.1.4.1	Backup Cache Tag Parity Error	9-7
9.1.1.4.2	Backup Cache Tag Control Parity Error	9-7
9.1.1.4.3	Backup Cache Data Single Bit EDC Error	9-7
9.1.1.4.4	Backup Cache Data Uncorrectable EDC Error	9-7
9.1.1.4.5	Backup Cache Data Single Bit EDC Error	9-8
9.1.1.4.6	Backup Cache Data Uncorrectable EDC Error	9-8
9.1.1.4.7	21064 Data Bus Single-bit EDC Error	9-8
9.1.1.4.8	21064 Data Bus Uncorrectable EDC Error	9-8
9.1.1.4.9	Backup Cache Tag or Tag Control Parity Error	9-8
9.1.1.4.10	System Bus Parity Error	9-9
9.1.1.4.11	Invalid System Bus Address	9-9
9.1.1.4.12	Other CPU Errors	9-9
9.1.1.4.13	Main Memory Uncorrectable EDC Errors	9-9
9.2	Non-processor Generated Interrupts	9-9
9.2.1	Interrupt Handling	9-10
9.2.1.1	PAL Priority Level	9-10
9.2.1.2	PALcode 00E0 Entry Characteristics	9-10
9.2.1.3	Parse Tree PALcode Entry 00E0 <sub>16</sub>	9-11
9.2.1.4	Hardware 0 Hardware Error	9-15
9.2.1.5	Hardware 1 Local I/O	9-17
9.2.1.6	Hardware 2 Futurebus+	9-18
9.2.1.7	Hardware 3 Interprocessor	9-19
9.2.1.8	Hardware 4 Interval Timer	9-19
9.2.1.9	Hardware 5 System Events	9-20
9.2.1.10	Software X Interrupt	9-20
9.2.1.11	Serial Line Interrupt	9-20
9.2.1.12	Asynchronous System Trap Interrupt	9-20

## 10 Fault Management and Error Recovery

10.1	Processor Errors	10-1
10.2	Backup Cache Errors	10-1
10.2.1	Tag and Tag Control Store Parity Errors	10-1
10.2.2	Data Store EDC Errors	10-3
10.2.2.1	Correctable	10-3
10.2.2.2	Uncorrectable Errors	10-6
10.3	Duplicate Primary Cache Tag Store Parity Errors	10-8
10.4	System Bus Errors	10-9
10.4.1	C/A Parity Error	10-9
10.4.2	Data Parity Error	10-9
10.4.3	Invalid Address - Bus Time-out	10-10
10.5	I/O Subsystem Errors	10-10
10.6	C_ERR L Assertion	10-10

## 11 CPU Power-Up and Initialization

11.1	Internal Processor Registers and the Internal JSR Stack	11-1
11.2	Backup Cache Initialization	11-1
11.2.1	LDQ Data Format, BCC ENABLE BACKUP CACHE INIT Set	11-2
11.3	Duplicate Tag Store Initialization	11-3
11.4	System Bus Interface Initialization	11-3
11.5	CPU Clocks and Reset	11-3
11.6	Power-Up Sequence	11-3
11.7	Powering Up with Bad Main Memory	11-3

## Part III The I/O Module

### 12 The KFA40 I/O Module

### 13 Address Mapping

13.1	System Memory Space	13-1
13.2	Primary I/O Space	13-2
13.3	Diagnostic Mode Address Registers	13-5
13.4	I/O Control/Status Register (IOCSR)	13-6
13.5	System Bus Error Register 1 (CERR1)	13-11
13.6	System Bus Error Register 2 (CERR2)	13-17
13.7	System Bus Error Register 3 (CERR3)	13-18
13.8	Lbus Mailbox Pointer Register (LMBPR)	13-18
13.9	Futurebus+ Mailbox Pointer Register (FMBPR)	13-19
13.10	Diagnostic Control/Status Register (DIAGCSR)	13-20
13.11	Futurebus+ Interrupt Vector Register (FIVECT)	13-21
13.12	Futurebus Halt Vector Register (FHVECT)	13-22
13.13	Futurebus Error Register 1 (FERR1)	13-22
13.14	Futurebus Error Register 2 (FERR2)	13-23
13.15	Local Interrupt Register (LINT)	13-24
13.16	Lbus Error Register 1 (LERR1)	13-26
13.17	Lbus Error Register 2 (LERR2)	13-27
13.18	Secondary I/O Space	13-28
13.18.1	Mailbox Data Structure	13-28
13.18.2	Software-visible Control Flow	13-29
13.18.3	Mailbox Pointer Registers	13-29
13.18.4	Mailbox Operation	13-36

### 14 Interrupts

14.1	Futurebus+ Interrupts	14-1
14.2	Local I/O Interrupts	14-2
14.3	Hardware Error Interrupts	14-2
14.4	System Event Interrupt Requests	14-3



## 15 Futurebus+ Adapter Architecture

15.1	Futurebus+ Address Space . . . . .	15-1
15.2	Futurebus+ Adapter Registers . . . . .	15-3
15.3	Futurebus+ Interrupt Request Register (FIRQ) . . . . .	15-4
15.4	Futurebus+ Halt Request Register (FHRQ) . . . . .	15-4
15.5	Supported Data Transfer Protocols . . . . .	15-5
15.5.1	Futurebus+ Slave . . . . .	15-5
15.5.2	Futurebus+ Master . . . . .	15-5
15.6	Futurebus+ Error Handling . . . . .	15-6

## 16 Local I/O Devices

16.1	Lbus Addressing . . . . .	16-1
16.2	EEPROM . . . . .	16-2
16.3	Serial Line Ports . . . . .	16-2
16.4	AUX_CTRL Register (ACR) . . . . .	16-4
16.5	AUX_Data Register (AD) . . . . .	16-4
16.6	Console Control Register (CON_CTRL) . . . . .	16-5
16.7	Console Data Register (CD) . . . . .	16-5
16.8	Time-of-Year Clock (TOY) . . . . .	16-6
16.9	TOY Register A (TOYA) . . . . .	16-7
16.10	TOY Register B (TOYB) . . . . .	16-9
16.11	TOY Register C (TOYC) . . . . .	16-11
16.12	TOY Register D (TOYD) . . . . .	16-12

## 17 Ethernet Adapters (TGEC)

17.1	TGEC Programming . . . . .	17-2
17.1.1	Programming Overview . . . . .	17-2
17.1.2	TGEC Command and Status Registers . . . . .	17-3
17.1.3	Host Access to CSRs . . . . .	17-3
17.1.3.1	TGEC Physical CSRs . . . . .	17-4
17.1.3.2	TGEC Virtual CSRs . . . . .	17-4
17.1.3.2.1	CSR Write . . . . .	17-4
17.1.3.2.2	CSR Read . . . . .	17-4
17.1.4	Vector Address, IPL, Sync/Asynch (CSR0) . . . . .	17-5
17.1.5	Transmit Polling Demand (CSR1) . . . . .	17-7
17.1.6	Receive Polling Demand (CSR2) . . . . .	17-8
17.1.7	Descriptor List Addresses (CSR3, CSR4) . . . . .	17-9
17.1.8	Status Register (CSR5) . . . . .	17-12
17.1.8.1	CSR5 Status Report . . . . .	17-15
17.1.9	Command and Mode Register (CSR6) . . . . .	17-17
17.1.10	System Base Register (CSR7) . . . . .	17-22
17.1.11	Reserved Register (CSR8) . . . . .	17-23
17.1.12	Watchdog Timers (CSR9) . . . . .	17-23
17.1.13	TGEC Identification and Missed Frame Count (CSR10) . . . . .	17-25
17.1.14	Boot Message (CSR11, 12, 13) . . . . .	17-26
17.1.15	Diagnostic Registers (CSR14, 15) . . . . .	17-28
17.2	Descriptors and Buffers Format . . . . .	17-28

17.2.1	Receive Descriptors .....	17-28
17.2.1.1	RDES0 Word .....	17-29
17.2.1.2	RDES1 Word .....	17-31
17.2.1.3	RDES2 Word .....	17-32
17.2.1.4	RDES3 Word .....	17-32
17.2.1.5	Receive Descriptor Status Validity .....	17-33
17.2.2	Transmit Descriptors .....	17-33
17.2.2.1	TDES0 Word .....	17-34
17.2.2.2	TDES1 word .....	17-35
17.2.2.3	TDES2 Word .....	17-37
17.2.2.4	TDES3 word .....	17-37
17.2.2.5	Transmit Descriptor Status Validity .....	17-37
17.2.3	Setup Frame .....	17-38
17.2.3.1	First Setup Frame .....	17-38
17.2.3.2	Subsequent Setup Frame .....	17-38
17.2.3.3	Setup Frame Descriptor .....	17-38
17.2.3.4	Perfect Filtering Setup Frame Buffer .....	17-40
17.2.3.5	Imperfect Filtering Setup Frame Buffer .....	17-42
17.3	TGEC Hardware and Software Reset .....	17-44
17.4	TGEC Interrupts .....	17-45
17.4.1	Startup Procedure .....	17-46
17.5	Reception Process .....	17-46
17.5.1	Transmission Process .....	17-48
17.5.2	Loopback Operations .....	17-50
17.5.3	DNA CSMA/CD Counters and Events Support .....	17-51

## 18 SCSI/DSSI Adapters

18.1	NCR 53C710 SCSI Adapter Registers .....	18-1
18.1.1	SCSI Register Map .....	18-2
18.2	SCSI Control 0 (SCNTL0) .....	18-4
18.3	SCSI Control 1 (SCNTL1) .....	18-6
18.4	SCSI Destination ID (SDID) .....	18-8
18.5	SCSI Interrupt Enable (SIEN) .....	18-8
18.6	SCSI Chip ID (SCID) .....	18-9
18.7	SCSI Transfer (SXFER) .....	18-10
18.8	SCSI Output Data Latch (SODL) .....	18-13
18.9	SCSI Output Control Latch (SOCL) .....	18-13
18.10	SCSI First Byte Received (SFBR) .....	18-14
18.11	SCSI Input Data Latch (SIDL) .....	18-15
18.12	SCSI Bus Data Lines (SBDL) .....	18-16
18.13	SCSI Bus Control Lines (SBCL) .....	18-16
18.14	DMA Status (DSTAT) .....	18-17
18.15	SCSI Status Zero (SSTAT0) .....	18-19
18.16	SCSI Status One (SSTAT1) .....	18-21
18.17	SCSI Status Two (SSTAT2) .....	18-22
18.18	Data Structure Address (DSA) .....	18-23
18.19	Chip Test Zero (CTEST0) .....	18-24
18.20	Chip Test One (CTEST1) .....	18-25
18.21	Chip Test Two (CTEST2) .....	18-26
18.22	Chip Test Three (CTEST3) .....	18-27
18.23	Chip Test Four (CTEST4) .....	18-28
18.24	Chip Test Five (CTEST5) .....	18-29
18.25	Chip Test Six (CTEST6) .....	18-31

18.26	Chip Test Seven (CTEST7)	18-32
18.27	Temporary Stack (TEMP)	18-33
18.28	DMA FIFO (DFIFO)	18-33
18.29	Interrupt Status (ISTAT)	18-34
18.30	Chip Test Eight (CTEST8)	18-37
18.31	Longitudinal Parity (LCRC)	18-38
18.32	DMA Byte Counter (DBC)	18-39
18.33	DMA Command (DCMD)	18-39
18.34	DMA Next Data Address (DNAD)	18-40
18.35	DMA Scripts Pointer (DSP)	18-40
18.36	DMA Scripts Pointer Save (DSPS)	18-41
18.37	Scratch Register (SCRATCH)	18-41
18.38	DMA Mode (DMODE)	18-42
18.39	DMA Interrupt Enable (DIEN)	18-43
18.40	DMA Watchdog Timer (DWT)	18-44
18.41	DMA Control Register (DCNTL)	18-45
18.42	Adder Sum Output (ADDER)	18-47
18.42.1	SCSI Script RAM Buffer	18-47
18.42.2	Ethernet Station Address ROM	18-48
18.42.3	Serial Control Bus Interface	18-48
18.42.4	FEPRM	18-48

## Part IV The System Bus

### 19 The DEC 4000 System Bus

19.1	Supported Transactions	19-1
19.2	Address Space	19-1
19.3	System Bus Transactions	19-2
19.3.1	Bus Arbiter	19-2
19.3.2	Transaction Process	19-2
19.4	Cache Protocol	19-3
19.5	System Bus Signals	19-6
19.5.1	Command and Address Format	19-9
19.5.2	Arbitration Signals	19-9
19.5.3	Protocol Signals	19-10
19.5.4	Address Field	19-11
19.5.5	Response and Interrupt Signals	19-14
19.5.6	Clocking and Initialization Signals	19-18
19.5.6.1	Synchronous Clock Signals	19-18
19.5.7	Initialization Signals	19-18
19.6	System Bus Transactions and Timing	19-19
19.6.1	Read and Read Exclusive Transactions	19-20
19.6.2	Null Transactions	19-24
19.6.3	Memory Exchange Transactions	19-26
19.6.4	Memory Write Transactions	19-28
19.6.5	Noncacheable Address Space Write Transactions	19-30

## Part V The Firmware

### 20 System State Transitions

#### 21 The Power-Up and Initialization State

21.1	The Powered Off State .....	21-2
21.2	The Power-up Process .....	21-2
21.2.1	AC Power-Up .....	21-4
21.2.2	DC Power-Up .....	21-4
21.2.3	ASYNC_RESET_L and CRESET_L .....	21-5
21.3	System Initialization .....	21-6
21.3.1	FEPROM Unloading .....	21-10
21.3.2	Console Initialization .....	21-12
21.3.3	Memory Testing .....	21-13
21.3.3.1	Memory Testing and the Bitmap .....	21-15
21.3.4	Driver Initialization .....	21-15
21.3.4.1	I/O Adapter Configuration .....	21-17

#### 22 The Halted State

22.1	The HALT command .....	22-2
22.2	The HALT Switch on the OCP .....	22-2
22.3	System Error .....	22-2

#### 23 The Bootstrap State

23.1	DEC 4000 Bootstrap Algorithm .....	23-2
23.2	Environment Variables .....	23-2
23.3	Hardware Restart Parameter Block .....	23-3
23.4	Console Callback Routines .....	23-3
23.5	Boot Block Calling Interface .....	23-4
23.6	Boot Devices .....	23-5
23.7	Boot Parameters .....	23-5
23.8	Disk and Tape Booting .....	23-6
23.9	Ethernet Booting .....	23-6
23.9.1	MOP Booting .....	23-7
23.9.1.1	MOP Network "Listening" .....	23-7
23.9.2	BOOTP booting .....	23-8
23.9.2.1	Setting up the Console for BOOTP Boots .....	23-8
23.9.2.2	Using BOOTP bootstrap .....	23-10

#### 24 The Restart State

24.1	Restart Failure .....	24-2
24.2	Restart Parameters .....	24-3

## 25 Console

25.1	Console Prompt	25-1
25.2	Command Conventions	25-1
25.3	Console Special Characters	25-2
25.4	Environment Variables	25-3
25.5	Command Overview	25-3
25.6	Getting Information About the System	25-4
25.7	Online HELP	25-5
25.8	Examining and Depositing	25-7
25.8.1	Accessing Memory	25-8
25.8.2	Examining registers	25-9
25.9	Using Pipes ( ) and grep to Filter Output	25-10
25.10	Using I/O Redirection (>)	25-11
25.11	Running Commands in the Background "&"	25-11
25.12	Monitoring Status	25-12
25.13	Killing a Process	25-13
25.14	Creating Scripts	25-13
25.15	Using Flow Control	25-14
25.16	Console Shell	25-16

## 26 Diagnostics

26.1	Diagnostic Overview	26-1
26.1.1	Diagnostic Classes	26-1
26.1.2	Diagnostic User Interface	26-2
26.1.2.1	Starting Diagnostic Tests and Exercisers	26-2
26.1.2.2	Diagnostic Control Flags	26-2
26.1.2.2.1	Global Diagnostic Environment Variables	26-3
26.1.2.2.2	Local Diagnostic Command Qualifiers for most diagnostics	26-3
26.1.2.2.3	Local Diagnostic Command Qualifiers for specific diagnostics	26-3
26.1.2.3	Monitoring the Exerciser Status	26-3
26.1.2.4	Terminating Diagnostic Tests and Exercisers	26-4
26.1.3	Diagnostic Output Displays	26-4
26.1.3.1	Operator Control Panel LED Displays	26-4
26.1.3.2	Console Terminal Error Messages	26-5
26.1.4	Startup Message	26-6
26.1.5	Completion Message	26-7
26.1.6	End-of-pass Message	26-7
26.1.7	Test Trace Message	26-8
26.1.8	Diagnostic Power-up Flow	26-8
26.2	CPU/Cache Subsystem	26-9
26.2.1	Diagnostic Strategy	26-9
26.2.2	Power-up Strategy	26-9
26.2.3	Fault Architecture	26-9
26.2.4	Interpreting Error Printouts	26-9
26.2.5	Error Message Display	26-9
26.2.6	Test name: B-Cache init	26-10
26.2.7	Test name: SRAM Address Test	26-10
26.2.8	Test name: SRAM Tag Store Test	26-11
26.2.9	Test name: SRAM B-Cache Data Line Test	26-12
26.2.10	Test name: SRAM ECC test	26-13
26.2.11	Test name: CPU Spin on Processor Mailbox	26-14
26.2.12	Test name: Cbus (System Bus) test	26-14

26.2.13	Test name: Disable failing CPU .....	26-15
26.2.14	Test name: Memory test .....	26-15
26.2.15	Test name: Unload .....	26-16
26.3	Memory Subsystem .....	26-19
26.3.1	Initialization .....	26-19
26.3.1.1	Initialization Tests .....	26-19
26.3.1.2	Memory Configuration .....	26-20
26.3.1.3	Failure Reporting and Logging .....	26-21
26.3.2	Exerciser Tests .....	26-21
26.4	SCSI/DSSI Subsystem .....	26-21
26.4.1	Expected Failures .....	26-22
26.4.2	Diagnostic Strategy .....	26-22
26.4.3	Power-up Tests .....	26-22
26.4.3.1	IO Module Self Tests .....	26-23
26.4.3.2	Driver Startup/Sizing .....	26-23
26.4.3.3	Disk/Tape Unit Self Tests .....	26-23
26.4.4	Functional Tests .....	26-23
26.4.5	Exerciser Tests .....	26-24
26.5	Network Subsystem .....	26-24
26.5.1	Initialization .....	26-24
26.5.1.1	Driver Initialization Tests .....	26-24
26.5.2	Exerciser Tests .....	26-25
26.6	Futurebus+ Subsystem .....	26-25
26.6.0.1	Hardware Overview .....	26-25
26.7	Futurebus+ Devices .....	26-26
26.7.1	Hardware Overview .....	26-26
26.7.1.1	Diagnostic Strategy .....	26-26
26.7.2	Futurebus+ Sizing .....	26-26
26.8	SLU Subsystem .....	26-27
26.8.1	Overview .....	26-27
26.8.1.1	Hardware Overview .....	26-27
26.8.2	85C30 Register Test .....	26-27
26.8.3	85C30 Internal Loop-back Test .....	26-28
26.8.4	85C30 External Loop-back Test .....	26-28
26.9	Multi-Processor .....	26-28
26.10	Serial Control Bus Subsystem .....	26-29
26.10.1	Controller Register Test (iic_reg_test) .....	26-29
26.10.2	Bus Access Test (iic_acc_test) .....	26-29

## A IPR State on Power-up and RESET

A.1	TB Miss Flows .....	A-2
A.1.1	ITB Miss .....	A-3
A.1.2	DTB Miss .....	A-3

## B Error Flows

B.1	I-stream ECC error .....	B-1
B.2	D-stream ECC error .....	B-1
B.3	BIU: tag address parity error .....	B-2
B.4	BIU: tag control parity error .....	B-2
B.5	BIU: system external transaction terminated with CACK_SERR .....	B-2
B.6	BIU: system transaction terminated with CACK_HERR .....	B-2
B.7	BIU: I-stream parity error (parity mode only) .....	B-2

B.8	BIU: D-stream parity error (parity mode only) .....	B-3
-----	---	-----

## C AC and DC Characteristics

## D Console Command Repository

alloc .....	D-2
bin .....	D-3
boot .....	D-4
cat .....	D-6
cbcc .....	D-7
cdp .....	D-9
check .....	D-11
chmod .....	D-12
chown .....	D-13
clear .....	D-14
cmp .....	D-15
continue .....	D-17
crc .....	D-18
date .....	D-19
deposit .....	D-20
dynamic .....	D-23
echo .....	D-25
edit .....	D-26
eval .....	D-29
examine .....	D-31
exer .....	D-34
exer_read .....	D-40
exer_write .....	D-41
exit .....	D-42
fbus_diag .....	D-43
fbus_sizer .....	D-45
find_field .....	D-46
free .....	D-47
grep .....	D-48
hd .....	D-50
help or man .....	D-51
initialize .....	D-53
io_diag .....	D-54
kill .....	D-55
kill_diags .....	D-56
line .....	D-57
ls .....	D-58
memexer .....	D-59
memexer_mp .....	D-60
memtest .....	D-61
net .....	D-64

nettest .....	D-67
ps .....	D-70
psc .....	D-71
rm .....	D-72
sa .....	D-73
semaphore .....	D-74
set .....	D-75
set host .....	D-77
sh .....	D-79
show .....	D-80
show cluster .....	D-83
show config .....	D-84
show device .....	D-85
show error .....	D-87
show fru .....	D-89
show hwrpb .....	D-90
show map .....	D-91
show memory .....	D-92
show_status .....	D-93
sleep .....	D-94
sort .....	D-95
sp .....	D-96
start .....	D-97
stop .....	D-98
sw .....	D-99
test .....	D-100
tr .....	D-102
uniq .....	D-103
update .....	D-104
wc .....	D-106

## E Environment Variables

### Glossary

### Index

### Examples

5-1	Update versus Invalidate Algorithm .....	5-19
17-1	Perfect Filtering Buffer .....	17-42
17-2	Imperfect Filtering Buffer .....	17-44



## Figures

1	Timing Diagram Conventions . . . . .	xxxiv
1-1	System Block Diagram . . . . .	1-4
1-2	The DEC 4000 System Front View . . . . .	1-5
1-3	BA640 Enclosure Views . . . . .	1-6
1-4	Operator Control Panel . . . . .	1-7
1-5	The OCP Circuit Board Block Diagram . . . . .	1-8
1-6	The Backplane Assembly . . . . .	1-10
1-7	Storage Backplane . . . . .	1-12
1-8	System Backplane . . . . .	1-14
1-9	KN430 CPU Subsystem Block Diagram . . . . .	1-16
1-10	CPU Subsystem Module Order . . . . .	1-17
1-11	KN430 Processor Module Functional Block Diagram . . . . .	1-19
1-12	The KN430 Processor Module . . . . .	1-20
1-13	I/O Module Functional Diagram . . . . .	1-23
1-14	The KFA40 I/O Module . . . . .	1-24
1-15	MS430 Module Functional Block Diagram . . . . .	1-26
1-16	The MS430 Memory Module . . . . .	1-28
1-17	Physical Address Space Layout in the System . . . . .	1-32
1-18	The Mass Storage Area . . . . .	1-35
1-19	Mass Storage Front Panels . . . . .	1-36
1-20	I/O Expansion Area . . . . .	1-37
1-21	The Power Supply Indicators . . . . .	1-38
1-22	The DEC 4000 Power Scheme . . . . .	1-40
1-23	System Firmware Locations . . . . .	1-44
3-1	Byte Data Format . . . . .	3-1
3-2	Word Data Format . . . . .	3-2
3-3	Longword Data Format . . . . .	3-2
3-4	Quadword Data Format . . . . .	3-2
3-5	F_floating Data Format . . . . .	3-3
3-6	F_floating Register . . . . .	3-3
3-7	G_floating Operand . . . . .	3-4
3-8	G_floating Data Format . . . . .	3-4
3-9	D_floating Data Format . . . . .	3-5
3-10	D_floating Register Format . . . . .	3-5
3-11	S_floating operand . . . . .	3-6
3-12	S_floating Register Format . . . . .	3-6
3-13	T_floating Datum . . . . .	3-8
3-14	T_floating Register Format . . . . .	3-8
3-15	Longword Integer Datum . . . . .	3-9
3-16	Longword Integer Floating-Register Format . . . . .	3-9
3-17	Quadword Integer Datum . . . . .	3-10
3-18	Quadword Integer Floating-Register Format . . . . .	3-10
4-1	TB_TAG Register Format . . . . .	4-3
4-2	ITB_PTE Register Format . . . . .	4-4
4-3	ICCSR Register Format . . . . .	4-5

4-24	ITB_PTE_TEMP Register (ITB_PTE_TEMP) . . . . .	4-7
4-5	Exception Address Register (EXC_ADDR) . . . . .	4-9
4-6	Serial Line Clear Register (SL_CLR) . . . . .	4-9
4-7	Serial Line Receive Register (sl_rcv) . . . . .	4-10
4-8	Processor State (PS) Register Format . . . . .	4-11
4-9	Exception Summary Register (EXC_SUM) . . . . .	4-12
4-10	PAL Base Address Register (PAL_BASE) . . . . .	4-13
4-11	Hardware Interrupt Request Register (HIRR) . . . . .	4-14
4-12	SIRR Register Format . . . . .	4-15
4-13	ASTRR Register Format . . . . .	4-16
4-14	HIER Format . . . . .	4-17
4-15	SIER Format . . . . .	4-17
4-16	ASTER Format . . . . .	4-18
4-17	Serial Line Transmit Register (SL_XMIT) . . . . .	4-18
4-18	TB Control Register (TB_CTL) . . . . .	4-19
4-19	DTB_PTE Register Format . . . . .	4-20
4-20	DTB_PTE_TEMP Register Format . . . . .	4-21
4-21	MM CSR (MM_CSR) . . . . .	4-21
4-22	A-box Control (Abox_CTL) . . . . .	4-23
4-23	ALT_MODE Register Format . . . . .	4-24
4-24	Bus Interface Control Unit Register (ITB_PTE_TEMP) . . . . .	4-26
4-25	DC_STAT Register Format . . . . .	4-29
4-26	BIU_STAT Register Format . . . . .	4-31
4-27	Fill Syndrome Register Format . . . . .	4-34
4-28	BC_TAG Register Format . . . . .	4-36
5-1	Back-up Cache Entry . . . . .	5-1
5-2	Backup Cache Control Register (BCC) . . . . .	5-4
5-3	Backup Cache Correctable Error Register (BCCE) . . . . .	5-9
5-4	Backup Cache Correctable Error Address Register (BCCEA) . . . . .	5-11
5-5	Backup Cache Uncorrectable Error Register (BCUE) . . . . .	5-13
5-6	Backup Cache Uncorrectable Error Address Register (BCUEA) . . . . .	5-15
5-7	Duplicate Tag Error Register (DTER) . . . . .	5-20
6-1	System Bus Control Register (CBCTL) . . . . .	6-2
6-2	System Bus Error Register (CBE) . . . . .	6-6
6-3	System Bus Error Address Low Register (CBEAL) . . . . .	6-10
6-4	System Bus Error Address High Register (CBEAH) . . . . .	6-11
6-5	Processor Mailbox Register (PMBX) . . . . .	6-13
6-6	Interprocessor Interrupt Request Register (IPIR) . . . . .	6-14
6-7	System Interrupt Clear Register (SIC) . . . . .	6-15
6-8	Address Lock Register (ADLK) . . . . .	6-16
6-9	Miss Address Register Low (MADRL) . . . . .	6-17
6-10	C <sup>3</sup> Revision Register (CRR) . . . . .	6-18
6-11	21064 Serial Load Data Format . . . . .	6-20
7-1	Address Space Map . . . . .	7-9
9-1	Machine-check Exception Parse Tree . . . . .	9-5
9-2	Interrupt Parse Tree . . . . .	9-12
9-3	Local I/O Interrupt Flow . . . . .	9-18

9-4	Futurebus+ Interrupt Flow .....	9-19
11-1	LDQ Data Format (LDF) .....	11-2
13-1	System Bus Address Map .....	13-1
13-2	I/O Module Register Map .....	13-2
13-3	Lbus Diagnostic Mode Register Map .....	13-3
13-4	Futurebus+ Diagnostic Mode Register Map .....	13-4
13-5	Diagnostic Address Register Map .....	13-5
13-6	I/O Control/Status Register (IOCSR) .....	13-7
13-7	System Bus Error Register 1 (CERR1) .....	13-12
13-8	System Bus Error Register 2 (CERR2) .....	13-18
13-9	System Bus Error Register 3 (CERR3) .....	13-18
13-10	Lbus Mailbox Pointer Register (LMBPR) .....	13-19
13-11	Futurebus+ Mailbox Pointer Register (FMBPR) .....	13-20
13-12	Diagnostic Control/Status Register (DIAGCSR) .....	13-20
13-13	Futurebus Interrupt Vector Register (FIVECT) .....	13-21
13-14	Futurebus Halt Vector Register (FHVECT) .....	13-22
13-15	Futurebus Error Register 1 (FERR1) .....	13-23
13-16	Futurebus Error Register 2 (FERR2) .....	13-24
13-17	Local Interrupt Register (LINT) .....	13-25
13-18	Lbus Error Register 1 (LEERR1) .....	13-26
13-19	Lbus Error Register 2 (LEERR2) .....	13-27
13-20	Mailbox Mechanism .....	13-30
13-21	Futurebus+ Mailbox Command Field (FCMD) .....	13-31
13-22	Futurebus+ Mailbox Status Field (FSTAT) .....	13-32
13-23	Lbus Mailbox Command Field (LCMD) .....	13-34
13-24	Lbus Mailbox Status Field (LSTAT) .....	13-36
15-1	A32 Futurebus+ Address Space .....	15-2
15-2	A64 Futurebus+ Address Space .....	15-2
15-3	Futurebus+ Node Addressing .....	15-3
15-4	Futurebus+ Interrupt Request Register (firq) .....	15-4
15-5	Futurebus+ Halt Request Register (fhrq) .....	15-4
16-1	Serial Line Unit Register Map .....	16-3
16-2	AUX_CTRL Register (acr) .....	16-4
16-3	AUX_Data Register (ad) .....	16-4
16-4	Console Control Register (CCTRL) .....	16-5
16-5	Console Data Register (CD) .....	16-5
16-6	TOY Clock Register Map .....	16-7
16-7	TOY Register A (toyA) .....	16-7
16-8	TOY Register B (toyB) .....	16-9
16-9	TOY Register C (toyC) .....	16-11
16-10	TOY Register D (TOYD) .....	16-12
17-1	TGEC Register Map .....	17-2
17-2	Vector Address, IPL, Sync/Asynch (CSR0) .....	17-5
17-3	Transmit Polling Demand (CSR1) .....	17-7
17-4	Receive Polling Demand (CSR2) .....	17-8
17-5	Descriptor List Addresses Format .....	17-10
17-6	Status Register 5 (CSR5) .....	17-12

17-7	Command and Mode Register (CSR6) .....	17-17
17-8	System Base Register (CSR7) .....	17-22
17-9	Watchdog Timer (CSR9) .....	17-23
17-10	TGEC Identification and Missed Frame Count (CSR10) .....	17-25
17-11	Boot Message Registers Format .....	17-26
17-12	Receive descriptor format .....	17-29
17-13	Transmit Descriptor Format .....	17-34
17-14	Setup frame descriptor format .....	17-38
17-15	Perfect Filtering setup frame buffer format .....	17-41
17-16	Imperfect Filtering Setup Frame Format .....	17-43
18-1	SCSI/DSSI Controller Register Map .....	18-3
18-2	SCSI Control 0 (SCNTL0) .....	18-4
18-3	SCSI Control 1 (SCNTL1) .....	18-7
18-4	SCSI Destination ID (SDID) .....	18-8
18-5	SCSI Interrupt Enable (SIEN) .....	18-9
18-6	SCSI Chip ID (SCID) .....	18-10
18-7	SCSI Transfer (SXFER) .....	18-10
18-8	SCSI Output Data Latch ( SODL) .....	18-13
18-9	SCSI Output Control Latch (SOCL) .....	18-14
18-10	SCSI First Byte Received ( SFBR) .....	18-15
18-11	SCSI Input Data Latch (SIDL) .....	18-15
18-12	SCSI Bus Data Lines (SBDL) .....	18-16
18-13	SCSI Bus Control Lines (SBCL) .....	18-17
18-14	DMA Status (DSTAT) .....	18-18
18-15	SCSI Status 0 (SSTAT0) .....	18-19
18-16	SCSI Status One (SSTAT1) .....	18-21
18-17	SCSI Status Two (SSTAT2) .....	18-23
18-18	Chip Test Zero (CTEST0) .....	18-24
18-19	Chip Test One (CTEST1) .....	18-25
18-20	Chip Test Two (CTEST2) .....	18-26
18-21	Chip Test Three (CTEST3) .....	18-27
18-22	Chip Test Four (CTEST4) .....	18-28
18-23	Chip Test Five (CTEST5) .....	18-30
18-24	Chip Test Six (CTEST6) .....	18-31
18-25	Chip Test Seven (CTEST7) .....	18-32
18-26	DMA FIFO (DFIFO) .....	18-34
18-27	Interrupt Status (ISTAT) .....	18-35
18-28	Chip Test Eight (CTEST8) .....	18-37
18-29	Longitudinal Parity (LCRC) .....	18-38
18-30	DMA Byte Counter (DBC) .....	18-39
18-31	DMA Command (DCMD) .....	18-39
18-32	DMA Next Data Address (DNAD) .....	18-40
18-33	DMA Scripts Pointer (DSP) .....	18-40
18-34	DMA Scripts Pointer Save (DSPS) .....	18-41
18-35	Scratch Register (SCRATCH) .....	18-41
18-36	DMA Mode (DMODE) .....	18-42
18-37	DMA Interrupt Enable (DIEN) .....	18-44

18-38	DMA Watchdog Timer (DWT) .....	18-45
18-39	DMA Control Register (DCNTL) .....	18-45
18-40	Adder Sum Output (ADDER) .....	18-47
18-41	Serial Control Bus Interface Register Map .....	18-48
19-1	Protocol Transition Summary .....	19-5
19-2	Command-Address Cycle Format, Data Longword Shuffle .....	19-7
19-3	Exchange Address Layout .....	19-12
19-4	System Bus Read Timing .....	19-23
19-5	System Bus NULL Transaction Timing .....	19-25
19-6	System Bus Memory Exchange Timing .....	19-27
19-7	System Bus Write Timing .....	19-29
19-8	System Bus Noncacheable Address Space Write Timing .....	19-31
20-1	DEC 4000 System State Transitions .....	20-3
21-1	Initialization Block Diagram .....	21-3
21-2	Initialization Flow Diagram .....	21-8
21-3	Initialization Flow Diagram .....	21-10
21-4	Initialization Flow Diagram .....	21-12
23-1	Alpha AXP Boot Block .....	23-4
26-1	Diagnostic Status Display .....	26-4
26-2	Example Error Message .....	26-5
26-3	Startup Message .....	26-6
26-4	Completion Message .....	26-7
26-5	End of Pass Message .....	26-7
26-6	Test Trace Message .....	26-8
26-7	Test 02 sample printout .....	26-11

## Tables

1	Conventions .....	xxxii
2	Bit Type Conventions .....	xxxiii
1-1	System Features Summary .....	1-3
1-2	PCF8574 ADDRESS 42/43 BAUD RATE BIT DEFINITIONS .....	1-9
1-3	Storage Backplane Connectors .....	1-11
1-4	System Backplane Connectors .....	1-13
1-5	CPU differences .....	1-18
1-6	Serial Control Bus Node Addresses .....	1-43
1-7	Serial Line Features .....	1-47
3-1	Alpha F_floating Load Exponent Mapping .....	3-3
3-2	S_floating Load Exponent Mapping .....	3-7
4-1	ICCSR Bits .....	4-6
4-2	BHE, BPE Branch Prediction Selection .....	4-7
4-3	ITB_PTE_TEMP Register Description .....	4-7
4-4	Exception Address Register Description .....	4-9
4-5	Serial Line Clear Register Description .....	4-9
4-6	Serial Line Receive Register Description .....	4-10
4-7	Exception Summary Register Description .....	4-12
4-8	PAL Base Address Register Description .....	4-13

4-9	Hardware Interrupt Request Register Description . . . . .	4-14
4-10	Serial Line Transmit Register Description . . . . .	4-18
4-11	TB Control Register Description . . . . .	4-19
4-12	MM CSR Description . . . . .	4-21
4-13	A-box Control Description . . . . .	4-23
4-14	ALT Mode . . . . .	4-25
4-15	Bus Interface Control Unit Register Description . . . . .	4-26
4-16	Backup Cache Size . . . . .	4-28
4-17	BC_PA_DIS . . . . .	4-28
4-18	Data cache Status Register . . . . .	4-29
4-19	Data cache status Error Modifiers . . . . .	4-30
4-20	BIU STAT . . . . .	4-32
4-21	Syndromes for Single-Bit Errors . . . . .	4-35
5-1	Cache Block Status Flags . . . . .	5-1
5-2	Backup Cache Control Register Description . . . . .	5-5
5-3	Backup Cache Correctable Error Register Description . . . . .	5-9
5-4	Backup Cache Correctable Error Address Register Description . . . . .	5-11
5-5	Backup Cache Uncorrectable Error Register Description . . . . .	5-13
5-6	Backup Cache Uncorrectable Error Address Register Description . . . . .	5-15
5-7	System Bus Backup Cache Access Time . . . . .	5-17
5-8	Duplicate Tag Error Register Description . . . . .	5-20
5-9	Data Integrity Reference . . . . .	5-22
6-1	System Bus Control Register Description . . . . .	6-2
6-2	System Bus Error Register Description . . . . .	6-7
6-3	System Bus Error Address Low Register Description . . . . .	6-10
6-4	System Bus Error Address High Register Description . . . . .	6-11
6-5	Processor Mailbox Register Description . . . . .	6-13
6-6	Interprocessor Interrupt Request Register Description . . . . .	6-14
6-7	System Interrupt Clear Register Description . . . . .	6-15
6-8	Address Lock Register Description . . . . .	6-16
6-9	Miss Address Register Low Description . . . . .	6-17
6-10	C <sup>3</sup> Revision Register Description . . . . .	6-18
6-11	D-bus Micro Port Mapping . . . . .	6-20
6-12	D-bus Microcontroller Clock Frequency . . . . .	6-21
6-13	D-bus Microcontroller System Control Bus Address . . . . .	6-22
6-14	Nonvolatile EEPROM System Control Bus Address . . . . .	6-23
7-1	Processor Initiated Transactions . . . . .	7-2
7-2	System Bus Initiated Transactions . . . . .	7-9
7-3	Processor Initiated Transactions Control Flow . . . . .	7-11
7-4	System Bus Initiated Transactions Control Flow . . . . .	7-16
8-1	Invalidate Management System Bus Caused . . . . .	8-2
9-1	General Exception Isolation Matrix . . . . .	9-1
9-2	Machine Check Isolation Matrix . . . . .	9-2
9-3	Exception Priority/PAL Offset/SCB Offset/IPL . . . . .	9-3
9-4	Hardware Interrupt Configuration . . . . .	9-9
9-5	Interrupt Priority/SCB Offset/IPL . . . . .	9-10
10-1	Tag/Tag Control Error Severity Matrix . . . . .	10-1

10-2	Backup Cache Tag Control or Tag Store Errors . . . . .	10-3
10-3	Backup Cache Data Correctable Errors . . . . .	10-5
10-4	Uncorrectable Data Store Error Severity Matrix . . . . .	10-6
10-5	Backup Cache Uncorrectable Errors . . . . .	10-7
10-6	C/A Parity Errors . . . . .	10-9
10-7	Data Parity Errors . . . . .	10-10
11-1	LDQ Data Format Description . . . . .	11-2
13-1	Diagnostic Mode Address Register Access . . . . .	13-6
13-2	I/O Control and Status Register Description . . . . .	13-8
13-3	System Bus Error Register 1 Description . . . . .	13-13
13-4	System Bus Error Register 2 Description . . . . .	13-18
13-5	System Bus Error Register 3 Description . . . . .	13-18
13-6	Lbus Mailbox Pointer Register Description . . . . .	13-19
13-7	Futurebus+ Mailbox Pointer Register Description . . . . .	13-20
13-8	Diagnostic Control/Status Register Description . . . . .	13-20
13-9	Futurebus Interrupt Vector Register Description . . . . .	13-22
13-10	Futurebus Halt Vector Register Description . . . . .	13-22
13-11	Futurebus Error Register 1 Description . . . . .	13-23
13-12	Futurebus Error Register 2 Description . . . . .	13-24
13-13	Local Interrupt Register Description . . . . .	13-25
13-14	Lbus Error Register 1 Description . . . . .	13-26
13-15	Lbus Error Register 2 Description . . . . .	13-27
13-16	Mailbox Data Structure . . . . .	13-30
13-17	Futurebus+ Mailbox Command Field Description . . . . .	13-31
13-18	Futurebus+ Mailbox Status Field Description . . . . .	13-33
13-19	Lbus Mailbox Command Field Description . . . . .	13-35
13-20	Lbus Mailbox Status Field Description . . . . .	13-36
14-1	DEC 4000 AXP System Interrupt Encoding . . . . .	14-1
14-2	I/O Module Hardware Error Interrupt Conditions . . . . .	14-3
15-1	Futurebus+ Address Mapping . . . . .	15-2
15-2	Futurebus+ Interrupt Request Register Description . . . . .	15-4
15-3	Futurebus+ Halt Request Register Description . . . . .	15-5
16-1	SCSI Controller Address Decoding . . . . .	16-2
16-2	AUX_CTRL Register Description . . . . .	16-4
16-3	AUX_Data Register Description . . . . .	16-4
16-4	Console Control Register Description . . . . .	16-5
16-5	Console Data Register Description . . . . .	16-5
16-6	Auxiliary Serial Line Modem Control . . . . .	16-6
16-7	TOY Register A Description . . . . .	16-8
16-8	TOY Clock Square Wave Frequencies . . . . .	16-8
16-9	TOY Register B Description . . . . .	16-10
16-10	TOY Register C Description . . . . .	16-11
16-11	TOY Register D Description . . . . .	16-12
17-1	Ethernet Interface Selection . . . . .	17-1
17-2	Vector Address, IPL, Sync/Asynch Description . . . . .	17-5
17-3	CSR0 access . . . . .	17-6
17-4	Transmit Polling Demand Description . . . . .	17-7

17-5	CSR1 Access . . . . .	17-7
17-6	Receive Polling Demand Description . . . . .	17-8
17-7	CSR2 Access . . . . .	17-8
17-8	Descriptor List Addresses Bits . . . . .	17-10
17-9	CSR3 Access . . . . .	17-10
17-10	CSR4 access . . . . .	17-11
17-11	Status Register 5 Description . . . . .	17-12
17-12	CSR5 Access . . . . .	17-15
17-13	Command and Mode Register Description . . . . .	17-17
17-14	CSR6 Access . . . . .	17-21
17-15	System Base Register Description . . . . .	17-22
17-16	CSR7 access . . . . .	17-22
17-17	Watchdog Timer Description . . . . .	17-23
17-18	CSR9 Access . . . . .	17-24
17-19	TGEC Identification and Missed Frame Count Description . . . . .	17-25
17-20	CSR10 access . . . . .	17-25
17-21	CSR11,12,13 bits . . . . .	17-26
17-22	CSR11,12,13 access . . . . .	17-27
17-23	RDES0 Bits . . . . .	17-29
17-24	RDES1 Bits . . . . .	17-31
17-25	RDES2 Bits . . . . .	17-32
17-26	RDES3 Bits . . . . .	17-33
17-27	Receive Descriptor Status Validity . . . . .	17-33
17-28	TDES0 bits . . . . .	17-34
17-29	TDES1 bits . . . . .	17-36
17-30	TDES2 Bits . . . . .	17-37
17-31	TDES3 Bits . . . . .	17-37
17-32	Transmit descriptor status validity . . . . .	17-37
17-33	Setup frame descriptor bits . . . . .	17-39
17-34	Reception Process State Transitions . . . . .	17-47
17-35	Transmission process state transitions . . . . .	17-49
17-36	CSMA/CD Counters . . . . .	17-51
18-1	SCSI/DSSI Selection . . . . .	18-2
18-2	SCSI Control 0 Description . . . . .	18-4
18-3	Arbitration Mode Bit Interpretations . . . . .	18-6
18-4	SCSI Control 1 Description . . . . .	18-7
18-5	SCSI Destination ID Description . . . . .	18-8
18-6	SCSI Interrupt Enable Description . . . . .	18-9
18-7	SCSI Chip ID Description . . . . .	18-10
18-8	SCSI Transfer Description . . . . .	18-10
18-9	. . . . .	18-12
18-10	. . . . .	18-12
18-11	. . . . .	18-12
18-12	SCSI Output Data Latch Description . . . . .	18-13
18-13	SCSI Output Control Latch Description . . . . .	18-14
18-14	SCSI First Byte Received Description . . . . .	18-15
18-15	SCSI Input Data Latch Description . . . . .	18-15



18-16	SCSI Bus Data Lines Description . . . . .	18-16
18-17	Bits <1:0> SSCF1-SSCFO (Synchronous SCSI Clock Control bits) . . . . .	18-16
18-18	SCSI Bus Control Lines Description . . . . .	18-17
18-19	DMA Status Description . . . . .	18-18
18-20	SCSI Status 0 Description . . . . .	18-19
18-21	SCSI Status One Description . . . . .	18-22
18-22	SCSI Status Two Description . . . . .	18-23
18-23	Chip Test Zero Description . . . . .	18-24
18-24	Chip Test One Description . . . . .	18-25
18-25	Chip Test Two Description . . . . .	18-26
18-26	Chip Test Three Description . . . . .	18-28
18-27	Chip Test Four Description . . . . .	18-28
18-28	Chip Test Five Description . . . . .	18-30
18-29	Chip Test Six Description . . . . .	18-32
18-30	Chip Test Seven Description . . . . .	18-32
18-31	DMA FIFO Description . . . . .	18-34
18-32	Interrupt Status Description . . . . .	18-35
18-33	Chip Test Eight Description . . . . .	18-37
18-34	Longitudinal Parity Description . . . . .	18-38
18-35	DMA Byte Counter Description . . . . .	18-39
18-36	DMA Command Description . . . . .	18-40
18-37	DMA Next Data Address Description . . . . .	18-40
18-38	DMA Scripts Pointer Description . . . . .	18-41
18-39	DMA Scripts Pointer Save Description . . . . .	18-41
18-40	Scratch Register Description . . . . .	18-42
18-41	DMA Mode Description . . . . .	18-42
18-42	DMA Interrupt Enable Description . . . . .	18-44
18-43	DMA Watchdog Timer Description . . . . .	18-45
18-44	DMA Control Register Description . . . . .	18-45
18-45	Adder Sum Output Description . . . . .	18-47
19-1	System Bus Signals (143 Total) . . . . .	19-7
19-2	Command and Address Format . . . . .	19-9
19-3	TRANS<2:0> Bit Interpretations . . . . .	19-13
19-4	Commander CID Assignments . . . . .	19-13
19-5	CAD Parity Coverage . . . . .	19-14
19-6	CPU Cache Line States . . . . .	19-16
19-7	Memory Slot ID . . . . .	19-19
23-1	Alpha AXP Boot Block Field Definitions . . . . .	23-5
23-2	Supported Boot Devices . . . . .	23-5
23-3	Boot Parameters . . . . .	23-5
25-1	Console Special Characters . . . . .	25-2
25-2	Frequently Used Commands . . . . .	25-4
25-3	Console Shell Operators . . . . .	25-17
25-4	DEC 4000 Console Command Summary . . . . .	25-18
26-1	Interleave environment variable syntax . . . . .	26-20
26-2	Interleave environment variable examples . . . . .	26-21
A-1	IPR Reset State . . . . .	A-1

C-1	System bus AC and DC Characteristics .....	C-1
E-1	Environment Variables .....	E-1

---

## Preface

The *DEC 4000 Technical Manual* documents the functional, physical and environmental characteristics of the DEC 4000 system with the emphasis placed on the KN430 CPU subsystem. A brief description of each subsystem is given in Part I.

This manual is intended for a design engineer or applications programmer and should be used along with the *Alpha Architecture Reference Manual* as a programmer's reference to the system.

The manual is divided into five parts, five appendices, a glossary and an index.

- Part I provides a list of the distinctive characteristics of the DEC 4000 system and its enclosure, the BA640. It gives a technical description of the system including physical and functional descriptions of each subsystem. Figure 1–10 shows the order in which the KN430 CPU Subsystem modules must be installed in the backplane.
- Part II provides in depth information about the 21064 CPU chip and a brief description of basic Alpha AXP architecture. This chapter lists all the internal processor registers used in the processor design. Some information on error handling is given also.
- Part III provides in depth description of the operation of the I/O module. It describes the registers used by the Ethernet controller as well as the NCR SCSI controller registers.
- Part IV provides information about how the system bus handles transactions and arbitration. It lists the system bus signal characteristics.
- Part V describes the operation of the system firmware. It lists the console commands and gives a block diagram showing the locations of the different pieces of firmware in the system.
- Appendix A discusses the state of the internal processor registers on power-up and system reset.
- Appendix B lists the errors flows that occur on system errors.
- Appendix C lists the electrical, environmental, and physical characteristics of the DEC 4000 system.
- Appendix D lists the console commands.
- Appendix E lists the environment variables used by the Alpha AXP Architecture and the DEC 4000 system.
- The Glossary provides descriptions of terms used in the DEC 4000 documentation set.

## Documentation Conventions

The following table lists the conventions used throughout this manual and explains their usage.

**Table 1 Conventions**

Convention	Meaning
<x:y>	Represents a bit field, or an "extent", a set of lines, or signals, ranging from x through y. For example, R0 <7:4> indicates bits 7 through 4 in a general purpose register R0. The associated field name (if defined) typically follows the field definition and appears in parentheses. For example, PSL<20:16> (IPL) represents the five-bit field for the Interrupt Priority Level in the Processor Status Longword.
x..y	Represents a range of bits, from y through x.
n, n <sub>16</sub> , n <sub>2</sub>	Numbers are decimal unless otherwise marked with a subscript number. Where there is ambiguity, the radix is explicitly stated.
2.0123.FFFF	Nine-digit numbers throughout this document typically represent 34-bit hexadecimal addresses and are grouped in four digit clusters separated by periods. This number is implied as hexadecimal and if marked will be marked with a subscript sixteen.
<span style="border: 1px solid black; padding: 2px;">Return</span>	A label enclosed in a box represents a key (usually a control or a special character key) on the keyboard (in this case, the carriage return key.)
<b>Note</b>	Flags special information that should be considered while using the manual.
<b>Caution</b>	Flags information that will help you prevent damage to equipment.
<b>n</b>	Boldface small n indicates a variable.
{ }	Represents a console command element
[ ]	Represents a console command element that is optional
...	Represents a list command element
SIGNAL NAME L	Signal names are emphasized in small capital letters

## Internal Code Names

Certain LSI components in the DEC 4000 system were developed within Digital and are referred to with internal code names. These are:

- C<sup>3</sup>, the command, communication, and control gate array on the processor module in some cases it is expressed as simply C3.
- IONIC, the system bus interface and write-merge buffer gate array on the I/O module.
- CMIC, the ECC and bus interface gate array on the memory modules.
- EV4, the 21064 CPU chip on the processor module, also known as the DECchip™.

This manual contains many figures that show the format of various registers followed by a description of each bit field. In general, the fields on the register are labeled with a signal name or mnemonic. The description of each field includes the name or mnemonic, the bit range, and the bit type.







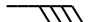


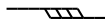
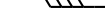




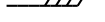
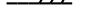
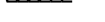


**Table 2 Bit Type Conventions**

Bit Type	Description
<b>Bit types</b>	A bit type denotes the mode used to access it.
0	Denotes a bit reserved for future expansion - Ignored on Write, Read as "0"
1	Reserved for future expansion - Ignored on Write, Read as "1"
Read-write	A read-write bit or field, may be read and written by software
Read-only	A read-only bit, may be read by software, It is written by hardware. Software writes are ignored
Write-only	A write-only bit may be written by software. It is used by hardware and reads by software return an unpredictable result.
Write	A write bit or field may be written by software. It is used by hardware and reads by software return a 0.
Write-one-to-clear	If reads are allowed to the register, then the value may be read by software. If it is a write-only register, then a read by software returns an unpredictable result. Software writes of a 1 cause the bit to be cleared by hardware. Software writes of a 0 do not modify the state of the bit.
Write-zero-to-clear	If reads are allowed to the register then the value may be read by software. If it is a write-only register then a read by software returns an unpredictable result, Software writes if a 0 cause the bit to be cleared by hardware, Software writes of a 1 do not modify the state of the bit.
Write-anything	If reads are allowed to the register then the value may be read by software. If it is a write-only register, then a read by software returns an unpredictable result. Software writes of any value to the register cause the bit to be cleared by hardware.
Read-to-clear	The value is written by hardware and remains unchanged until read. The value may be read by software, at which point hardware may write a new value into the field.
IGN	Ignored. Fields specified as ignore are ignored when written.
RAZ	Read as zero. Fields specified as such return a zero when read.
MBZ	Fields specified as must be zero must never be filled by software with a non-zero value. If the processor encounters a nonzero number in a field specified as MBZ, a reserved operand exception occurs.
SBZ	Fields specified as should be zero should be filled by software with a zero value. These fields may be used at some future time. Nonzero values in SBZ fields produce unpredictable results.

# Timing Diagram Conventions

The conventions used in the timing diagrams are shown in Figure 1.

**Figure 1 Timing Diagram Conventions**

HIGH	
LOW	
INTERMEDIATE	
VALID_HIGH_OR_LOW	
CHANGING	
INVALID_BUT_NOT_CHANGING	
HIGH_TO_LOW	
HIGH_TO_VALID	
HIGH_TO_INVALID	
INTERMEDIATE_TO_LOW	
HIGH_TO_INTERMEDIATE	
LOW_TO_HIGH	
LOW_TO_VALID	
LOW_TO_INVALID	
INTERMEDIATE_TO_HIGH	
LOW_TO_INTERMEDIATE	
VALID_TO_INTERMEDIATE	
INVALID_TO_INTERMEDIATE	
INTERMEDIATE_TO_VALID	
INTERMEDIATE_TO_INVALID	

**Related Documents**

The following documents are related to the DEC 4000 CPU.

- Alpha Architecture Handbook (EC-H1689-10)
- Alpha Architecture Reference Manual (EY-L520E-DP)
- DECchip 21064-AA Microprocessor Hardware Reference Manual (EC-N0079-72)

**Comments and Suggestions**

We welcome your comments and suggestions on this book. Write to:

Entry Systems Engineering Group  
Digital Equipment Corporation  
MLO5-5/E71  
146 Main Street  
Maynard, Massachusetts USA, 01754-2571

Attention: Susan Yuryan

# Part I

---

## DEC 4000 System Description

This part contains a technical overview of the DEC 4000 system.



---

## DEC 4000 System Features

This chapter introduces the DEC 4000 system and provides a list of system features to familiarize the reader with the DEC 4000 system's capabilities.

The DEC 4000 product is a member of a new family of reduced instruction set computer (RISC) systems that support both the VMS and OSF/1 operating system software. It is designed for use in high-speed, real-time applications in an open office environment.

The DEC 4000 is Digital's entry product in the multi-user and symmetric multiprocessing technical server market using the first generation Alpha microprocessor. This chapter describes the main configuration features of the DEC 4000 system.

### **CPU**

The DEC 4000 system is equipped with the DECchip™ 21064-AA microprocessor, running at a clock speed of 160 MHz. In addition to the EV4's 16 kB on-chip cache, the DEC 4000 provides 1 MB cache on the processor module, for significantly enhanced system performance. DEC 4000 can be configured with one or two processor modules. The second processor module provides a near-doubling of system compute performance, and allows operation in a symmetric multiprocessing mode.

### **Memory capacity**

The DEC 4000 system can be configured using up to four memory boards, each of which may contain 64 or 128 MB, for a maximum total of 512 MB (4x128).

### **Fixed storage I/O**

DEC 4000 is configured with one of two fixed storage I/O options, depending on the user's functional requirements. One option supports up to four fast SCSI buses (10 MB/S each), for users that require maximum single stream data transfer rates to and from disk.

The second option supports up to four buses (5 MB/S each) that can each be set to run either the DSSI or the SCSI protocol. The DSSI option is available for those users who want to take advantage of DSSI's clustering and multihosting features.

Depending on which fixed storage I/O option is selected, either a fast SCSI or DSSI/SCSI I/O board is installed in the DEC 4000 cabinet. The fast SCSI I/O board provides integral support for one Ethernet port. The DSSI/SCSI I/O board provides integral support for two Ethernet ports.

### **Fixed storage devices**

There are four storage compartments in the DEC 4000 cabinet. Each compartment can hold up to four 3.5" disks or one 5.25" disk. Each fixed storage I/O bus connects to the fixed disks installed inside the respective compartment; thus all disks in a given compartment must be of the same type (for example, SCSI or DSSI), as appropriate to that compartment's bus type. Each compartment (except for the Fast SCSI compartment) provides a front-panel connector for expansion of the bus outside the DEC 4000 cabinet. <sup>1</sup> DEC 4000 requires that at least one compartment have a fixed disk mounted to act as a system disk.

### **System I/O**

For system I/O, all DEC 4000 configurations provide six Futurebus+ (Profile B) slots, with a peak data transfer rate of 160 MB/S. These support both Digital and 3rd party Futurebus+ options for various communications and expansion purposes.

### **Removable storage I/O**

For removable storage, DEC 4000 provides one SCSI bus (5 MB/S) to support the installed RRD42 CD reader and various tape backup options. This bus is not optional, and is provided in all DEC 4000 configurations. The DEC 4000 front panel provides a port that allows expansion of this SCSI bus outside the cabinet, for connection to additional SCSI devices.

### **Removable storage**

DEC 4000 comes standard with one RRD42 CD reader. Additional mounting space is provided for up to three half-height tape backup devices (or one full-height and one half-height drive), depending on the user's backup requirements. These devices all connect to the removable storage SCSI bus.

Table 1-1 gives a summary of system features:

---

<sup>1</sup> For enhanced data transfer integrity, the fast SCSI bus must run at 5 MB/S when expanded outside the DEC 4000 cabinet.

**Table 1-1 System Features Summary**

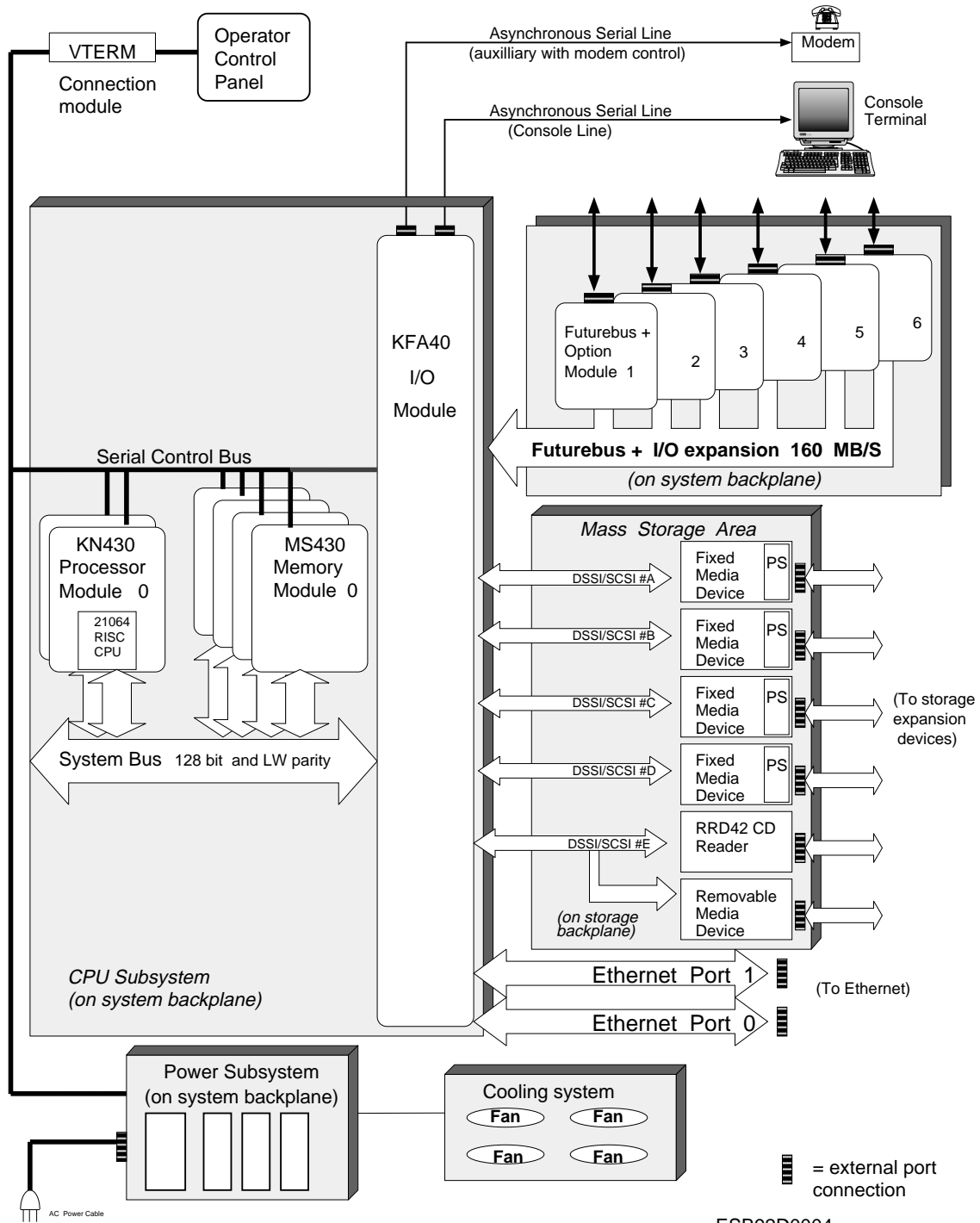
---

CPU	1 or 2 processor module, 21064-AA CPU w/ 1 MB cache
Memory	1 to 4 Memory Boards; 64 or 128 MB
Fixed Storage I/O	1 I/O Board, either:  Fast SCSI (4 buses @ 10 MB/S each; each bus also operates at Standard SCSI speeds of 5 MB/s), plus 1 Ethernet port DSSI/SCSI (4 buses @ 5 MB/S each), each bus selectable DSSI or SCSI, plus 2 Ethernet ports
Fixed Storage	1 to 4 Storage Compartments  Compartment 1: 1 - 5.25" -or- up to 4 - 3.5" fixed disks Compartments 2-4: 1 - 5.25" -or- up to 4 - 3.5" fixed disks Note: Empty compartments are allowed.
System I/O	6 Futurebus+ (Profile B) slots, 160 MB/S
Removable Storage	1 SCSI bus (5 MB/S) I/O
Removable Storage	1 RRD42 CD Reader; 0 to 3 half-height tape drives or 1 full-height and 1 half-height tape drive.

---

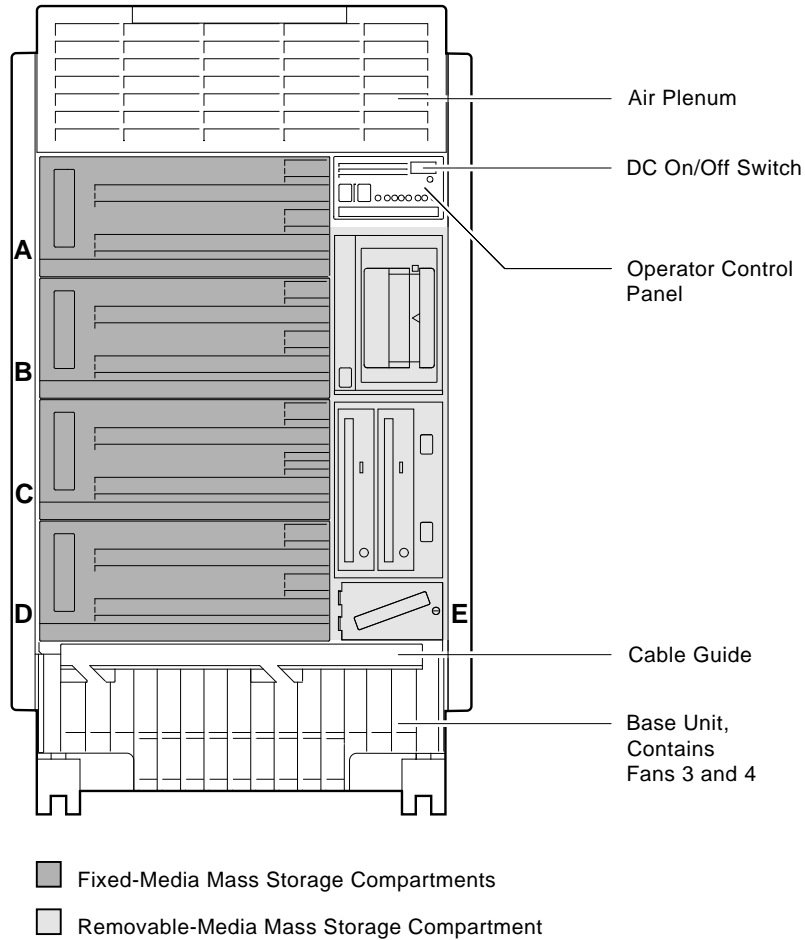
Figure 1-1 shows a system block diagram.

**Figure 1-1 System Block Diagram**



The DEC 4000 system consists of a CPU subsystem, an I/O expansion area implemented with the Futurebus+ Profile B, a mass storage area, and a power supply subsystem integrated together and housed in the BA640 cabinet. Figure 1-2 provides a front view of the DEC 4000 system with the front door removed.

**Figure 1-2 The DEC 4000 System Front View**



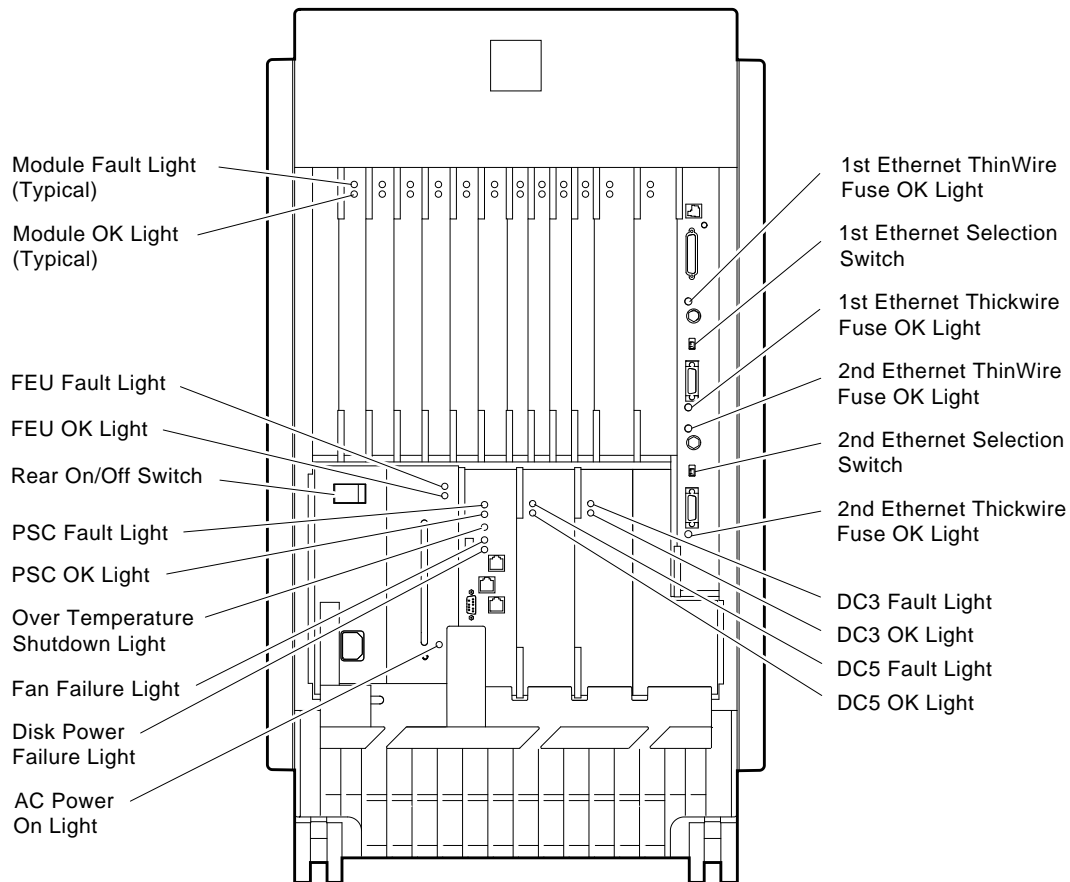
MLO-007714

## 1.1 BA640 Cabinet

The BA640 cabinet has a plastic frame covered with plastic panels and sits on four casters. This cabinet provides both front and back access doors for the DEC 4000 system. The front of the system has the mass storage devices for both fixed and removable media, and the operator control panel (OCP). The back of the system has the CPU subsystem modules, an area to plug in Futurebus+ modules, and the power supply modules. The cooling fans are at the bottom of the system. Figure 1-3 shows the front and back views of an opened DEC 4000 system.

## 1.1 BA640 Cabinet

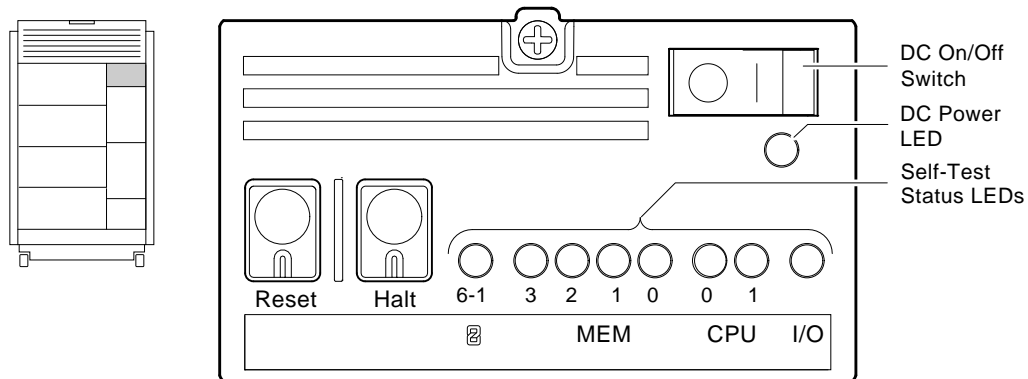
Figure 1-3 BA640 Enclosure Views



MLO-008199

The operator control panel measures approximately 4.6" x 2.6" and sits in the front of the system above the removable storage media devices. This panel contains 4 switches and 9 indicator lights. This panel works with the serial control bus to provide the user with a way to interface with and control the system. Figure 1-4 shows the operator control panel detail.

Figure 1-4 Operator Control Panel



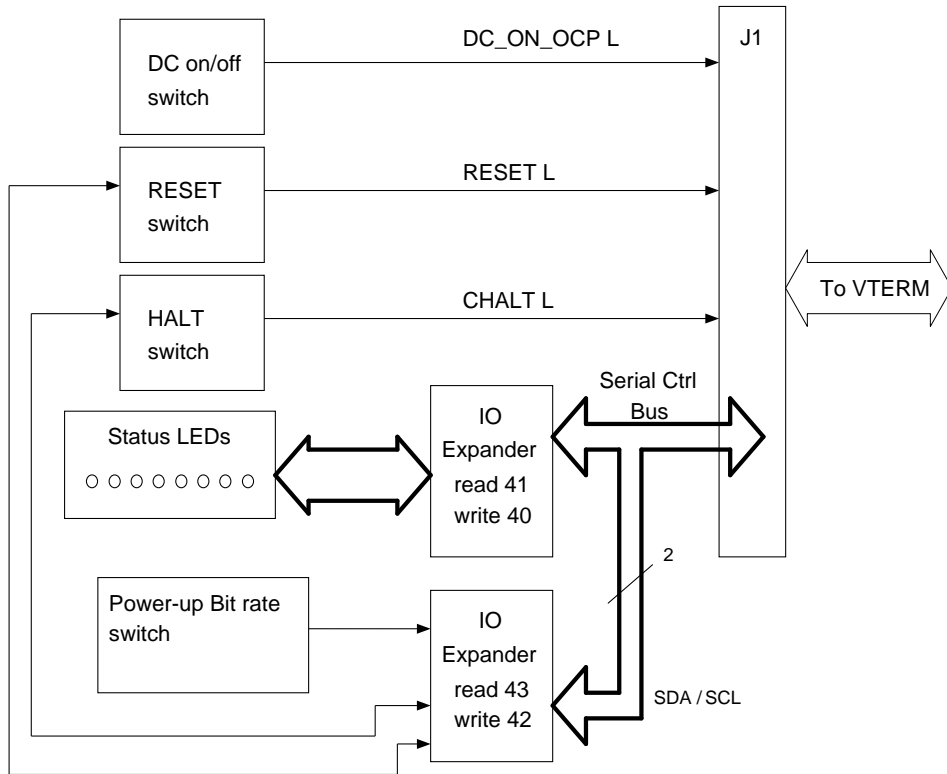
LJ-02008-T10

- DC on/off switch (Enables DC power to flow to the systems modules)
- Halt switch (Halts the CPU, forcing the system to invoke console mode)
- Reset switch (Resets the system)
- Power-up baud rate select switch (Provides the console baud rate for system power-up)
- The nine status LEDs indicate the presence of DC power, and faults in the I/O, CPU, Memory, and Futurebus+ modules.

The circuit board behind the OCP panel contains an 8-bit I/O expander chip (PCF8574) that interfaces to the serial control bus. This module also contains the switch hardware and LEDs, and a connector to the Vterm module that links the operator control panel to the storage backplane module. Figure 1-5 shows a functional block diagram of the operator control panel's circuit board.

## 1.1 BA640 Cabinet

Figure 1–5 The OCP Circuit Board Block Diagram



ESB92D0007

### 1.1.1 DC on/off Switch

The DC on/off switch is a latching rocker switch. This switch is connected to an external green LED located on the OCP below the DC on/off switch. The DC on/off switch debounce logic resides on the PSC. The LED is controlled entirely by the operator control panel module. There is no control of this LED by the firmware. When the DC on/off switch is moved to the "1" position, the DC\_ON\_OCP L signal is asserted and sent to the power supply controller (PSC). The DC LED is lit by the power system controller module whenever the 5 volt line is within tolerance. When the switch is moved to the "0" position, the PSC negates DC\_ON\_OCP L and extinguishes the LED.

### 1.1.2 RESET Switch

The Reset Switch is a momentary switch with an integral green LED. The integral LED is controlled entirely by the firmware. There is no control of this LED by the OCP module. The RESET L signal is an open-collector output. Its pull-up resistor is located on the PSC and its debounce circuitry is located on the OCP module. The switch, when pressed activates a one-shot device that asserts the RESET L signal pulse to the power supply (PSC) which resets the system by



providing the ASYNC\_RESET L signal. The duration of this pulse is minimum of 10.1 ms and a maximum of 12.5 ms.

The reset switch LED is illuminated by the firmware when the reset switch is pressed and extinguished by the firmware when the reset switch is released. The firmware illuminates the reset LED by writing a zero to bit P4 at the I/O expander chip write address 42. Writing a one to this bit extinguishes the reset LED.

### 1.1.3 The Halt Switch

The Halt switch is a latching switch with an integral green LED. The CHALT L signal is a open-collector output. A pull-up resistor is provided on the processor module. The Halt switch debounced circuitry is located on the OCP module. When the halt switch is pressed, it activates a one-shot circuit that asserts the CHALT L signal pulse to the processor module for a minimum of 375 ns and a maximum of 470 ns. The CHALT L signal places the system into console halt mode.

The firmware determines the position of the halt switch by reading bit P3 at serial control bus read address 43. If bit P3 is read as a zero, it indicates that the switch has been pressed and the system is in halt mode. If bit P3 is read as a one, it indicates that the Halt switch has been released and the system is in run mode. The position of the Halt switch must always be determined whenever the system executes its initialization process or during normal operation. In conjunction with the Halt switch, there is a halt LED controlled by port bit P2 at write address 42. This LED is written with a zero (Halt switch depressed) to illuminate the LED and written with a one (Halt switch released) to extinguish the LED.

### 1.1.4 Power-up Console Baud rate switch

The power-up console baud rate selection switch is a thumbwheel switch numbered 0 through 7. The selected number can be confirmed through a window on the switch body. This switch provides three bitrate codes to the system firmware at power-up time in order to program the speed of the console terminal serial line. Table 1-2 lists the baud rate codes.

**Table 1-2 PCF8574 ADDRESS 42/43 BAUD RATE BIT DEFINITIONS**

Switch Number	Baud Rate	Bit Code 2,1,0
1	600	1 1 0
2	1200	1 0 1
3	2400	1 0 0
4	4800	0 1 1
5	9600	0 1 0
6	19,200	0 0 1

## 1.1 BA640 Cabinet

### 1.1.5 Status LEDs

Status LEDs exist for both CPU<sub>0</sub> and CPU<sub>1</sub>. On a power-up and *only* on a power-up of the system, the CPU<sub>0</sub> and CPU<sub>1</sub> status LEDs are illuminated by logic on the operator control panel module. Thereafter, the system firmware controls and illuminates/extinguishes these two LEDs. To extinguish the CPU<sub>0</sub>/CPU<sub>1</sub> LEDs on the first write after power-up, the firmware must write P1 bit in each I/O expander to a zero. Thereafter, a write to the P0 bit in each I/O expander (0 for illuminate and 1 for extinguish) controls these two LEDs and is more in line with the programming control of the remaining status LEDs.

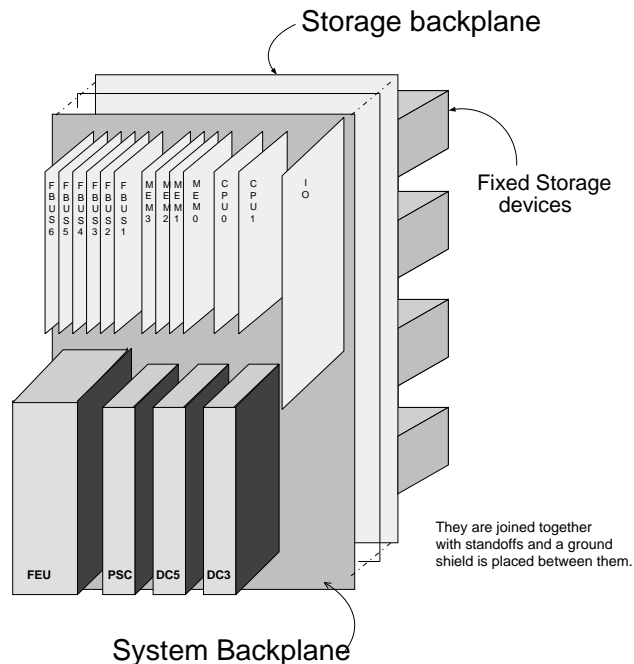
## 1.2 The Backplane Assembly

Inside the cabinet, the backplane assembly holds two backplane modules positioned back-to-back in the center of the system. The modules are joined by several tin-plated brass standoffs that carry power from the system backplane to the storage backplane. Several cables carry signals between backplane modules.

All modules and integrated storage assemblies plug directly into designated slots in the backplane modules eliminating the need for cables.

Figure 1-6 shows a view of the system backplane module in the backplane assembly facing the system backplane module. The metal card cage that holds the modules and storage devices is not shown.

Figure 1-6 The Backplane Assembly



ESB9200015

### 1.2.1 The Storage Backplane Module

The storage backplane module is a 10-layer module that measures 13.33" by 19.29". This backplane contains 16 connectors for connecting storage devices and carrying signals. It uses a combination of press-pin and surface-mount components. All components are mounted on side one.

Connectors J14 and J15 connect to the system backplane with flat ribbon cables and carry local I/O bus signals that originate on the I/O module to the fixed media devices. Local I/O bus 3 also extends to the removable media devices. Connector J16 connects to the system backplane with a flat ribbon cable and carries the serial control bus signals to the operator control panel via the vterm module.

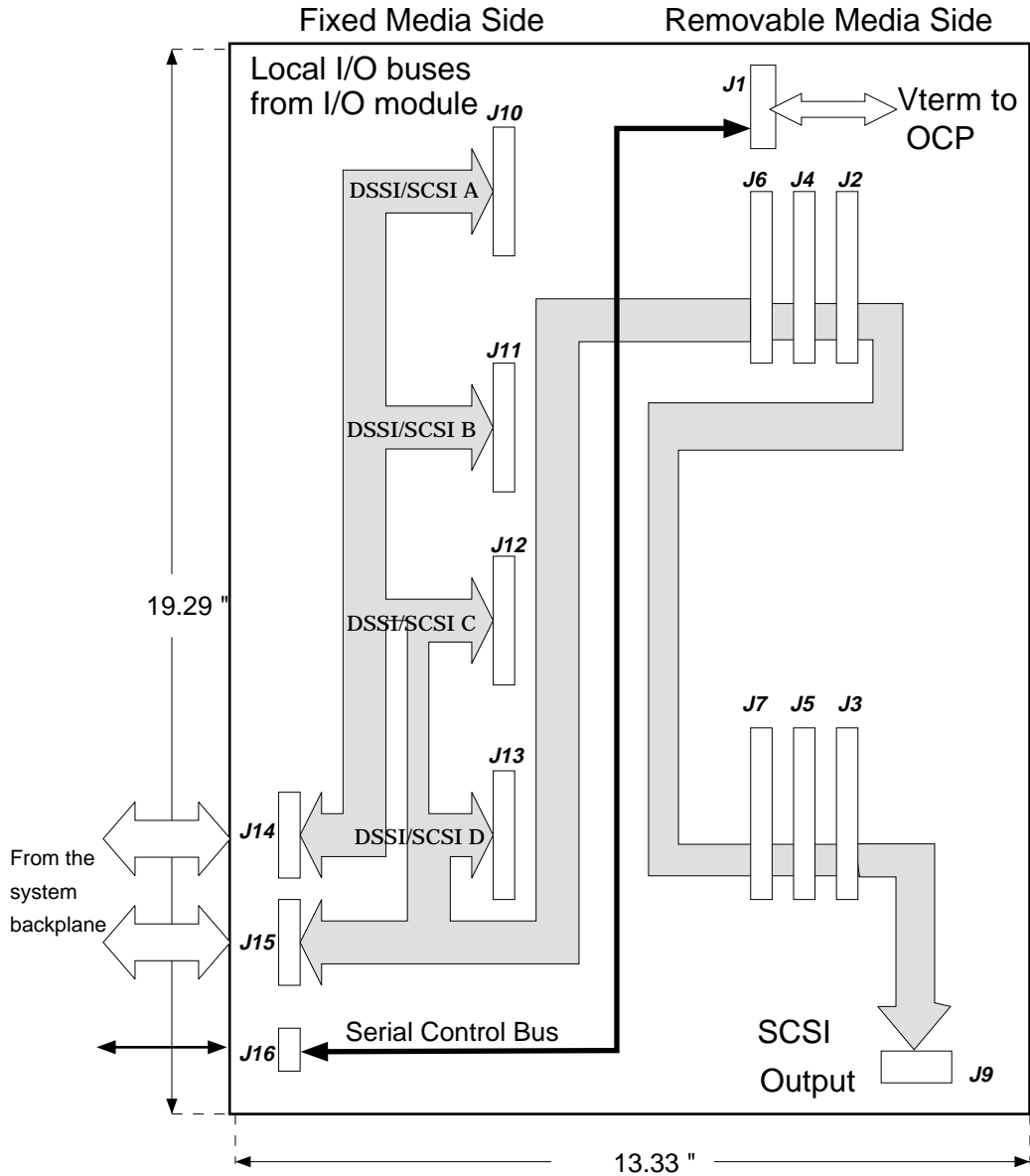
Figure 1-7 shows the layout of the storage backplane module. Table 1-3 describes the function of each connector.

**Table 1-3 Storage Backplane Connectors**

Connector	Number of pins	Function
J1	30	Provides power and control signals to the OCP terminator card
J2	78	Accepts a removable media device
J3	78	Accepts a removable media device
J4	78	Accepts a removable media device
J5	78	Accepts a removable media device
J6	78	Accepts a removable media device
J7	78	Accepts a removable media device
J9	50	Provides SCSI-2 output signals
J10	80	Accepts a fixed media storage device
J11	80	Accepts a fixed media storage device
J12	80	Accepts a fixed media storage device
J13	80	Accepts a fixed media storage device
J14	100	Carries I/O bus signals between backplane modules
J15	100	Carries I/O bus signals between backplane modules
J16	40	Serial Control Bus

## 1.2 The Backplane Assembly

Figure 1-7 Storage Backplane



ESB92D0019

### 1.2.2 System Backplane Module

The system backplane module is a 12-layer module that contains a total of 86 connectors for connecting the system modules and carrying signals. A combination of press-pin and surface-mount components are used. All components are mounted on side one.

### 1.2.3 System Bus Routing

The system bus is routed to the processor memory modules and the topmost 192-pin connector of the I/O module. Because CPU<sub>0</sub> always creates and distributes the system clocks, the slot for CPU<sub>0</sub> contains a clock connector where the slot for CPU<sub>1</sub> does not. This scheme allows identical processor modules to be configured as CPU<sub>0</sub> or CPU<sub>1</sub>.

### 1.2.4 Futurebus+ Routing

The Futurebus+ interconnect is routed to the 6 Futurebus+ slots and the I/O module. The Futurebus+ signals do not connect to the processor and memory modules. All communication between the Futurebus+ devices and the subsystem modules is accomplished through the I/O module.

### 1.2.5 Local I/O Bus Routing

The local I/O buses are routed from the I/O module to connectors J and J that connect to the storage backplane.

### 1.2.6 Serial Control Bus Routing

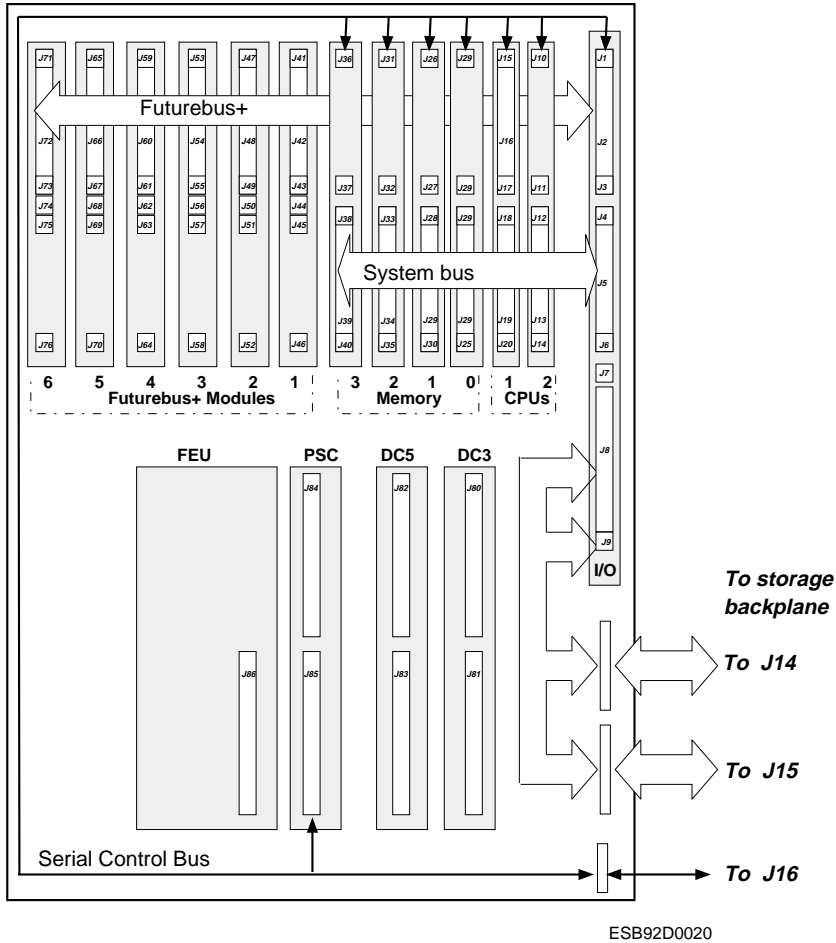
The serial control bus is routed to the processor, memory, the I/O modules, as well as the power system control module and to Jxx and routed across a flat ribbon cable to be connected to the operator control panel on the storage backplane. Figure 1-8 shows the layout of the system backplane module. Table 1-4 describes how the connectors are used.

**Table 1-4 System Backplane Connectors**

Connector	Usage	Connector	Usage
J1 - J9	I/O module slot	J53 - J58	Futurebus+ slot 3
J10 - J14	CPU <sub>1</sub> module slot	J59 - J64	Futurebus+ slot 4
J15 - J20	CPU <sub>0</sub> module slot	J65 - J70	Futurebus+ slot 5
J21 - J25	memory module 0 slot	J71 - J76	Futurebus+ slot 6
J26 - J30	memory module 1 slot	J80 - J81	DC3 power module slot
J31 - J35	memory module 2 slot	J82 - J83	DC5 power module slot
J36 - J40	memory module 3 slot	J84 - J85	PSC power module slot
J41 - J46	Futurebus+ slot 1	J86	FEU power module slot
J47 - J52	Futurebus+ slot 2		

## 1.2 The Backplane Assembly

Figure 1–8 System Backplane



## 1.3 KN430 CPU Subsystem

The KN430 CPU subsystem has one or two processor modules, up to four memory modules, and one I/O module. At least one of each of these modules must be present in order to form the most basic DEC 4000 system.

### 1.3.1 Processor Module Differences

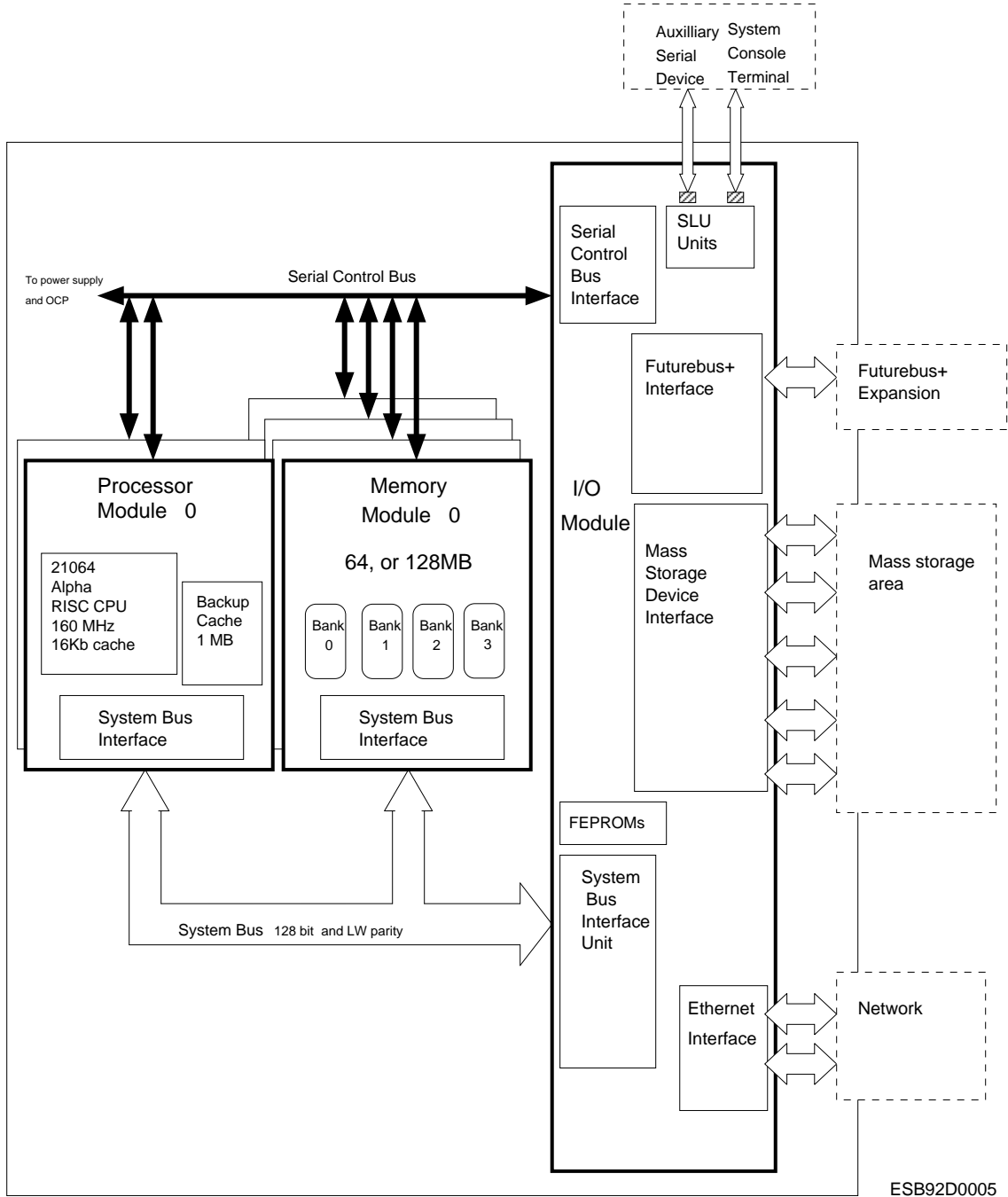
Both processor modules are identical. However, their functions differ slightly depending on the way they are configured. The difference in the way they operate is a function of the CPU backplane slot they occupy. The module configured as CPU<sub>0</sub> must occupy the slot immediately to the right of memory module 0. Similarly CPU<sub>1</sub> must occupy the immediately left of the I/O module. Each CPU configures itself based on its position in the backplane.

## 1.3 KN430 CPU Subsystem

The modules in this subsystem represent the "brain" of the DEC 4000 system. The processor module uses the DECchip 21064™ CPU chip to interpret and execute instructions. The memory module contains the system's main memory space. System programs, including console emulation firmware are loaded into the main memory for execution. The I/O module facilitates communication between the processor module and all I/O devices on the system including the mass storage devices, the console terminal, the I/O expansion modules, and the Ethernet interfaces. Figure 1-9 shows a block diagram of the CPU subsystem.

### 1.3 KN430 CPU Subsystem

Figure 1-9 KN430 CPU Subsystem Block Diagram

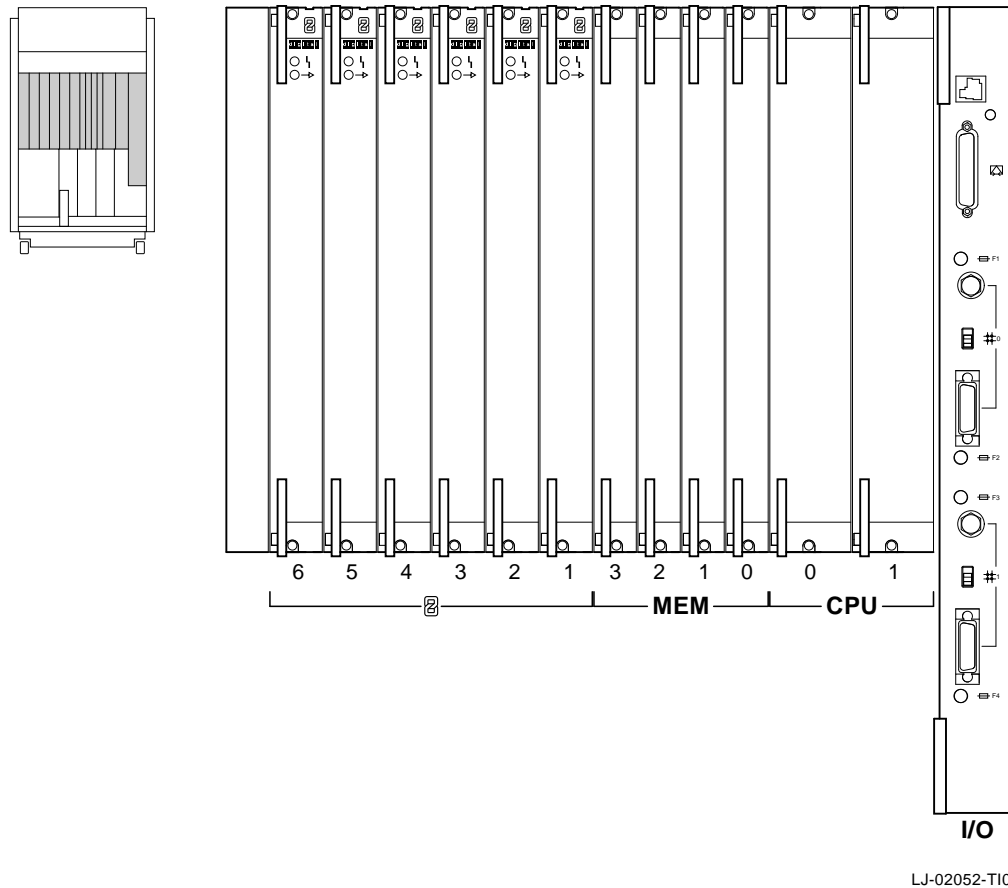




## 1.3.2 Module order

The CPU subsystem modules are located on the back of the DEC 4000 system on the top half of the system backplane. They occupy the seven rightmost slots of the system. Each module has a dedicated slot that it must occupy. The order of the CPU subsystem modules is shown in Figure 1-10.

**Figure 1-10 CPU Subsystem Module Order**



## 1.3.3 KN430 Processor Module

The processor module contains the 21064 Alpha processor, the backup cache memory chips, the system bus interface chips and a serial control bus interface. This module also contains the bus clock generator/distributor circuitry, and the clock power and detect circuit.

The CPU chip interprets and executes Digital Alpha AXP instructions. The backup cache holds copies of data recently used by the processor and fetches several bytes of data from the memory in anticipation that the processor will use the next sequential series of bytes. The system bus interface logic allows the CPU to communicate with other modules on the system bus.

### 1.3 KN430 CPU Subsystem

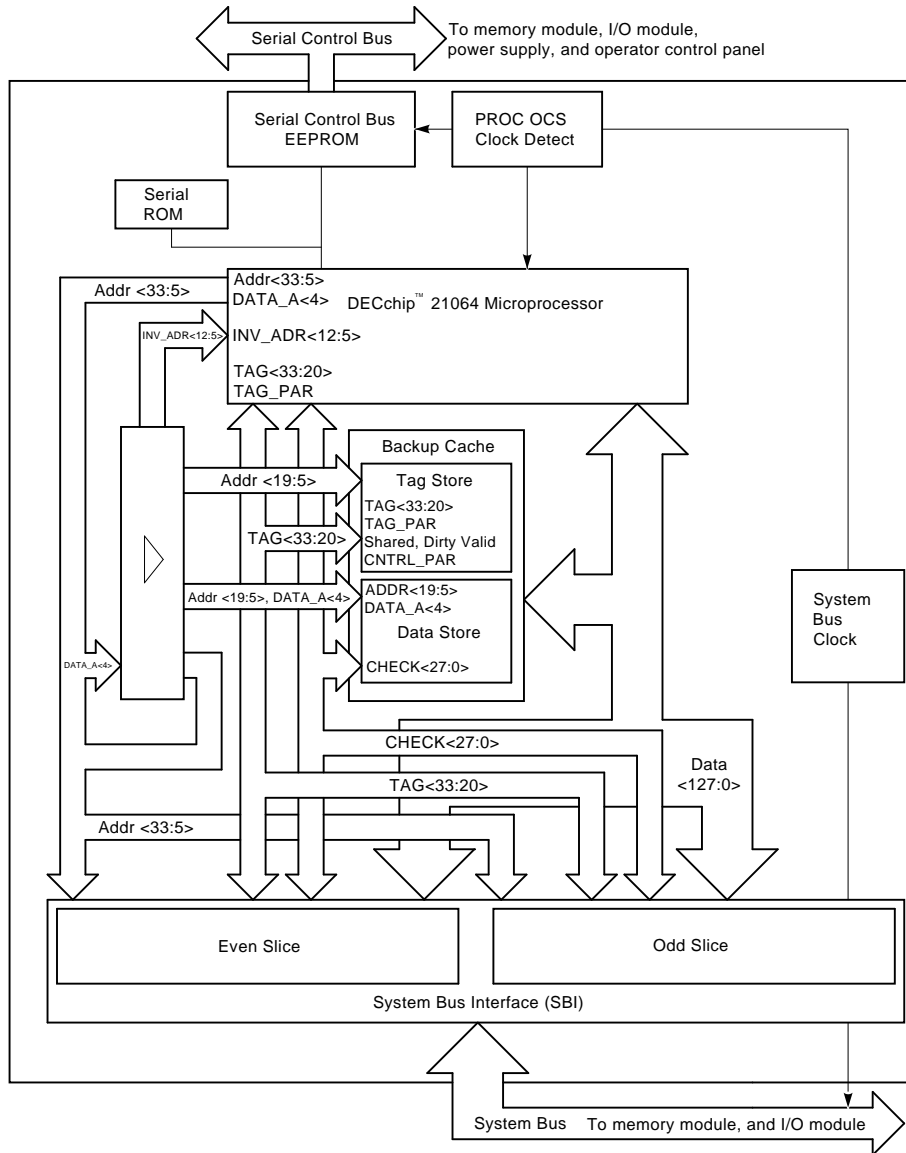
Table 1–5 shows the major differences between CPU<sub>0</sub> and CPU<sub>1</sub>.

**Table 1–5 CPU differences**

<b>CPU<sub>0</sub> (primary)</b>	<b>CPU<sub>1</sub> (secondary)</b>
Performs system bus arbitration	—
Generates the system clock	—
Generates CRESET	—
Has HALT signal buffer	—
Addresses	
System bus CSRs: 2.0000.0XXX	system bus CSRs: 2.0800.0XXX
Serial control bus micro: Read/Write B1/B0	Serial control bus micro: Read/Write B3/B2
Serial control bus NVRAM: Read/Write A9/A8	Serial control bus NVRAM Read/Write AB/AA

Figure 1–11 shows a block diagram of the KN430 processor module.

Figure 1–11 KN430 Processor Module Functional Block Diagram



LJ-02057-T10

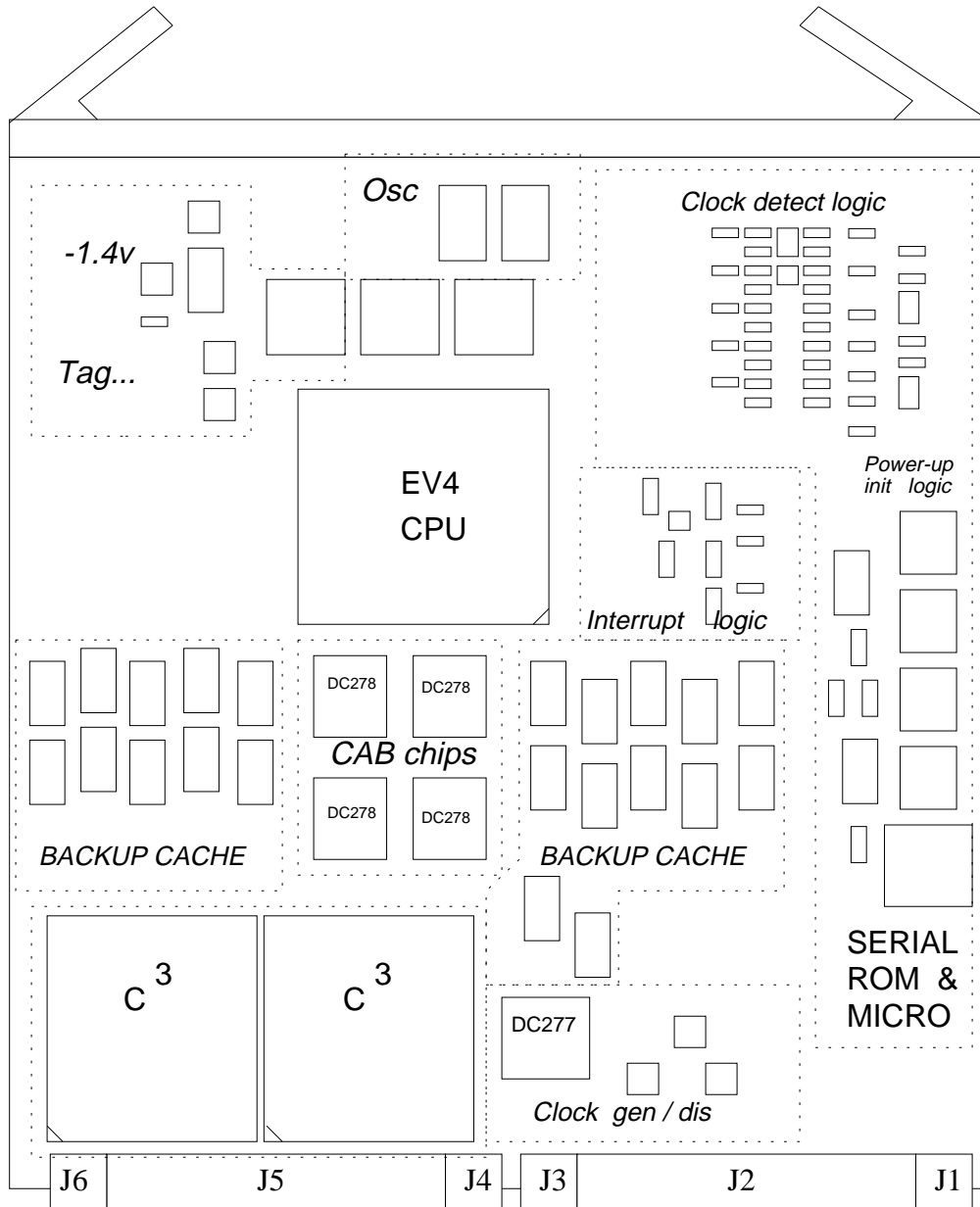
### 1.3.3.1 Processor module components

Most of the processor module's components reside on side 1 of the module. However a significant amount of the chips for the backup cache reside on side 2. The majority of components are implemented in surface-mount technology with the exception of the 21064 CPU chip, the C<sup>3</sup> bus interface unit, and some discrete dip packages.

## 1.3 KN430 CPU Subsystem

Refer to Figure 1-12.

Figure 1-12 The KN430 Processor Module



ESB92D0014

### 1.3.3.2 DECchip 21064™ Alpha CPU

The CPU chip is a 431-pin PGA CMOS-4 chip that contains approximately 2.3 million transistors. This chip is a super-scaler, super-pipelined, implementation of the Digital Alpha architecture.

### 1.3.3.3 Backup Cache Memory Devices

The backup cache consists of 44 64kx4 RAM chips. These devices are 28-pin surface-mount packages located mostly in the center of the module with chips on sides 1 and 2.

### 1.3.3.4 System bus interface

The bus interface is implemented with two Command, Control, and Communication chips called C<sup>3</sup>s. The C<sup>3</sup> chip is a 299-pin PGA chip. Two of these chips allow the processor module to interface with the system bus. They check and generate parity and maintain cache coherency. The 128-bit system bus is split into two slices with one slice handling the even longwords and the other slice handling the odd longwords of any given octaword. The C<sup>3</sup> chips also contain the system bus arbitration logic which is implemented if the C<sup>3</sup> chip resides on CPU<sub>0</sub>.

### 1.3.3.5 System bus clock generator/distributor circuitry

The system bus clock frequency is generated from an 8x, 200PPM PECL oscillator and divided by 8 using the 100E131 configured as a synchronous counter. System bus clocks are PECL level clocks derived from the same 100E131, to minimize skew.

The clocking scheme supports a bus cycle time of 24ns, which translates to a clock frequency of 41.66Mhz. The system bus clock generator/distribution circuitry is enabled only on CPU<sub>0</sub>. The 100E111 drivers are disabled when the module is configured as CPU<sub>1</sub>.

### 1.3.3.6 Clock/ power detect circuit

This circuit is responsible for detecting power and clock and reporting when either of these signals is not present. It is implemented with several discrete surface-mount voltage supervisor chips and discrete transistors, capacitors, and resistors.

### 1.3.3.7 Serial Control Bus Interface

This interface is achieved with an 87C652 8-bit microcontroller, a 256x4 EEPROM, and an octal driver. The microcontroller provides an interface between the 2-line serial control bus implemented with the I<sup>2</sup>C bus and the processor module. The serial control bus is intended as a system maintenance bus that attaches to each CPU subsystem module and captures error data from failed modules and stores the data in EEPROMs located on each CPU subsystem module. A copy of this interface circuit is present on all CPU subsystem modules (CPUs 0 and 1, memories 1 through 4, and the I/O module.)

## 1.3 KN430 CPU Subsystem

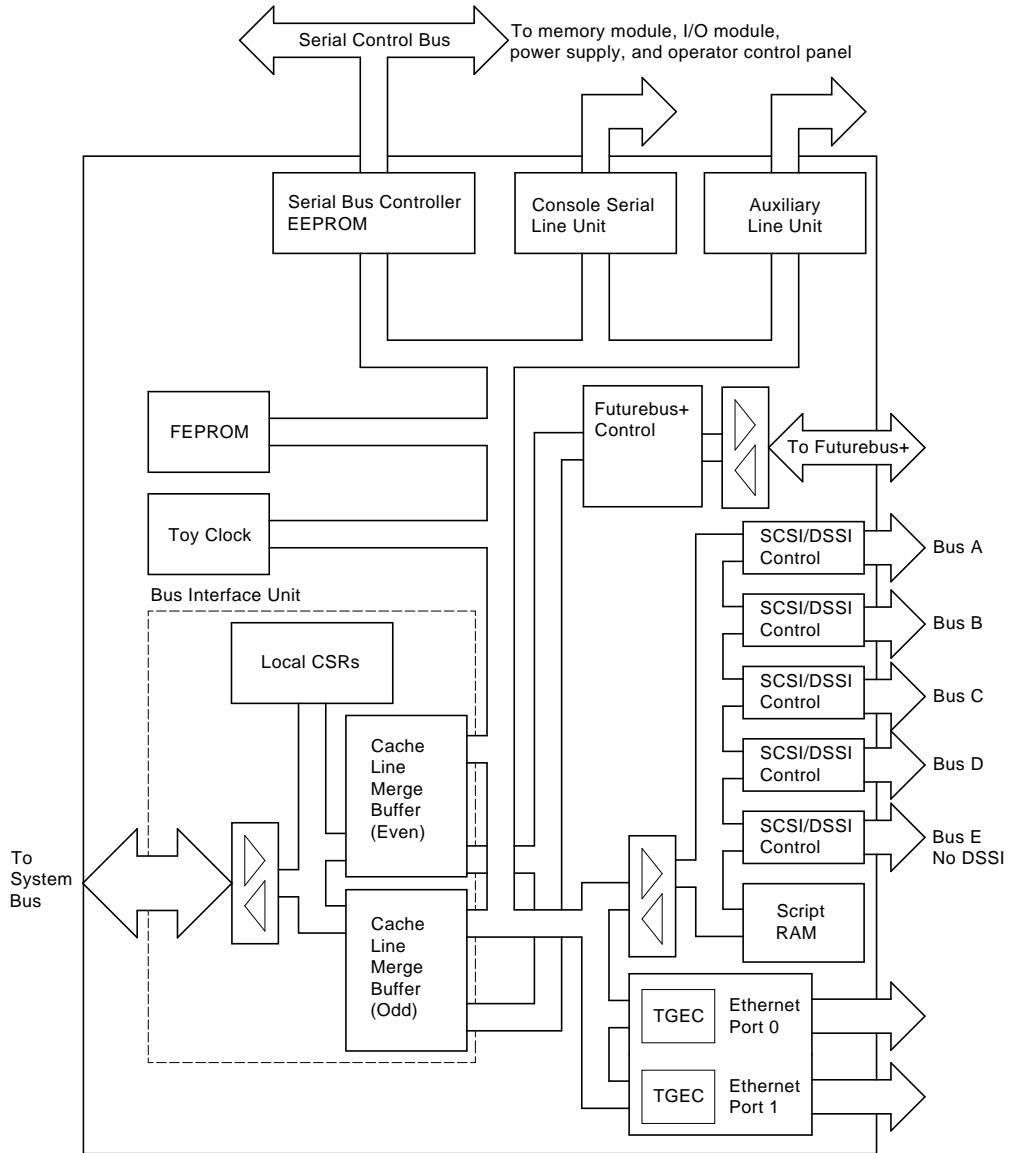
### 1.3.4 KFA40 I/O Module

The KFA40 module contains the SCSI/DSSI controllers, the Ethernet controllers, the console firmware FEPR0M and EEPR0M, the system toy clock, a serial control bus interface, and a system bus interface.

The I/O module performs all I/O transactions on behalf of the processor modules. This module provides data and message routing and control between the system bus and all I/O devices associated with the processing complex. The bus interface unit allows the I/O module to communicate with other modules on the systems bus. The I/O module interfaces to the system bus through coherent cache line buffers. The SCSI controller chips interface to the local I/O buses routed to the mass storage disks on the front of the system. The Ethernet controllers interface to the local area network. Thinwire and standard thickwire Ethernet connections are available on the module. The FEPR0Ms store console code program. The toy clock keeps system time. The serial control bus interface provides an interrupt driven master interface to the serial control bus.

Figure 1-13 shows high level functional partition for the module.

**Figure 1-13 I/O Module Functional Diagram**

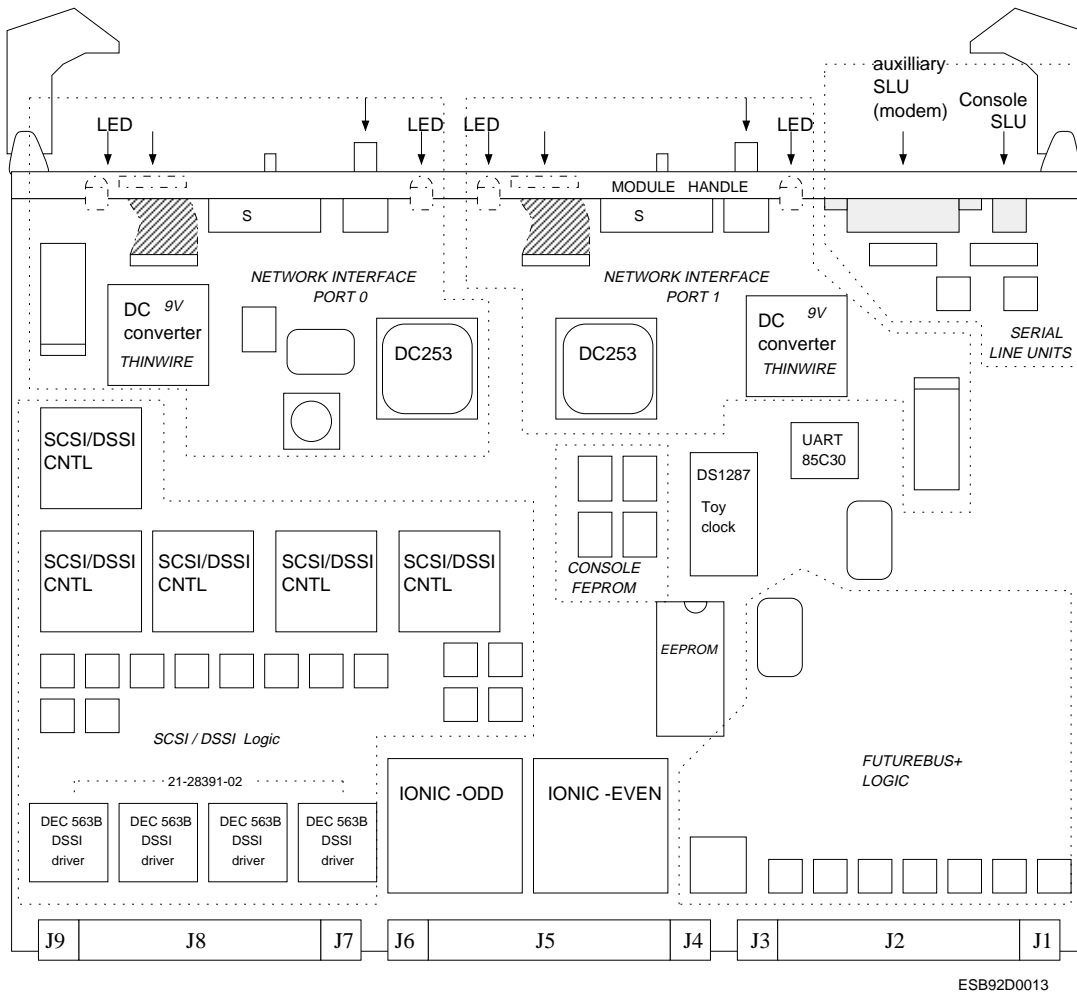


LJ-02056-T10

The I/O module is physically the largest of the CPU subsystem modules. The backplane connectors are right angle 129-pin and 24-pin connectors located on the bottom of the module. The module handle, complete with serial line connectors and Ethernet connectors, is located at the top of the module. The majority of the I/O module's components are implemented in surface-mount technology and inserted onto side 1 of the module. The exceptions are the IONIC PGA chips, some discrete DIP and SIP packages, and the EPROM. Figure 1-14 is a drawing of the I/O module.

## 1.3 KN430 CPU Subsystem

Figure 1-14 The KFA40 I/O Module



### 1.3.4.1 EPROM

The EPROM is implemented in a 28-pin X2864AP ROM dip package.

### 1.3.4.2 FEPROMS

The FEPROMS are implemented in four 32-pin CMOS flash-erasable PROMS used to store console programs as well as the Digital Alpha privileged library architecture code used by the 21064 microprocessor.

### 1.3.4.3 Ethernet Controllers

Two Third Generation Ethernet Controller chips (ports 0 and 1) with 9 volt converters, are located close to the top of the module where the thinwire and thickwire Ethernet port connectors attach to the handle.



### 1.3.4.4 Serial Line Units

Two serial line units (SLU) and a 44-pin 85C30 UART provide for serial line communications with the console terminal and one other serial device. The accompanying serial line connectors mount in the I/O module handle.

### 1.3.4.5 SCSI Controllers

Five 160-pin SCSI controller chips and four 68-pin DSSI driver chips, along with support circuitry implemented in 22V10 pals provide the control for the mass storage SCSI drivers.

### 1.3.4.6 IONIC gate arrays

Two 299-pin PGA chips called IONICs contain the system bus interface logic and Futurebus+ interface logic as well as some discrete Futurebus+ interface support logic.

### 1.3.4.7 Time of Year Clock (TOY)

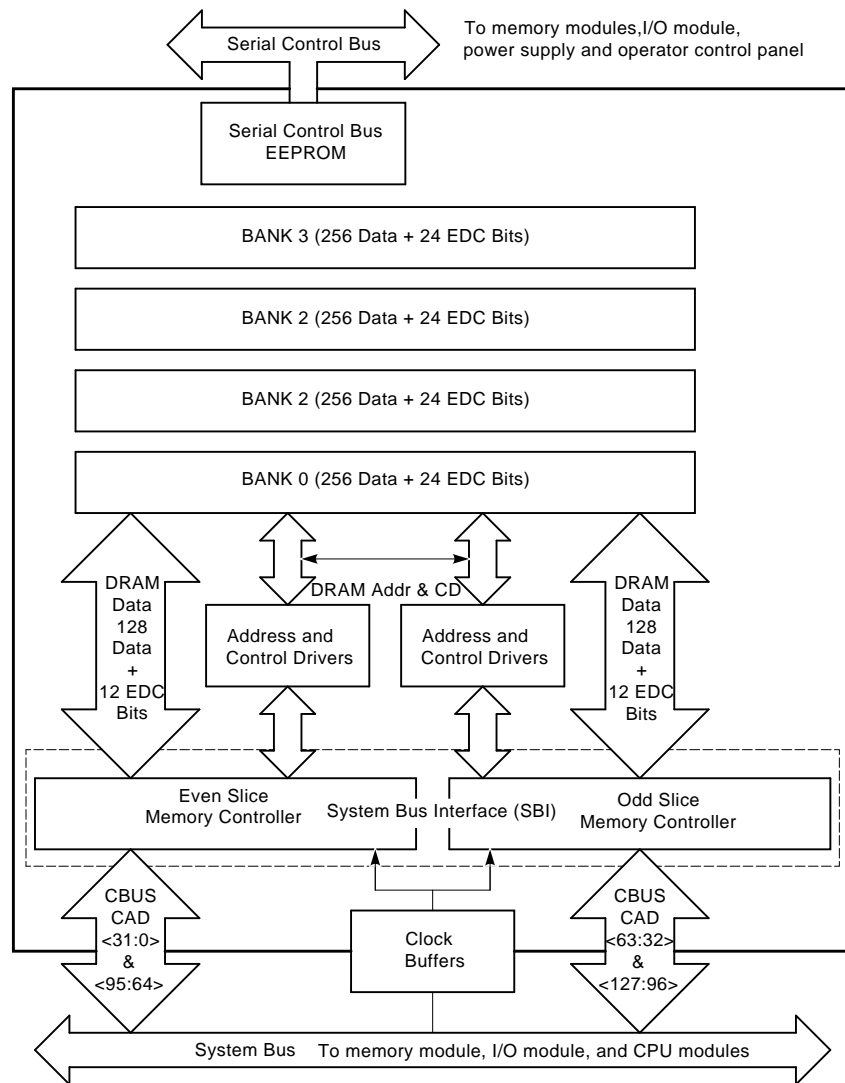
The time of year clock is a non-volatile clock that keeps the system time and date. There are several registers associated with the TOY clock function. These registers are described in detail in the I/O module chapter on functionality.

## 1.3 KN430 CPU Subsystem

### 1.3.5 MS430 Memory Module

The memory module provides high-bandwidth, low latency program and data storage for the DEC 4000 system. These modules use DRAM memory chips for storage and high-performance CMOS ASIC chips for all interface and control logic. The memory design uses error detection and correction circuitry that enhances data integrity and reliability, and availability. Figure 1-15 shows a block diagram of the MS430 Memory module.

Figure 1-15 MS430 Module Functional Block Diagram



LJ-02055-T10

The module has the same mechanical dimension as the CPU module. The memory module's backplane connectors are right-angle 129-pin and 24-pin connectors located on the bottom of the module. The module handle is located at the top of the module.

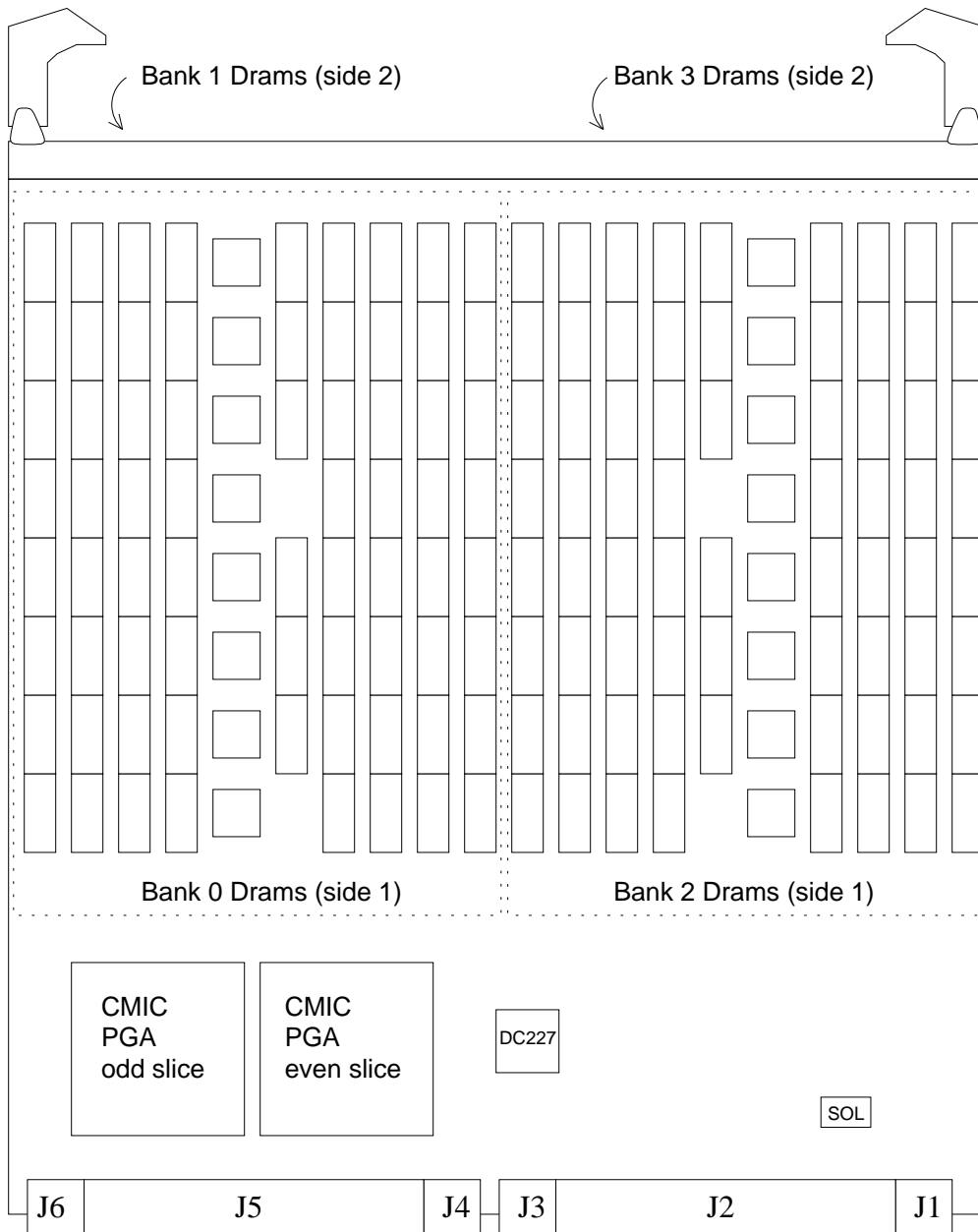
## 1.3 KN430 CPU Subsystem

The module's components are predominantly DRAM memory chips. Driver chips and CMIC chips are also present. The majority of the memory module's components are implemented with surface-mount DIP packages, with the exception of the CMIC chips. The module's components are split almost equally between side 1 and side 2 of the module, with memory banks 0 and 2 residing on side 1 of the module and banks 1 and 3 residing on side 2.

The CMIC chips are located at the bottom of the module near the J5 backplane connector. The DEC 4000 system can use up to four memory modules. The modules must occupy backplane slots 4 through 7 from the right. The slots are well marked. Refer to Figure 1-10 to see where the CPU subsystem modules are located. Figure 1-16 is a drawing of a fully populated memory module.

## 1.3 KN430 CPU Subsystem

Figure 1-16 The MS430 Memory Module



A list of significant components on the memory module follows:

### 1.3.5.1 DRAMS

DRAMS 20-pin surface-mount (140 per side)

### 1.3.5.2 Driver chips

Driver chips 28-pin surface-mount

### 1.3.5.3 CMIC chips

Two 299-pin PGA CMIC bus interface chips

## 1.4 The System Bus

The DEC 4000 system bus is a synchronous multiplexed interconnect between the processor module(s), the I/O module, and up to four memory modules. The system bus is used exclusively between these modules and is not intended as a public bus. This bus uses a five-state snooping protocol that allows a processor's first level write-through caches and second level writeback cache to maintain consistent data with another processor's caches, the system memory, and the I/O port on a transaction-by-transaction basis.

### 1.4.1 Supported transactions

The system bus supports four types of transactions: Null, Exchange, Read, and Write.

The system bus provides a snooping protocol for write back cache coherent 32-byte block read and write transactions to system memory address space. The protocol also facilitates non-cacheable address space directed read and write transactions to primary registers which do not stall longer than a memory transaction's time.

A system bus commander reads or writes a hexword of data to or from up to four buffered memory modules, or to registers in memory, on the I/O module, or CPU CSRs. A system bus commander node is constrained to performing one transaction per bus grant.

When commander nodes perform non-cacheable address space hexword write transactions, acceptance of the hexword aligned longword or quadword data is the responsibility of the responder node, as no mask is provided. However, the commander does not perform transactions to unaligned octawords within the aligned hexword.

When commander nodes perform non-cacheable address space hexword read transactions, the responder node is responsible for supplying an aligned hexword of data and the commander node consumes the needed longword or quadword, as no mask is provided. The non-cacheable node will not perform commander transactions to system bus non-cacheable address space, however the system bus provides no prevention mechanism.

---

#### Note

---

Primary non-cacheable address space responder nodes must acknowledge every read or write transaction, regardless of the read only or write only nature of the addressed register. The responder node is responsible for the effect of these transactions.

---

The memory connectors provide a unique slot identification code to each memory which is used to configure the CSR registers address space. The CPU<sub>1</sub> connector provides an identification code which is used to disable the clock drivers and to configure the CSR registers address space.

## 1.4 The System Bus

The memory modules provide 1-, 2- or 4-way hexword interleave if the modules are the same size and are populated with the same size DRAM chips. The size of each module's memory banks is determined by reading each module's configuration CSR.

### Arbitration

Arbitration on the system bus is performed by the two processor modules and the I/O module. An arbitration control on CPU<sub>0</sub> maintains a "round-robin" scheme of arbitration. This scheme allows the I/O module to interleave with the two processors. CPU<sub>0</sub> is responsible for system bus arbitration of CPU<sub>1</sub>. The I/O module is responsible for its own arbitration.

This bus can transfer a 34-bit address or a 128-bit piece of data with 32-bit parity in a single cycle. DEC 4000 system bus transactions always transfer hexword aligned data as two aligned octawords. The order of the octawords within a hexword is specified at address time, for example, data wrap is supported in memory address space based on address 4 which is a DEC 4000 system bus signal CAD<2>. Octaword within hexword wrap is not supported in I/O address space.

The CPU<sub>0</sub> node is responsible for system bus arbitration of the CPU<sub>1</sub> node and the I/O node is responsible for itself. I/O node arbitration is interleaved with the CPU nodes to minimize I/O latency. This round robin scheme of: CPU<sub>0</sub> - I/O - CPU<sub>1</sub> - I/O - CPU<sub>0</sub>, where if an idle cycle passes a first come granted policy, is effected and maintained by the arbitration control logic located on the CPU<sub>0</sub> node.

An I/O node or CPU node is permitted to hold bus request asserted to perform continuous system bus transactions, the arbitration controller continuously samples requests when the system bus is idle and resamples requests near the end of each current transaction.

The arbiter does not require system bus request signals to negate before they reassert. However, a granted node must perform a system bus transaction without hesitation.

A null transaction is provided to enable a commander to nullify the active transaction request or to acquire the system bus to avoid resource contention. A requesting commander is not permitted to withdraw a request prior to being granted. The request and grant must result in a valid system bus transaction.

The arbiter drives the signal on the system bus indicating that a commander is about to drive the address and command packet, and the granted commander is obligated to comply.

Request signals may assert or negate during stalled cycles of a transaction. The arbitration controller must monitor the bus transaction type and follow the transactions, cycle by cycle, in order to know when to rearbitrate and signal a new address and command cycle.

### Idle cycles

The arbitration controller has provision for masking CPU<sub>0</sub>, CPU<sub>1</sub> or I/O node arbitration in support of diagnostics, it also maintains a counter of contiguous transactions which is used to inject idle arbitration cycles. These idle cycles prevent probe traffic from consuming the backup cache from a CPU node for extended periods.

### Interrupts

The I/O module drives two device related interrupt signals which are received by both CPU nodes. One interrupt is associated with the Futurebus+ and the other is associated with all the device controllers local to the I/O module.

The I/O module provides a readable non-cacheable address space silo register of Futurebus+ interrupt pointers and a readable noncacheable address space device request register of local device interrupt requests.

It is assumed that either CPU<sub>0</sub> or CPU<sub>1</sub> will be the designated primary interrupt dispatch node, responsible for reading the device interrupt register or Futurebus+ interrupt register to determine which local devices require service or which Futurebus+ offset to dispatch into.

---

#### Note

---

Either CPU node is capable of servicing all or either I/O device interrupts.

---

A system bus readable and writable interprocessor interrupt register and interprocessor mailbox register is provided on each CPU node. The interrupt register has one bit which is set to cause an interrupt and cleared by the interrupted processor. It is possible for a CPU to interrupt another CPU and itself. A CPU should test that the bit is cleared prior to setting the bit. The interprocessor mailbox register is longword wide and at least one bit should be used as an ownership flag.

The interval timer interrupt clock is distributed to both CPU nodes from the I/O node, because the time of year clock and 1 ms interval timer clock are located on the I/O module.

**Soft errors** are errors that are dynamically correctable. The system bus assumes that soft errors, are logged with or without a posted interrupt, to be serviced or polled by system software in proximity to the event or at periodic intervals. An error interrupt signal is available to all system bus nodes; it is used for soft and hard error interrupt reporting. Hard errors are generally not recoverable in a level of operating system software, but may be recoverable in driver or user levels of system software.

The system power supply, the operator control panel, the Futurebus+ halt CSR, the Ethernet interface controller, or the console serial line interface are capable of causing a system event interrupt. A non-maskable interrupt is provided to signal both CPU nodes of the system event. System events could be one or more of the following:

- AC or DC power failure
- Fan failure
- Overtemperature shutdown
- Reserve battery activated
- Reserve battery low voltage warning
- Halt
- Remote reboot

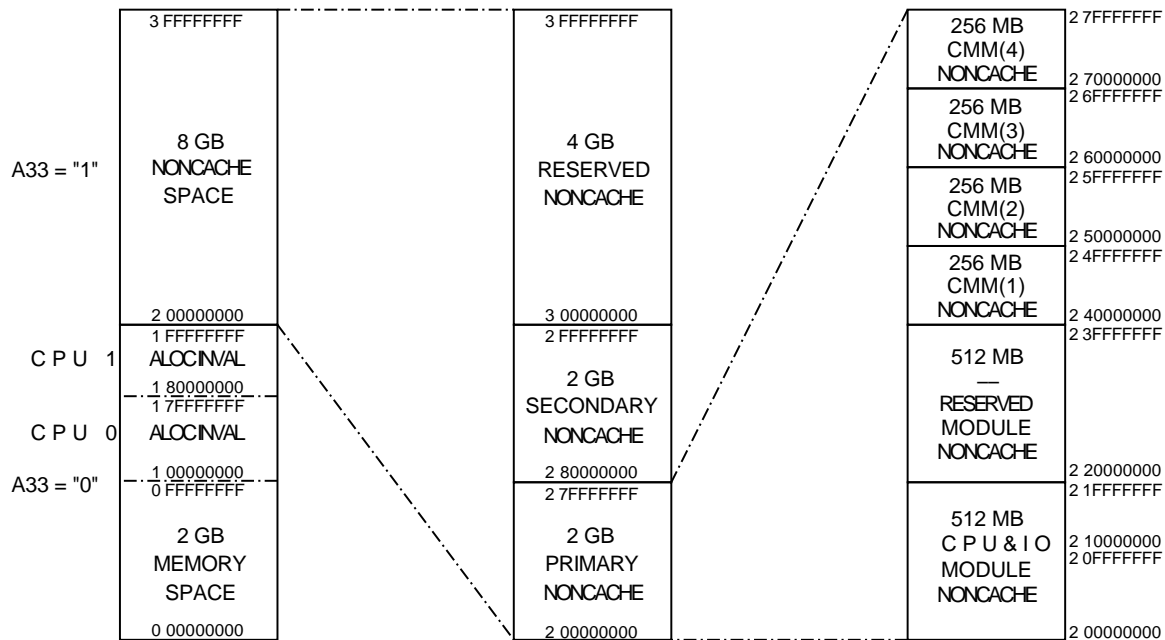
## 1.4 The System Bus

### 1.4.2 Physical Address Space Layout

Figure 1–17 illustrates the division of the address space into memory space and non-cacheable space. The upper 4 GB of the 8 GB memory space is reserved and called allocate invalid address space. This space is divided into two sections, each used by a CPU node as a backup cache address alias for flushing dirty lines without allocation. The 512MB of primary I/O module CSR space for CPU and I/O nodes is partitioned as follows:

- 2 0000 0000 - 2 07FF FFFF for CPU<sub>0</sub>
- 2 0800 0000 - 2 0FFF FFFF for CPU<sub>1</sub>
- 2 1000 0000 - 2 1FFF FFFF for I/O

Figure 1–17 Physical Address Space Layout in the System



NOTES: 1. ALL ADDRESSES ARE PHYSICAL (PA<33:0>).

ADDMAP

All 32 bits of address are significant for non-cacheable space addresses, because device registers may have to be decoded uniquely on hexword boundaries. There is no length field for system bus transactions. All hexword transactions in non-cacheable space must occur to aligned hexword addresses. The actual amount of data read or written may be a quadword or less.



### 1.4.3 System clock signals

The processor clock is the source of the timing signals in the DEC 4000 system. It is an ECL level signal with a nominal frequency of 151.51 Mhz. Each processor module generates its own CPU clock independently of the other processor module.

The system bus clock is a free running clock with a frequency that is derived by dividing the processor clock by four. A system bus clock generator and distributor is located on the CPU<sub>0</sub> processor module. Each clock signal is source terminated on the CPU<sub>0</sub> module and delivered to the backplane so that the absence of a module will not negatively affect the clock driver chip.

The primary processor module, CPU<sub>0</sub>, always generates the system bus clock. The backplane connector for CPU<sub>1</sub> provides an identification code which is used to disable the clock drivers and configure the CSR register address space if CPU<sub>1</sub> is plugged into the backplane.

Two pairs of differential clock signals are generated. Each module on the system bus receives a differential pair of clock signals. The module must terminate the signals and AC couple them to a chip that converts them to CMOS level signals for on-module use. Two of each of these signals are dedicated to driving the system bus interface chip. (Each module on the system bus uses a system bus interface chip.)

In all, there are three time domains in the DEC 4000 system. The DECchip 21064 CPU chip (6.6 ns), the system bus (24 ns), and the CPU chip to C<sup>3</sup> (18 ns).

### 1.4.4 Subsystem communications with secondary buses

The secondary buses in the DEC 4000 system are the Futurebus+, which is an open bus available for the user, and the L-bus which is a 32-bit bus that provides communication to the local I/O devices. The L-bus is not available to users, it is contained on the I/O module. Read operations directed at the secondary buses are very slow relative to the processor clock speed.

A CSR read may take 1 to 10 µsec or even longer, while a processor clock cycle is under 10 nsec. In order to decouple the comparatively long access latencies of these buses from the system bus (which the processor uses for access to main memory) these buses are not directly accessible by the processor. Communications between the system bus and the secondary buses is achieved through the interfaces on the I/O module.

A mechanism called a *mailbox* provides access to devices on either of the secondary buses. There are two mailbox pointers, the L-bus Mailbox Pointer Structure (LMBPR) for operations to the local I/O devices, and the F-bus Mailbox Pointer Register (FMBPR) for operations to devices on the Futurebus+.

The processor builds a structure in main memory called the Mailbox Data structure that describes the operation to be performed. The processor then writes a pointer to this structure into a Mailbox Pointer Register. The I/O module reads this Mailbox Data Structure and performs the operation specified (read or write) and returns status and any data to the structure in memory.

## 1.5 The Mass Storage Area

### 1.5 The Mass Storage Area

The mass storage area has four bays for fixed media and removable media devices. Each fixed media device has a drawer part, a disk power supply and the individual disks.

These disks are responsible for holding system software. One disk is required to hold the operating system software. Each can be expanded. The disk power supply converts power from the system power supply to power that the disk assemblies can use.

The mass storage area at the front of the system contains mechanical assemblies that accept a combination of different storage devices. Each storage assembly includes the following:

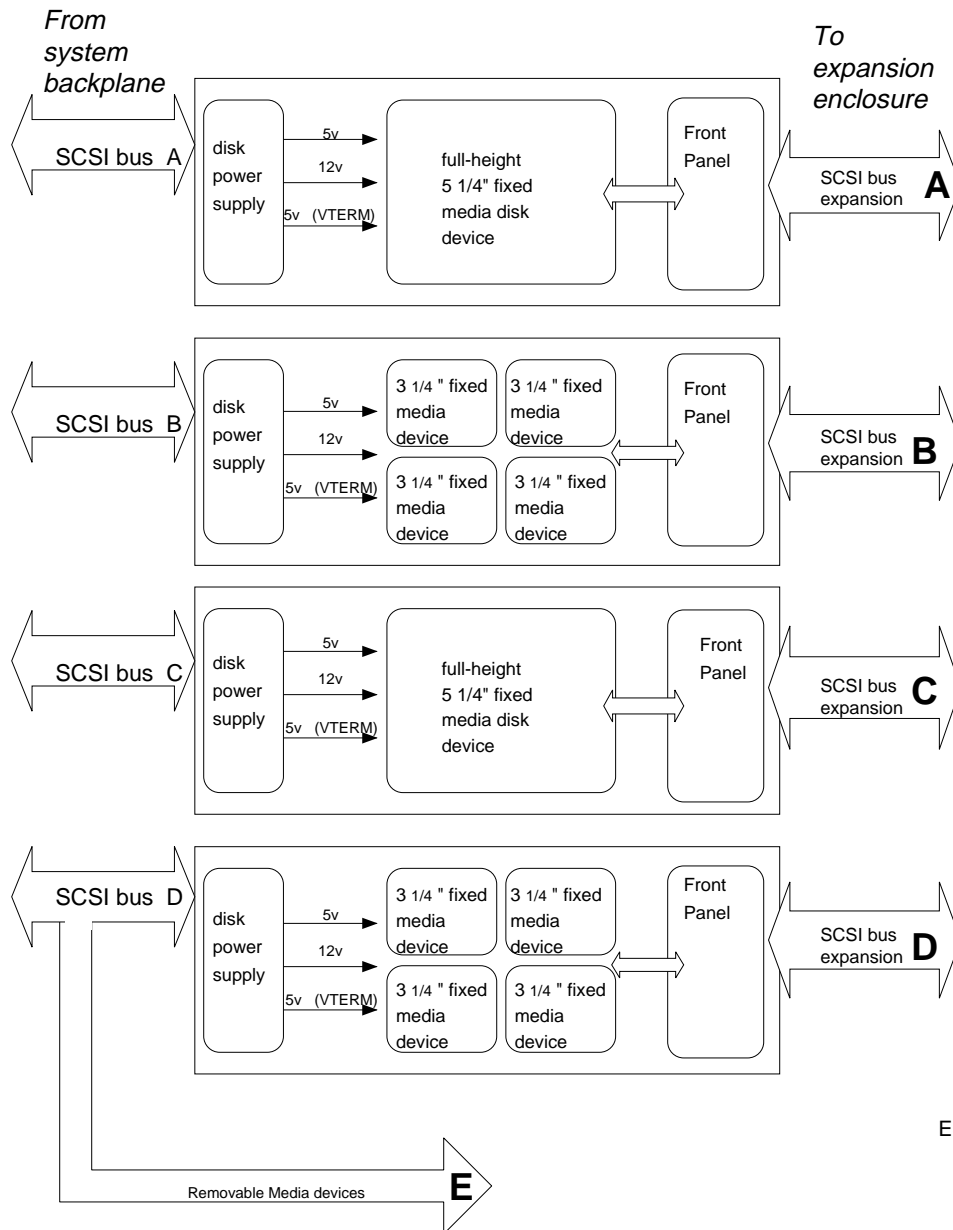
#### **Disk power source**

- A dedicated power source module known as the local disk converter that converts 48 volts dc bus to three voltages which all share a common return:
  - 5 volts DC for powering the storage device logic
  - 5 volts DC termination power for the I/O bus terminators
  - 12 volts DC SCSI voltages
- A front panel that contains unit ID plugs and fault and online indicators
- A connector to allow for SCSI bus expansion

Figure 1-18 shows a functional block diagram of the mass storage area.

## 1.5 The Mass Storage Area

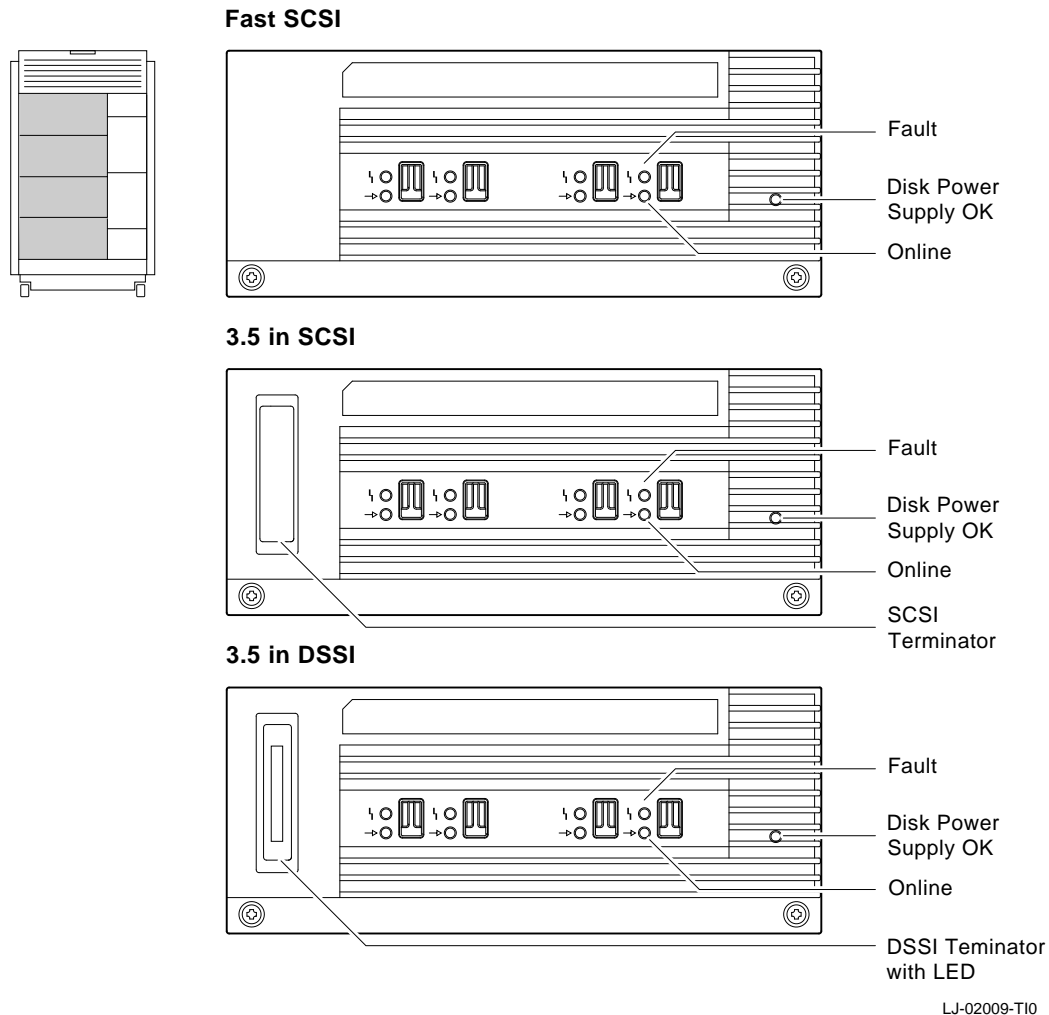
Figure 1-18 The Mass Storage Area



The mass storage area is on the front of the system. The system can have up to four fixed-media storage devices and two removable-media storage devices. Each fixed media device may contain a combination of full-height and 3-1/4 inch devices, so there are several different front panels available for the front of the mass storage area. Figure 1-19 shows the typical mass storage front panels.

## 1.5 The Mass Storage Area

Figure 1-19 Mass Storage Front Panels



## 1.6 I/O Expansion Area

The I/O expansion area has six slots for Futurebus+ modules.

These slots provide space for the system to be expanded with Futurebus+ I/O devices, both Digital and non-Digital made. The I/O expansion bus is implemented with the Futurebus+ Profile B open standard. Futurebus+ Profile B is a general purpose, high performance, technology independent bus. A profile is a specification which calls out subset functions from a larger Futurebus+ specification. All I/O expansion option modules must conform to the dimensions for the Futurebus+ Profile B specification.

The I/O expansion area provides space to install six Futurebus+ option modules on the system backplane. Futurebus+ modules must occupy Futurebus+ slots (1-6) to the left of the memory modules. The Futurebus+ is separate from the system bus and all communications between the Futurebus+ modules and the CPU subsystem are facilitated by the CPU subsystem's I/O module.

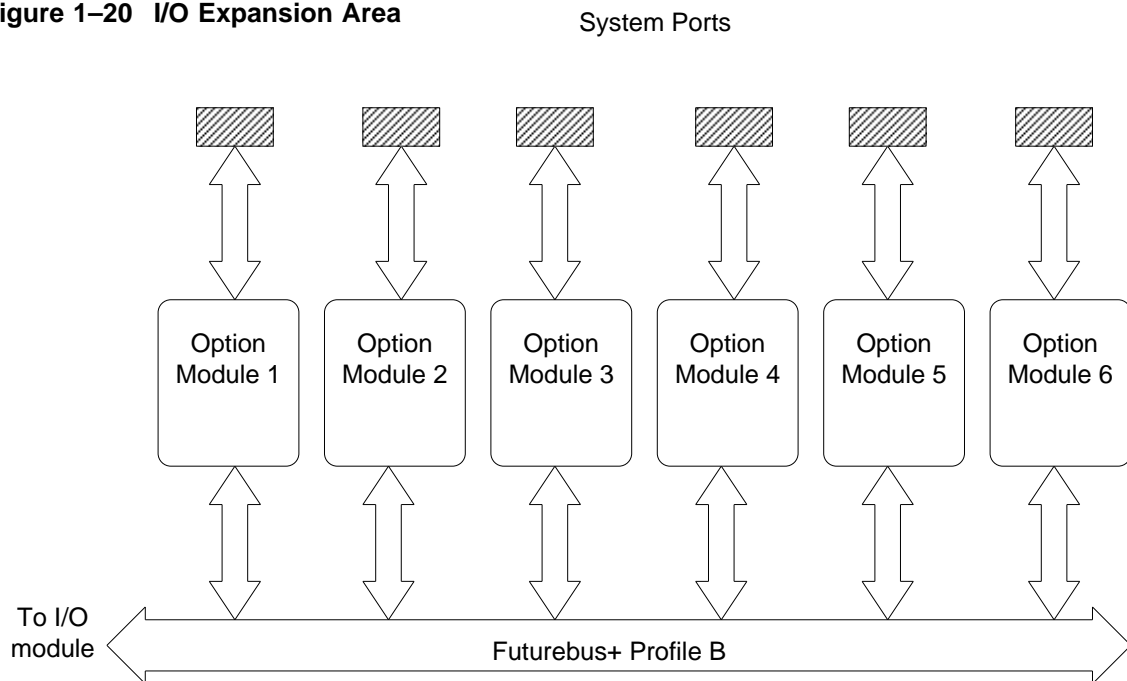
## 1.6 I/O Expansion Area

The Futurebus+ Profile B bus is a 32- or 64-bit, multiplexed address and data bus located in the system backplane. It uses the following:

- Central arbitration scheme
- Simple transaction set
- Futurebus+ CSR set
- A simple interrupt scheme
- A high-performance DMA architecture

The Futurebus+ will support communication, disk controllers, and special purpose devices as required. Figure 1–20 shows a functional block diagram of the I/O expansion area.

Figure 1–20 I/O Expansion Area



ESB91D0024

## 1.7 The Power Supply Subsystem

The power supply subsystem is an integral part of the BA640 enclosure. It is accessed from the back of the system and has the following components:

- H7853 front end unit (FEU)
- H7851 power system controller (PSC)
- H7179 DC-DC converter unit, 5 volt (DC5)
- H7178 DC-DC converter unit, 12/3.3/2.1 volt (DC3)

## 1.7 The Power Supply Subsystem

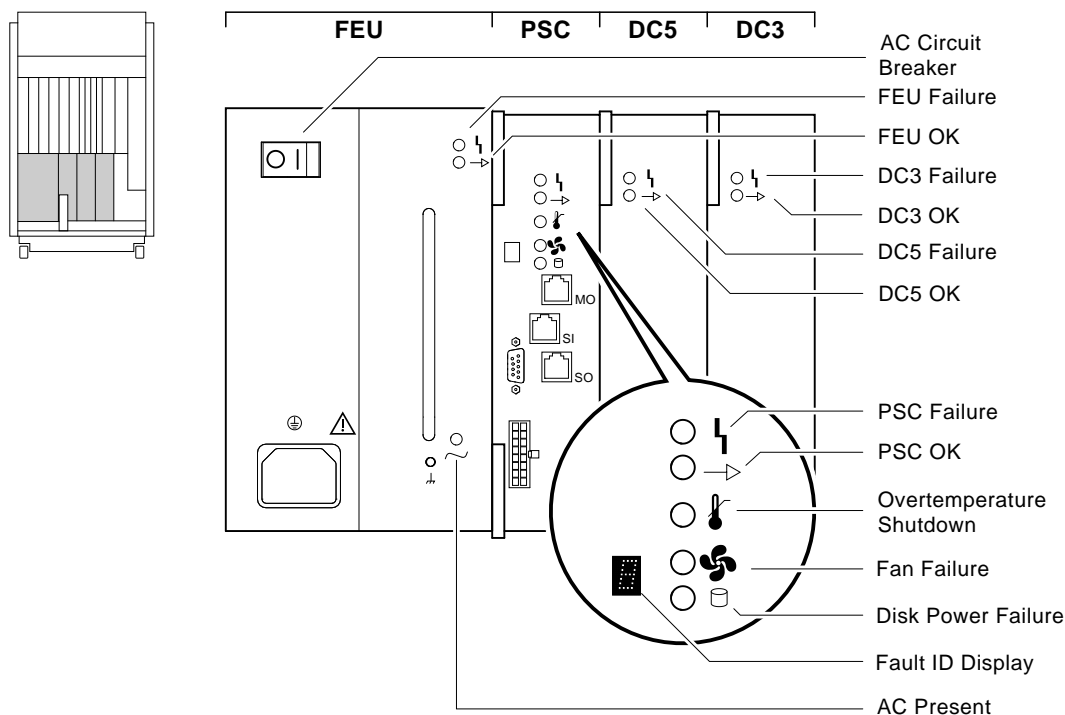
- Up to four remote local disk power converters called "disk power supplies" that provide power for the removable-media storage devices.

The FEU, PSC, DC5 and the DC3 modules are replaceable units that plug into the system backplane. The local disk converters are DC-DC power supplies that are built into the fixed media assemblies. All interconnects between the modules are routed through the backplanes. Cooling is provided by air flowing from the Futurebus+ card cage area, through the power supply system, and into the fan plenum below.

The front end unit module is enclosed in sheet metal and screening to prevent contact with the high voltages inside it. The power system controller module and the two DC-DC converter modules are open modules with backplane connectors on the back and modular bulkhead handles on the front.

Figure 1-21 shows the location and meaning of the power supply indicator LEDs.

Figure 1-21 The Power Supply Indicators



LJ-02011-T10

The power supply subsystem converts AC power to DC power that can be used by the system. This universal supply contains a high-power-factor AC-DC converter that is capable of working over the entire 120-240 volts AC input line voltage without any reconfiguration of its circuitry. This eliminates the need for the user to set a switch to select 120 or 240 volts AC.

Logic on the PSC module monitors system voltages and temperature and adjusts the fans accordingly. Two DC-DC converter units (DC5 and DC3) provide 5, 2.1, 3.3, and 12.0 volts DC for various uses throughout the system.

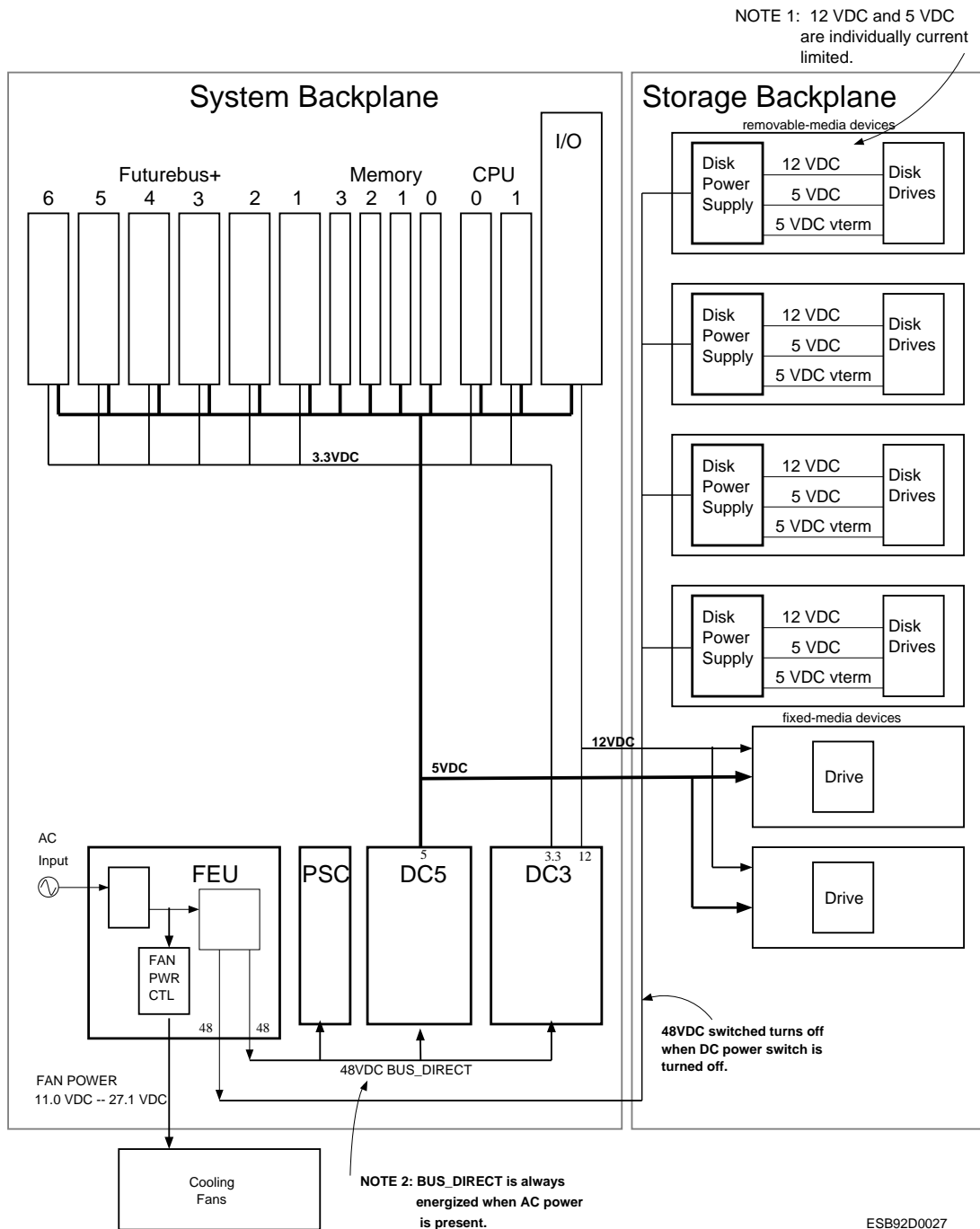
## 1.7 The Power Supply Subsystem

The disk power supplies provide 12.0 and 5.0 volts DC and a termination power called Vterm (5 volts DC) power for the removable-media disk assemblies.

Figure 1-22 shows a high-level block diagram of the power supply subsystem and shows the way power is routed in the system.

## 1.7 The Power Supply Subsystem

Figure 1–22 The DEC 4000 Power Scheme



Each power supply component is described in more detail in the following paragraphs.



## 1.7 The Power Supply Subsystem

### 1.7.1 H7853 Front End Unit (FEU)

The front end unit takes in AC power from the mains at near unity power factor. The input AC is processed through a high-voltage power factor (HPF) AC-DC converter to create a high-voltage DC (HVDC) bus at 385 VDC. The HVDC is connected to the inputs of two DC-DC converters. These converters, known as the step-down converters, process the 385 volts DC into a nominal 48 volts DC. The outputs of these two converters are connected together and operate in parallel.

Circuitry on each converter assures that the load current is shared, almost equally between the two converters. It is important to note that neither the supply nor the return of these 48 VDC buses is referenced directly to chassis or earth. Instead, each rail is connected to chassis through a parallel combination of a large value resistor with a small value capacitor. This creates a floating 48 volts DC that, if measured with a volt meter, would indicate the supply rail to be at +24 VDC and the return rail to be at -24 VDC.

It is also important to note that any loading on the 48 VDC bus be isolated from the chassis. All current taken from the supply side of the bus *must* be returned to the return side of the bus. No current can be returned to chassis or earth connection. This means that all DC-DC converters drawing power from either of the 48 VDC sources must be galvanically isolated input to output.

### 1.7.2 H7851 Power System Controller (PSC)

The power system controller is the brain of the power supply subsystem. It provides all of the monitoring and interface functions of the power system. The main processing component of this module is the Intel 80C196 microprocessor. The PSC module provides the following services:

- Communicates system status information to the KN430 system processor module via the serial control bus
- Receives commands, status, and configuration information and other communications from the system via the serial control bus
- Initiates power-down sequence if the FEU indicates that less than 4 msec of reserve energy exist
- Provides a power fail signal (POK H) to the mass storage devices and the I/O module
- Receives on/off and system restart commands from the OCP
- Generates a system reset (ASYNC\_RESET L) on power-up and when requested by the OCP
- Drives the power system visual status indicators
- Provides power-up sequencing for the DC-DC converters
- Passes the fan power from the backplane to the fans through anti-beat diodes
- Provides the system with warning if temperatures are beyond normal operating range
- Initiates the power-down sequence when temperatures are excessive
- Controls the speed of the fans
- Interfaces to external UPS and conveys status to the system
- Initiates a shutdown if the FEU indicates that one of the 48 VDC buses is out of tolerance

## 1.7 The Power Supply Subsystem

- Monitors the system temperature and cooling with thermal emulators
- Monitors the backplane voltages and shuts down when there is an undervoltage fault
- Monitors the backplane voltages and fires the crowbar and shutdown on an overvoltage fault
- Monitors status of the disk power supplies and reports failures to the system
- Monitors fan rotation and initiates a power-down sequence on detection of a failure
- Interfaces to the power control bus

### 1.7.3 H7179 5 Volt Converter (DC5)

This unit receives power at 48 volts DC, processes it through three DC-DC converters operating in parallel and generates an output of 5 volts DC @ 90 amps for the system's logic modules.

### 1.7.4 H7178 3 Volt Converter (DC3)

This module has three DC-DC converters on it, all operating from the 48 volts DC input. This unit produces the following outputs:

- 12 VDC @ 12.5 A (150 W)
- 3.3 VDC @ 30 A (100 W)
- 2.1 VDC @ 10 A (21 W)

### 1.7.5 Disk Power Supplies

The disk power supply module is a DC-DC converter specialized for powering small mass storage devices. Input power is taken from the 48 VDC bus and three outputs are generated:

- 12.0 volts DC with a fast transient response and tolerance to the short-term loading during spinup
- 5.0 volts DC for powering storage device logic
- 5 volts DC Vterm termination power, a 5 Volt output that is diode-isolated and current limited for powering the local I/O bus terminators.

### 1.7.6 System Voltages

The following lists all of the system voltages:

- 48 volts DC BUS\_DIRECT
- 48 volts DC SWITCHED
- 5 volts DC
- 2.1 volts DC
- 3.3 volts DC
- 12.0 volts DC
- Fan power 11.0 - 27.1 volts DC
- Disk power supply voltages:
  - 12.0 volts DC
  - 5 volts DC

5 volts DC Vterm termination power

## 1.8 The Serial Control Bus

The serial control bus is a two-conductor serial interconnect that is independent of the system bus. The bus master for the serial bus is the I/O module. This bus links the processor modules, the IO, the memory, the power subsystem and the OCP. Each module on the bus has an EEPROM that captures data that can later be interpreted by a field service technician. This bus reports any failed devices to the processor module so the processor module can illuminate LEDs on the OCP. The system accomplishes a data write to this bus with the execution of a single privileged instruction. The write is completed without further system software intervention. The write to EEPROMS requires 25 ms maximum. Table 1-6 lists the node addresses on the serial control bus.

**Table 1-6 Serial Control Bus Node Addresses**

Device	Bus Node	Read Address	Write Address
D-bus microcontroller	CPU <sub>0</sub>	B1	B0
D-bus microcontroller	CPU <sub>1</sub>	B3	B2
NVRAM 256 x 8	CPU <sub>0</sub>	A9	A8
NVRAM 256 x 8	CPU <sub>1</sub>	AB	AA
NVRAM 256 x 8	I/O	AD	AC
NVRAM 256 x 8	Mem <sub>0</sub>	A1	A0
NVRAM 256 x 8	Mem <sub>1</sub>	A3	A2
NVRAM 256 x 8	Mem <sub>2</sub>	A5	A4
NVRAM 256 x 8	Mem <sub>3</sub>	A7	A6
8-bit register	OCP	41	40
8-bit register	OCP	43	42
8-bit register	VTERM	45	44
8-bit register	PSC	49	48
8-bit register	PSC	4B	4A
I/O port to serial control bus		B5	B4

## 1.9 System Firmware

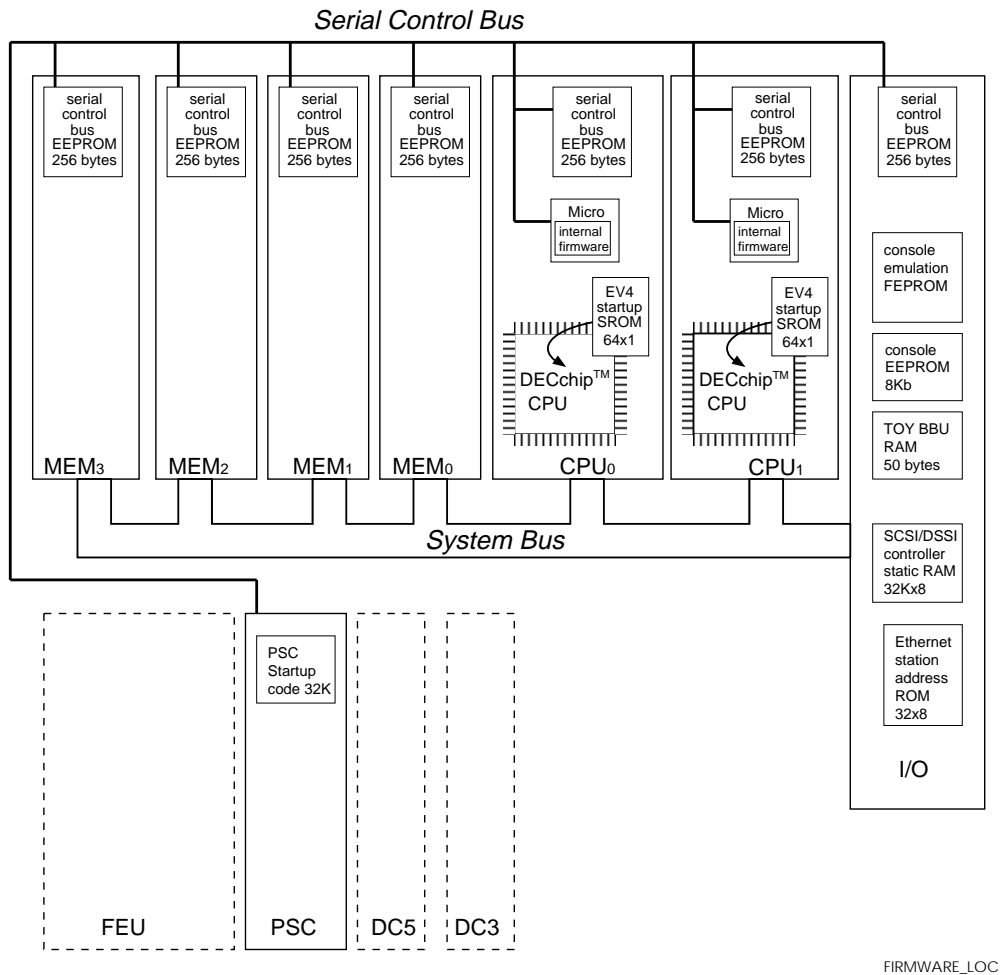
The term *firmware* refers to code that is stored in a fixed or firm way usually in a read-only memory device. It contains instructions designed to help hardware perform its assigned functions. The DEC 4000 system contains several such pieces of firmware located in various devices throughout the system.

In general the DEC 4000 system firmware is "soft" in that it executes out of memory. On power-up, code is transferred from the FEPROMS to the main memory and the CPU runs this code out of main memory. This is different from previous products where the console program is run out of firmware EPROMs.

## 1.9 System Firmware

The functionality of each piece of firmware is described in the following paragraphs. The locations of the firmware in the DEC 4000 system are shown in Figure 1–23.

Figure 1–23 System Firmware Locations



### PSC Startup Firmware

The power system controller startup code is stored in one ultraviolet EPROM (device number 27256). The UVPROM is packaged in a 28-pin, dual in-line chip located on the power system controller module (currently in position E10). The UVPROM holds 32K of lower memory that may be expanded with another 32K of upper memory. This code is invoked on power-up after the deassertion of the system reset signal. This code is responsible for starting and controlling the 80C196 microprocessor by performing the following functions:

- Monitors the power supply voltages
- Monitors the AC power line
- Facilitates communication with the processor modules

- Monitors and controls system temperature

### **The DECchip 21064™ CPU Startup Firmware**

The CPU chip startup code is stored in four 20-pin, PLCC, 64x1 serial configuration PROMS (SROM). A copy of this code is present on both processor modules. This code is loaded into the 21064 instruction cache after the deassertion of the system reset signal. This code works with the micro controller to issue commands to the 21064. This code tests the most basic functions of the processor module doing the following:

- Tests the backup cache
- Tests the system bus
- Tests the console memory
- Unloads the console program from the FEPROMS

### **Micro Controller Startup Firmware**

The 87C652 microcontroller used on the processor modules contains internal firmware that is invoked on initialization after the deassertion of system reset. The 87C652 is a 40-pin PLCC chip. A copy of this code exists on both processor modules. The 87C652 resides in location EX on both CPU<sub>0</sub> and CPU<sub>1</sub>. This code supports the initialization procedure by performing the following functions:

- Monitors serial ROM load progress
- Enables serial port UART when serial ROM load completes.
- Sizes for optional support module on the serial control bus
- Reads memory module configuration data from the memory modules EEPROMs on the serial control bus.
- Sends the memory module configuration data to the CPU via the serial port.
- Controls the CPU power-up test flow and FEPRM unloading
- Sends the progress information to the LEDs on the Operator Control Panel by way of the serial control bus.

### **Console Emulation and Diagnostics Firmware**

Console Emulation and Diagnostics code is stored in four surface mount, 32-pin flash EPROMS located on the I/O module. Chips E77, E91, E78, E92 represent bytes 0, 1, 2, and 3 respectively. Each chip has 128K of non-partitioned flash memory. This code participates in the system initialization process and is responsible for system boot, console emulation, and console diagnostics. An FEPRM is an electrically erasable EPROM where all the bits are erased in parallel. FEPROMS can be erased and programmed while remaining installed on the module. An upgrade takes approximately 30 seconds.

**FEPRM Code Structure** The DEC 4000 FLASH PROM firmware has several major functional blocks of code. The firmware is invoked following system halts, or severe errors. It is responsible for saving the machine state. It controls the transition from halted state to the running state and it performs a restoration of the saved context prior to the transition.

### **Console Based Diagnostics**

The console diagnostics consist of kernel initialization, driver initialization, subsystem tests, and system exercisers.

## 1.9 System Firmware

### Restart and Boot Code

On invocation, depending on the nature of the halt and the system context, the firmware attempts either an operating system restart, a bootstrap operation, or transitions to console I/O mode.

The console is responsible for restarting a processor halted by powerfail or by error halt. The console follows the same sequence for a primary processor or a secondary processor.

### Standard Console Services

On a Digital Alpha APX system, underlying control of the system hardware is provided by the console. The console provides the following:

- Initializes, tests, and prepares the system hardware for Alpha system software.
- Bootstraps (loads into memory and starts the execution of) system software.
- Controls and monitors the state and state transitions of each processor in a multiprocessor system.
- Provides services to system software that simplify system software control of and access to hardware.
- Provides a means for a *console operator* to monitor and control the system.

The console interacts with system hardware to accomplish the first three tasks. The actual mechanisms of these interactions are specific to the hardware; however, the net effects are common to all systems.

The console interacts with system software once control of the system hardware has been transferred to that software.

The console interacts with the console operator through a virtual display device or *console terminal*. The console operator may be a human being or a management application.

### Console EEPROM

The console uses 8kb of non-volatile EEPROM located on the I/O module. The EEPROM is implemented in a 28-pin PLCC surface mount package. Logically this EEPROM sits on the LBUS and provides space for scripts used by the console emulation software. This EEPROM also holds environment variables used by the console code.

### SCSI/DSSI Controller Static RAM

The SCSI/DSSI controllers have 32Kx8 of CMOS, parity protected, static RAM space available to use as a shared resource for the controllers. This RAM is implemented in eight 28-pin surface mount chips located on the I/O module.

### Ethernet Station Address ROM

The Ethernet station address is implemented with two 32x8 ROMs, one ROM for each Ethernet port. Each ROM holds a unique address for the system. These ROMs are located on the I/O module.

### I/O Module TOY Battery Backup RAM

The time-of-year clock on the I/O module is implemented with a Dallas Semiconductor DS1287 real-time clock. It is contained in a 24-pin dip package. The real-time clock provides the I/O module with 50 bytes of non-volatile static RAM to accommodate the TOY clock registers.

### Serial Control Bus EEPROM

Each module on the system bus has an integrated serial control bus EEPROM. The EEPROM is implemented in an 8-pin, 2K bits static CMOS device arranged as 256 by 8-bits. It has 256 bytes of byte addressable, with byte erase/write, non-volatile storage. Each EEPROM stores information about the module on which it resides. Some of the information consists of the following:

- Administrative information
  - Material tracking
  - Revision control
  - Configuration tracking
  - Product functionality
- Failure/Diagnosis information
  - Manufacturing repair
  - Customer service logistics repair
  - Systemic problem analysis
  - Fault management

## 1.10 System Ports

The DEC 4000 system contains an AC power port, two serial line ports, an Ethernet port, and up to four SCSI storage expansion ports. This section describes the external ports that exist on the DEC 4000 system.

### 1.10.1 The serial line ports (SLU)

The DEC 4000 contains two serial line ports for the console serial line and the auxiliary serial line. Both serial lines are implemented using a single 85C30 device and operate in asynchronous mode only.

The console serial line uses a 6-pin DECconnect MMJ connector mounted on the I/O module handle. This line is compatible with the RS-232-C,D,E, EIA 423, and CCITT V.28 and V.10 standards.

The auxiliary serial line provides modem control and uses a 25-pin DIN connector mounted on the I/O module handle. This line is compatible with RS-232-C,D,E, EIA 423, CCITT V.28 and V.24 standards. It is also compatible with BELL 101, 103 and 112 modems.

Table 1–7 lists a summary of the characteristics of the serial lines.

**Table 1–7 Serial Line Features**

Console Serial Line	Auxiliary Serial Line
Asynchronous	Asynchronous
No modem control	Modem control
6-pin (DECconnect MMJ) connector	25-pin (DIN)
Connected to I/O module handle	Connected to I/O module handle
Compatible with RS-232-C,D,E EIA 432	Compatible with RS-232-C,D,E EIA 432

(continued on next page)

## 1.10 System Ports

**Table 1-7 (Cont.) Serial Line Features**

Console Serial Line	Auxiliary Serial Line
CCITT V.28 and V.10	CCITT V.28 and V.24, BELL 101, 103 AND 112

### 1.10.2 The Ethernet Port

The Ethernet port on the DEC 4000 system provides connectors that accept thickwire or Thinwire standard Ethernet cables. This connection allows the system to communicate with other systems, information processing products, or office equipment at a local business site. The Ethernet port is located on the I/O module.

### 1.10.3 SCSI Storage Expansion Ports

The mass storage SCSI devices use a standard SCSI 50-conductor shielded cable. This cable is terminated at both ends and provides for daisy chain expansion. All signals are common between all SCSI devices.

### 1.10.4 AC Power Connector Port

The AC power connector port provides the connection from the DEC 4000 system to an AC power wall outlet. The cable is a standard AC power cable.



# Part II

---

## The Processor Module

This part contains a detailed functional description of the DEC 4000 processor module.

---

**The KN430 Processor Module**

The KN430 processor module is the central processing unit of the DEC 4000 system. The processor contains 11 subsystems, many of which are found in the C<sup>3</sup> chip developed at Digital Equipment Corporation. In the following list, these subsystems are found in brackets next to the C<sup>3</sup> chip.

- The DECchip™ 21064 CPU chip
- Backup cache
- C<sup>3</sup> chip {
  - Write merge buffer
  - Duplicate tag store
  - System bus interface
  - System bus arbitrator
  - Address lock
  - Bcache Control machines
  - System bus snooping control
- Clock generator/distributor
- Reset generator
- D-bus serial ROM and microcontroller
- Clock/power detect circuitry
- Address Lock to support the Load Lock/Store Conditional construct (C<sup>3</sup>)

---

## Alpha Architecture

The DEC 4000 system is an implementation of the Digital Alpha AXP architecture. The following paragraphs give a brief overview of the Alpha addressing and data types. For a complete description refer to the *Alpha Architecture Reference Manual*.

### 3.1 Addressing

The basic addressable unit in the Alpha architecture is the 8-bit byte. Virtual addresses are 64 bits long. An implementation may support a smaller virtual address space. The minimum virtual address size is 43 bits.

Virtual addresses as seen by the program are translated into physical memory addresses by the memory management mechanism.

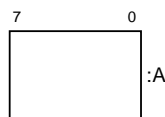
### 3.2 Data Types

An explanation of the Digital Alpha supported data types is given in the following paragraphs.

#### Byte

A byte is eight contiguous bits starting on an addressable byte boundary. The bits are numbered from right to left, 0 through 7 as shown in Figure 3-1.

Figure 3-1 Byte Data Format



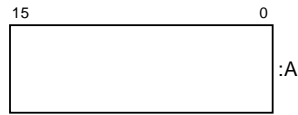
A byte is specified by its address A. A byte is an 8-bit value. The byte is supported in Alpha only by the extract, mask, insert, and zap instructions.

#### Word

A word is two contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 15 as shown in Figure 3-2.

## 3.2 Data Types

**Figure 3–2 Word Data Format**

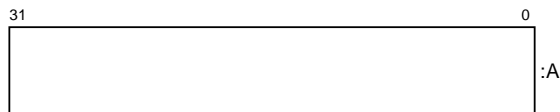


A word is specified by its address, the address of the byte containing bit 0. A word is a 16-bit value. The word is supported in Alpha only by the extract and insert instructions.

### **Longword**

A longword is four contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 31 as shown in Figure 3–3.

**Figure 3–3 Longword Data Format**



A longword is specified by its address A, the address of the byte containing bit 0. A longword is a 32-bit value. When interpreted arithmetically, a longword is a two's-complement integer with bits of increasing significance going 0 through 30. Bit 31 is the sign bit. The longword is supported in Alpha only by sign-extended load and store instructions, and by longword arithmetic instructions.

---

**Note**

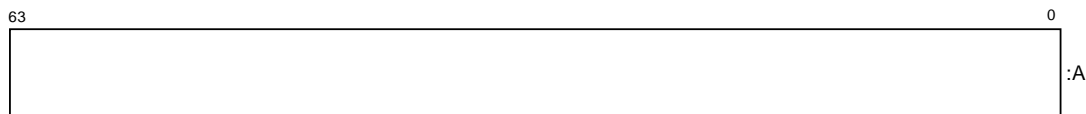
Alpha implementations impose a significant performance penalty when accessing longword operands that are not naturally aligned. (A naturally aligned longword has 0 as the low-order two bits of its address.)

---

### **Quadword**

A quadword is eight contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 63 as shown in Figure 3–4.

**Figure 3–4 Quadword Data Format**



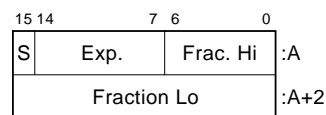
A quadword is specified by its address A, the address of the byte containing bit 0. A quadword is a 64-bit value. When interpreted arithmetically, a quadword is either a two's-complement integer with bits of increasing significance going 0 through 62 and bit 63 as the sign bit, or an unsigned integer with bits of increasing significance going 0 through 63.

**Note**

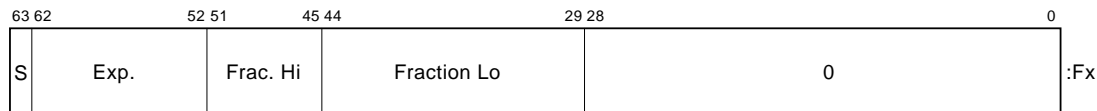
Alpha implementations impose a significant performance penalty when accessing quadword operands that are not naturally aligned. (A naturally aligned quadword has 0 as the low-order three bits of its address.)

**F\_floating**

An `F_floating` datum is four contiguous bytes in memory starting on an arbitrary byte boundary. The bits are labeled from right to left, 0 through 31 as shown in Figure 3–5.

**Figure 3–5 F\_floating Data Format**

An `F_floating` operand occupies 64 bits in a floating register, left-justified in the 64-bit register as shown in Figure 3–6.

**Figure 3–6 F\_floating Register**

The `F_floating` load instruction reorders bits on the way in from memory, expands the exponent from 8 to 11 bits, and sets the low-order fraction bits to 0. This produces in the register an equivalent `G_floating` number, suitable for either `F_floating` or `G_floating` operations. The mapping from 8-bit memory-format exponents to 11-bit register-format exponents is explained in Table 3–1.

**Table 3–1 Alpha F\_floating Load Exponent Mapping**

Memory	Register
1 1111111	1 000 1111111
1 xxxxxxx	1 000 xxxxxxx
0 xxxxxxx	0 111 xxxxxxx (xxxxxxx not all 1's)
0 0000000	0 000 0000000 (xxxxxxx not all 0's)

This mapping preserves both normal values and exceptional values.

The `F_floating` store instruction reorders register bits on the way to memory and does no checking of the low-order fraction bits. Register bits `<61:59>` and `<28:0>` are completely ignored by the store instruction.

## 3.2 Data Types

An *F\_floating* datum is specified by its address *A*, the address of the byte containing bit 0. The memory form of an *F\_floating* datum is sign magnitude with bit 15 the sign bit, bits <14:7> an access-128 binary exponent, and bits <6:0> and <31:16> a normalized 24-bit fraction with the redundant most significant fraction bit not represented. Within the fraction, bits of increasing significance go from 16 through 31 and 0 through 6. The 8-bit exponent field encodes the values 0 through 255. An exponent value of 0, together with a sign bit of 0, is taken to indicate that the *F\_floating* datum has a value of 0.

If the result of a VAX floating-point format instruction has a value of 0, the instruction always produces a datum with a sign bit of 0, an exponent of 0, and all fraction bits of 0. Exponent values of 1..255 indicate true binary exponents of -127..127. An exponent value of 0, together with a sign bit of 1, is taken as a reserve operand. Floating-point instructions processing a reserve operand take an arithmetic exception. The value of an *F\_floating* datum is in the approximate range of  $0.29 \times 10^{-38}$ .. $1.7 \times 10^{38}$ . The precision of an *F\_floating* datum is approximately one part in  $2^{23}$ , typically 7 decimal digits.

---

### Note

---

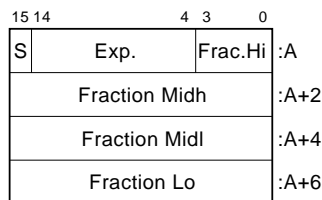
Alpha implementations impose a significant performance penalty when accessing *F\_floating* operands that are not naturally aligned. (A naturally aligned *F\_floating* datum has 0 as the low-order two bits of its address.)

---

### G\_floating

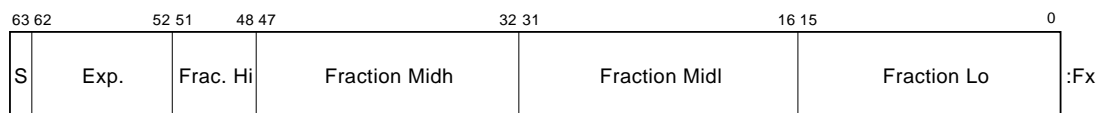
A *G\_floating* datum in memory is eight contiguous bytes starting on an arbitrary byte boundary. The bits are labeled from right to left, 0 through 63 as shown in Figure 3-7.

**Figure 3-7 G\_floating Operand**



A *G\_floating* operand occupies 64 bits in a floating register, arranged as shown in Figure 3-8.

**Figure 3-8 G\_floating Data Format**



A *G\_floating* datum is specified by its address *A*, the address of the byte containing bit 0. The form of a *G\_floating* datum is sign magnitude with bit 15 the sign bit, bits <14:4> an excess-1024 binary exponent, and bits <3:0> and <63:16> a normalized 53-bit fraction with the redundant most significant fraction

bit not represented. Within the fraction, bits of increasing significance go from 48 through 63, 32 through 47, 16 through 31, and 0 through 3. The 11-bit exponent field encodes the values of 0 through 2047. An exponent value of 0, together with a sign bit of 0, is taken to indicate that the G\_floating datum has a value of 0.

If the result of a floating-point instruction has a value of 0, the instruction always produces a datum with a sign bit of 0, an exponent of 0, and all fraction bits of 0. Exponent values of 1..2047 indicate true binary exponents of -1023..1023. An exponent value of 0, together with a sign bit of 1, is taken as a reserve operand. Floating-point instructions processing a reserve operand take a user-visible arithmetic exception. The value of a G\_floating datum is in the approximate range of  $0.56 \times 10^{-308}$ .. $0.9 \times 10^{308}$ . The precision of a G\_floating datum is approximately one part in  $2^{52}$ , typically 15 decimal digits.

---

**Note**

---

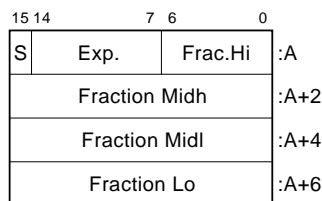
Alpha implementations impose a significant performance penalty when accessing G\_floating operands that are not naturally aligned. (A naturally aligned G\_floating datum has 0 as the low-order three bits of its address.)

---

### D\_floating

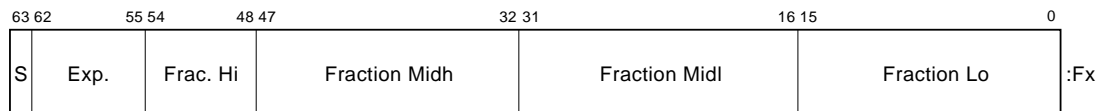
A D\_floating datum in memory is 8 contiguous bytes starting on an arbitrary byte boundary. The bits are labeled from right to left, 0 through 63 as shown in Figure 3–9.

**Figure 3–9 D\_floating Data Format**



A D\_floating operand occupies 64 bits in a floating register, arranged as shown in Figure 3–10.

**Figure 3–10 D\_floating Register Format**



The reordering of bits required for a D\_floating load or store is identical to those required for a G\_floating load or store. The G\_floating load and store instructions are therefore used for loading or storing D\_floating data.

A D\_floating datum is specified by its address A, the address of the byte containing bit 0. The memory form of a D\_floating datum is identical to an F\_floating datum except for an additional 32 low significance fraction bits. Within the fraction, bits of increasing significance go from 48 through 63, 32 through 47, 16 through 31, and 0 through 6. The exponent conventions and approximate



## 3.2 Data Types

range of values is the same for D\_floating as F\_floating. The precision of a D\_floating datum is approximately one part in  $2^{55}$ , typically 16 decimal digits.

---

**Note**

---

D\_floating is not a fully-supported data type; no D\_floating arithmetic operations are provided in the architecture. For backward compatibility, exact D\_floating arithmetic may be provided via software emulation. D\_floating "format compatibility," in which binary files of D\_floating numbers may be processed but without the last 3 bits of fraction precision, can be obtained via conversions to G\_floating, G arithmetic operations, then conversion back to D\_floating.

---



---

**Note**

---

Alpha implementations impose a significant performance penalty on access to D\_floating operands that are not naturally aligned. (A naturally aligned D\_floating datum has the low-order three bits of its address 0.)

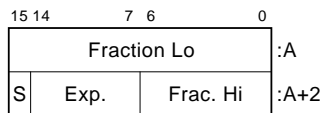
---

### S\_floating

An IEEE single precision, or S\_floating, datum occupies four contiguous bytes in memory starting on an arbitrary byte boundary.

The bits are labeled from right to left, 0 through 31 as shown in Figure 3–11.

**Figure 3–11 S\_floating operand**



An S\_floating operand occupies 64 bits in a floating register, left-justified in the 64-bit register, as shown in Figure 3–12.

**Figure 3–12 S\_floating Register Format**



The S\_floating load instruction reorders bits on the way in from memory, expanding the exponent from 8 to 11 bits, and sets the low-order fraction bits to 0. This produces in the register an equivalent T\_floating number, suitable for either S\_floating or T\_floating operations. The mapping from 8-bit memory-format exponents to 11-bit register-format exponents is shown in Table 3–2.

Table 3–2 S\_floating Load Exponent Mapping

Memory	Register
1 1111111	1 111 1111111
1 xxxxxxx	1 000 xxxxxxx
0 xxxxxxx	0 111 xxxxxxx (xxxxxxx not all 1's)
0 0000000	0 000 0000000 (xxxxxxx not all 0's)

This mapping preserves both normal values and exceptional values. Note that the mapping for all 1's differs from that of F\_floating load, because for S\_floating all 1's is an exceptional value and for F\_floating all 1's is a normal value.

The S\_floating store instruction reorders register bits on the way to memory and does no checking of the low-order fraction bits. Register bits <61:59> and <28:0> are completely ignored by the store instruction. The S\_floating load instruction does no checking of the input.

The S\_floating store instruction does no checking of the data; the preceding operation should have specified an S\_floating result.

An S\_floating datum is specified by its address A, the address of the byte containing bit 0. The memory form of an S\_floating datum is sign magnitude with bit 31 the sign bit, bits <30:23> an excess-127 binary exponent, and bits <22:0> a 23-bit fraction.

The value (V) of an S\_floating number is inferred from its constituent sign (S), exponent (E), and fraction (F) fields as follows:

1. If  $E=255$  and  $F \neq 0$ , then V is NaN, regardless of S
2. If  $E=255$  and  $F=0$ , then  $V = (-1)^S \times \text{Infinity}$
3. If  $0 < E < 255$ , then  $V = (-1)^S \times 2^{(E-127)} \times (1.F)$
4. If  $E=0$  and  $F \neq 0$ , then  $V = (-1)^S \times 2^{(-126)} \times (0.F)$
5. If  $E=0$  and  $F=0$ , then  $V = (-1)^S \times 0$  (0)

Floating-point operations on S\_floating numbers may take an arithmetic exception for a number of different reasons, including invalid operations, overflow, underflow, division by 0, and inexact results.

---

**Note**

---

Alpha implementations impose a significant performance penalty when accessing S\_floating operands that are not naturally aligned. (A naturally aligned S\_floating datum has 0 as the low-order two bits of its address)

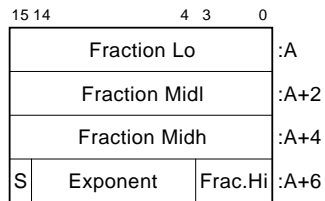
---

### T\_floating

An IEEE double-precision, or T\_floating, datum occupies 8 contiguous bytes in memory starting on an arbitrary byte boundary. The bits are labeled from right to left, 0 thru 63, as shown in Figure 3–13.

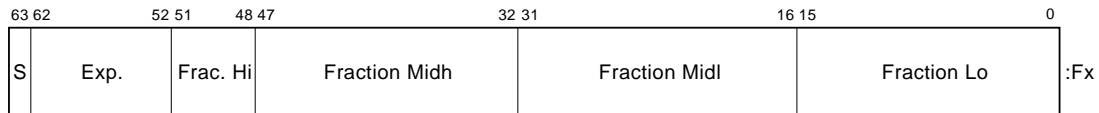
## 3.2 Data Types

**Figure 3–13 T\_floating Datum**



A T\_floating operand occupies 64 bits in a floating register, arranged as shown in Figure 3–14.

**Figure 3–14 T\_floating Register Format**



The T\_floating load instruction performs no bit reordering on input, nor does it perform checking of the input data.

The T\_floating store instruction performs no bit reordering on output. This instruction does no checking of the data; the preceding operation should have specified a T\_floating result.

A T\_floating datum is specified by its address A, the address of the byte containing bit 0. The form of a T\_floating datum is sign magnitude with bit 63, the sign bit, bits <62:52> an excess-1023 binary exponent, and bits <51:0> a 52-bit fraction.

The value (V) of a T\_floating number is inferred from its constituent sign (S), exponent (E), and fraction (F) fields as follows:

1. If  $E=2047$  and  $F \neq 0$ , then V is NaN, regardless of S.
2. If  $E=2047$  and  $F=0$ , then  $V = (-1)^S \times \text{Infinity}$ .
3. If  $0 < E < 2047$ , then  $V = (-1)^S \times 2^{(E-1023)} \times (1.F)$ .
4. If  $E=0$  and  $F \neq 0$ , then  $V = (-1)^S \times 2^{(-1022)} \times (0.F)$ .
5. If  $E=0$  and  $F=0$ , then  $V = (-1)^S \times 0$  (zero).

Floating-point operations on T\_floating numbers may take an arithmetic exception for a variety of reasons, including invalid operations, overflow, underflow, division by zero, and inexact results.

---

**Note**

---

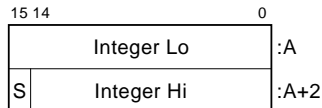
Alpha implementations will impose a significant performance penalty when accessing T\_floating operands that are not naturally aligned. (A

naturally aligned T\_floating datum has zero as the low-order three bits of its address.)

### Longword Integer Format in Floating-Point Unit

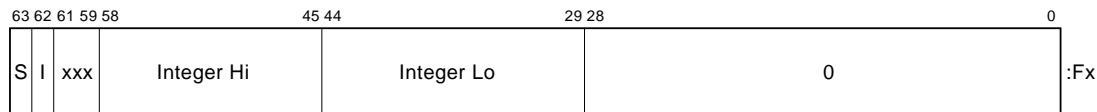
A longword integer operand occupies 32 bits in memory, arranged as shown in Figure 3–15.

**Figure 3–15 Longword Integer Datum**



A longword integer operand occupies 64 bits in a floating register, arranged as shown in Figure 3–16.

**Figure 3–16 Longword Integer Floating-Register Format**



There is no explicit longword load or store instruction; the S\_floating load/store instructions are used to move longword data into or out of the floating registers. The register bits <61:59> are set by the S\_floating load exponent mapping. They are ignored by S\_floating store. They are also ignored in operands of a longword integer operate instruction, and they are set to 000 in the result of a longword operate instruction.

The register format bit <62>, "I", in Figure 3–16 is part of the Integer Hi field in Figure 3–15 and represents the high-order bit of that field. Bits <58:45> of Figure 3–16 are the remaining bits of the Integer Hi field of Figure 3–15.

#### Note

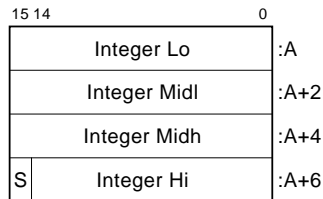
Alpha implementations will impose a significant performance penalty when accessing longwords that are not naturally aligned. (A naturally aligned longword datum has zero as the low-order two bits of its address).

### Quadword Integer Format in Floating-Point Unit

A quadword integer operand occupies 64 bits in memory, arranged as shown in Figure 3–17.

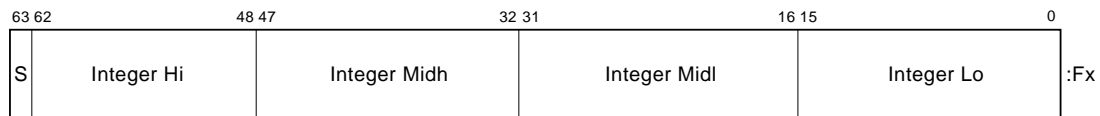
## 3.2 Data Types

**Figure 3–17 Quadword Integer Datum**



A quadword integer operand occupies 64 bits in a floating register, arranged as shown in Figure 3–18.

**Figure 3–18 Quadword Integer Floating-Register Format**



There is no explicit quadword load or store instruction; the T\_floating load/store instructions are used to move quadword data into or out of the floating registers.

The T\_floating load instruction performs no bit reordering on input. The T\_floating store instruction performs no bit reordering on output. This instruction does no checking of the data; when used to store quadwords, the preceding operation should have specified a quadword result.

---

**Note**

---

Alpha implementations will impose a significant performance penalty when accessing quadwords that are not naturally aligned. (A naturally aligned quadword datum has zero as the low-order three bits of its address.)

---

### Data Types with No Hardware Support

- Octaword
- H\_floating
- D\_floating (except load/store and convert to/from G\_floating)
- Variable Length Bit Field
- Character String
- Trailing Numeric String
- Leading Separate Numeric String
- Packed Decimal String

### 3.3 Alpha Registers

The following paragraphs describe the registers defined by the Digital Alpha AXP architecture.

#### 3.3.1 Program Counter Register

The Program Counter (PC) is a special register that addresses the instruction stream. As each instruction is decoded, the PC is advanced to the next sequential instruction. This is referred to as the "updated PC." Any instruction that uses the value of the PC uses the updated PC. The PC includes only bits <63:2> with bits <1:0> treated as RAZ/IGN. This quantity is a longword-aligned byte address. The PC is an implied operand on conditional branch and subroutine jump instructions. The PC is not accessible as an integer register.

#### 3.3.2 Processor Status Register

The Processor Status (PS) is a special register that contains the current status of the processor. It can be read by all CALL\_PAL RD\_PS routine. The PS<SW> field can be written by a CALL\_PAL WR\_PS SW routine.

#### 3.3.3 Integer Registers

There are 32 integer registers (R0 through R31), each 64 bits wide.

Certain of the registers are assigned special meaning by the Alpha architecture:

- **R30** is the stacked pointer (SP). SP contains the address of the top of the stack in the current mode.

Certain PALcode (for example, REI) uses R30 as an implicit operand. During such operations, the address value in R30, interpreted as an unsigned 64 bit integer, decreases (predecrements) when items are pushed onto the stack, and increases (postincrements) when they are popped from the stack. After pushing (writing) an item to the stack, SP points to that item.

- **R31** When R31 is specified as a register source operand, a 0-valued operand is supplied. With one exemption, results of an instruction that specifies R31 as a destination operand are discarded and it is UNPREDICTABLE whether the other destination operands (implicit and explicit) are changed by the instructions. In this case, it is implementation-dependent to what extent the instruction is actually executed once it has been fetched. It is also UNPREDICTABLE whether exceptions are signaled during the execution of such an instruction. Note, however, that exceptions associated with the instruction fetch of such an instruction are always signaled. The exemption to the above rule is for the following branch instructions when R31 is specified as the Ra operand: the unconditional branch (BR and BSR) and jump to subroutine (JMP, JSR, RET, and JSR\_COROUTINE) instructions. These instructions execute normally and update the PC with the target virtual address when R31 is specified as the Ra operand – of course no PC value can be saved in R31. Applying the rule above, there are some interesting cases involving R31 as a destination.

1. STx\_C R31, disp(Rb) although this might seem like a good way to 0 out a shared location and reset the lock\_flag, this instruction causes the lock\_flag and virtual location {Rbv + SEXT(disp)} to become UNPREDICTABLE.

### 3.3 Alpha Registers

2. LDx\_L R31,disp(Rb) this instruction produces no useful result because it causes both lock\_flag and locked\_physical\_address to become UNPREDICTABLE.

#### 3.3.4 Floating-point Registers

There are 32 floating-point registers (FO through F31), each 64 bits wide.

When F31 is specified as a register source operand, a true 0-valued operand is supplied.

Results of an instruction that specifies F31 as a destination operand are discarded and it is UNPREDICTABLE whether the other destination operands (implicit and explicit) are changed by the instruction. In this case, it is implementation-dependent to what extent the instruction is actually executed once it has been fetched. It is also UNPREDICTABLE whether exceptions are signaled during the execution of such an instruction. Note, however, that exceptions associated with the instruction fetch of such an instruction are always signaled.

A floating-point instruction that operates on single-precision data reads all bits <63:0> of the source floating-point register. A floating-point instruction that produces a single-precision result writes all bits <63:0> of the destination floating-point register.

#### 3.3.5 Lock Registers

There are two per-processor registers associated with the LDx\_L and STx\_C instructions, the lock\_flat and the locked\_physical\_address register.

#### 3.3.6 Alpha Internal Processor Registers

There are a number of internal processor registers with specialized uses that are available only to privileged software via the MTPR and MFPR PALcode routines.

#### 3.3.7 Optional Registers

Some Alpha implementations may include optional memory prefetched or VAX compatibility processor registers.

##### Memory Prefetched Registers

If the prefetched instructions FETCH and FETCH\_M are implemented, then an implementation will include two sets of state prefetch registers used by those instructions. These registers are not directly accessible by software and are just listed here for completeness.

##### VAX Compatibility Registers

If the VAX compatibility instructions RC and RS are implemented, then an implementation will include the intr-flag register.

### 3.4 Instruction Formats

There are five basic Digital Alpha instruction formats as listed below:

- Memory
- Branch
- Operate
- Floating-point Operate
- PALcode

## 3.4 Instruction Formats

All instruction formats are 32 bits long with a 6-bit major opcode field in bits <31:26> of the instruction.



---

## The DECchip™ 21064 CPU chip

The 21064 central processing unit is a superscalar, superpipelined implementation of the Digital Alpha AXP architecture. This 431-pin CPU is fabricated in complementary metal-oxide semiconductor technology and packaged in a 24x24, 100-mil pin pitch pin grid array (PGA). The 21064 features include the following:

- Supports Alpha instruction and data types for byte, word, longword, quadword, DEC floating point data types ( F\_floating, D\_floating, and G\_floating) and IEEE floating point data types ( S\_floating, and T\_floating). The architecturally optional instructions RCC is also implemented.
- Contains a demand-page memory management unit that in conjunction with properly written privileged architecture library code (PALcode) stored in firmware FEPROMs, fully implements the Alpha memory management architecture. The translation buffer can be used with alternative PALcode to implement a different page table structure.
- Contains on-chip, 8-entry, instruction stream translation buffer for mapping 8 KB physical pages and a 4-entry instruction stream translation buffer for mapping groups of up to 512 contiguous 8KB pages. It also contains a 32-entry data stream translation buffer for mapping 8-KB physical pages, and a 4-entry data stream translation buffer for mapping aligned groups of 512 contiguous 8-KB pages.
- Implements a dynamic branch prediction algorithm using a 2 KB x 1-bit branch history table. An internal control register bit selects one of these two algorithms.
- Contains an integer execution unit that supports scaled add instructions that improve the performance of address calculations for longword- and quadword-length array elements.
- Uses a 6.6 ns cycle time as the DECchip's nominal frequency.
- Provides low average cycles per instruction (CPI). The 21064 can issue two Alpha instructions in a single cycle, minimizing the average CPI. Branch history tables minimize the branch latency, further reducing the average CPI.
- Contains a fully pipelined floating point execution unit capable of executing both DEC and IEEE floating-point instructions. The floating point unit can accept a new instruction every cycle, except for divide instructions. The operate-to-operate latency for all instructions other than divide is six CPU cycles. The latencies for single and double precision divide instructions are 17 and 59 cycles, respectively.
- Contains an on-chip 8 KB direct mapped, write-through physical data cache with a block size of 32 bytes.

- Contains an on-chip 8 KB direct-mapped, read-only physical instruction cache with a block size of 32 bytes (managed as a virtual cache).
- A single-entry stream buffer to prefetch 32-byte instruction cache blocks.
- An on-chip 4 entry (32 bytes/entry) write buffer with byte merging capability.

The 21064 CPU chip consists of three independent execution units: integer execution unit (E-box), floating point unit (F-box), and the address generation, memory management, write buffer and bus interface unit (A-box). Each unit can accept at most one instruction per cycle, however if code is properly scheduled, this CPU chip can issue two instructions to two independent units in a single cycle. A fourth box, the (I-box), is the central control unit. It issues instructions, maintains the pipeline, and performs all of the PC calculations.

## 4.1 I-box Internal Processor Registers

The primary function of the I-box is to issue instructions to the E-box, A-box, and F-box. The I-box decodes two instructions in parallel and checks that the required resources are available for both instructions. The following sections describe the registers that facilitate this process.

### 4.1.1 Translation Buffer Tag Register (TB\_TAG)

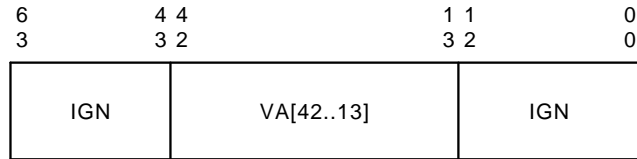
The TB\_TAG register is a write-only register that holds the tag for the next TB update operation in either the ITB or DTB. To insure the integrity of the TB, the tag is written to a temporary register and not transferred to the ITB or DTB until the ITB\_PTE or DTB\_PTE register is written. The entry to be written is chosen at the time of the ITB\_PTE or DTB\_PTE write operation by a not-last-used algorithm implemented in hardware.

The ITB\_TAG register is written only while in PAL mode regardless of the state of the HWE bit in the ICCSR IPR. The ITB\_TAG format is shown in Figure 4-1

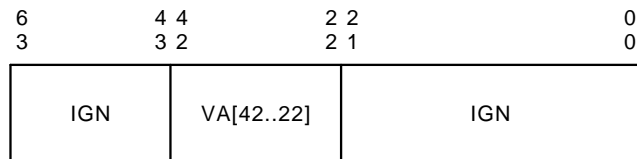
## 4.1 I-box Internal Processor Registers

Figure 4–1 TB\_TAG Register Format

Small Page Format:



GH = 11(bin) Format (DTB only):



### 4.1.2 Instruction Translation Buffer Page Table Entry Register (ITB\_PTE)

The ITB PTE register is a read/write register representing the twelve ITB page table entries. The first eight provide small-page (8K byte) translations while the remaining four provide large-page (4M byte) translations. The entry to be written is chosen by a not-last-used algorithm implemented in hardware. Writes to the ITB\_PTE use the memory-format bit positions as described in the *Alpha Architecture Reference Manual*, except some fields are ignored.

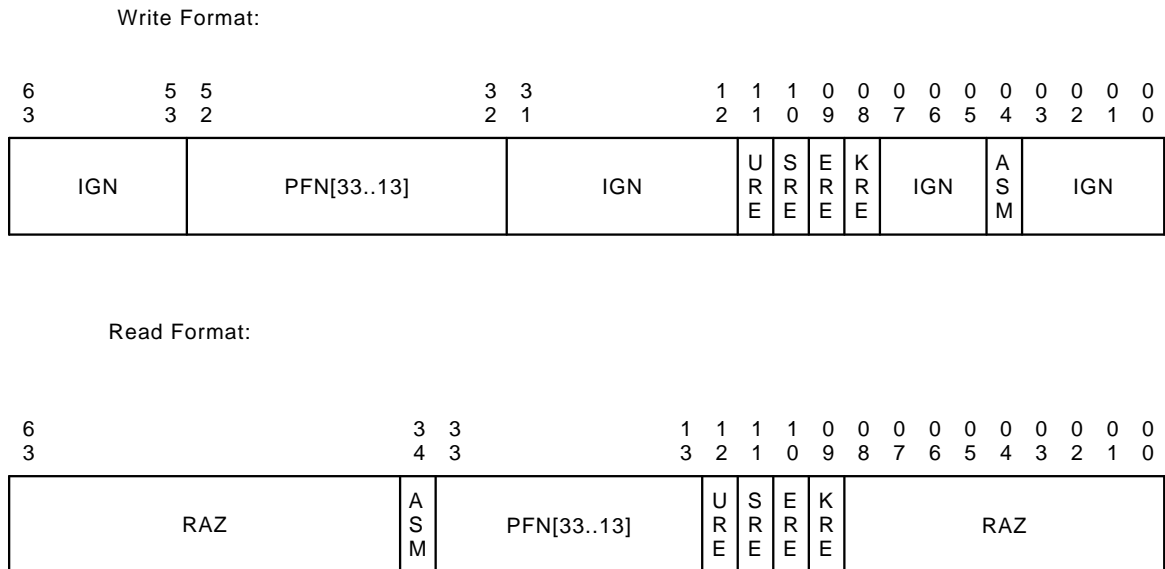
To ensure the integrity of the ITB, the ITB's tag array is updated from the internal tag register when the ITB\_PTE register is written. Reads of the ITB\_PTE require two instructions. First, a read from the ITB\_PTE sends the PTE data to the ITB\_PTE\_TEMP register.

Then, a second instruction reading from the ITB\_PTE\_TEMP register returns the PTE entry to the register file. Reading or writing the ITB\_PTE register increments the TB entry pointer corresponding to the large/small page selection indicated by the TB\_CTL, which allows reading the entire set of ITB\_PTE register entries.

The ITB\_PTE register is read and written in PAL mode only, regardless of the state of the HWE bit in the ICCSR IPR. The format is shown in Figure 4–2.

## 4.1 I-box Internal Processor Registers

Figure 4–2 ITB\_PTE Register Format



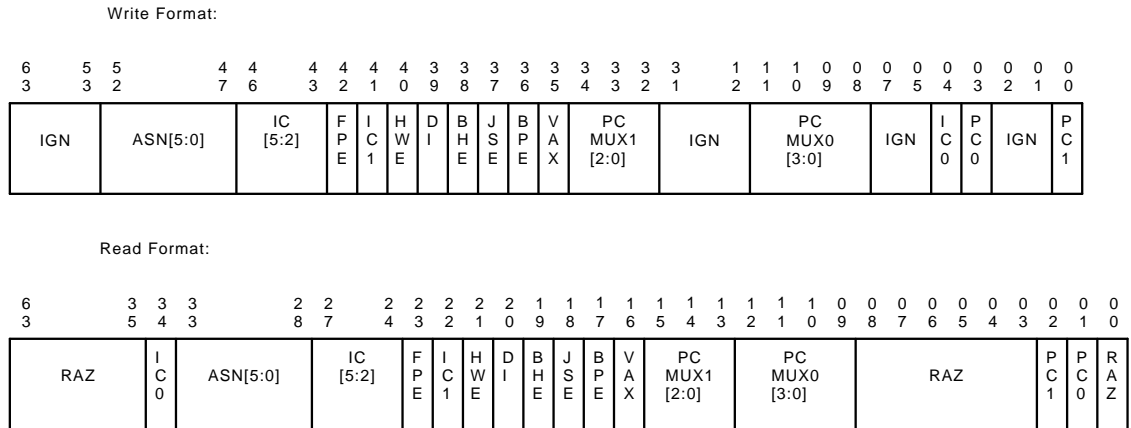
### 4.1.3 Instruction Cache Control and Status Register (ICCSR)

The ICCSR register contains various I-box hardware enables. The only architecturally defined bit in this register is the floating point enable (FPE), which enables floating-point instruction execution. When clear, all floating-point instructions generate FEN exceptions. Most of this register is cleared by hardware at reset. Fields not cleared include ASN, PC0, and PC1.

The HWE bit allows the special PAL instructions to execute in kernel mode. This bit is intended for diagnostics or operating system alternative PAL routines only. The HWE bit does not allow access to the ITB registers outside of PAL mode. Therefore, some PALcode flows may require the PAL mode environment to execute properly (for example, ITB fill). The format is shown in Figure 4–3 and the register fields are described in Table 4–1.

## 4.1 I-box Internal Processor Registers

Figure 4–3 ICCSR Register Format



ICCSR

## 4.1 I-box Internal Processor Registers

Table 4–1 ICCSR Bits

Field	Type	Description
FPE	RW,0	If set, floating point instructions can be issued. If clear, floating point instructions cause FEN exceptions.
MAP	RW,0	If set, allows super page instruction stream memory mapping of VPC<33:13> directly to physical PC<33:13> essentially bypassing ITB for VPC addresses containing VPC<42:41>=2. Super-page mapping is allowed in kernel mode only. The ASM bit is always set. If clear, super-page mapping is disabled.
HWE	RW,0	If set, allows the five PALRES instructions to be issued in kernel mode. If cleared, attempted execution of PALRES instructions while not in PALmode results in OPCDEC exceptions. Use of HW_MTPR instruction to update the EXC_ADDR IPR while in native mode is restricted to values with bit<0> equal to 0. The combination of native mode and EXC_ADDR<0> equal to 1 causes UNDEFINED behavior.
MAP	RW,0	If set, allows super-page instruction stream mapping of VPC<33:13> directly to physical PC<33:13> essentially bypassing ITB for VPC addresses containing VPC <42:41>. Super-page mapping is allowed in kernel mode only. The ASM bit is always set. If clear, super-page mapping is disabled.
DI	RW,0	If set, enables dual issue. If cleared, instructions can only single issue.
BHE	RW,0	Used with BPE. See Table 4–2 for programming information.
JSE	RW,0	If set, enables the JSR stack to push return addresses. If cleared, JSR stack is disabled.
BPE	RW,0	Used with BHE. See Table 4–2 for programming information.
PIPE	RW,0	If clear, causes all hardware-interlocked instructions to drain the machine and waits for the write buffer to empty before issuing the next instruction. Examples of instructions that do not cause the pipe to drain include HW_MTPR, HW_REI, conditional branches, and instructions that have a destination register of R31. If set, pipeline proceeds normally.
PCMUX1	RW,0	
PCMUX0	RW,0	
PC1	RW,0	If clear, enables a performance counter 1 interrupt request after $2^{12}$ events are counted. If set, enables a performance counter 1 interrupt request after $2^8$ events are counted.
PC0	RW,0	If clear enables a performance counter 0 interrupt request after $2^{16}$ events are counted. If set enables a performance counter 0 interrupt request after $2^{12}$ events are counted.
ASN	RW,0	Address Space Number field is used in conjunction with the instruction cache (I-cache) in the 21064 to further qualify cache entries and avoid some cache flushes. The ASN is written to the instruction cache during fill operations and compared with the instruction stream (Instruction stream) data on fetch operations. Mismatches invalidate the fetch without affecting the I-cache.
RES	RW,0	The RES state bits are reserved by Digital and should not be used by any software.
ICCSR<45:44>	RW,0	When set, the performance counters will be enabled and will increment. During normal operation, these bits should be written with 0 in order to disable the performance counters.

## 4.1 I-box Internal Processor Registers

**Table 4–2 BHE, BPE Branch Prediction Selection**

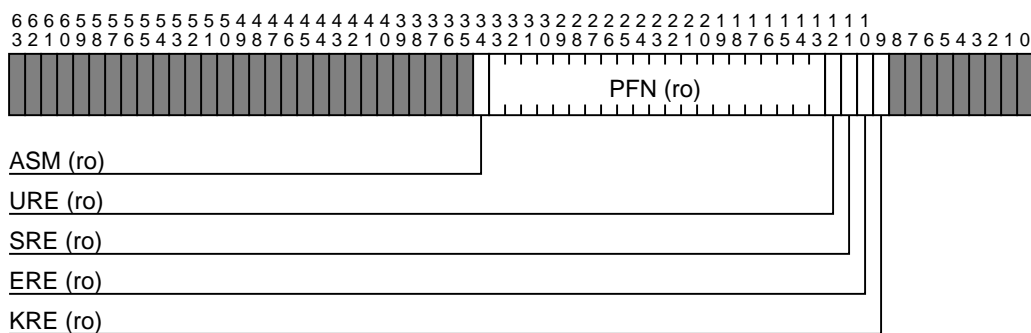
BPE	BHE	Prediction
0	X	Not Taken
1	0	Sign of Displacement
1	1	Branch History Table

### 4.1.4 Instruction Translation Buffer Page Table Entry Temporary Register (ITB\_PTE\_TEMP)

The ITB\_PTE\_TEMP register is a read-only holding register for instruction translation buffer page table entry read data. Reads of the ITB\_PTE require two instructions to return the data to the register file. The first reads the ITB\_PTE register to the ITB\_PTE\_TEMP register. The second returns the ITB\_PTE\_TEMP register to the integer register file. The ITB\_PTE\_TEMP register is updated on all ITB accesses, both read and write. A read of the ITB\_PTE to the ITB\_PTE\_TEMP should be followed closely by a read of the ITB\_PTE\_TEMP to the register file.

The ITB\_PTE\_TEMP register is read only while in PALmode, regardless of the state of the HWE bit in the ICCSR IPR.

**Figure 4–24 ITB\_PTE\_TEMP Register (ITB\_PTE\_TEMP)**



**Table 4–3 ITB\_PTE\_TEMP Register Description**

Field	Description
<b>34</b>	<b>ASM</b> <i>[read-only]</i> ASM
<b>33:13</b>	<b>PFN</b> <i>[read-only]</i> PFN
<b>12</b>	<b>URE</b> <i>[read-only]</i> URE
<b>11</b>	<b>SRE</b> <i>[read-only]</i> SRE

(continued on next page)

## 4.1 I-box Internal Processor Registers

Table 4–3 (Cont.) ITB\_PTE\_TEMP Register Description

Field	Description
10	<b>ERE</b> <i>[read-only]</i> ERE
9	<b>KRE</b> <i>[read-only]</i> KRE

### 4.1.5 Exception Address Register (EXC\_ADDR)

The EXC\_ADDR register is a read/write register used to restart the machine after exceptions or interrupts. The EXC\_ADDR register can be read and written by software via the HW\_MTPR instruction as well as directly by hardware. The HW\_REI instruction executes a jump to the address contained in EXC\_ADDR.

The EXC\_ADDR register is written by hardware after an exception to provide a return address for PALcode. The instruction pointed to by the EXC\_ADDR register did not complete execution. Because the PC is longword aligned, the least significant bit (LSB) of EXC\_ADDR is used to indicate PAL mode to the hardware. When the LSB is clear, the HW\_REI instruction executes a jump to native (non-PAL) mode, enabling address translation.

CALL\_PAL exceptions load the EXC\_ADDR with the PC of the instruction following the CALL\_PAL. This function allows CALL\_PAL service routines to return without needing to increment the value in EXC\_ADDR.

This feature, however, requires careful treatment in PALcode. Arithmetic traps and machine check exceptions can pre-empt CALL\_PAL exceptions resulting in an incorrect value being saved in the EXC\_ADDR register. In the cases of an arithmetic trap or a machine check exception, and only in these cases, EXC\_ADDR<1> takes on special meaning. PAL code servicing these two exceptions should interpret a 0 in EXC\_ADDR<1> as indicating that the PC in EXC\_ADDR<63:2> is too large by a value of 4 bytes and subtract 4 before executing a HW\_REI from this address. PALcode should interpret a 1 in EXC\_ADDR<1> as indicating that the PC in EXC\_ADDR<63:2> is correct, and clear the value of EXC\_ADDR<1>. All other PALcode entry points, except reset, can expect EXC\_ADDR<1> to be 0.

This logic allows the following code sequence to conditionally subtract 4 from the address in the EXC\_ADDR register without use of an additional register. This code sequence should be present in arithmetic trap and machine check flows only.

```
HW_MFPR    Rx, EXC_ADDR    ; read EXC_ADDR into GPR
SUBQ      Rx, #2, Rx      ; subtract 2 causing borrow if [1]=0
BIC       Rx, #2, Rx      ; clear [1]
HW_MTPR    Rx, EXC_ADDR    ; write back to EXC_ADDR
```

Note that bit<1> is undefined when the EXC\_ADDR is read. The actual hardware ignores this bit, however PALcode must explicitly clear this bit before it pushes the exception address on the stack.

IPR Format:



## 4.1 I-box Internal Processor Registers

Figure 4–5 Exception Address Register (EXC\_ADDR)

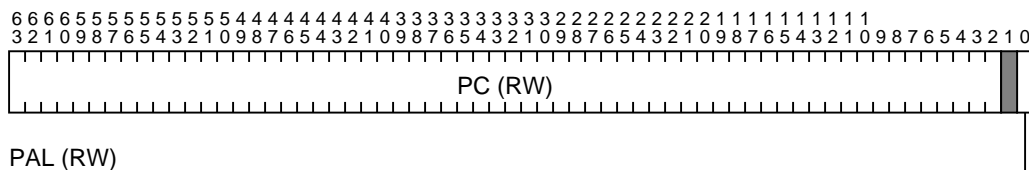


Table 4–4 Exception Address Register Description

Field	Description
<b>63:2</b>	<b>PC</b> <i>[read/write]</i> PC
<b>0</b>	<b>PAL</b> <i>[read/write]</i> Pal

### 4.1.6 Serial Line Clear Register (SL\_CLR)

This write-only register clears the serial line interrupt request, the performance counter interrupt request and the correctable read (CRD) interrupt request. Therefore, the write of any data to the SL\_CLR register clears the remaining serial line interrupt request. The 21064 requires that the indicated bit be written with a 0 to clear the selected interrupt source.

Figure 4–6 Serial Line Clear Register (SL\_CLR)

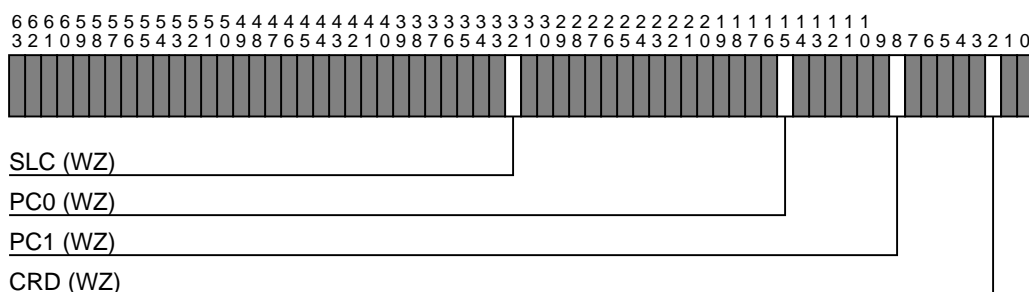


Table 4–5 Serial Line Clear Register Description

Field	Description
<b>32</b>	<b>SLC</b> <i>[write]</i> Clears the serial line interrupt request
<b>15</b>	<b>PC0</b> <i>[write]</i> Clears the performance counter 0 interrupt request
<b>8</b>	<b>PC1</b> <i>[write]</i>

(continued on next page)

## 4.1 I-box Internal Processor Registers

Table 4–5 (Cont.) Serial Line Clear Register Description

Field	Description
	Clears the performance counter 1 interrupt request
2	<b>CRD</b> <i>[write]</i> Clears the correctable read error interrupt request

### 4.1.7 Serial Line Receive Register (SL\_RCV)

The serial line receive register contains a single read-only bit used with the interrupt control registers and the SROMD\_H and SROMCLK\_H pins to provide an on-chip serial line function.

Figure 4–7 Serial Line Receive Register (sl\_rcv)

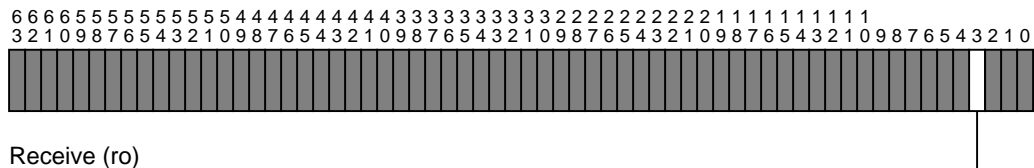


Table 4–6 Serial Line Receive Register Description

Field	Description
3	<b>Receive</b> <i>[read-only]</i> The RCV bit is functionally connected to the SROMD_H pin after the instruction cache is loaded from the external serial ROM. The RCV bit can be read to receive external data one bit at a time under a software timing loop. A serial line interrupt is requested on detection of any receive line transition that sets the SL_REQ bit in the HIRR. Using a software timing loop, the RCV bit can be read to receive data one bit at a time. The serial line interrupt can be disabled by clearing the HIER register's SL_ENA bit.

### 4.1.8 Instruction Translation Buffer Zap Register (ITBZAP)

Writing any value to this register invalidates all twelve ITB entries. It also resets the not-last-used (NLU) pointer to its initial state. The ITBZAP register should be written *only* in PAL mode.

### 4.1.9 Instruction Translation Buffer ASM (ITBASM)

Writing any value to this register invalidates all ITB entries in which the ASM bit is equal to 0. The ITBASM register should be written *only* in PAL mode.

### 4.1.10 Instruction Translation Buffer IS Register (ITBIS)

Writing any value to this register invalidates all twelve ITB entries. It also resets both not-last-used (NLU) pointers to their initial state. The ITBIS register should be written *only* in PAL mode.

## 4.1 I-box Internal Processor Registers

### 4.1.11 Processor Status (PS)

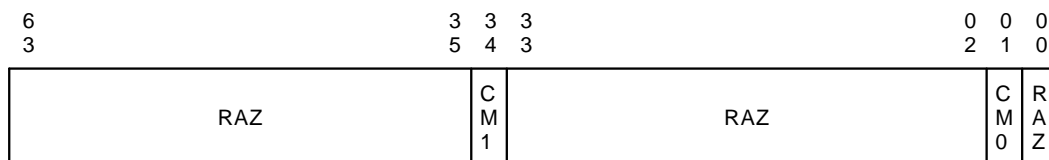
The processor status register is a read/write register containing only the current mode bits of the architecturally defined PS.

**Figure 4–8 Processor State (PS) Register Format**

Write Format:



Read Format:



### 4.1.12 Exception Summary Register (EXC\_SUM)

The exception summary register records the various types of arithmetic traps that have occurred since the last time the EXC\_SUM was written (cleared). When the result of an arithmetic operation produces an arithmetic trap, the corresponding EXC\_SUM bit is set.

In addition, the register containing the result of that operation is recorded in the exception register write mask IPR, as a single bit in a 64-bit field specifying registers F31-F0 and I31-I0. This IPR is visible only through the EXC\_SUM. The EXC\_SUM register provides a 1-bit window to the exception register write mask. Each read to EXC\_SUM shifts one bit in order F31-F0, then I31-I0. The read also clears the corresponding bit. Therefore, the EXC\_SUM must be read 64 times to extract the complete mask and clear the entire register.

Any write to EXC\_SUM clears bits [8..2] and does not affect the write mask.

The write mask register bit clears three cycles after a read. Therefore, code intended to read the register must allow at least three cycles between reads to allow the clear and shift operation to complete in order to ensure reading successive bits.

## 4.1 I-box Internal Processor Registers

Figure 4–9 Exception Summary Register (EXC\_SUM)

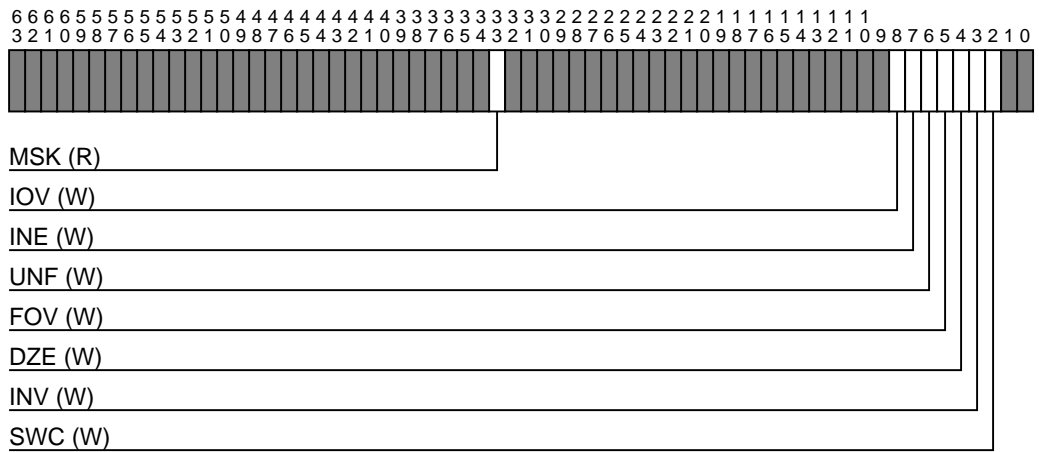


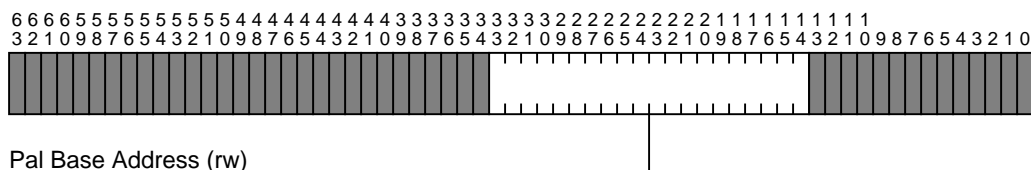
Table 4–7 Exception Summary Register Description

Field	Description
<b>33</b>	<b>MSK</b> Exception register write mask IPR window.
<b>8</b>	<b>IOV</b> Indicates an Fbox convert to integer overflow or integer arithmetic overflow.
<b>7</b>	<b>INE</b> Indicates a floating inexact error.
<b>6</b>	<b>UNF</b> Indicates a floating point underflow.
<b>5</b>	<b>FOV</b> Indicates a floating point overflow.
<b>4</b>	<b>DZE</b> Indicates a divide by 0.
<b>3</b>	<b>INV</b> Indicates an Invalid operation.
<b>2</b>	<b>SWC</b> Indicates software completion possible. The bit is set after a floating-point instruction containing the /S modifier completes with an arithmetic trap, if all previous floating point instructions that trapped since the last MTPR EXC_SUM also contained the /S modifier. The SWC bit is cleared whenever a floating point instruction without the /S modifier completes with an arithmetic trap. The bit remains cleared regardless of additional arithmetic traps, until the register is written using an MTPR instruction. The bit is always cleared upon any MTPR write to the EXC_SUM register.

### 4.1.13 Privileged Architecture Library Base Register (PAL\_BASE)

The PAL base register is a read/write register containing the base address for PALcode. This register is cleared by hardware at reset.

**Figure 4–10 PAL Base Address Register (PAL\_BASE)**



**Table 4–8 PAL Base Address Register Description**

Field	Description
<b>33:14</b>	<b>Pal Base Address</b> <i>[read/write]</i> Privileged Architecture Library code base address

### 4.1.14 Hardware Interrupt Request Register (HIRR)

The hardware interrupt request register is a read-only register providing a record of all currently outstanding interrupt requests and summary bits at the time of the read. For each bit of the HIRR [5:0] there is a corresponding bit of the HIER (hardware interrupt enable register) that must be set to request an interrupt. In addition to returning the status of the hardware interrupt requests, a read of the HIRR returns the state of the software interrupt and AST requests. Note that a read of the HIRR may return a value of 0 if the hardware interrupt was released before the read (passive release). The register guarantees that the HWR bit reflects the status as shown by the HIRR bits. All interrupt requests are blocked while executing in PAL mode.

## 4.1 I-box Internal Processor Registers

Figure 4–11 Hardware Interrupt Request Register (HIRR)

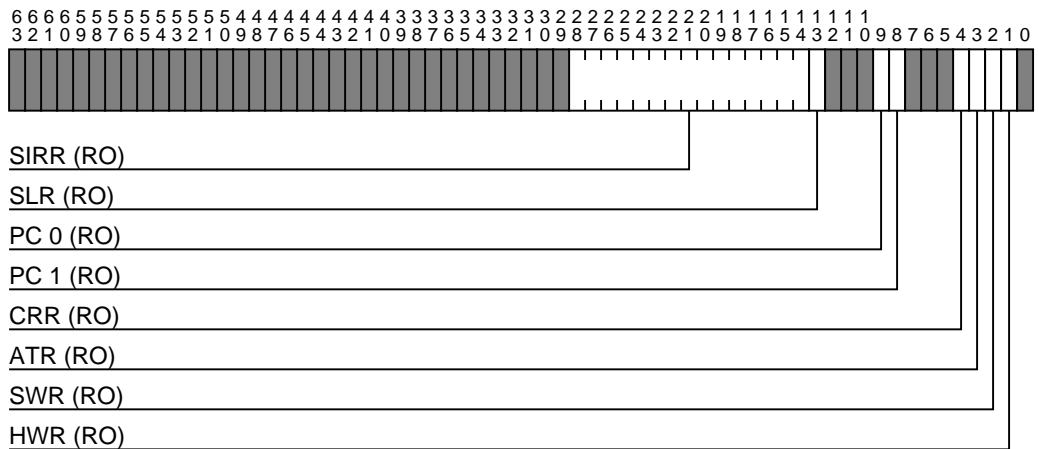


Table 4–9 Hardware Interrupt Request Register Description

Field	Description
<b>32:29</b>	<b>ASTRR &lt;3:0&gt;</b> <i>[read-only]</i> Corresponds to AST Request 3 to 0 (USEK).
<b>28:14</b>	<b>SIRR</b> <i>[read-only]</i> Corresponds to Software Interrupt Request 15 to 1.
<b>13</b>	<b>SLR</b> <i>[read-only]</i> Serial-line-interrupt request. <i>See also</i> SL_RCV, SL_XMIT, and SL_CLR.
<b>12:10</b>	<b>HIRR&lt;2:0&gt;</b> <i>[read-only]</i>  2 - Fbus+ interrupt 1 - Local I/O interrupt 0 - Hardware error interrupt
<b>9</b>	<b>PC 0</b> <i>[read-only]</i> Performance counter 0 interrupt request.
<b>8</b>	<b>PC 1</b> <i>[read-only]</i> Performance counter 1 interrupt request.
<b>7:5</b>	<b>HIRR&lt;5:3&gt;</b> <i>[read-only]</i>  5 - System event interrupt 4 - Interval timer interrupt 3 - Interprocessor interrupt
<b>4</b>	<b>CRR</b> <i>[read-only]</i> CRD correctable read error interrupt request. This interrupt is cleared by the SL_CLR register.

(continued on next page)

## 4.1 I-box Internal Processor Registers

**Table 4–9 (Cont.) Hardware Interrupt Request Register Description**

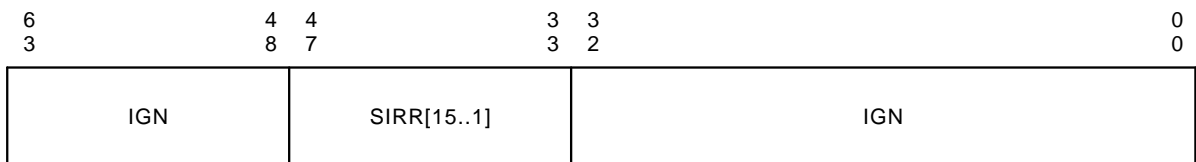
Field	Description
<b>3</b>	<b>ATR</b> <i>[read-only]</i> Is set if any AST request and corresponding enable is set. This bit also requires that the processor mode be equal to or higher than the request mode. In the 21064 chip, SIER<2> must also be set, to allow AST interrupt requests.
<b>2</b>	<b>SWR</b> <i>[read-only]</i> Is set if any Software Interrupt Request and corresponding enable is set
<b>1</b>	<b>HWR</b> <i>[read-only]</i> Is set if any Hardware Interrupt Request and corresponding enable is set

### 4.1.15 Software Interrupt Request Register (SIRR)

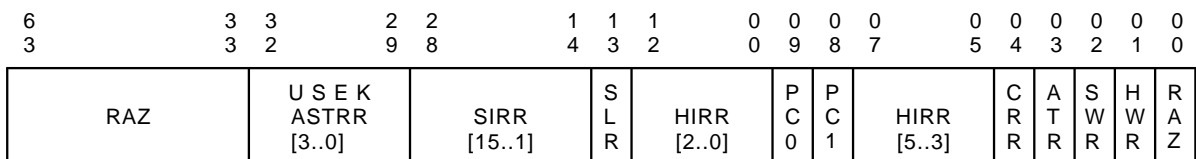
The software interrupt request register is a read/write register used to control software interrupt requests. For each bit of the SIRR there is a corresponding bit of the software interrupt enable register (SIER) that must be set to request an interrupt. Reads of the SIRR return the complete set of interrupt request registers and summary bits. (See Section 4.1.14 for details.) All interrupt requests are blocked while executing in PAL mode. The SIRR format is shown in Figure 4–12.

**Figure 4–12 SIRR Register Format**

Write Format:



Read Format:



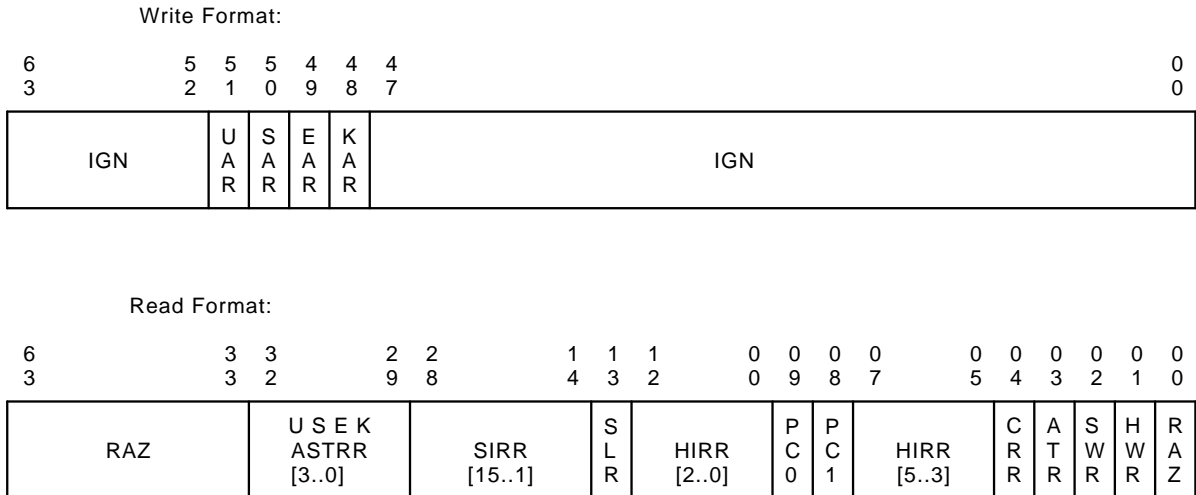
### 4.1.16 Asynchronous Trap Request Register (ASTRR)

The asynchronous trap request register is a read/write register. It contains bits to request AST interrupts in each of the processor modes. In order to generate an AST interrupt, the corresponding enable bit in the ASTER must be set and the processor must be in the selected processor mode or higher privilege as described by the current value of the PS CM bits. In addition, AST interrupts are enabled in the 21064 processor only if SIER<2> is set.

## 4.1 I-box Internal Processor Registers

This process provides a mechanism to lock out AST requests over certain IPL levels. All interrupt requests are blocked while executing in PAL mode. Reads of the ASTRR return the complete set of interrupt request registers and summary bits. (See Section 4.1.14 for details.) Figure 4–13 shows the ASTRR register format.

**Figure 4–13 ASTRR Register Format**

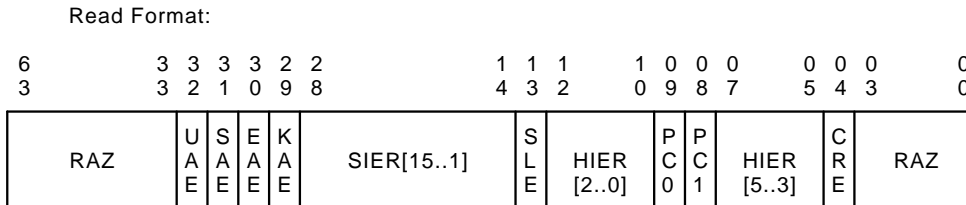
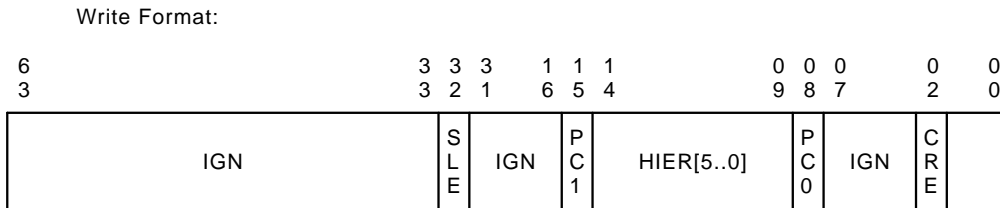


### 4.1.17 Hardware Interrupt Enable Register (HIER)

The hardware interrupt enable register is a read/write register. It is used to enable corresponding bits of the HIRR requesting interrupt. The PC0, PC1, SLE, and CRE bits of this register enable the performance counters, serial line and correctable read interrupts. There is a one-to-one correspondence between the interrupt requests and enable bits. As with the reads of the interrupt request IPRs, reads of the HIER return the complete set of interrupt enable registers. (See Section 4.1.14 for details.) Figure 4–14 shows the HIER register format.



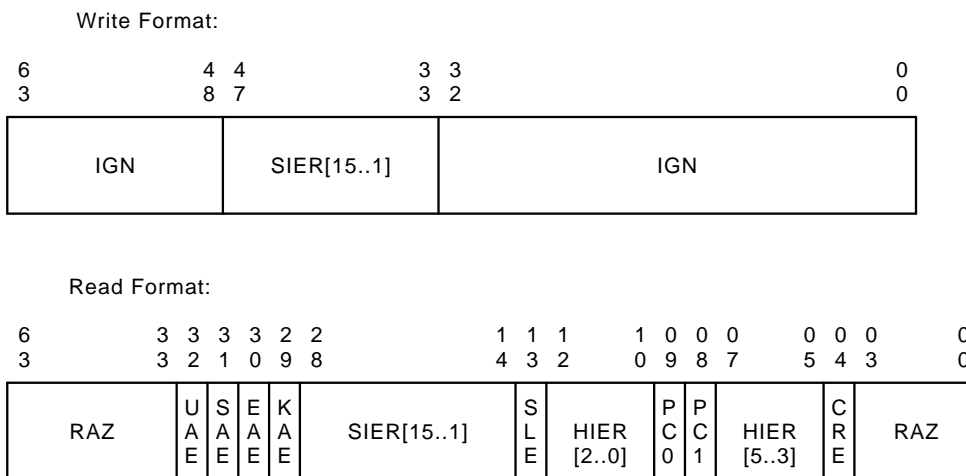
Figure 4–14 HIER Format



**4.1.18 Software Interrupt Enable Register (SIER)**

The software interrupt enable register is a read/write register. It is used to enable corresponding bits of the SIRR requesting interrupts. There is a one-to-one correspondence between the interrupt requests and enable bits. As with the reads of the interrupt request IPRs, reads of the SIER return the complete set of interrupt enable registers. See Section 4.1.14 for details. Figure 4–15 shows the SIER register format.

Figure 4–15 SIER Format



**4.1.19 Asynchronous System Trap Enable Register (ASTER)**

The AST interrupt enable register is a read/write register. It is used to enable corresponding bits of the ASTRR requesting interrupts. There is a one-to-one correspondence between the interrupt requests and enable bits. As with the reads of the interrupt request IPRs, reads of the ASTER return the complete set



## 4.2 A-box Internal Processor Registers

The A-box in the 21064 contains six major sections:

- Address translation data path
- Load silo
- Write buffer
- Data cache interface
- External bus interface unit (BIU)
- Internal processor registers

The address translation data path has a displacement adder that generates the effective virtual address for the load and store instructions and a pair of translation buffers that generate the corresponding physical address. The following sections describe the registers contained within the 21064 processor's A-box unit.

### 4.2.1 Translation Buffer Control Register (TB\_CTL)

The TB\_CTL register is write-only.

Figure 4–18 TB Control Register (TB\_CTL)

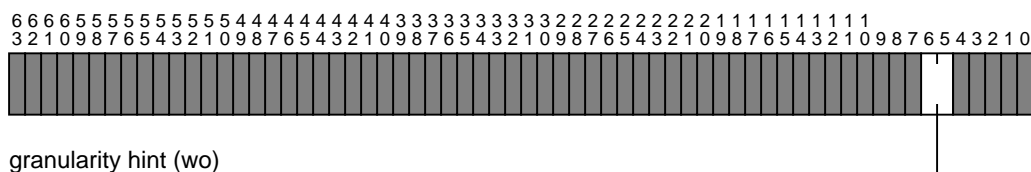


Table 4–11 TB Control Register Description

Field	Description
<b>6:5</b>	<b>granularity hint</b> <i>[write-only]</i> The granularity hint (GH) field selects between the 21064 TB page mapping sizes. The 21064 CPU provides two sizes in the ITB and all four sizes in the DTB. When only two sizes are provided, the large-page-select (GH=11(bin)) field selects the largest mapping size (512 * 8 Kbytes) and all other values select the smallest (8-Kbyte) size. The GH field affects both reads and writes to the ITB and DTB in the 21064.

### 4.2.2 Data Translation Buffer Page Table Entry Register (TB\_PTE)

The DTB PTE register is a read/write register representing the 32-entry small-page and 4-entry large-page DTB page table entries. The entry to be written is chosen by a not-last-used algorithm implemented in hardware and the value in the DTB\_CTL register. Writes to the DTB\_PTE use the memory format bit positions as described in the *Alpha Architecture Reference Manual* except some fields are ignored. In particular the valid bit is not represented in hardware.

## 4.2 A-box Internal Processor Registers

To ensure the integrity of the DTBs, the DTB's tag array is updated from the internal tag register when the DTB\_PTE register is written. Reads of the DTB\_PTE require two instructions. First, a read from the DTB\_PTE sends the PTE data to the DTB\_PTE\_TEMP register. Then a second instruction reading from the DTB\_PTE\_TEMP register returns the PTE entry to the register file. Reading or writing the DTB\_PTE register increments the TB entry pointer of the DTB indicated by the DTB\_CTL IPR. This allows the entire set of DTB PTE entries to be read.

**Figure 4–19 DTB\_PTE Register Format**

Small Page Format:

6 3	5 3	5 2	3 2	3 1	1 6	1 5	1 4	1 3	1 2	1 1	1 0	0 9	0 8	00 75	0 4	0 3	0 2	0 1	0 0
IGN	PFN[33..13]			IGN	U W E	S W E	E W E	K W E	U R E	S R E	E R E	K R E	I G N	A S M	I G N	F O W	F O R	I G N	

Large Page Format:

6 3	5 3	5 2	4 1	4 0	1 6	1 5	1 4	1 3	1 2	1 1	1 0	0 9	0 8	00 75	0 4	0 3	0 2	0 1	0 0
IGN	PFN[33..22]			IGN	U W E	S W E	E W E	K W E	U R E	S R E	E R E	K R E	I G N	A S M	I G N	F O W	F O R	I G N	

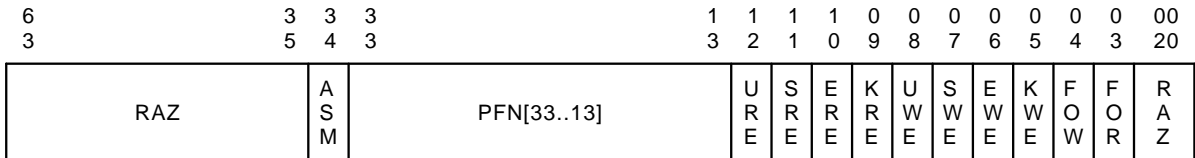
### 4.2.3 Data Translation Buffer Page Table Entry Temporary Register (DTB\_PTE\_TEMP)

The DTB\_PTE\_TEMP register is a read-only holding register for DTB\_PTE read data. Reads of the DTB\_PTE require two instructions to return the data to the register file. The first reads the DTB\_PTE register to the DTB\_PTE\_TEMP register. The second returns the DTB\_PTE\_TEMP register to the integer register file.

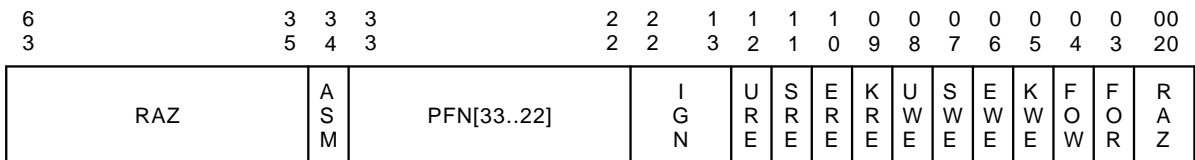
## 4.2 A-box Internal Processor Registers

**Figure 4–20 DTB\_PTE\_TEMP Register Format**

Small Page Format:



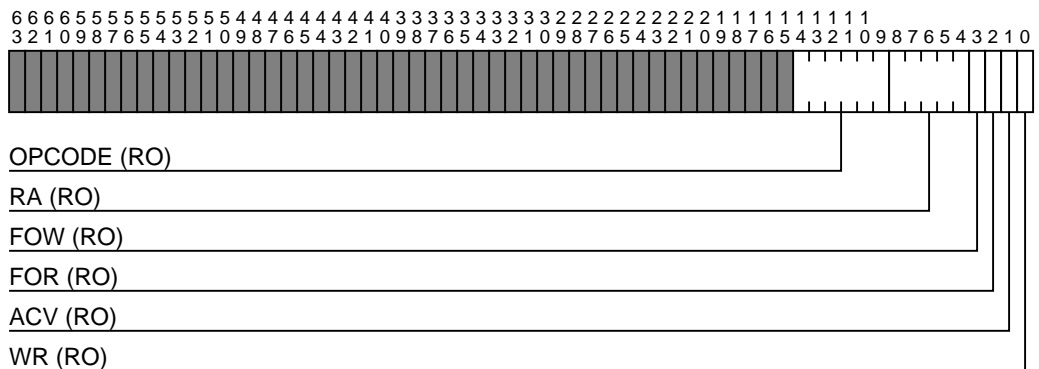
Large Page Format:



### 4.2.4 Memory Management Control and Status Register (MM\_CSR)

When data-stream faults occur, the information about the fault is latched and saved in the MM\_CSR register. The VA and MMCSR registers are locked against further updates until software reads the VA register. Palcode must explicitly unlock this register whenever its entry point was higher in priority than a DTB miss. MM\_CSR bits are modified only by hardware when the register is not locked and a memory management error or a DTB miss occurs. The MM\_CSR is unlocked after a reset.

**Figure 4–21 MM CSR (MM\_CSR)**



**Table 4–12 MM CSR Description**

Field	Description
14:9	<b>OPCODE</b> <i>[read-only]</i>

(continued on next page)

## 4.2 A-box Internal Processor Registers

Table 4–12 (Cont.) MM CSR Description

Field	Description
	Opcode field of the faulting instruction.
<b>8:4</b>	<b>RA</b> <i>[read-only]</i> Ra field of the faulting instruction.
<b>3</b>	<b>FOW</b> <i>[read-only]</i> Set if reference was a read and the PTE's FOR bit was set.
<b>2</b>	<b>FOR</b> <i>[read-only]</i> Set if reference was a read and the PTE's FOR bit was set.
<b>1</b>	<b>ACV</b> <i>[read-only]</i> Set if reference caused an access violation.
<b>0</b>	<b>WR</b> <i>[read-only]</i> Set if reference that caused error was a write.

### 4.2.5 Virtual Address Register(VA)

When data-stream faults or DTB misses occur the effective virtual address associated with the fault or miss is latched in the read-only VA register. The VA and MMCSR registers are locked against further updates until software reads the VA register. The VA IPR is unlocked after reset. Palcode must explicitly unlock this register whenever its entry point was higher in priority than a DTB miss.

### 4.2.6 Data Translation Buffer Zap Register (DTBZAP)

A write of any value to this IPR invalidates all 32 small-page and four large-page DTB entries. A write also resets the NLU pointer to its initial state. The DTBZAP is a pseudo register.

### 4.2.7 Data Translation Buffer ASM Register (DTBASM)

A write of any value to this IPR invalidates all 32 small-page and four large-page DTB entries in which the ASM bit is equal to 0. The DTBASM is a pseudo register.

### 4.2.8 Data Translation Buffer Invalidate Signal Register (DTBIS)

Any write to this pseudo register will invalidate the DTB entry that maps the virtual address held in the integer register. The integer register is identified by the Rb field of the HW\_MTPR instruction, used to perform the write.

### 4.2.9 Flush Instruction Cache Register (FLUSH\_IC)

A write of any value to this pseudo-IPR flushes the entire instruction cache.

### 4.2.10 Flush Instruction Cache ASM Register (FLUSH\_IC\_ASM)

In the 21064 CPU, a write of any value to this pseudo-IPR invalidates all instruction cache blocks in which the ASM bit is clear.

## 4.2 A-box Internal Processor Registers

### 4.2.11 A-box Control Register (A-BOX\_CTL)

The A-box control register directs the actions of the 21064 processor's A-box unit.

Figure 4–22 A-box Control (Abox\_CTL)

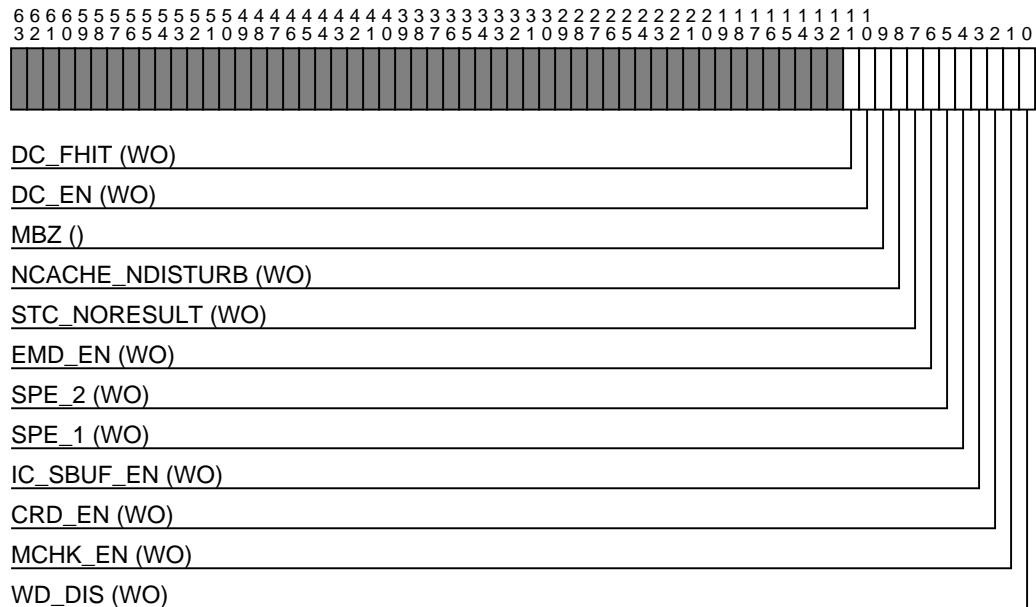


Table 4–13 A-box Control Description

Field	Description
<b>11</b>	<b>DC_FHIT</b> <i>[write-only]</i> D-cache force hit. When set, this bit forces all data-stream references to hit in the data cache. This bit takes precedence over DC_EN, for example, when DC_FHIT is set and DC_EN is clear all data-stream references hit in the data cache.
<b>10</b>	<b>DC_EN</b> <i>[write-only]</i> D-cache enable. When clear, this bit disables and flushes the D-cache. When set, this bit enables the D-cache.
<b>9</b>	<b>MBZ</b>
<b>8</b>	<b>NCACHE_NDISTURB</b> <i>[write-only]</i> When set, any reference to non-cacheable memory space will not cause an arbitrary invalidation of a primary D-cache location. This should be set.
<b>7</b>	<b>STC_NORESULT</b> <i>[write-only]</i> Must be cleared
<b>6</b>	<b>EMD_EN</b> <i>[write-only]</i> Limited hardware support is provided for big endian data formats by way of bit <6> of the A-box_CTL register. When set, this bit, inverts physical address bit <2> for all D-stream references. It is intended that chip endian mode be selected during initialization PALcod only.

(continued on next page)

## 4.2 A-box Internal Processor Registers

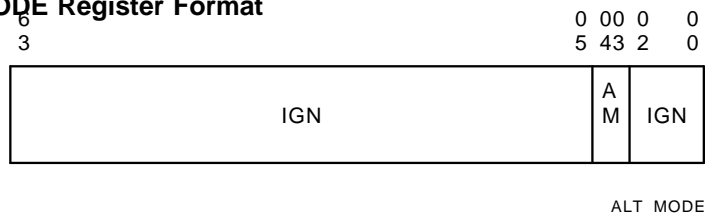
**Table 4–13 (Cont.) A-box Control Description**

Field	Description
5	<p><b>SPE_2</b> <i>[write-only]</i></p> <p>This bit, when set, enables one-to-one super-page mapping of D-stream virtual addresses with VA&lt;33:13&gt;, if virtual address bits VA&lt;42:41&gt; = 2. Virtual address bits VA&lt;40:34&gt; are ignored in this translation. Access is only allowed in kernel mode.</p>
4	<p><b>SPE_1</b> <i>[write-only]</i></p> <p>This bit, when set, enables one-to-one super-page mapping of D-stream virtual addresses with VA&lt;42:30&gt; = 1FFE(Hex) to physical addresses with PA&lt;33:30&gt; = 0(Hex). Access is only allowed in kernel mode.</p>
3	<p><b>IC_SBUF_EN</b> <i>[write-only]</i></p> <p>I-cache stream buffer enable. When set, this bit enables operation of a single entry I-cache stream buffer.</p>
2	<p><b>CRD_EN</b> <i>[write-only]</i></p> <p>Corrected read data interrupt enable. When this bit is set the A-box generates an interrupt request whenever a pin bus transaction is terminated with a cAck_h code of SOFT_ERROR.</p>
1	<p><b>MCHK_EN</b> <i>[write-only]</i></p> <p>Machine Check Enable. When this bit is set the A-box generates a machine check when errors (which are not correctable by the hardware) are encountered. When this bit is cleared, uncorrectable errors do not cause a machine check, but the BIU_STAT, DC_STAT, BIU_ADDR, FILL_ADDR and DC_ADDR registers are updated and locked when the errors occur.</p>
0	<p><b>WD_DIS</b> <i>[write-only]</i></p> <p>Write Buffer unload Disable. When set, this bit prevents the write buffer from sending write data to the BIU. It should be set for diagnostics only.</p>

### 4.2.12 Alternate Processor Mode Register (ALT\_MODE)

ALT\_MODE is a write-only internal processor register. The AM field specifies the alternate processor mode used by HW\_LD and HW\_ST instructions that have their ALT bit (bit 14) set. The format of the register is shown in Figure 4–23.

**Figure 4–23 ALT\_MODE Register Format**





**Table 4–14 ALT Mode**

ALT_MODE[4..3]	Mode
0 0	Kernel
0 1	Executive
1 0	Supervisor
1 1	User

### 4.2.13 Cycle Counter Register

The 21064 supports a cycle counter as described in the *Alpha Architecture Reference Manual*. This counter, when enabled, increments once for each CPU cycle. HW\_MTPR Rn,CC writes CC<63..32> with the value held in Rn<63..32>, and CC<31..0> are not changed. This register is read by the RCC instruction defined in the *Alpha Architecture Reference Manual*.

### 4.2.14 Cycle Counter Control Register (CC\_CTL)

The cycle counter register is a write-only IPR. HW\_MTPR Rn,CC\_CTL writes CC<31:0> with the value held in Rn<31:0>, and CC<63:32> are not changed. CC<3:0> must be written with 0. If Rn<32> is set then the counter is enabled, otherwise the counter is disabled.

## 4.2 A-box Internal Processor Registers

### 4.2.15 Bus Interface Control Unit Register (BIU\_CTL)

The BIU\_CTL register directs the actions of the bus interface unit.

Figure 4–24 Bus Interface Control Unit Register (ITB\_PTE\_TEMP)

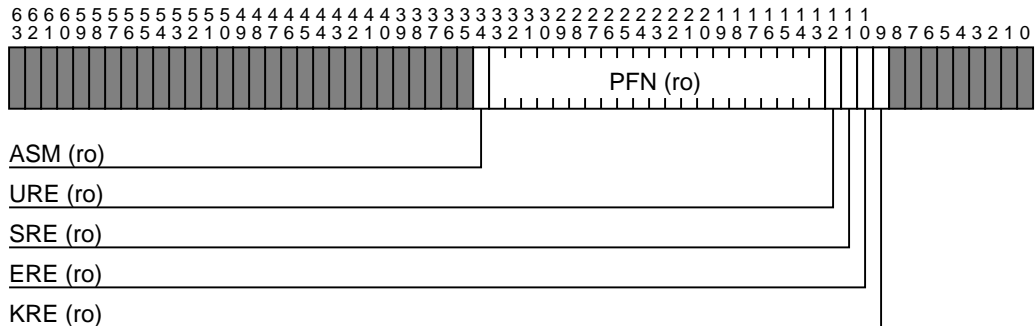


Table 4–15 Bus Interface Control Unit Register Description

Field	Description
<b>43</b>	<b>BC_BURST_ALL</b> <i>[write-only]</i>
<b>42:40</b>	<b>BC_BURST_SPD</b> <i>[write-only]</i>
<b>39</b>	<b>MBZ</b> <i>[write-only]</i>
<b>38</b>	<b>SYS_WRAP</b> <i>[write-only]</i>
<b>37</b>	<b>BYTE_PARITY</b> <i>[write-only]</i>
<b>36</b>	<b>BAD_DP</b> <i>[write-only]</i> Bad Data Parity - When set, BAD_DP causes the 21064 CPU to invert the value placed on bits <0>,<7>,<14> and <21> of the check_h<27:0> field during off-chip writes. This produces bad parity when the 21064 CPU is in parity mode, and bad check bit codes when the CPU is in EDC mode.
<b>35:32</b>	<b>BC_PA_DIS</b> <i>[write-only]</i> Backup cache physical address disable - This 4-bit field may be used to prevent the CPU chip from using the external cache to service reads and writes based on the quadrant of physical address space which they reference. Table 4–17 shows the correspondence between this bit field and the physical address space. When a read or write reference is presented to the bus interface unit (BIU), the values of BC_PA_DIS, BC_ENA, and physical address bits <33:32> together determine whether or not to try using the external cache to satisfy the reference. If the external cache is not to be used for a given reference, the bus interface unit does not probe the tag store and makes the appropriate system request immediately. The value of BC_PA_DIS has no impact on which portions of the physical address space may be cached in the primary caches. System components control this through the RDACK field of the pin bus. BC_PA_DIS is not initialized by a reset.
<b>31</b>	<b>BAD_TCP</b> <i>[write-only]</i> BAD Tag Control Parity - When set, BAD_TCP causes the 21064 CPU to write bad parity into the tag control RAM whenever it does a fast external RAM write.

(continued on next page)

## 4.2 A-box Internal Processor Registers

**Table 4–15 (Cont.) Bus Interface Control Unit Register Description**

Field	Description
<b>30:28</b>	<p><b>BC_SIZE</b> <i>[write-only]</i></p> <p>Backup Cache Size - This field is used to indicate the size of the external cache. BC_SIZE is not initialized by a reset and must be explicitly written before enabling the backup cache. (See Table 4–16 for the encodings.)</p>
<b>27:13</b>	<p><b>BC_WE_CTL[15:1]</b> <i>[write-only]</i></p> <p>Backup Cache Write Enable Control. This field controls the timing of the write enable and chip enable pins during writes into the data and tag control RAMs. It consists of 15 bits, where each bit determines the value placed on the write enable and chip enable pins during a given CPU cycle of the RAM write access. When a given bit of BC_WE_CTL is set, the write enable and chip enable pins are asserted during the corresponding CPU cycle of the RAM access. BC_WE_CTL&lt;0&gt; (bit 13 in BIU_CTL) corresponds to the second cycle of the write access, BC_WE_CTL&lt;1&gt; (bit 14 in BIU_CTL) to the third CPU cycle, and so on. The write enable pins are never asserted in the first CPU cycle of a RAM write access.</p> <p>Unused bits in the BC_WE_CTL field must be written with 0s.</p> <p>BC_WE_CTL is not initialized on reset and must be explicitly written before enabling the external backup cache.</p>
<b>12</b>	<p><b>DELAY_WDATA</b> <i>[write-only]</i></p> <p>DELAY_DATA</p>
<b>11:8</b>	<p><b>BC_WR_SPD</b> <i>[write-only]</i></p> <p>Backup cache write speed. This field indicates to the bus interface unit the write cycle time of the RAMs used to implement the off-chip external cache, (Backup cache on the CPU module), measured in CPU cycles. It should be written with a value equal to one less the write cycle time of the external cache RAMs.</p> <p>Access times for writes must be in the range 16..2 CPU cycles, which means the values for the BC_WR_SPD field are in the range of 15..1.</p> <p>BC_WR_SPD is not initialized on reset and must be explicitly written before enabling the external cache.</p>
<b>7:4</b>	<p><b>BC_RD_SPD</b> <i>[write-only]</i></p> <p>Backup (external) cache read speed. This field indicates to the bus interface unit the read access time of the RAMs used to implement the off-chip external cache, measured in CPU cycles. This field should be written with a value equal to one less the read access time of the external cache RAMs.</p> <p>Access times for reads must be in the range 16..3 CPU cycles, which means the values for the BC_RD_SPD field are in the range of 15..2.</p> <p>BC_RD_SPD are not initialized on reset and must be explicitly written before enabling the external cache.</p>
<b>3</b>	<p><b>BC_FHIT</b> <i>[write-only]</i></p> <p>Backup cache force hit. (This cache is external to the 21064 chip.) When this bit and BC_EN are set, all pin bus READ_BLOCK and WRITE_BLOCK transactions are forced to hit in the backup cache. Tag and tag control parity are ignored when the BIU operates in this mode. BC_EN takes precedence over BC_FHIT. When BC_EN is clear and BC_FHIT is set, no tag probes occur and external requests are directed to the CREQ_H pins.</p> <p>Note that the BC_PA_DIS field takes precedence over the BC_FHIT bit.</p>
<b>2</b>	<p><b>OE</b> <i>[write-only]</i></p>

(continued on next page)

## 4.2 A-box Internal Processor Registers

**Table 4–15 (Cont.) Bus Interface Control Unit Register Description**

Field	Description
	Output enable - When this bit is set, the 21064 CHP chip does not assert its chip enable pins during RAM write cycles, thus enabling these pins to be connected to the output enable pins of the cache RAMs.
<b>1</b>	<b>EDC</b> <i>[write-only]</i> Error detection and correction. When this bit is set, the 21064 CPU chip generates/expects EDC on the CHECK_H pins. When this bit is clear the CPU chip generates/expects parity on four of the CHECK_H pins.
<b>0</b>	<b>BC_ENA</b> <i>[write-only]</i> External cache enable. When clear, this bit disables the external cache. When the external cache is disabled, the BIU does not probe the external cache tag store for read and write references; it initiates a request on CREQ_H immediately.

The BC\_SIZE bits are interpreted as shown in Table 4–16.

**Table 4–16 Backup Cache Size**

BC_SIZE bits	Backup Cache Size
0 0 0	128 kbytes
0 0 1	256 kbytes
0 1 0	512 kbytes
0 1 1	1 Mbytes
1 0 0	2 Mbytes
1 0 1	4 Mbytes
1 1 0	8 Mbytes

The BC\_PA\_DIS field is interpreted as shown in Table 4–17.

**Table 4–17 BC\_PA\_DIS**

BIU_CTL Bits	Physical Address
<32>	PA[33..32] = 0
<33>	PA[33..32] = 1
<34>	PA[33..32] = 2
<35>	PA[33..32] = 3

## 4.3 Privileged Architecture Library Temporary Registers (PAL\_TEMP)

The 21064 contains 32 registers that provide temporary storage for PALcode. These registers are accessible through HW\_MXPR instructions.

## 4.3 Privileged Architecture Library Temporary Registers (PAL\_TEMP)

### 4.3.1 Data Cache Status Register (DC\_STAT)

The DC\_STAT is a read-only internal processor register for diagnostic use only.

When an external EDC or parity error is recognized during a primary cache fill operation, the DC\_STAT register is locked against further updates. If the cache fill was due to data stream activity, PALcode may use the DC\_STAT contents in conjunction with information latched elsewhere to recover from some single-bit EDC errors. DC\_STAT is unlocked when DC\_ADDR is read. The format of the DC\_STAT register is shown in Figure 4-25.

Figure 4-25 DC\_STAT Register Format

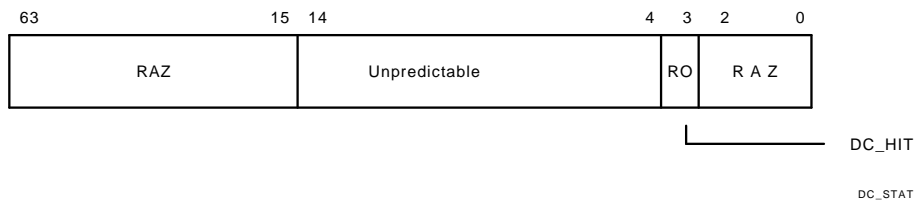


Table 4-18 Data cache Status Register

Field	Type	Description
DC_HIT	RO	Data cache hit - This bit indicates whether the last load or store instruction processed by the A-box hit (DC_HIT set) or missed (DC_HIT clear) the data cache. In the 21064 CPU, loads that miss the data cache may be completed without requiring external reads. that is, pending fill or pending store hits.
SEO	RO	Second Error Occurred. Set when an error that normally locks the DC_STAT register occurs while the DC_STAT register is already locked.

The other DC\_STAT bit (Table 4-19) following bits are meaningful only if the FILL\_EDC or FILL\_DPERR bit in the BIU\_STAT register is set.

### 4.3 Privileged Architecture Library Temporary Registers (PAL\_TEMP)s

**Table 4–19 Data cache status Error Modifiers**

Field	Type	Description
RA	RO	This bit is the Ra field of the instruction that caused the error.
INT	RO	Integer, When set, indicates an integer load or store.
LW	RO	Longword, When set, indicates that the data length of the load or store was longword.
VAX_FP	RO	VAX floating-point, When INT is clear, this bit is set to indicate that a VAX floating-point format load or store caused the error.
LOCK	RO	Lock, This bit is set to indicate that the error stemmed from a LDLL, LDQL, STLC, or STQC instruction.
STORE	RO	Store - This bit is set to indicate that the error stemmed from a store instruction.

## 4.3 Privileged Architecture Library Temporary Registers (PAL\_TEMP)

### 4.3.2 Bus Interface Unit Status Register (BIU\_STAT)

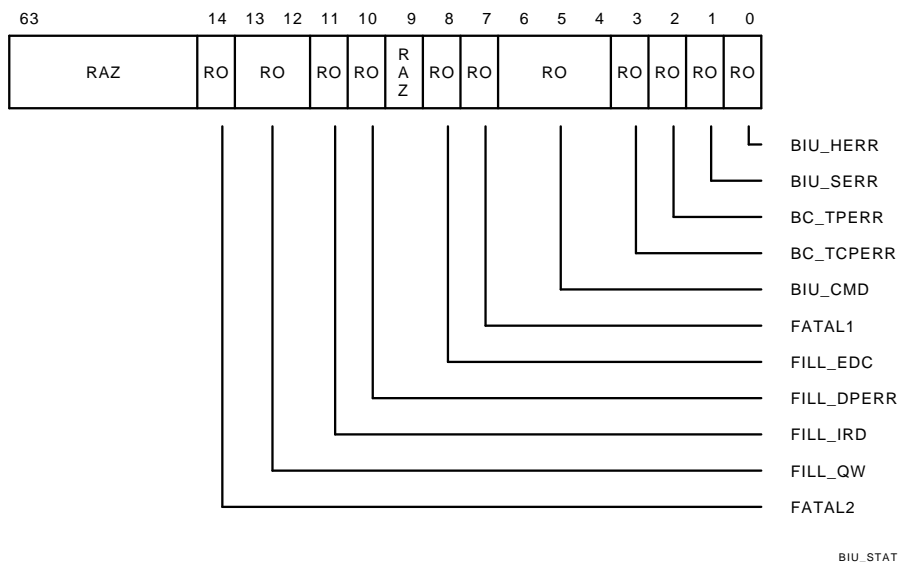
BIU\_STAT is a read-only internal processor register.

When the BIU\_HERR, BIU\_SERR, BC\_TPERR, or BC\_TCPERR, is set, BIU\_STAT<6:0> are locked against further updates; the address associated with the error is latched and locked in the BIU\_ADDR register. BIU\_STAT<6:0> and the BIU\_ADDR are also spuriously locked when a parity error or an uncorrectable EDC error occurs during a primary cache fill operation. BIU\_STAT<7:0> and BIU\_ADDR are unlocked when the BIU\_ADDR register is read.

When FILL\_EDC or BIU\_DPERR is set, BIU\_STAT<13:8> are locked against further updates; the address associated with the error is latched and locked in the FILL\_ADDR register. BIU\_STAT<14:8> and FILL\_ADDR are unlocked when the FILL\_ADDR register is read.

BIU\_STAT is not unlocked or cleared by a reset and needs to be explicitly cleared by PALcode. The BIU\_STAT register format is shown in Figure 4-26.

Figure 4-26 BIU\_STAT Register Format



## 4.3 Privileged Architecture Library Temporary Registers (PAL\_TEMP)

Table 4–20 BIU STAT

Field	Type	Description
BIU_HERR	RO	Hard Error, When set, indicates that an external cycle was terminated with the CACK_H pins indicating HARD_ERROR.
BIU_SERR	RO	Soft Error, When set, indicates that an external cycle was terminated with the CACK_H pins indicating SOFT_ERROR. Note that this should never occur on a DEC 4000 system.
BC_TPERR	RO	Backup Cache Tag Parity Error, When set, indicates that an external cache tag probe encountered bad parity in the tag address RAM.
BC_TCPERR	RO	Backup Cache Tag Control Parity Error, When set, indicates that an external cache tag probe encountered bad parity in the tag control RAM.
BIU_CMD	RO	Bus Interface Unit CMD, This field latches the cycle type on the CREQ_H pins when a BIU_HERR, BIU_SERR, BC_TPERR, or BC_TCPERR error occurs.
BIU_SEO	RO	Bus Interface Unit SEO, When set, indicates one of two things: (1) that an external cycle was terminated with the CACK_H pins indicating a HARD_ERROR or (2) that an external cache tag probe encountered bad parity in the tag address RAM or the tag control RAM while BIU_HERR, BIU_SERR, BC_TPERR, or BC_TCPERR was set.
FILL_EDC	RO	EDC Error, When set, indicates that primary cache fill data received from outside the CPU chip contained an EDC error.
FILL_DPERR	RO	Fill Parity Error, When set, indicates that the BIU received data with a parity error from outside the CPU chip while performing either a data cache or instruction cache fill. FILL_DPERR is meaningful only when the CPU chip is in parity mode, as opposed to EDC mode.
FILL_IRD	RO	This bit is meaningful only when either FILL_EDC or FILL_DPERR is set. When set, FILL_IRD indicates that the error that caused FILL_EDC or FILL_DPERR to set occurred during an instruction cache fill. When cleared, this bit indicates that the error occurred during a data cache fill.
FILL_QW	RO	This field is meaningful only when either FILL_EDC or FILL_DPERR is set. FILL_QW identifies the quadword within the hexaword primary cache fill block that caused the error. FILL_QW can be used with FILL_ADDR<33:5> to get the complete physical address of the bad quadword.
FILL_SEO	RO	Fill SEO, When set, indicates that (1) a primary cache fill operation resulted in either an uncorrectable EDC error or in a parity error while FILL_EDC or (2) FILL_DPERR was already set.

### 4.3.3 Bus Interface Unit Address Register (BIU\_ADDR)

This read-only register contains the physical address associated with errors reported by BIU\_STAT<7:0>. BIU\_ADDRs contents are meaningful only when BIU\_HERR, BIU\_SERR, BC\_TPERR, or BC\_TCPERR are set. Reads of BIU\_ADDR unlock both BIU\_ADDR and BIU\_STAT<7:0>.



### 4.3 Privileged Architecture Library Temporary Registers (PAL\_TEMP)s

In the 21064 CPU, BIU\_ADDR<33:5> contain the values of ADR\_H<33:5> associated with the pin bus transaction which resulted in the error indicated in BIU\_STAT<7:0>.

In the 21064, if the BIU\_CMD field of the BIU\_STAT register indicates that the transaction that received the error was READ\_BLOCK or LDx/L. The state of BIU\_STAT<4:2> is unpredictable. If the BIU\_CMD field of the BIU\_STAT register encodes any pin bus command other than READ\_BLOCK or LDx/L, then BIU\_ADDR<4:2> contains 0s. BIU\_ADDR<63:34> and BIU\_ADDR<1:0> always read as 0.

#### 4.3.4 Fill Address Register (FILL\_ADDR)

This read-only register contains the physical address associated with errors reported by BIU\_STAT<14:8>. Its contents are meaningful only when FILL\_EDC or FILL\_DPERR is set. Reads of FILL\_ADDR unlock FILL\_ADDR, BIU\_STAT<14:8> and FILL\_SYNDROME.

In the 21064, FILL\_ADDR<33:5> identify the 32-byte cache block that the CPU was attempting to read when the error occurred.

If the FILL\_IRD bit of the BIU\_STAT register is clear (indicating that the error occurred during a data stream cache fill) then FILL\_ADDR<4:2> contain bits <4:2> of the physical address generated by the load instruction that triggered the cache fill. If FILL\_IRD is set, then the state of FILL\_ADDR<4:2> is unpredictable. FILL\_ADDR<63:34> and FILL\_ADDR<1:0> are read as 0.

## 4.3 Privileged Architecture Library Temporary Registers (PAL\_TEMP)

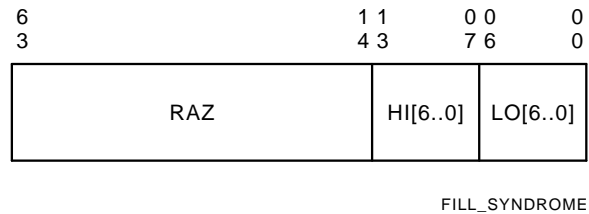
### 4.3.5 Fill Syndrome Register (FILL\_SYNDROME)

The FILL\_SYNDROME register is a 14-bit read-only register.

If the CPU chip is in EDC mode and an EDC error is recognized during a primary cache fill operation, the syndrome bits associated with the bad quadword are locked in the FILL\_SYNDROME register. FILL\_SYNDROME<6:0> contain the syndrome associated with the lower longword of the quadword, and FILL\_SYNDROME<13:7> contain the syndrome associated with the higher longword of the quadword. A syndrome value of 0 means that no errors were found in the associated longword. (See Table 4–21 for a list of syndromes associated with correctable single-bit errors.) The FILL\_SYNDROME register is unlocked when the FILL\_ADDR register is read.

If the chip is in parity mode and a parity error is recognized during a primary cache fill operation, the FILL\_SYNDROME register indicates which longword in the quadword has bad parity. FILL\_SYNDROME<0> is set to indicate that the low longword was corrupted, and FILL\_SYNDROME<7> is set to indicate that the high longword was corrupted. FILL\_SYNDROME<13:8> and <6:1> are RAZ in parity mode. Figure 4–27 shows the format of the Fill\_syndrome register and Table 4–21 lists the syndromes for single-bit errors.

**Figure 4–27 Fill Syndrome Register Format**



### 4.3 Privileged Architecture Library Temporary Registers (PAL\_TEMP)

Table 4–21 Syndromes for Single-Bit Errors

Data Bit	Syndrome <sub>16</sub>	Check Bit	Syndrome <sub>16</sub>
00	4F	00	01
01	4A	01	02
02	52	02	04
03	54	03	08
04	57	04	10
05	58	05	20
06	5B	06	40
07	5D		
08	23		
09	25		
10	26		
11	29		
12	2A		
13	2C		
14	31		
15	34		
16	0E		
17	0B		
18	13		
19	15		
20	16		
21	19		
22	1A		
23	1C		
24	62		
25	64		
26	67		
27	68		
28	6B		
29	6D		
30	70		
31	75		

**Note**

A syndrome of 1F is invalid and indicates that a bad EDC code was written intentionally into the cache. This is done when bad data is provided by the system bus to the C<sup>3</sup> interface. A syndrome of 0 indicates that no EDC error was detected.

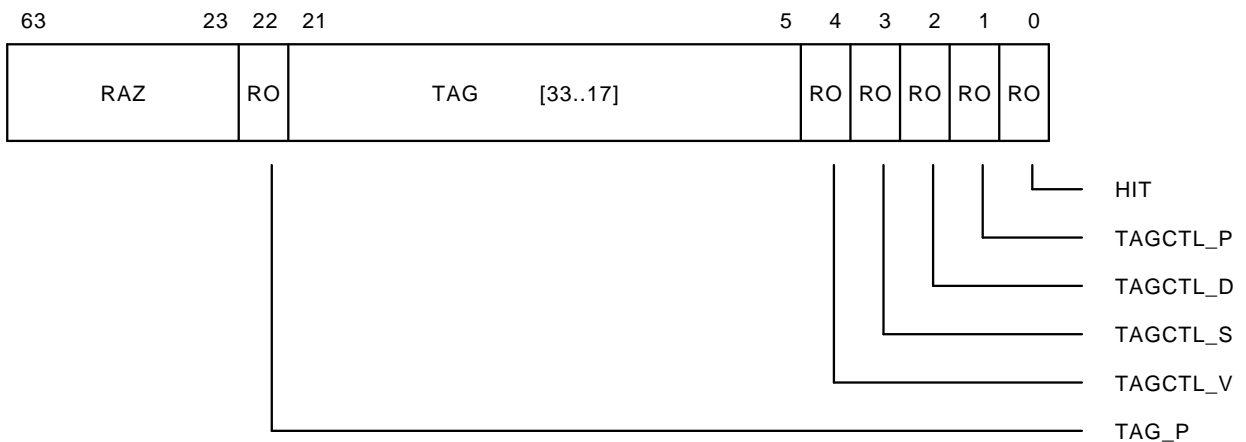
## 4.3 Privileged Architecture Library Temporary Registers (PAL\_TEMP)s)

### 4.3.6 Backup Cache Tag Register (BC\_TAG)

BC\_TAG is a read-only internal processor register. Unless locked, the BC\_TAG register is loaded with the results of every backup cache tag probe. When a tag or tag control parity error or primary fill data error (parity or EDC) occurs, this register is locked against further updates. Software may read the LSB of this register by using the HW\_MFPR instruction. Each time an HW\_MFPR from BC\_TAG completes, the contents of BC\_TAG is shifted one bit position to the right. Then the entire register may be read using a sequence of HW\_MFPRs. Software may unlock the BC\_TAG register using a HW\_MTPR instruction to BC\_TAG.

Successive HW\_MFPRs from the BC\_TAG register must be separated by at least one null cycle. Figure 4–28 shows the format of the BC\_TAG register.

Figure 4–28 BC\_TAG Register Format



Unused tag bits in the TAG field of this register are always cleared, based on the size of the external cache as determined by the BC\_SIZE field of the BIU\_CTL register.

### 4.4 Error Detection and Correction (EDC)

When in EDC mode, the 21064 CPU generates longword EDC on writes, and checks EDC on reads. The CPU chip does contain hardware to correct single-bit errors.

When an EDC error is recognized during a data cache fill, the bus interface unit places the affected fill block into the data cache unchanged, validates the block, and posts a machine check. The load instruction that triggered the data cache fill is completed by writing the requested longword or longwords into the register file. Whether or not the longword read by the load instruction was the cause of the error, a machine check is posted.

The I-box reacts to the machine check by (1) aborting instruction execution before any instruction issued after the load can overwrite the register containing the load data, and (2) vectoring to the PALcode machine check handler. Sufficient state is retained in various status registers for PALcode to determine whether or not the error affects the longword read by the load instruction, and whether the error is correctable. In any event, PALcode must explicitly flush the data cache.

If the longword containing the error was written into the register file, PALcode must either correct it and restart the machine, or report an uncorrectable hardware error to the operating system. Whether or not the failing longword was read by the load instruction, PALcode may scrub memory by explicitly reading the longword with the physical/lock variant of the HW\_LD instruction, flipping the necessary bit, and writing the longword with the physical/conditional variant of the HW\_ST instruction. When PAL rereads the affected longword, the hardware may report no errors. This indicates that the longword has been overwritten.

When an EDC error occurs during an instruction cache fill, the bus interface unit places the affected fill block into the instruction cache unchanged, validates the block, and posts a machine check. The I-box vectors to the PALcode machine check handler before it executes any of the instructions in the bad block. PAL code may then flush the instruction cache and scrub memory as described in this section.

Compared with hardware error correction, this approach is vulnerable to the following:

- Single-bit errors during instruction stream reads of the PALcode machine check handler
- Single-bit errors in multiple quadwords of a cache fill block
- Single-bit errors resulting from multiple silo'ed load misses

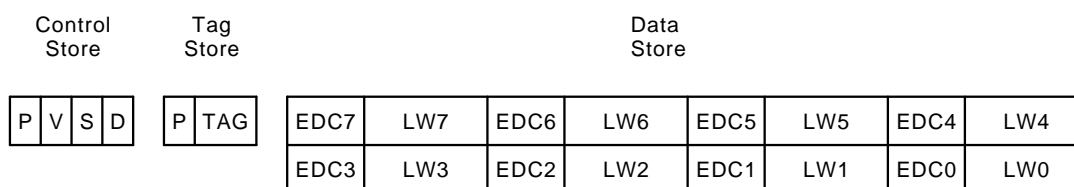
## Backup Cache (B-cache)

The DEC 4000 backup cache is a 1 megabyte, direct mapped, physical write back cache. It has a fixed 32-byte block size and supports the system bus snooping protocol to allow for a dual processor implementation.

Each cache block entry consists of three functionally identifiable storage element arrays:

- The control store, which is parity protected, contains the binary flags that indicate whether a particular cache block entry is valid, dirty, and/or shared.
- The tag store, which is also parity protected, contains the high order address bits of the data currently stored in that particular cache block entry.
- The data store, which is EDC protected, and contains the actual 32 bytes of “cached” data.

Figure 5–1 Back-up Cache Entry



### 5.1 Control Store

The control store of the backup cache contains the binary flags indicating the status of the cache block. These flags are defined in Table 5–1.

Table 5–1 Cache Block Status Flags

Flag	Description
VALID	When set, this flag indicates that the data found in the other bits of the control store, the tag store, and the data store contain valid information. The VALID bit is set only by the backup cache controller when a cache block is filled with new data. The VALID bit is cleared only by the backup cache controller when a system bus write requires a cache block invalidation. To initialize the VALID bit at power-up. See Section 11.2.

(continued on next page)

## 5.1 Control Store

Table 5–1 (Cont.) Cache Block Status Flags

Flag	Description
DIRTY	<p>When set, the DIRTY and VALID bits indicate the data store contains an updated copy of a main memory location. This cache data must be written back to main memory when the cache location is victimized. In a DEC 4000 system, there is only one copy of any given memory location marked DIRTY. The DIRTY bit is set by the processor performing a fast backup cache write hit cycle, or by the backup cache assisting the processor perform a STxC cycle. To learn how to initialize the DIRTY bit at power-up. See Section 11.2. When the VALID bit is cleared the backup cache controller guarantees that the DIRTY bit is also cleared.</p>
SHARED	<p>When set, SHARED and the VALID bit indicates that the data store contains a copy of a main memory location that another system bus node also has a copy of. Writes to this location must “write-through” the cache to maintain a coherent view of memory.</p> <p>The SHARED bit is set (1) by the backup cache controller when the system bus CSHARED_L signal is asserted during cache block allocation or (2) when a system bus read hits a location found in the backup cache (CSHARED_L must be pulled).</p> <p>The SHARED bit is cleared by the backup cache controller when it performs a system bus WRITE and the CSHARED_L signal is not asserted. To initialize the SHARED bit at power-up, refer to Section 11.2. When the VALID bit is cleared the backup cache controller guarantees that the DIRTY bit is also cleared.</p>
PARITY	<p>This flag contains EVEN parity over the contents of the Control Store. The PARITY flag is valid even if the VALID bit is not set. To learn how to initialize the VALID bit at power-up, see Section 11.2. Parity is checked by the processor during every backup cache probe cycle, and by the backup cache controller during every system bus initiated probe cycle. For details regarding parity error detection, see Chapter 10.</p>

## 5.2 Tag Store

The Tag Store contains the high order address bits <30:20> (<30:19> for a 512KB cache) of the memory location that currently resides in the cache entry. There is a single parity bit that provides EVEN parity over the complete Tag Store. Once power-up initialization has occurred, the parity bit contains valid parity regardless of the value of the control stores VALID bit.

Parity is checked by the processor during every B-Cache probe cycle, and by the B-Cache controller during every system bus initiated probe cycle. For details regarding parity error detection, see Chapter 10.

## 5.3 Data Store

The data store contains the actual data of the memory location that is “cached”. Every cache entry contains 8 longwords. Each longword is protected by 7 bits of EDC. Physically the cache is only 4 longwords wide, so a cache block consists of 2 consecutive addresses aligned on a 32-byte block boundary. After initialization, the EDC bits are valid regardless of the validity of the control store’s VALID bit.

Error checking occurs whenever the processor hits the cache on a read. Error checking and possible correction is performed whenever a system bus read probe hits dirty, the victimization of a dirty location, or a masked processor write to a shared location. For details on EDC error detection, see Chapter 10.

## 5.4 Backup Cache Control Register Definitions

The backup cache subsystem is manipulated at separate times by two different controllers. The first is the processor, which probes the tag field. If hit, it reads or writes data into the backup cache. The behavior of the processor relative to the backup cache is controlled or monitored by the processors BIU\_STAT, BIU\_ADDR, FILL\_ADDR, BIU\_CTL, AND BC\_TAG internal processor registers. See Section 4.3.

The second is the backup cache controller which processes all miss traffic and system bus probe activity. The behavior of the backup cache controller is determined by the settings of the backup cache control and status registers defined in the following sections.

### 5.4.1 Backup Cache Control Register (CSR0)

The backup cache control register contains all of the control fields to enable and disable various functions of the backup cache. This register is primarily for the use of diagnostic self-test code running during system initialization.

---

**Note**

---

The CPU ensures that the state of control flags in the low longword are consistent with control flags in the upper longword, by performing quadword writes.

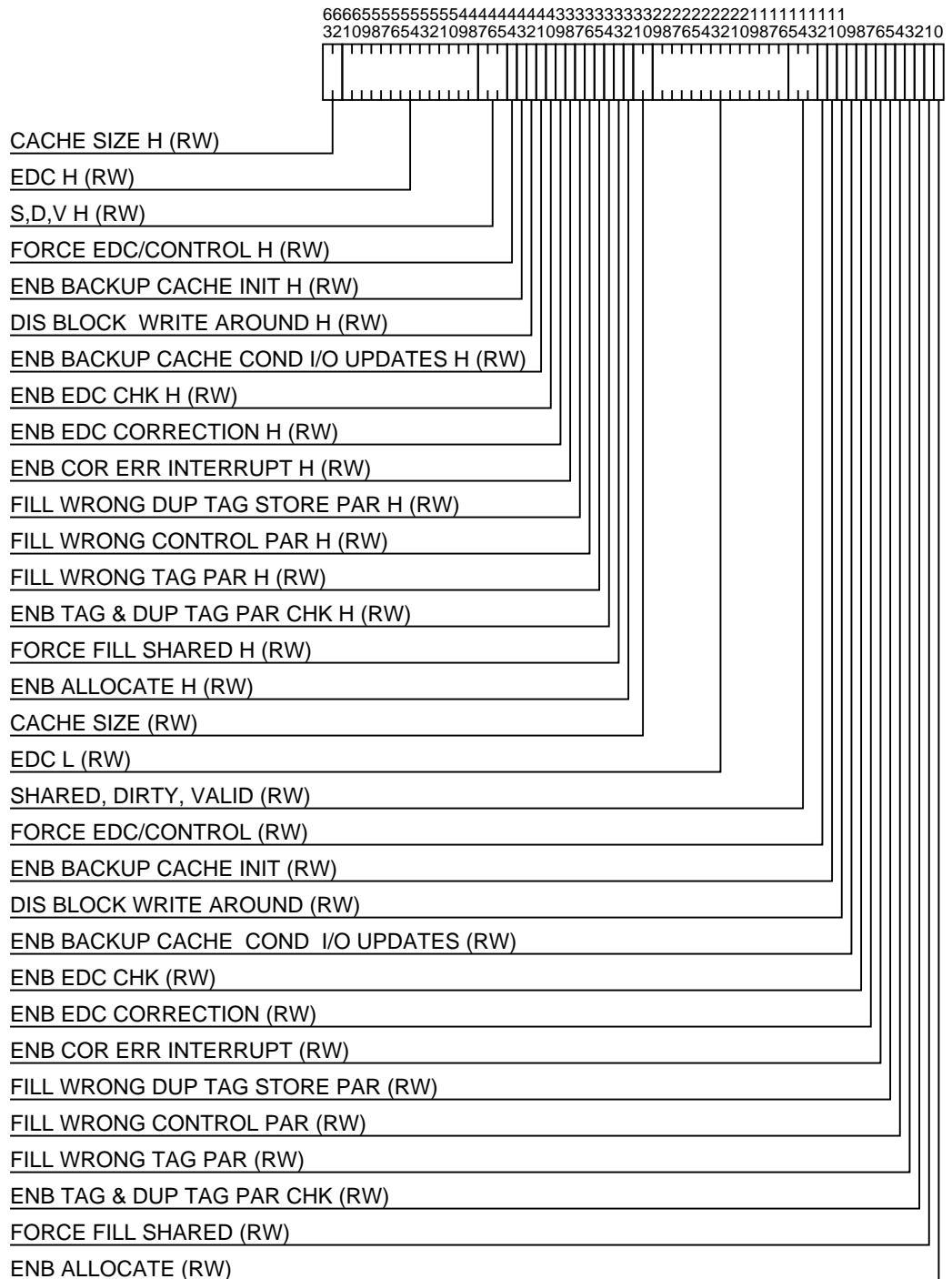
---

The backup cache control register for CPU<sub>0</sub> is located at address 2.0000.0000<sub>16</sub>, and for CPU<sub>1</sub> at address 2.0800.0000<sub>16</sub>.



## 5.4 Backup Cache Control Register Definitions

Figure 5–2 Backup Cache Control Register (BCC)



## 5.4 Backup Cache Control Register Definitions

Table 5–2 Backup Cache Control Register Description

Field	Description
<b>63:62</b>	<p><b>CACHE SIZE H</b> <i>[read/write]</i></p> <p>Cleared on power-up. They must be set correctly and they must match bits &lt;31:30&gt; before the backup cache is accessed.</p> <ul style="list-style-type: none"> <li>• 11 4 MB cache</li> <li>• 10 4 MB cache</li> <li>• 01 1 MB cache</li> <li>• 00 512 kB cache</li> </ul> <p>The value of this field not only controls which tag signals are used in determining the proper parity value for the tag, but also determines whether address bits &lt;21&gt;, &lt;20&gt;, and &lt;19&gt; should be driven by the C<sup>3</sup> during access to the backup cache.</p>
<b>61:48</b>	<p><b>EDC H</b> <i>[read/write]</i></p> <p>These bits are cleared on power-up. When the FORCE EDC/CONTROL bit 12 is set during backup cache initialization, the values specified in this register are forced on the EDC field of longwords 3 and 1 of any filled backup cache locations. See Section 11.2 for details.</p>
<b>47:45</b>	<p><b>S,D,V H</b> <i>[read/write]</i></p> <p>Cleared on power-up. These bits must always match bits &lt;15:13&gt; in this register.</p>
<b>44</b>	<p><b>FORCE EDC/CONTROL H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;12&gt; in this register.</p>
<b>43:43</b>	<p><b>ENB BACKUP CACHE INIT H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;11&gt; in this register.</p>
<b>42</b>	<p><b>DIS BLOCK WRITE AROUND H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;10&gt; in this register.</p>
<b>41</b>	<p><b>ENB BACKUP CACHE COND I/O UPDATES H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;9&gt; in this register.</p>
<b>40</b>	<p><b>ENB EDC CHK H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;8&gt; in this register.</p>
<b>39</b>	<p><b>ENB EDC CORRECTION H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;7&gt; in this register.</p>
<b>38</b>	<p><b>ENB COR ERR INTERRUPT H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;6&gt; in this register.</p>
<b>37</b>	<p><b>FILL WRONG DUP TAG STORE PAR H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;5&gt; in this register.</p>
<b>36</b>	<p><b>FILL WRONG CONTROL PAR H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;4&gt; in this register.</p>
<b>35</b>	<p><b>FILL WRONG TAG PAR H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;3&gt; in this register.</p>
<b>34</b>	<p><b>ENB TAG &amp; DUP TAG PAR CHK H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit must always match bit &lt;2&gt; in this register.</p>
<b>33</b>	<p><b>FORCE FILL SHARED H</b> <i>[read/write]</i></p>

(continued on next page)

## 5.4 Backup Cache Control Register Definitions

**Table 5–2 (Cont.) Backup Cache Control Register Description**

Field	Description
	This bit is cleared on power-up. This bit must always match bit <1> in this register.
<b>32</b>	<b>ENB ALLOCATE H</b> <i>[read/write]</i>
	This bit is cleared on power-up. This bit must always match bit <0> in this register.
<b>31:30</b>	<b>CACHE SIZE</b> <i>[read/write]</i>
	These bits are cleared on power-up. These bits must be correctly set before the backup cache is accessed.
	<ul style="list-style-type: none"> <li>• 11 4 MB cache</li> <li>• 10 4 MB cache</li> <li>• 01 1 MB cache</li> <li>• 00 512 KB cache</li> </ul>
	The value of this field controls not only which tag signals are used in determining the proper parity value for the tag, but also in determining whether address bits 21,20, and 19 should be driven by the C <sup>3</sup> during access to the backup cache.
<b>29:16</b>	<b>EDC L</b> <i>[read/write]</i>
	These bits are cleared on power-up. When the FORCE EDC/CONTROL bit 12 is set during backup cache initialization, the values specified into this register are written forced on the EDC field of longwords 2 and 0 of any filled backup cache locations. See Section 11.2 for details.
<b>15:13</b>	<b>SHARED, DIRTY, VALID</b> <i>[read/write]</i>
	These bits are cleared on power-up. When the FORCE EDC/CONTROL bit is set during backup cache initialization, the values specified in SHARED, DIRTY, and VALID are filled into the control field of the backup cache location referenced by a READ_BLOCK. See Section 11.2 for details. When FORCE EDC/CONTROL is cleared, these bits have no effect.
<b>12</b>	<b>FORCE EDC/CONTROL</b> <i>[read/write]</i>
	This bit is cleared on power-up. The state of bits <8,7,5:2> should be checked when setting this bit to avoid machine check responses. When set, the appropriate backup cache location (indicated by address bits <19:5>) is updated for every processor READ_BLOCK cycle to a cacheable location.
	<ol style="list-style-type: none"> <li>1. The Tag Probe at READ_BLOCK address is forced clean, to avoid victims.</li> <li>2. The value of address bits &lt;30:19&gt; with its associated parity is filled into the backup cache tag store.</li> <li>3. The value of the control bits specified in this register's SHARED, DIRTY, and VALID and their associated parity are filled into the backup cache control store. The EDC bits specified in EDC H and L are filled into the EDC field of the data store.</li> <li>4. The backup cache is updated with the data returned on the system bus or the CSR data if the ENB BACKUP CACHE INIT bit &lt;11&gt; is set. This data is also returned to the processor to satisfy the READ BLOCK request.</li> </ol>
	This bit does not clear itself. See Section 11.2 for details.
<b>11</b>	<b>ENB BACKUP CACHE INIT</b> <i>[read/write]</i>

(continued on next page)

## 5.4 Backup Cache Control Register Definitions

**Table 5–2 (Cont.) Backup Cache Control Register Description**

Field	Description
	<p>This bit is cleared on power-up. When set, read transactions to memory address space are addressed as follows:</p> <p style="padding-left: 40px;">1.0000.0000 - 1.7FFF.FFFF for CPU<sub>0</sub> 1.8000.0000 - 1.FFFF.FFFF for CPU<sub>1</sub></p> <p>The reads are forced to return data from the corresponding CPU's CSR8-CBEAL register.</p> <p>When set, memory write allocate transactions are not supported, however CSR write transactions are supported. This causes the transactions to the system bus reserved memory address space to supply a known pattern, which is the transaction address as it appears on the system bus, to fill into the backup cache. For a CPU commander, the CSR data is filled to the backup cache tag and data entry indexed by the transaction physical address as follows :</p> <p style="padding-left: 40px;">Index    tag filled with address bits &lt;18:5&gt; &lt;30:19&gt;    for 512 KB cache &lt;19:5&gt; &lt;30:20&gt;    for 1 MB cache &lt;21:5&gt; &lt;30:22&gt;    for 4 MB cache</p> <p>Also, when this bit is set, the tag probe at the transaction address is forced clean to avoid victims, and if bits &lt;32&gt; and &lt;0&gt; ENB ALLOCATE are also set the backup cache tag control bits are filled with the value specified in bits &lt;15:13&gt; of this register correct parity unless parity is forced bad, regardless of the state of bit &lt;12&gt; and the returned data is written into the backup cache data store.</p> <p>See Section 11.2 for the actual returned data format.</p>
<b>10</b>	<p><b>DIS BLOCK WRITE AROUND</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When clear, only masked processor writes result in allocation. Full block unmasked writes write around the backup cache and do not result in allocation. When set, processor writes result in allocation regardless of the block mask.</p>
<b>9</b>	<p><b>ENB BACKUP CACHE COND I/O UPDATES</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, conditional updates of the backup cache occur due to system bus writes when the I/O module is the commander. A system bus write causes an update if both the contents of the duplicate tag store and the backup cache indicate a hit. When clear, Unconditional updates of the backup cache occurs when a system bus write hits a VALID backup cache location when the I/O module is the system bus commander. See Chapter 8 and Section 5.6 for details about conditional invalidation and updating.</p>
<b>8</b>	<p><b>ENB EDC CHK</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, the EDC of the BACKUP CACHE Data Store and processor written data is checked. EDC is checked by the backup cache controller only when (1) a system bus read or exchange hits dirty, (2) a masked write to a shared location occurs, or (3) during a victim write. When clear, checking for single and multiple bit errors is disabled, but EDC generation still occurs.</p>
<b>7</b>	<p><b>ENB EDC CORRECTION</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up to disable EDC correction of data during any transaction. Correctable single bit errors are reported as uncorrectable. When set, enables EDC correction of processor write data, victim data, and dirty read hit data.</p>
<b>6</b>	<p><b>ENB COR ERR INTERRUPT</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, EDC correctable errors detected during any transaction that is logged by CSR1 results in the assertion of the HARDWARE ERROR INTERRUPT. When clear, the interrupt is not sent but error information is captured in CSR1.</p>
<b>5</b>	<p><b>FILL WRONG DUP TAG STORE PAR</b> <i>[read/write]</i></p>

(continued on next page)

## 5.4 Backup Cache Control Register Definitions

Table 5–2 (Cont.) Backup Cache Control Register Description

Field	Description
	<p>This bit is cleared on power-up. When set, the wrong parity value is written in the primary data cache duplicate tag store parity location when a primary data cache location is updated during a processor primary data cache allocation.</p> <p>This bit is NOT self clearing. When this bit is set, the ENB TAG &amp; DUP TAG PAR CHK bits 34 and 2 should be cleared; otherwise a HARDWARE ERROR interrupt is signaled every time the duplicate tag store is written.</p>
<b>4</b>	<p><b>FILL WRONG CONTROL PAR</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, wrong parity is forced on the backup cache tag control store during the next backup cache allocation/updates (system bus read from this processor or write accept). This bit is self clearing.</p>
<b>3</b>	<p><b>FILL WRONG TAG PAR</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, wrong parity is forced on the backup cache tag address store during the next backup cache allocation/update (system bus read from this processor or write accept). This bit is self clearing.</p>
<b>2</b>	<p><b>ENB TAG &amp; DUP TAG PAR CHK</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, it enables parity checking of backup cache tag address, tag control, and duplicate tag memory contents whenever the backup cache controller accesses the backup cache and/or the duplicate tag store of the primary cache. When clear, checking of tag address, tag control, and duplicate tag address parity is ignored.</p>
<b>1</b>	<p><b>FORCE FILL SHARED</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, forces the setting of the SHARED bit in the Tag Control Store when a new entry is allocated in the backup cache.</p>
<b>0</b>	<p><b>ENB ALLOCATE</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, it enables the filling of the backup cache when cacheable cycles occur. When clear, backup cache allocation is disabled, but victimization of dirty cache lines still occurs. This means that the cache line associated with the read/write request is written to memory if dirty and then marked invalid; or if not dirty then no change in the cache line state occurs. In either case the processor is informed not to cache the data in either internal cache.</p> <p>The backup cache provides data to the system bus when read cycles probe dirty, system bus write accepting continues. Invalidation of the primary cache due to system bus writes still occurs. Because coherence is maintained, flushing the primary and backup caches is not necessary before re-enabling them. This bit can be cleared by writing or by any backup cache error reported by CSR3 bits 1 or 3. This bit cannot be set if CSR3 bits 1 or 3 are set.</p>

### 5.4.2 Backup Cache Correctable Error Register (CSR1)

The backup cache correctable error register latches the state of the backup cache tag and control stores when a correctable EDC error (during the data portion of the cycle) is detected. The contents of backup cache correctable error address register are not updated while error flags are set. Lost error flags do not inhibit error logging.

These errors are detected only as a result of a processor masked write hit to a shared location, victimization of a cache location, or a system bus read or exchange to a dirty location.

The backup cache correctable error register for CPU<sub>0</sub> is located at address 2.0000.0020<sub>16</sub>, and for CPU<sub>1</sub> at address 2.0800.0020<sub>16</sub>.

## 5.4 Backup Cache Control Register Definitions

Figure 5–3 Backup Cache Correctable Error Register (BCCE)

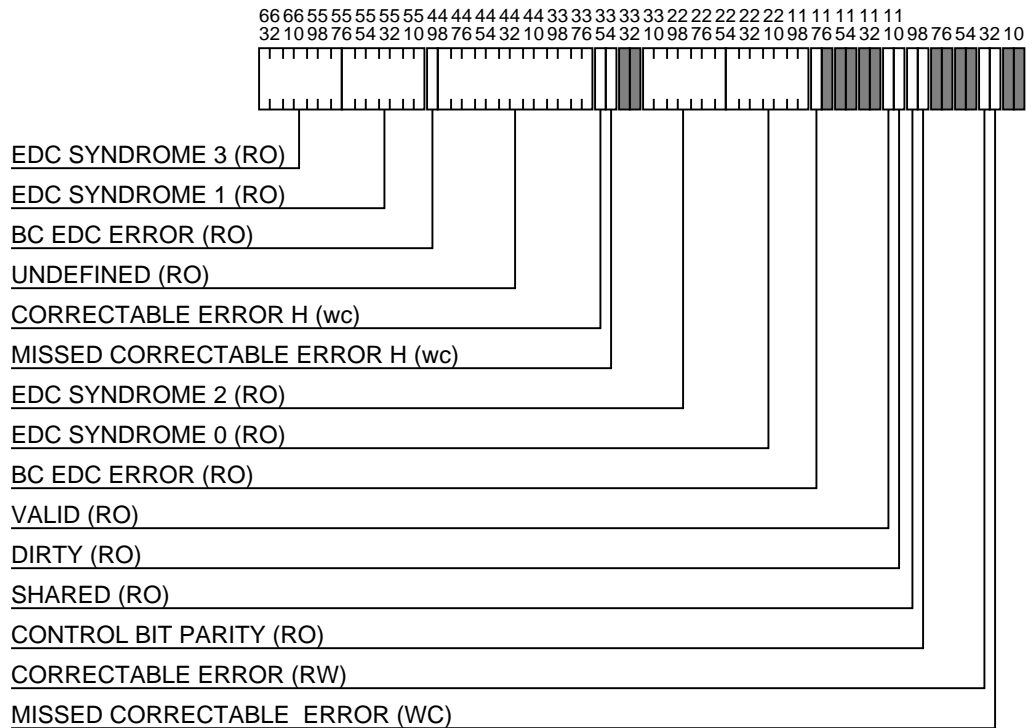


Table 5–3 Backup Cache Correctable Error Register Description

Field	Description
<b>63:57</b>	<b>EDC SYNDROME 3</b> <i>[read-only]</i> This bit is undefined on power-up. EDC SYNDROME 3 is valid when a correctable error has occurred. This register is updated when the BCCE ERROR H bit is clear. The syndrome contained in this register is relevant to longword 3 of data. See Table 4–21 for the single bit error syndrome list.
<b>56:50</b>	<b>EDC SYNDROME 1</b> <i>[read-only]</i> This bit is undefined on power-up. EDC SYNDROME 1 is Valid when a correctable error has occurred. This register is updated when the BCCE ERROR H bit is clear. The syndrome contained in this register is relevant to longword 1 of data. See Table 4–21 for the single bit error syndrome list.
<b>49</b>	<b>BC EDC ERROR</b> <i>[read-only]</i> This bit is undefined on power-up. It is valid when a correctable error has occurred indicating that the CPU or the backup cache data was the cause of the error. When set, indicates the backup cache was the cause of the data error. This register is updated when the BCCE ERROR H bit is clear.
<b>48:36</b>	<b>UNDEFINED</b> <i>[read-only]</i> Undefined
<b>35</b>	<b>CORRECTABLE ERROR H</b> <i>[read/write 1 to clear]</i>

(continued on next page)

## 5.4 Backup Cache Control Register Definitions

Table 5–3 (Cont.) Backup Cache Correctable Error Register Description

Field	Description
	This bit is cleared on power-up. This must be cleared when CORRECTABLE ERROR is cleared. This bit is not set if a single bit error occurs when EDC correction is disabled. (The UNCORRECTABLE ERROR bit is set in CSR3.) Write 1 to clear.
<b>34</b>	<b>MISSED CORRECTABLE ERROR H</b> <i>[read/write 1 to clear]</i> This bit is cleared on power-up. It must be cleared when MISSED CORRECTABLE ERROR is cleared. Write 1 to clear.
<b>31:25</b>	<b>EDC SYNDROME 2</b> <i>[read-only]</i> This bit is undefined on power-up. It is valid when a correctable error has occurred. This register is updated when the BCCE ERROR bit is clear. The syndrome contained in this register is relevant to longword 2 of data. See Table 4–21 for the single bit error syndrome list.
<b>24:18</b>	<b>EDC SYNDROME 0</b> <i>[read-only]</i> This bit is undefined on power-up. It is valid when a correctable error has occurred. This register is updated when the BCCE ERROR bit is clear. The syndrome contained in this register is relevant to longword 0 of data. See Table 4–21 for the single bit error syndrome list.
<b>17</b>	<b>BC EDC ERROR</b> <i>[read-only]</i> This bit is undefined on power-up. It is valid when a correctable error has occurred indicating that the CPU or the backup cache data was the cause of the error. When set, it indicates backup cache was the cause of the data error. This register is updated when the BCCE ERROR bit is clear.
<b>11</b>	<b>VALID</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of the VALID bit for the last backup cache location accessed by the backup cache controller. This register is updated when the BCCE ERROR bit is clear.
<b>10</b>	<b>DIRTY</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of the DIRTY bit for the last backup cache location accessed by the backup cache controller. This register is updated when the BCCE ERROR bit is clear.
<b>9</b>	<b>SHARED</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of the SHARED bit for the last backup cache location accessed by the backup cache controller. This register is updated when the BCCE ERROR bit is clear.
<b>8</b>	<b>CONTROL BIT PARITY</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of control bit parity for the last backup cache location accessed by the backup cache controller. This register is updated when the BCCE ERROR bit is clear.
<b>3</b>	<b>CORRECTABLE ERROR</b> <i>[read/write]</i> This bit is cleared on power-up. It is set when a correctable EDC error occurs in the backup cache causing the contents of this register (non-error bits) and the contents of the backup cache correctable error address register to freeze. This bit is not set if a single bit error occurs when EDC correction is disabled. (The UNCORRECTABLE ERROR bit is set in CSR3. The UNCORRECTABLE ERROR bit will be set in CSR3. ) Write 1 to clear.
<b>2</b>	<b>MISSED CORRECTABLE ERROR</b> <i>[read/write 1 to clear]</i>

(continued on next page)

## 5.4 Backup Cache Control Register Definitions

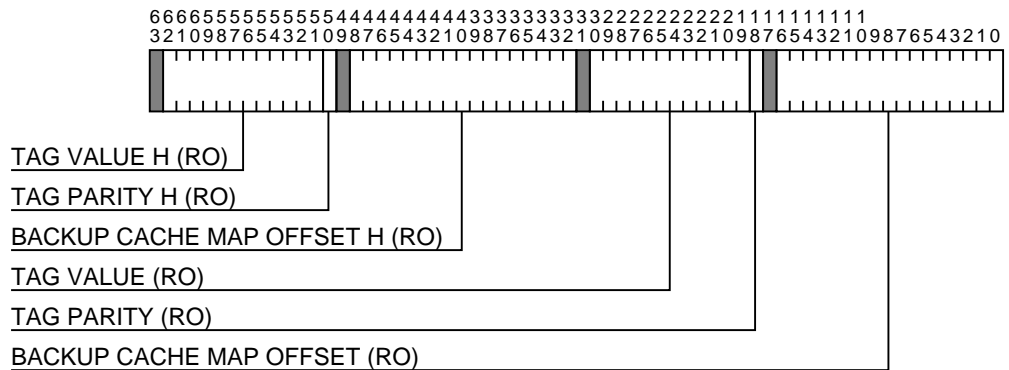
**Table 5–3 (Cont.) Backup Cache Correctable Error Register Description**

Field	Description
	This bit is cleared on power-up. It is set when a correctable EDC error occurs in the backup cache and CORRECTABLE ERROR bit is set from a previous error. A single cache block logs one error only, even if both octawords of the cache block hexaword are bad. An error in a single cache block does not cause the MISSED CORRECTABLE ERROR bit to be set. Write 1 to clear.

### 5.4.3 Backup Cache Correctable Error Address Register (BCCEA) CSR2

When a backup cache correctable EDC error is detected, this register contains the index of the backup cache location that contains the error. The backup cache correctable error address register for CPU<sub>0</sub> is located at address 2.0000.0040<sub>16</sub>, and for CPU<sub>1</sub> at address 2.0800.0040<sub>16</sub>.

**Figure 5–4 Backup Cache Correctable Error Address Register (BCCEA)**



**Table 5–4 Backup Cache Correctable Error Address Register Description**

Field	Description
<b>62:51</b>	<b>TAG VALUE H</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of the Tag for the last backup cache location accessed by the backup cache controller. This register is updated when the BCCE ERROR bit is clear. If LW 3 or LW 1 has a correctable EDC error, this field provides an address pointer to the backup cache.
<b>50</b>	<b>TAG PARITY H</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of Tag Store Parity for the last backup cache location accessed by the backup cache controller. This register is updated when the BCCE ERROR bit is clear.
<b>48:32</b>	<b>BACKUP CACHE MAP OFFSET H</b> <i>[read-only]</i> This field is cleared on power-up. It contains the last backup cache MAP index. If a correctable EDC error has been detected, the contents of this register are frozen until the BCCE ERROR bit in the BCCE register is cleared. If LW 3 or LW 1 has a correctable EDC error, this field provides an address pointer into the backup cache.
<b>30:19</b>	<b>TAG VALUE</b> <i>[read-only]</i>

(continued on next page)



## 5.4 Backup Cache Control Register Definitions

Table 5–4 (Cont.) Backup Cache Correctable Error Address Register Description

Field	Description
	This bit is cleared on power-up. It contains the value of the tag for the last backup cache location accessed by the backup cache controller. This register is updated when the BCCE ERROR bit is clear. If LW 2 or LW 0 has a correctable EDC error, this field provides an address pointer to the backup cache.
<b>18</b>	<b>TAG PARITY</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of Tag Store Parity for the last backup cache location accessed by the backup cache controller. This register is updated when the BCCE ERROR bit is clear.
<b>16:0</b>	<b>BACKUP CACHE MAP OFFSET</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the last backup cache MAP index. If a correctable EDC error has been detected, the contents of this register are frozen until the BCCE ERROR bit in the BCCE Register has been cleared. If LW 2 or LW 0 has a correctable EDC error, this field provides an address pointer into the backup cache.

### 5.4.4 Backup Cache Uncorrectable Error Register (CSR3)

This register latches the state of the backup cache tag and control stores when a parity error or an EDC uncorrectable error (during the data portion of the cycle) is detected. The contents of backup cache uncorrectable error address register are not updated while error flags are set, the lost error flag does not inhibit error logging.

These errors are detected only by the backup cache controller as a result of (1) a processor masked write hit to a shared location, a LDxL or STxC, victimization of a cache location, or (2) a system bus read, write, or exchange transaction. EDC errors are not detected by the C<sup>3</sup> on LDxL, and system bus write transactions as a bystander or responder.

The backup cache uncorrectable error register for CPU<sub>0</sub> is located at address 2.0000.0060<sub>16</sub>, and for CPU<sub>1</sub> at address 2.0800.0060<sub>16</sub>.

## 5.4 Backup Cache Control Register Definitions

Figure 5–5 Backup Cache Uncorrectable Error Register (BCUE)

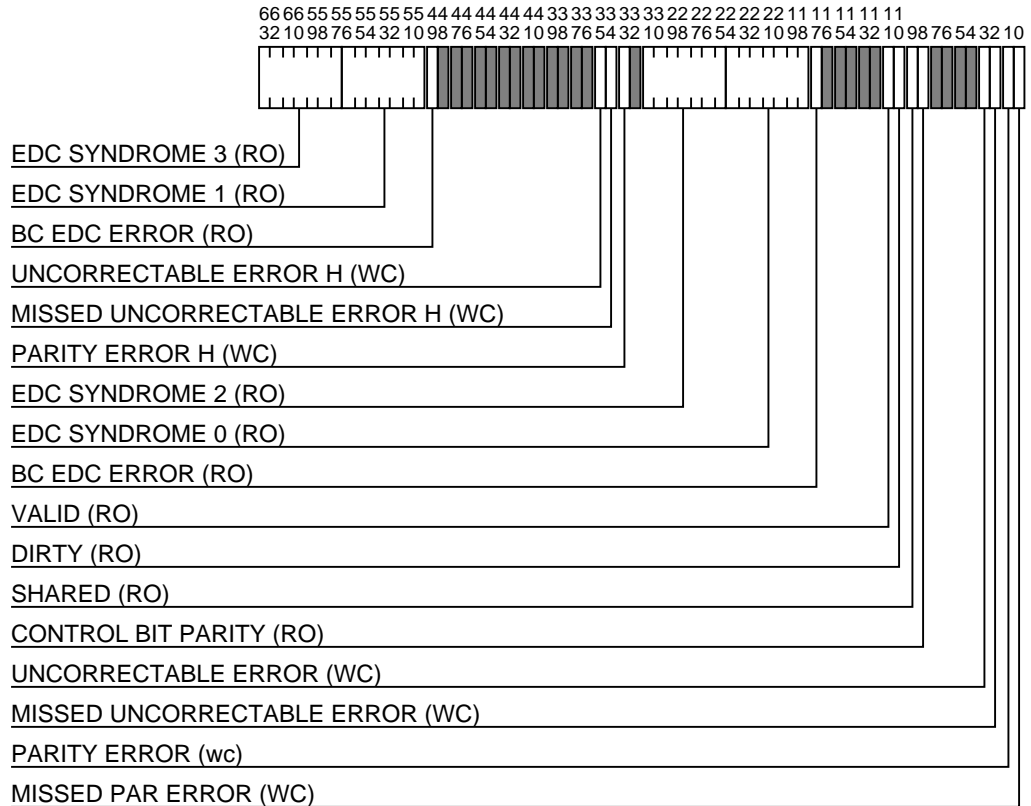


Table 5–5 Backup Cache Uncorrectable Error Register Description

Field	Description
<b>63:57</b>	<b>EDC SYNDROME 3</b> <i>[read-only]</i> Undefined on power-up. EDC SYNDROME 3 is valid when an uncorrectable error has occurred. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR H bits are clear. The syndrome contained in this register is relevant to longword 3 of data.
<b>56:50</b>	<b>EDC SYNDROME 1</b> <i>[read-only]</i> This field is undefined on power-up. EDC SYNDROME 1 is valid when a uncorrectable error has occurred. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR H bits are clear. The syndrome contained in this register is relevant to longword 1 of data.
<b>49</b>	<b>BC EDC ERROR</b> <i>[read-only]</i> This bit is undefined on power-up. It is valid when an uncorrectable error has occurred indicating that the CPU or the backup cache data was the cause of the error. When set, indicates the backup cache was the cause of the data error. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR H bits are clear.
<b>35</b>	<b>UNCORRECTABLE ERROR H</b> <i>[read/write 1 to clear]</i> This bit is cleared on power-up. This bit must be cleared when UNCORRECTABLE ERROR is cleared. Write 1 to clear.
<b>34</b>	<b>MISSED UNCORRECTABLE ERROR H</b> <i>[read/write 1 to clear]</i>

(continued on next page)

## 5.4 Backup Cache Control Register Definitions

**Table 5–5 (Cont.) Backup Cache Uncorrectable Error Register Description**

Field	Description
	This bit is cleared on power-up. It must be cleared when MISSED UNCORRECTABLE ERROR is cleared. Write 1 to clear.
<b>33</b>	<b>PARITY ERROR H</b> <i>[read/write 1 to clear]</i> This bit is cleared on power-up. It must be cleared when PAR ERROR is cleared. Write 1 to clear.
<b>31:25</b>	<b>EDC SYNDROME 2</b> <i>[read-only]</i> This field is undefined on power-up. It is valid when an uncorrectable error has occurred. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear. The syndrome contained in this register is relevant to longword 2 of data.
<b>24:18</b>	<b>EDC SYNDROME 0</b> <i>[read-only]</i> This field is undefined on power-up. It is valid when an uncorrectable error has occurred. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear. The syndrome contained in this register is relevant to longword 0 of data.
<b>17</b>	<b>BC EDC ERROR</b> <i>[read-only]</i> This bit is undefined on power-up. It is valid when an uncorrectable error has occurred indicating that the CPU or the backup cache data was the cause of the error. When set, it indicates that the backup cache was the cause of the data error. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear.
<b>11</b>	<b>VALID</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of the VALID bit for the last backup cache location accessed by the backup cache controller. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear.
<b>10</b>	<b>DIRTY</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of the dirty bit for the last backup cache location accessed by the backup cache controller. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear.
<b>9</b>	<b>SHARED</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of the SHARED bit for the last backup cache location accessed by the backup cache controller. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear.
<b>8</b>	<b>CONTROL BIT PARITY</b> <i>[read-only]</i> This bit is cleared on power-up. It contains the value of control bit parity for the last backup cache location accessed by the backup cache controller. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear.
<b>3</b>	<b>UNCORRECTABLE ERROR</b> <i>[read/write 1 to clear]</i> This bit is cleared on power-up. It is set when an uncorrectable EDC error occurs in the backup cache which causes the contents of this register's (non-error bits) and the contents of the backup cache uncorrectable error address register to freeze. Write 1 to clear. When this bit is set, CSRI bit 0 is cleared, to stop allocation.
<b>2</b>	<b>MISSED UNCORRECTABLE ERROR</b> <i>[read/write 1 to clear]</i> This bit is cleared on power-up. It is set when an uncorrectable EDC error occurs in the backup cache and UNCORRECTABLE ERROR or PAR ERROR bits are set from a previous error. A single cache block logs only one error even if both octawords of the cache block hexaword are bad. An error in a single cache block does not cause the MISSED UNCORRECTABLE ERROR bit to be set. Write 1 to clear.
<b>1</b>	<b>PARITY ERROR</b> <i>[read/write 1 to clear]</i>

(continued on next page)

## 5.4 Backup Cache Control Register Definitions

**Table 5–5 (Cont.) Backup Cache Uncorrectable Error Register Description**

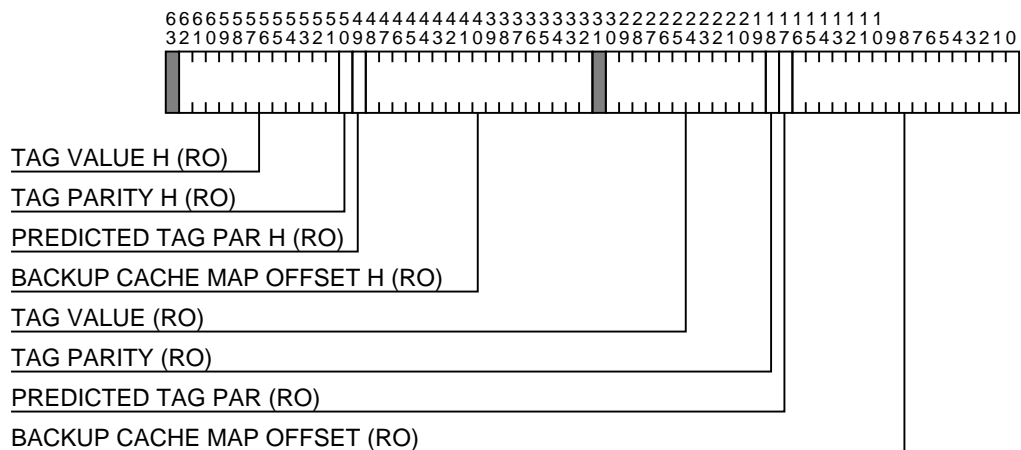
Field	Description
	This bit is cleared on power-up. When set, it indicates that the BCUE register contains information about a backup cache tag error or control store parity error. Setting this bit causes the contents of this register's (non-error bits) and the contents of the backup cache uncorrectable error address register to freeze. Write 1 to clear. When this bit is set, CSR1 bit 0 is cleared, stopping allocation and disabling parity checking on the backup cache tag address and control storage elements during local CPU initiated reads and writes.
<b>0</b>	<b>MISSED PAR ERROR</b> <i>[read/write 1 to clear]</i> This bit is cleared on power-up. When set, it indicates that a parity error occurred in the tag or tag control store while the PARITY ERROR or the UNCORRECTABLE ERROR bit was set. Write 1 to clear.

### 5.4.5 Backup Cache Uncorrectable Error Address Register (CSR4)

This register contains the index of the backup cache location containing the error when one of the following occur: (1) a backup cache tag store or control store parity error, or (2) an uncorrectable EDC error has been detected.

The backup cache uncorrectable error address register for CPU<sub>0</sub> is located at address 2.0000.0080<sub>16</sub>, and for CPU<sub>1</sub> at address 2.0800.0080<sub>16</sub>.

**Figure 5–6 Backup Cache Uncorrectable Error Address Register (BCUEA)**



**Table 5–6 Backup Cache Uncorrectable Error Address Register Description**

Field	Description
<b>62:51</b>	<b>TAG VALUE H</b> <i>[read-only]</i> This field is cleared on power-up. It contains the value of the tag for the last backup cache location accessed by the backup cache controller. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear. If LW 3 or LW 1 has an uncorrectable EDC error, this field provides an address pointer into the backup cache.

(continued on next page)

## 5.4 Backup Cache Control Register Definitions

**Table 5–6 (Cont.) Backup Cache Uncorrectable Error Address Register Description**

Field	Description
<b>50</b>	<p><b>TAG PARITY H</b> <i>[read-only]</i></p> <p>This bit is cleared on power-up. It contains the value of Tag Store Parity for the last backup cache location accessed by the backup cache controller. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear.</p>
<b>49</b>	<p><b>PREDICTED TAG PAR H</b> <i>[read-only]</i></p> <p>This bit is cleared on power-up. It contains the value of the last system bus predicted tag parity. The value of this bit is frozen if any of the following occur:</p> <ul style="list-style-type: none"> <li>• A parity error occurs in the tag, control store, or an EDC error is detected</li> <li>• A victim write</li> <li>• A masked write to a shared location</li> <li>• An LDxL or STxC, victimization of a cache location</li> <li>• A system bus read, write or exchange</li> </ul> <p>This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear. This provides power-up diagnostics with visibility to an internal parity tree. The value of this location is valid if the register is frozen and the cycle causing the error was a system bus read or write or if the register is not frozen and the register is read. (The even parity of the address bits &lt;33:20&gt; during the CSR read access are found in this location.) This bit should be disregarded during normal system operation.</p>
<b>48:32</b>	<p><b>BACKUP CACHE MAP OFFSET H</b> <i>[read-only]</i></p> <p>This field is cleared on power-up. It contains the last backup cache MAP index (address bits &lt;21:5&gt; for 1 MB cache bits 19:5 indicate cache block offset). If a parity error occurs in the tag, or control bits, or an EDC error has been detected, the contents of this register are frozen until the BCUE UNCORRECTABLE and PARITY ERROR bits are clear. If LW 3 or LW 1 has an uncorrectable EDC error this field provides an address pointer into the backup cache.</p>
<b>30:19</b>	<p><b>TAG VALUE</b> <i>[read-only]</i></p> <p>This field is cleared on power-up. It contains the value of the Tag for the last backup cache location accessed by the backup cache controller. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear. If LW 2 or LW 0 has an uncorrectable EDC error this field provides an address pointer into the backup cache.</p>
<b>18</b>	<p><b>TAG PARITY</b> <i>[read-only]</i></p> <p>This bit is cleared on power-up. It contains the value of the tag store parity for the last backup cache location accessed by the backup cache controller. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear.</p>
<b>17</b>	<p><b>PREDICTED TAG PAR</b> <i>[read-only]</i></p> <p>This bit is cleared on power-up. It contains the value of the last system bus predicted tag parity. The value of this bit is frozen if a parity error is detected in the tag, tag control store, or an EDC error is detected, when a system bus read, write or exchange probe occurs. This register is updated when the BCUE UNCORRECTABLE and PARITY ERROR bits are clear.</p> <p>This provides power-up diagnostics the visibility to an internal parity tree. The value of this location is valid if the register is frozen and the cycle causing the error was a system bus read or write (the EVEN parity of the address bits &lt;30:20&gt; for 1-MB cache). This bit should be disregarded during normal system operation.</p>
<b>16:0</b>	<p><b>BACKUP CACHE MAP OFFSET</b> <i>[read-only]</i></p>

(continued on next page)

## 5.4 Backup Cache Control Register Definitions

**Table 5–6 (Cont.) Backup Cache Uncorrectable Error Address Register Description**

Field	Description
	This field is cleared on power-up. It contains the last backup cache MAP index. If a parity error occurs in the tag, or control bits, or an EDC error has been detected, the contents of this register are frozen until the BCUE UNCORRECTABLE and PARITY ERROR bits are clear. If LW 2 or LW 0 has an uncorrectable EDC error, this field provides an address pointer into the backup cache.

### 5.4.6 System Bus Cycles

The backup cache is single ported, and as such the system bus and the processor must contend for its ownership. The arbitration algorithm provides top priority to the system bus. Whenever a system bus transaction is requested by a commander, every backup cache on the system bus (maximum of 2 in a DEC 4000 dual processor system) is unavailable to its processor for the number of system bus cycles shown in Table 5–7. The system bus guarantees that a processor is not held off its own cache for any more than two consecutive system bus WRITE cycles.

**Table 5–7 System Bus Backup Cache Access Time**

System Bus Transaction	Access Time (system bus cycles)†
Read Hit Dirty	6
Read Hit Clean	5
Read Miss	3
Write Hit	5
Write Miss	3
Exchange Hit Dirty	6
Exchange Hit Clean	5
Exchange Miss	3

†The cycle time of the system bus is 24 ns.

## 5.5 Cache Block Merge Buffer

The cache block merge buffer is a CPU- specific buffer used to prevent word tearing caused by simultaneous writes to different words in the same cache block. It is also used to form complete cache blocks of masked writes when the backup cache is disabled.

The merge buffer consists of a 32-byte data buffer, a valid bit associated with each longword in the buffer, and a complete address tag.

When a masked write miss occurs, the data longwords indicated by the 21064 CPU longword mask are written into the Cache Block Merge Buffer and the associated valid bits are set. This miss is handled like other misses in that a system bus read is performed to allocate the cache entry. As the read data is returned off the system bus, it is merged with the data already in the merge buffer and written into the backup cache data store.

## 5.5 Cache Block Merge Buffer

If an intervening system bus write occurs to that same cache entry (by a different system bus commander), the valid longwords in the merge buffer are merged with the new data provided by that system bus write cycle, and subsequent system bus write (if that datum is now shared) provides the newly merged data to the system bus.

## 5.6 Duplicate Primary Data Cache Tag Store

Every DEC 4000 processor module provides a mechanism to perform “conditional invalidation” of the cache memory subsystems. This is accomplished through the use of a copy of the primary data cache tag store. As updates to cacheable locations occur, invalidation of the internal data cache is 100% accurate. The duplicate tag store also provides a means to “selectively” accept updates to the backup cache. This allows optimal system performance for accesses to truly shared locations, however it protects the integrity of the write-back system by invalidating “shared cold” locations caused by phenomena such as process migration.

The general Update versus Invalidate algorithm when a system bus write cycle occurs is shown in Example 5-1

## 5.6 Duplicate Primary Data Cache Tag Store

### Example 5–1 Update versus Invalidate Algorithm

```
IF (Waiting in ARB for system bus)
{
  IF (HIT backup cache)
  {
    update_backup cache_entry();
  }

  IF (HIT primary data cache)
  {
    invalidate_primary_data_cache_entry();
  }
}
ELSE
{
  IF (system bus commander is a CPU OR ENB BACKUP CACHE COND I/O UPDATES (BCC Reg))
  {
    IF (HIT primary data cache)
    {
      invalidate_primary_data_cache_entry();
    }

    IF (HIT backup cache)
    {
      update_backup cache_entry();
    }
  }
  ELSE
  {
    IF (HIT backup cache)
    {
      invalidate_backup cache_entry();
    }
  }
}
ELSE
{
  IF (HIT primary data cache)
  {
    invalidate_primary_data_cache_entry();
  }

  IF (HIT backup cache)
  {
    update_backup cache_entry();
  }
}
}
```



## 5.6 Duplicate Primary Data Cache Tag Store

### 5.6.1 Duplicate Tag Error Register (CSR5)

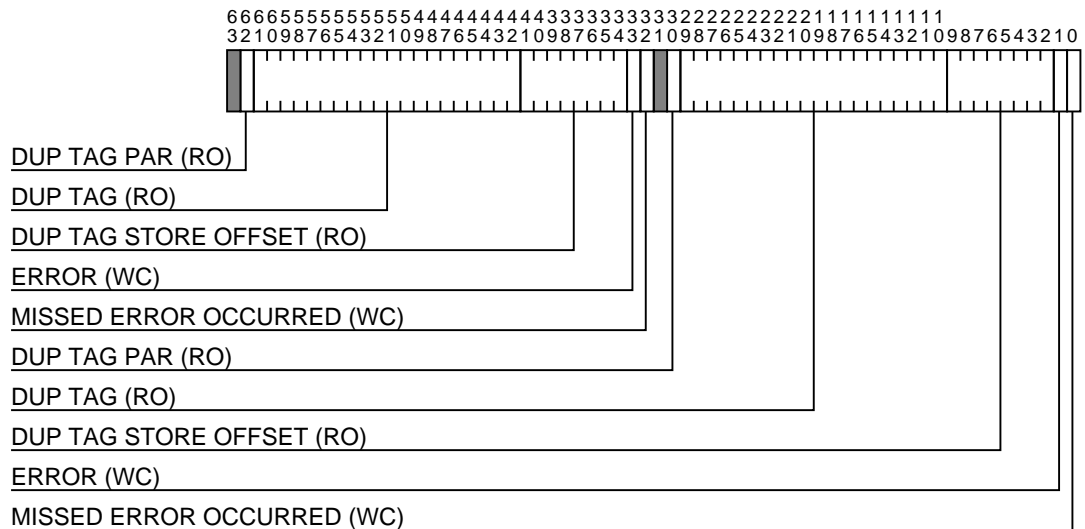
The duplicate tag error register is updated after each access to the duplicate tag store RAM in both slices. After a parity error is detected, the contents of the register are not updated while the error flag is set, the lost error flag does not inhibit error logging. The missed error bit in this register when set, suppresses further error interrupts from duplicate tag parity errors. When these parity errors are detected, invalidation of the primary and/or secondary caches result. Diagnostic Hint: filling the duplicate tag RAM with bad parity causes the coherence policy to degenerate from update to invalidate.

**Note**

The system software performs quadword writes to clear flags. This ensures that the state of error flags in the low longword is consistent with the state of control flags in the upper longword.

The duplicate tag error register for CPU<sub>0</sub> is located at address 2.0000.00A0<sub>16</sub>, and for CPU<sub>1</sub> at address 2.0800.00A0<sub>16</sub>.

**Figure 5–7 Duplicate Tag Error Register (DTER)**



**Table 5–8 Duplicate Tag Error Register Description**

Field	Description
62	<b>DUP TAG PAR</b> <i>[read-only]</i>

(continued on next page)

## 5.6 Duplicate Primary Data Cache Tag Store

**Table 5–8 (Cont.) Duplicate Tag Error Register Description**

Field	Description
	<p>This bit is undefined on power-up. It contains the most recent Primary data cache duplicate tag store parity. There are two cases to consider,</p> <ol style="list-style-type: none"> <li>1. probes from the system bus</li> <li>2. read misses from the processor</li> </ol> <p>If a parity error occurs due to a probe, this bit shows that bad parity as calculated from the DUP TAG&lt;61:42&gt; bits in this register and is a fatal error. If a parity error occurs due to a processor read miss which allocates into the duplicate tag, this bit shows the parity of the allocated address and does not reflect the calculated parity of DUP TAG&lt;61:42&gt;. The contents of this field are frozen until the ERROR bits have been cleared.</p> <p>This field is updated only when memory space references occur while the register is not frozen.</p>
<b>61:42</b>	<p><b>DUP TAG</b> <i>[read-only]</i></p> <p>This field is undefined on power-up. It contains the last primary cache duplicate tag. If a parity error occurs in the primary cache duplicate tag store, the contents of this register are frozen until the ERROR bits are cleared.</p> <p>This field is updated only when memory space references occur while the register is not frozen.</p>
<b>41:34</b>	<p><b>DUP TAG STORE OFFSET</b> <i>[read-only]</i></p> <p>This field is undefined on power-up. Contains the last duplicate tag store offset. If a parity error occurs in the duplicate tag store, the DUP TAG STORE OFFSET will remain for that cycle until the ERROR bits have been cleared.</p> <p>This field is updated only when memory space references occur while the register is not frozen.</p>
<b>33</b>	<p><b>ERROR</b> <i>[read/write 1 to clear]</i></p> <p>Cleared on power-up. Set when a primary data cache duplicate tag store parity error has been detected to hold the contents of this register until this flag is cleared. Write 1 to clear.</p>
<b>32</b>	<p><b>MISSED ERROR OCCURRED</b> <i>[read/write 1 to clear]</i></p> <p>Cleared on power-up. Set when a primary D-cache duplicate tag store parity error bit was set and another one is detected. When set logging in this CSR is not inhibited and further interrupts from this error are suppressed. Write 1 to clear.</p>
<b>30</b>	<p><b>DUP TAG PAR</b> <i>[read-only]</i></p> <p>This field is undefined on power-up. Contains the most recent primary D-cache duplicate tag store parity. There are two cases to consider, probes from the system bus and read misses from the processor. If a parity error occurs due to a probe, this bit shows that bad parity as calculated from the DUP TAG&lt;29:10&gt; bits in this register and is a fatal error.</p> <p>If a parity error occurs due to a processor read miss which allocates into the duplicate tag, this bit shows the parity of the allocated address and does not reflect the calculated parity of DUP TAG&lt;29:10&gt;. The contents of this field are frozen until the ERROR bits have been cleared.</p> <p>This field is updated only when memory space references occur while the register is not frozen.</p>
<b>29:10</b>	<p><b>DUP TAG</b> <i>[read-only]</i></p> <p>This field is undefined on power-up. It contains the last primary cache duplicate tag. If a parity error occurs in the primary cache duplicate tag store, the contents of this register are frozen until the ERROR bits have been cleared.</p> <p>This field is updated only when memory space references occur while the register is not frozen.</p>
<b>9:2</b>	<p><b>DUP TAG STORE OFFSET</b> <i>[read-only]</i></p>

(continued on next page)

## 5.6 Duplicate Primary Data Cache Tag Store

**Table 5–8 (Cont.) Duplicate Tag Error Register Description**

Field	Description
	This field is undefined on power-up. It contains the last duplicate tag store offset. If a parity error occurs in the duplicate tag store, the DUP TAG STORE OFFSET will remain for that cycle until the ERROR bits have been cleared.
	This field is updated only when memory space references occur while the register is not frozen.
<b>1</b>	<b>ERROR</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. It is set when a primary data cache duplicate tag store parity error is detected, to hold the contents of this register until this flag is cleared. Write 1 to clear.
<b>0</b>	<b>MISSED ERROR OCCURRED</b> <i>[read/write 1 to clear]</i> This bit is cleared on power-up. It is set when a primary data cache duplicate tag store parity error bit is set and another one is detected. When set, logging in this CSR is not inhibited and further interrupts from this error are suppressed. Write 1 to clear.

## 5.7 Lack of Duplicate Primary Instruction Cache Tag Store

Because the primary instruction cache is a virtual cache, all cache coherence is managed by the system software. Thus no hardware invalidates of the instruction cache are performed. Also, because very little writing to shared instruction stream locations occur, the benefit of “selective” updating is diminished. As such there is **no** duplicate primary instruction cache tag store on the DEC 4000 CPU module.

## 5.8 Lack of Cache Block Prefetch

Both FETCH and FETCHM instructions are handled in the same manner. When either instruction is issued, the processor is immediately released to continue executing.

## 5.9 Data Integrity

The DEC 4000 CPU module provides error detection mechanisms for its major data storage elements and buses. Table 5–9 provides an overview of the error detection mechanisms provided in DEC 4000 subsystems.

**Table 5–9 Data Integrity Reference**

Element	Protection Method	Reference
Primary I-cache tag Store	Parity	Appendix B
Primary I-cache data store	LW Parity	Appendix B
Primary D-cache tag Store	Parity	Appendix B
Primary D-cache data store	LW Parity	Appendix B
Backup cache control store	Parity	Section 5.1, Section 10.2.1
Backup cache tag store	Parity	Section 5.2, Section 10.3
Backup cache data store	EDC†	Section 5.3, Section 10.2.2
Duplicate tag store	Parity	Section 5.6, Section 10.3

†Single-bit correction, double bit detection

(continued on next page)

**Table 5–9 (Cont.) Data Integrity Reference**

Element	Protection Method	Reference
System bus	LW Parity	Chapter 19, Section 10.4
DEC 4000 processor data bus	EDC‡	Section 5.3, Section 10.2.2

‡Single-bit correction, double bit detection, Depending on the source of the data, this could include errors that occur in the backup cache data store as well as those occurring on the bus.

---

## System Bus Interface

The system bus interface is the DEC 4000 CPU module's "window to the world". All standard data processed by a DEC 4000 processor is obtained over the system bus from either a DEC 4000 memory module, the DEC 4000 I/O module, or another DEC 4000 CPU module. All system bus registers are visible to all system bus commanders. See Chapter 19.

### 6.1 CPU System Bus Register Definitions

#### 6.1.1 System Bus Control Register (CSR6)

The system bus control register provides a means of controlling the system bus arbitration and interface signals for diagnostic purposes, and contains the information required to support the WHOAMI requirement specified in the *Alpha Architecture Reference Manual*.

This register's arbitration control bits allow a system bus commander to selectively disallow other commander's ownership of the system bus. This register should be used only during system initialization; however it may also be useful in guarding the system bus from a failed second CPU module. Because the register can be accessed only through the system bus, a commander is not allowed to mask itself off as it would never be able to re-enable access to the system bus.

---

**Note**

---

The CPU ensures that the state of control flags in the low longword are consistent with control flags in the upper longword by performing quadword writes.

---

The system bus control register for CPU<sub>0</sub> is located at address 2.0000.00C0<sub>16</sub>, and for CPU<sub>1</sub> at address 2.0800.00C0<sub>16</sub>.

## 6.1 CPU System Bus Register Definitions

Figure 6–1 System Bus Control Register (CBCTL)

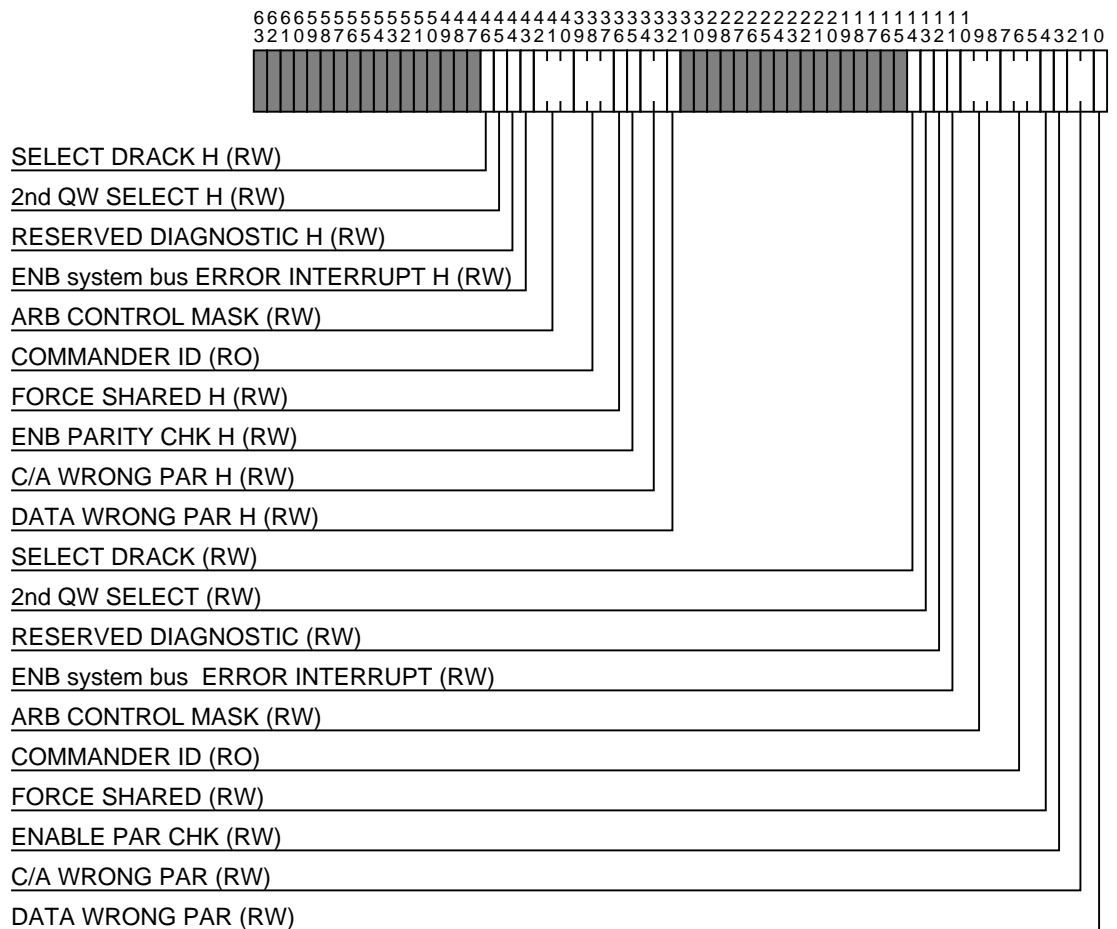


Table 6–1 System Bus Control Register Description

Field	Description
46	<b>SELECT DRACK H</b> <i>[read/write]</i> This field is cleared on power-up. This bit is reserved for timing configuration selection of the CPU response strobe DRACK<1>. When set, the DRACK<1> strobe timing is shifted by six nanoseconds. The state of this bit must match bit <14> in this register.
45	<b>2nd QW SELECT H</b> <i>[read/write]</i> This bit is cleared on power-up. This bit is reserved for timing configuration selection of the second quadword of returned data to the CPU chip. When set, the data mux control strobe is shifted by six nanoseconds. The state of this bit must match bit <13> in this register.
44	<b>RESERVED DIAGNOSTIC H</b> <i>[read/write]</i> This bit is cleared on power-up. It is reserved for diagnostic testing. When set, the behavior of the system bus arbiter is modified. The state of this bit must match bit <12> in this register.
43	<b>ENB system bus ERROR INTERRUPT H</b> <i>[read/write]</i>

(continued on next page)

## 6.1 CPU System Bus Register Definitions

**Table 6–1 (Cont.) System Bus Control Register Description**

Field	Description
	This bit is cleared on power-up. The state of this bit must match bit <11> in this register.
<b>42:40</b>	<p><b>ARB CONTROL MASK</b> <i>[read/write]</i></p> <p>Bits 40 and 41 are set on power-up and bit 42 is cleared on power-up. When the bit corresponding to a particular subsystem is set, that subsystem is granted the bus when requested. A processor cannot clear its own ARB control mask bit.</p> <ul style="list-style-type: none"> <li>• bit 42 set : I/O</li> <li>• bit 41 set : CPU<sub>1</sub></li> <li>• bit 40 set : CPU<sub>0</sub></li> </ul> <p>This field is non-functional in CPU<sub>1</sub>.</p>
<b>39:37</b>	<p><b>COMMANDER ID</b> <i>[read-only]</i></p> <p>This field identifies the CPU as being CPU<sub>0</sub> or CPU<sub>1</sub> based on the system bus commander ID. Writes have no effect. Regardless of being CPU<sub>0</sub> or CPU<sub>1</sub> performing a read of this CSR, the contents of this field are returned with the CPU ID of the CPU commanding the CSR read.</p> <p style="margin-left: 40px;">010 CPU<sub>1</sub> 001 CPU<sub>0</sub></p>
<b>36</b>	<p><b>FORCE SHARED H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, it asserts CSHARED_L on all system bus transactions.</p>
<b>35</b>	<p><b>ENB PARITY CHK H</b> <i>[read/write]</i></p> <p>This field is cleared on power-up. When set, the longword parity checking on the system bus for both the C/A and data portion of cycle (if responder) is enabled for C/A longwords 3 and 2 and data longwords 7,5,3 and 1.</p>
<b>34:33</b>	<p><b>C/A WRONG PAR H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, it forces wrong parity on longwords 3 and 1 respectively during the C/A portion of the next C/A cycle from this node to the system bus. Once this cycle has occurred, the C/A WRONG PAR bits are automatically cleared.</p>
<b>32</b>	<p><b>DATA WRONG PAR H</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. When set, it forces wrong parity on longwords 7,5,3 and 1 during both data portions of the next system bus cycle to which this node responds. Once this cycle has occurred the DATA WRONG PAR bit is automatically cleared. This should not be set when any of the C/A WRONG PAR bits are set, as a responder is not able to log both wrong C/A parity and wrong data parity in the same system bus transaction.</p>
<b>14</b>	<p><b>SELECT DRACK</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. It is reserved for timing configuration selection of the 21064 response strobe DRACK&lt;1&gt;. When set the DRACK&lt;1&gt; strobe timing is shifted by six nanoseconds.</p>
<b>13</b>	<p><b>2nd QW SELECT</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. It is reserved for timing configuration selection of the second quadword of returned data to the 21064 CPU. When set, the data MUX control strobe is shifted by six nanoseconds.</p>
<b>12</b>	<p><b>RESERVED DIAGNOSTIC</b> <i>[read/write]</i></p> <p>This bit is cleared on power-up. This bit is reserved for diagnostic testing. When set, the behavior of the system bus arbiter is modified to regulate the flow of back to back transactions.</p>

(continued on next page)

## 6.1 CPU System Bus Register Definitions

Table 6–1 (Cont.) System Bus Control Register Description

Field	Description
<b>11</b>	<b>ENB system bus ERROR INTERRUPT</b> <i>[read/write]</i> This field is cleared on power-up to disable the system bus C_ERR_L interrupt signal from being driven due to errors encountered by this node. This bit does not disable reception of this interrupt signal.
<b>10:8</b>	<b>ARB CONTROL MASK</b> <i>[read/write]</i> Bits 8 and 9 set, bit 10 cleared on power-up. When the bit corresponding to a particular subsystem is set, that subsystem is granted the bus when requested. A processor can not clear its own ARB Control Mask bit. <ul style="list-style-type: none"><li>• bit 10 set : I/O</li><li>• bit 9 set : CPU<sub>1</sub></li><li>• bit 8 set : CPU<sub>0</sub></li></ul> This field is non-functional in CPU <sub>1</sub> .
<b>7:5</b>	<b>COMMANDER ID</b> <i>[read-only]</i> Identifies the CPU as being CPU <sub>0</sub> or CPU <sub>1</sub> based on the system bus commander ID. Writes have no effect. Regardless of being CPU <sub>0</sub> or CPU <sub>1</sub> performing a read of this CSR, the contents of this field are returned with the CPU ID of the CPU commanding the CSR read.  010 CPU <sub>1</sub> 001 CPU <sub>0</sub>
<b>4</b>	<b>FORCE SHARED</b> <i>[read/write]</i> This field is cleared on power-up. When set, asserts CSHARED_L on all system bus transactions.
<b>3</b>	<b>ENABLE PAR CHK</b> <i>[read/write]</i> This field is cleared on power-up. When set, longword parity checking on the system bus for both the C/A and data portion of cycle (if responder) is enabled for C/A longwords 1 and 0 and data longwords 2 and 0.
<b>2:1</b>	<b>C/A WRONG PAR</b> <i>[read/write]</i> This bit is cleared on power-up. When set, forces wrong parity on longwords 2 and 0 respectively during the C/A portion of the next C/A cycle from this node to the system bus. Once this cycle has occurred the C/A WRONG PAR bits are automatically cleared.
<b>0</b>	<b>DATA WRONG PAR</b> <i>[read/write]</i> This bit is cleared on power-up. When set, forces wrong parity on longwords 6, 4, 2, and 0 during both data portions of the next system bus cycle to which this node responds. Once this cycle has occurred the DATA WRONG PAR bit is automatically cleared. This should not be set when any of the C/A WRONG PAR bits are set as a responder is not able to log both wrong C/A parity and wrong data parity in the same system bus transaction.

### 6.1.2 System Bus Error Register (CSR7)

The system bus error register is updated every system bus cycle. Whenever a system bus error is detected, the contents of this register are frozen until the ERROR bits are cleared. The contents of the register are not updated while the error flags are set. The lost error flags do not inhibit error logging.

#### Note

Even though the CBE register is updated with the latest cycle on the system bus, the contents of this register in the non-error case is always the cycle that was issued to actually read the CBE register. This is



## 6.1 CPU System Bus Register Definitions

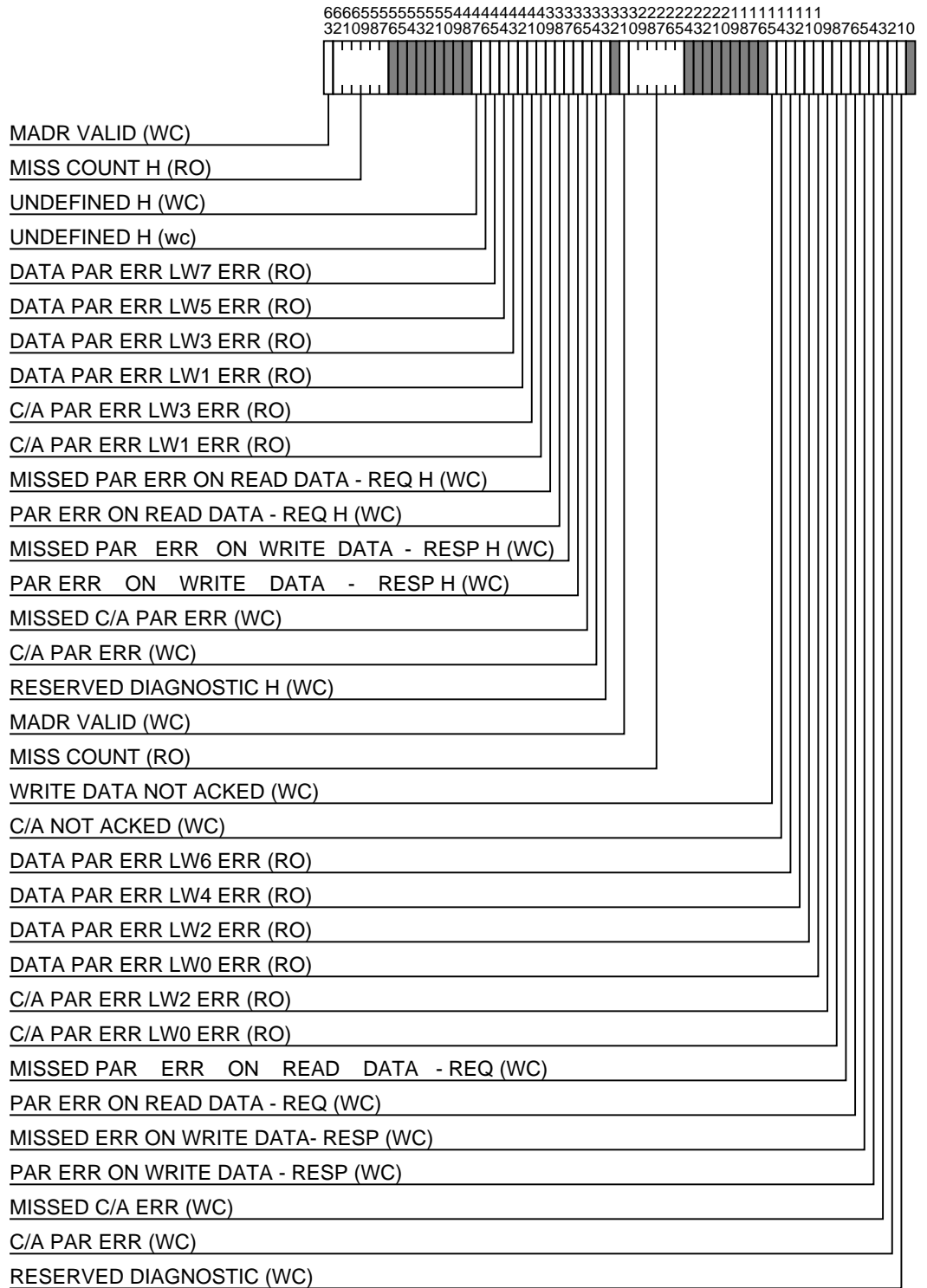
because the register is accessed via a system bus cycle. If a tag control or tag store parity error is detected then the C/A NOT ACKED and/or the WRITE DATA NOT ACKED bits are set.

---

The system bus error register for CPU<sub>0</sub> is located at address 2.0000.00E0<sub>16</sub>, and for CPU<sub>1</sub> at address 2.0800.00E0<sub>16</sub>.

## 6.1 CPU System Bus Register Definitions

Figure 6–2 System Bus Error Register (CBE)



## 6.1 CPU System Bus Register Definitions

**Table 6–2 System Bus Error Register Description**

Field	Description
<b>63</b>	<b>MADR VALID</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. It has the same function as bit <31>, but indicates another valid sample in MADRL <63:32> that occurs 32 miss counts offset from the sample in the other half of these registers. Effectively providing two samples every 64 misses. A copy of this bit is readable from bit 0 of the MADRL register. Write 1 to clear.
<b>62:57</b>	<b>MISS COUNT H</b> <i>[read-only]</i> This field is set to 000000 on power-up. The six bit miss address counter is readable in this field.
<b>47</b>	<b>UNDEFINED H</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. Undefined usage. Write 1 to clear.
<b>46</b>	<b>UNDEFINED H</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. Undefined usage. Write 1 to clear.
<b>45</b>	<b>DATA PAR ERR LW7 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on data longword 7 system bus bits <127:96>, during the second data cycle portion of a system bus transaction. DATA PAR ERR LW7 ERR remains valid until the Data ERR bits (<38> and <36>) are cleared.
<b>44</b>	<b>DATA PAR ERR LW5 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on data longword 5 system bus bits <95:64>, during the second data cycle portion of a system bus transaction. DATA PAR ERR LW5 ERR remains valid until the DATA ERR bits <38> and <36> are cleared.
<b>43</b>	<b>DATA PAR ERR LW3 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on data longword 3 system bus bits <127:96>, during the first data cycle portion of a system bus transaction. DATA PAR ERR LW3 ERR remains valid until the Data ERR bits (<38> and <36>) are cleared.
<b>42</b>	<b>DATA PAR ERR LW1 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on data longword 1 system bus bits <95:64>, during the first data cycle portion of a system bus transaction. DATA PAR ERR LW1 ERR remains valid until the Data ERR bits (<38> and <36>) are cleared.
<b>41</b>	<b>C/A PAR ERR LW3 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on command/address longword 3 system bus bits <127:96>, during the C/A cycle portion of a system bus transaction. C/A PAR ERR LW3 ERR remains valid until the Data ERR bit (<34>) is cleared.
<b>40</b>	<b>C/A PAR ERR LW1 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on command/address longword 1 system bus bits <95:32>, during the C/A cycle portion of a system bus transaction. C/A PAR ERR LW1 ERR remains valid until the Data ERR bit<34>is cleared.
<b>39</b>	<b>MISSED PAR ERR ON READ DATA - REQ H</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. It is set when a parity error is detected on returned read data longwords 3 or 1 as a commander and the error command and address cannot be saved in the CBEAH register. Write "1" to clear.
<b>38</b>	<b>PAR ERR ON READ DATA - REQ H</b> <i>[read/write 1 to clear]</i>

(continued on next page)

## 6.1 CPU System Bus Register Definitions

Table 6–2 (Cont.) System Bus Error Register Description

Field	Description
	This field is cleared on power-up. It is set when a parity error is detected on returned read data longwords 3 or 1 as a commander. Write 1 to clear.
<b>37</b>	<b>MISSED PAR ERR ON WRITE DATA - RESP H</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. It is set when a parity error is detected on longwords 3 or 1 as a responder or bystander the error command and address cannot not be saved in the CBEAH register. Write 1 to clear.
<b>36</b>	<b>PAR ERR ON WRITE DATA - RESP H</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. It is set when a parity error is detected on odd longwords 3 or 1 as a responder or a bystander in the case of accepting write data to update the backup cache. Write 1 to clear. It detects parity error only on longwords 3 and 1.
<b>35</b>	<b>MISSED C/A PAR ERR</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. Set when a parity error was detected on the even C/A longwords (3,1) and the error command and address could not be saved in the CBEAH register. Write 1 to clear.
<b>34</b>	<b>C/A PAR ERR</b> <i>[read/write 1 to clear]</i> This bit is cleared on power-up. It is set when a parity error was detected on the even C/A longwords (3,1) regardless of the address or which node was the commander. A CPU node checks its own parity as a commander. Write 1 to clear.
<b>33</b>	<b>RESERVED DIAGNOSTIC H</b> <i>[read/write 1 to clear]</i> This bit is cleared on power-up. It is set as a result of an error event being communicated between the even and odd interface chips. This bit is for chip debug and should not be used by system software. The address is not held in CSR 8 and 9 when this bit is set. Write 1 to clear.
<b>31</b>	<b>MADR VALID</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. This bit is set when the miss counter overflows and holds sampled miss address into the MADRL register. The miss address is sampled every 64th backup cache miss by a free running miss transaction counter. When this bit is set MADRL <31:0> contain valid miss contents. Setting this bit inhibits capturing a new sample, but does not inhibit the counter from incrementing. A copy of this bit is readable from bit 0 of the MADRL register. Write 1 to clear.
<b>30:25</b>	<b>MISS COUNT</b> <i>[read-only]</i> This field is set to 111110 on power-up. The six bit miss address counter is readable in this field.
<b>15</b>	<b>WRITE DATA NOT ACKED</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. This bit is set if either octaword portion of a system bus write cycle generated by this commander is not acknowledged. The error address is logged in CBEAL and CBEAH, hence this error may cause subsequent lost errors. Write 1 to clear. This error can be flagged if a double bit error occurs in the tag store and the exchange address of the dirty victim places its address outside the physical address space of the currently configured system.
<b>14</b>	<b>C/A NOT ACKED</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. This bit is set if the C/A portion of a system bus cycle generated by this commander is not acknowledged. The error address is logged in fields associated with the lower 32 bits of the CBEAL and CBEAH. The upper 32 bits of this register have no meaning when this error is detected. This error may cause subsequent lost errors. Write 1 to clear.
<b>13</b>	<b>DATA PAR ERR LW6 ERR</b> <i>[read-only]</i>

(continued on next page)

## 6.1 CPU System Bus Register Definitions

**Table 6–2 (Cont.) System Bus Error Register Description**

Field	Description
	This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on data longword 6 system bus bits <63:32>, during the second data cycle portion of a system bus transaction. DATA PAR ERR LW6 ERR remains valid until the Data ERR bits (<6>,<4>) are cleared.
<b>12</b>	<b>DATA PAR ERR LW4 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on data longword 4 system bus bits <31:0>, during the second data cycle portion of a system bus transaction. DATA PAR ERR LW4 ERR remains valid until the Data ERR bits (<6>,<4>) are cleared.
<b>11</b>	<b>DATA PAR ERR LW2 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on data longword 2 system bus bits <63:32>, during the first data cycle portion of a system bus transaction. DATA PAR ERR LW2 ERR remains valid until the Data ERR bits (<6>,<4>) are cleared.
<b>10</b>	<b>DATA PAR ERR LW0 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on data longword 0 system bus bits <31:0>, during the first data cycle portion of a system bus transaction. DATA PAR ERR LW0 ERR remains valid until the Data ERR bits (<6>,<4>) are cleared.
<b>9</b>	<b>C/A PAR ERR LW2 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on command/address longword 2 system bus bits <63:32>, during the C/A cycle portion of a system bus transaction. C/A PAR ERR LW2 ERR remains valid until the Data ERR bit (<2>) is cleared.
<b>8</b>	<b>C/A PAR ERR LW0 ERR</b> <i>[read-only]</i> This field is cleared on power-up. This bit is valid if a parity error is detected by this module, reading a 1 indicates an error on command/address longword 0 system bus bits <31:0>, during the C/A cycle portion of a system bus transaction. C/A PAR ERR LW0 ERR remains valid until the Data ERR bit (<2>) is cleared.
<b>7</b>	<b>MISSED PAR ERR ON READ DATA - REQ</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. It is set when a parity error is detected on returned read data longwords 2 or 0 as a commander and the error command and address could not be saved in the CBEAL register. Write 1 to clear.
<b>6</b>	<b>PAR ERR ON READ DATA - REQ</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. It is set when a parity error is detected on returned read data longwords 2 or 0 as a commander. Write 1 to clear.
<b>5</b>	<b>MISSED ERR ON WRITE DATA- RESP</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. It is set when a parity error is detected on longwords 2 or 0 as a responder or bystander or a write data cycle from this commander is not acknowledged and the error command and address cannot not be saved in the CBEAL register. Write 1 to clear.
<b>4</b>	<b>PAR ERR ON WRITE DATA - RESP</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. It is set when a parity error is detected on even longwords 2 or 0 as a responder or a bystander in the case of accepting write data to update the backup cache. Write 1 to clear. It detects parity error only on longwords 2 and 0.
<b>3</b>	<b>MISSED C/A ERR</b> <i>[read/write 1 to clear]</i>

(continued on next page)

## 6.1 CPU System Bus Register Definitions

Table 6–2 (Cont.) System Bus Error Register Description

Field	Description
	This field is cleared on power-up. It is set when a parity error is detected on the even C/A longwords (2,0) or the C/A cycle was not acknowledged from this commander and the error command and address cannot be saved in the CBEAL register. Write 1 to clear.
2	<b>C/A PAR ERR</b> <i>[read/write 1 to clear]</i> This field is cleared on power-up. It is set when a parity error is detected on the even C/A longwords (2,0) regardless of the address or which node is the commander. A CPU node checks its own parity as a commander. Write 1 to clear.
1	<b>RESERVED DIAGNOSTIC</b> <i>[read/write 1 to clear]</i> This bit is cleared on power-up. It is set as a result of an error event being communicated between the even and odd interface chips. This bit is for chip debug and should not be used by system software. The address is not held in CSR 8 and 9 when this bit is set. Write 1 to clear.

### 6.1.3 System Bus Error Address Low Register (CSR8)

The system bus error address low register is updated by this node's commander transactions or by system bus errors. It contains the actual data found on the system bus <63:0> during the latest C/A cycle. Whenever a system bus error is detected and logged in the CBE register, the contents of this register are frozen until all of the error indications, not the missed error indications in the CBE Register are cleared.

For system bus command/address cycles that are not acknowledged, the failing address is latched only in the error logging bits <31:0>. Bits <63:34> do not contain valid information when this type of error is logged.

The system bus error address low register for CPU<sub>0</sub> is located at address 2.0000.0100<sub>16</sub>, and for CPU<sub>1</sub> at address 2.0800.0100<sub>16</sub>.

Figure 6–3 System Bus Error Address Low Register (CBEAL)

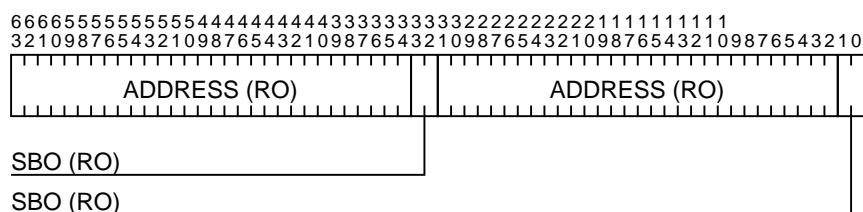


Table 6–3 System Bus Error Address Low Register Description

Field	Description
63:34	<b>ADDRESS</b> <i>[read-only]</i> This field is undefined on power-up. Address field <33:4>, from CAD<63:34>.
33:32	<b>SBO</b> <i>[read-only]</i> These bits should be ones.

(continued on next page)

## 6.1 CPU System Bus Register Definitions

**Table 6–3 (Cont.) System Bus Error Address Low Register Description**

Field	Description
<b>31:2</b>	<b>ADDRESS</b> <i>[read-only]</i> This field is undefined on power-up. Address field <33:4>, from CAD<31:2>.
<b>1:0</b>	<b>SBO</b> <i>[read-only]</i> These bits should be ones.

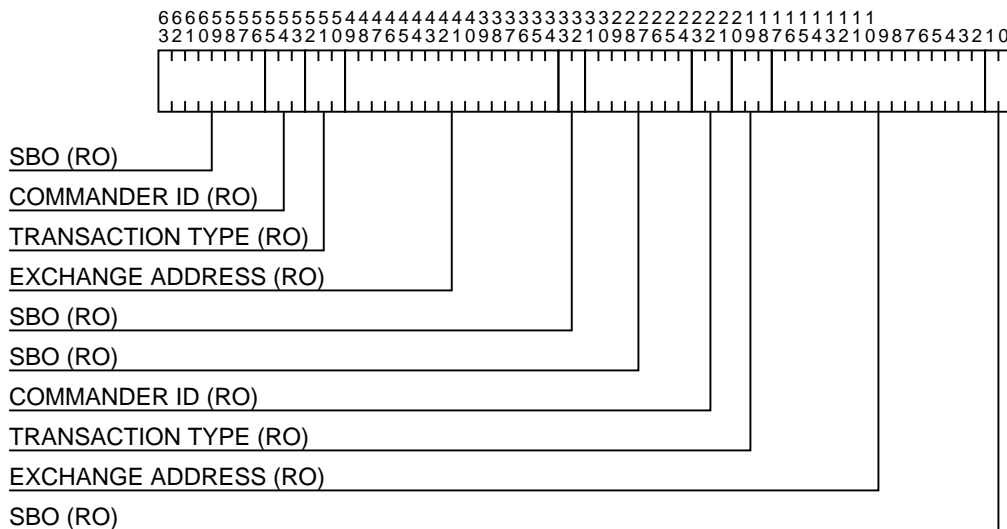
### 6.1.4 System Bus Error Address High Register (CSR9)

The system bus error address high register is updated by this node's commander transactions or by system bus errors and contains the actual data found on the system bus <127:64> during the latest C/A cycle. Whenever a system bus error is detected and logged in the CBE register, the contents of this register are frozen until all of the errors, not missed error, indications in the CBE register are cleared.

For system bus command/address cycles that are not acknowledged, the failing address is latched only in the error logging bits <31:0>. Bits <63:34> do not contain valid information when this type of error is logged.

The system bus error address high register for CPU<sub>0</sub> is located at address 2.0000.0120<sub>16</sub>, and for CPU<sub>1</sub> at address 2.0800.0120<sub>16</sub>.

**Figure 6–4 System Bus Error Address High Register (CBEAH)**



**Table 6–4 System Bus Error Address High Register Description**

Field	Description
<b>63:56</b>	<b>SBO</b> <i>[read-only]</i>

(continued on next page)

## 6.1 CPU System Bus Register Definitions

Table 6–4 (Cont.) System Bus Error Address High Register Description

Field	Description
	This bit is undefined on power-up. These bits should be ones.
<b>55:53</b>	<b>COMMANDER ID</b> <i>[read-only]</i> This field is undefined on power-up. Contains the commander identification code. The field is encoded as:  000 - RESERVED 001 - CPU <sub>0</sub> 010 - CPU <sub>1</sub> 011 - RESERVED 100 - I/O 101 - RESERVED 110 - RESERVED 111 - RESERVED
<b>52:50</b>	<b>TRANSACTION TYPE</b> <i>[read-only]</i> This field is undefined on power-up. The commander transaction request. The field is encoded as follows:  000 - READ 001 - RESERVED 010 - EXCHANGE 011 - RESERVED 100 - WRITE 101 - RESERVED 110 - RESERVED 111 - NUT
<b>49:34</b>	<b>EXCHANGE ADDRESS</b> <i>[read-only]</i> This field is undefined on power-up. It contains the tag of the victimized cache location. See Chapter 19 for details.
<b>33:32</b>	<b>SBO</b> <i>[read-only]</i> These bits should be ones.
<b>31:24</b>	<b>SBO</b> <i>[read-only]</i> These bits should be ones.
<b>23:21</b>	<b>COMMANDER ID</b> <i>[read-only]</i> This field is undefined on power-up. It contains the commander identification code. The field is encoded as follows:  000 - RESERVED 001 - CPU <sub>0</sub> 010 - CPU <sub>1</sub> 011 - RESERVED 100 - I/O 101 - RESERVED 110 - RESERVED 111 - RESERVED
<b>20:18</b>	<b>TRANSACTION TYPE</b> <i>[read-only]</i>

(continued on next page)



## 6.1 CPU System Bus Register Definitions

**Table 6–4 (Cont.) System Bus Error Address High Register Description**

Field	Description
	This field is undefined on power-up. The commander transaction request. The field is encoded as: 000 - READ 001 - RESERVED 010 - EXCHANGE 011 - RESERVED 100 - WRITE 101 - RESERVED 110 - RESERVED 111 - NUT
<b>17:2</b>	<b>EXCHANGE ADDRESS</b> <i>[read-only]</i> This field is undefined on power-up. It contains the tag of the victimized cache location. See Chapter 19 for details.
<b>1:0</b>	<b>SBO</b> <i>[read-only]</i> These bits should be ones.

## 6.2 Multiprocessor Configuration CSR Definitions

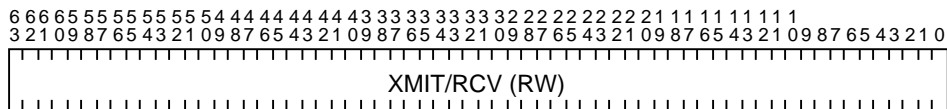
The following paragraphs describe the registers used in multiprocessor configurations.

### 6.2.1 Processor Mailbox Register (CSR10)

The processor mailbox register allows any system bus commander to communicate with any other system bus commander (implementing the processor mailbox register) without the need for any memory subsystem. This is primarily intended to be used during the initialization of the system and system exception processing.

The processor mailbox registers are located at address 2.0000.0140<sub>16</sub> for CPU<sub>0</sub> and address 2.0800.0140<sub>16</sub> for CPU<sub>1</sub>.

**Figure 6–5 Processor Mailbox Register (PMBX)**



**Table 6–5 Processor Mailbox Register Description**

Field	Description
<b>63:0</b>	<b>XMIT/RCV</b> <i>[read/write]</i> This field is cleared on power-up. The protocol used in communication between the two processors is completely under software control.

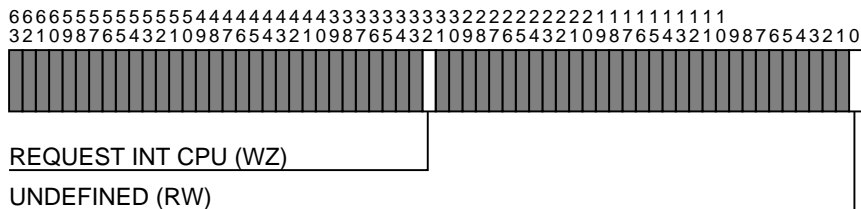
## 6.2 Multiprocessor Configuration CSR Definitions

### 6.2.2 Interprocessor Interrupt Request Register (CSR11)

Each CPU has an interprocessor interrupt request register to support the interprocessor interrupt IPR specified in the *Alpha Architecture Reference Manual*. Because these are system bus visible registers, any system bus commander, including the owner, can post an interprocessor interrupt to any CPU. The interrupt is driven into the processor signal identified internally as HIRR(3). Refer to the DECchip 21064 User Guide and Chapter 9.

The interprocessor interrupt request registers are located at address 2.0000.0160<sub>16</sub> for CPU<sub>0</sub> and address 2.0800.0160<sub>16</sub> for CPU<sub>1</sub>.

**Figure 6–6 Interprocessor Interrupt Request Register (IPIR)**



**Table 6–6 Interprocessor Interrupt Request Register Description**

Field	Description
<b>32</b>	<b>REQUEST INT CPU</b> <i>[write]</i> This field is cleared on power-up. Writing a 1 to this bit causes a hardware interrupt to be posted to the CPU defined by the address of the CSR. Reading this register returns the state of the interrupt request signal. This field is cleared by writing a '0'.
<b>0</b>	<b>UNDEFINED</b> <i>[read/write]</i> Undefined

### 6.3 System Interrupt Clear Register (CSR12)

The system interrupt clear register provides a path for the CPU to clear the edge triggered interrupts from the system bus C\_ERR\_L signal, the SYS\_EVENT\_L signal, and the interval timer interrupt, CINT\_TIM signal. The C\_ERR\_L and SYS\_EVENT\_L signals are broadcast to both CPU modules. The interval timer clock is received from the system bus and used to generate an interval timer interrupt to each processor. The interval timer interrupt is local to each CPU so that this interrupt occurs 180 degrees out of phase with the other processor node. The system generic system event interrupt is generated by any of the following:

- I/O halt request
- Operator control panel halt request
- An Enclosure event
- Power supply event

The generic transaction error signal C\_ERR\_L is generated by soft or hard errors related to data transactions.

## 6.3 System Interrupt Clear Register (CSR12)

The system interrupt clear register for CPU<sub>0</sub> is located at addresses 2.0000.0180<sub>16</sub>, and CPU<sub>1</sub> at address 2.0800.0180<sub>16</sub>.

Figure 6–7 System Interrupt Clear Register (SIC)

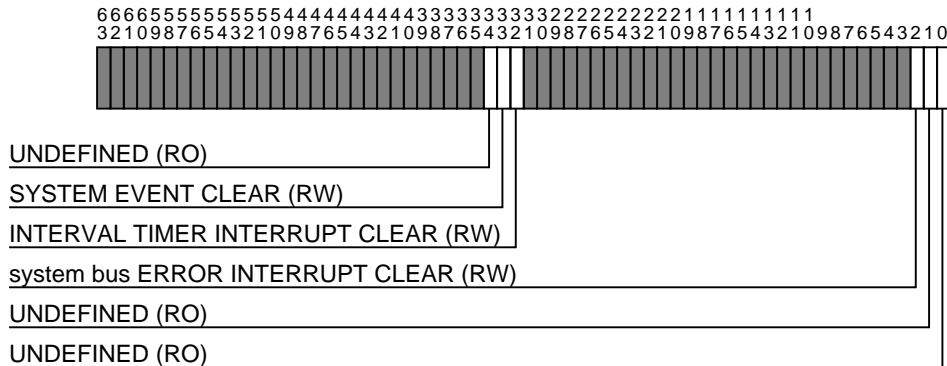


Table 6–7 System Interrupt Clear Register Description

Field	Description
<b>34</b>	<b>UNDEFINED</b> <i>[read-only]</i> Undefined
<b>33</b>	<b>SYSTEM EVENT CLEAR</b> <i>[read/write]</i> This field is cleared on power-up. A read of this register returns the state of the request signal to the processor. An STQ to this register with a 1 in this bit position clears the latched CSYS_EVENT_L interrupt signal driven to the local CPU. This interrupt can be masked in the 21064 CPU chip.
<b>32</b>	<b>INTERVAL TIMER INTERRUPT CLEAR</b> <i>[read/write]</i> This field is cleared on power-up. A read of this register returns the state of the interrupt signal to the processor. A STQ to this register with a 1 in this bit position clears the latched CINT_TIM interrupt signal driven to the local CPU. This interrupt can be masked in the EV chip.
<b>2</b>	<b>system bus ERROR INTERRUPT CLEAR</b> <i>[read/write]</i> This field is cleared on power-up. A read of this register returns the state of the interrupt signal to the processor. An STQ to this register with a 1 in this bit position clears the latched C_ERR_L interrupt signal driven to the local CPU. This interrupt can be masked in the 21064 chip. Interrupts generated by errors detected by this node may be disabled via bit <43,11> in CSR6.
<b>1</b>	<b>UNDEFINED</b> <i>[read-only]</i> Undefined
<b>0</b>	<b>UNDEFINED</b> <i>[read-only]</i> Undefined

## 6.4 Address Lock Register (CSR13)

The address lock register is required by the Alpha architecture to support the LDxL and STxC instructions. This is supported on DEC 4000 in “memory-like” regions only (Address<33> is 0). This register latches the address and sets the LOCK ADDRESS VALID bit when an LDxL instruction to memory address space is executed. The Lock Address Valid bit is cleared when a STxC to any location or a system bus write to the locked location occurs even if the write is from this node,

## 6.4 Address Lock Register (CSR13)

or by explicitly clearing it by writing bit <0>. If the Lock Address Valid bit is set, and a LDxL to a different location occurs, the contents of the Lock Address field is updated with the new address.

The resolution of the address lock in the DEC 4000 system is a single aligned 32 byte-block.

### Note

The system software ensures that the state of the lock VALID flag in the low longword is consistent with the lock flag in the upper longword by performing quadword writes.

The lock flag must always be cleared before returning from PAL mode.

The address lock register for CPU<sub>0</sub> is located at address 2.0000.01A0<sub>16</sub> and for CPU<sub>1</sub> address 2.0800.01A0<sub>16</sub>.

Figure 6–8 Address Lock Register (ADLK)

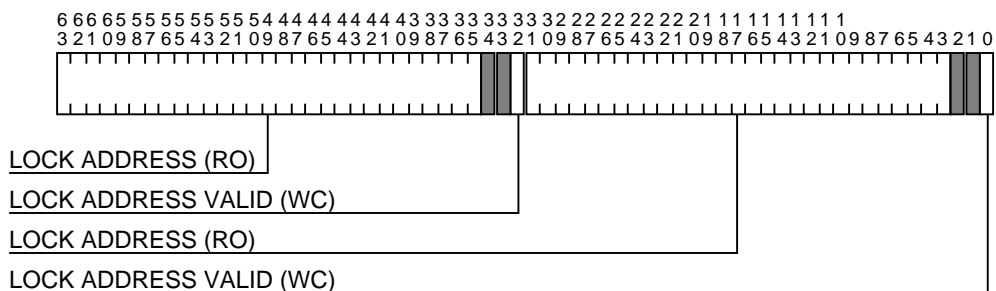


Table 6–8 Address Lock Register Description

Field	Description
<b>63:35</b>	<b>LOCK ADDRESS</b> <i>[read-only]</i> This field is set on power-up. An LDxL instruction to memory address space from this nodes processor causes the contents of this register to be updated with the lock address. The contents are invalid when the LOCK ADDRESS VALID bit is clear. Bit <63> always reads as 0.
<b>32</b>	<b>LOCK ADDRESS VALID</b> <i>[read/write 1 to clear]</i> When set, this bit indicates that the LOCK ADDRESS field is valid and a STxC succeeds. This bit is cleared on power-up, by a write from the system bus to the locked location, by a STxC from the processor, or by writing a 1 to this bit.
<b>31:3</b>	<b>LOCK ADDRESS</b> <i>[read-only]</i> Set on power-up. An LDxL instruction to memory address space from this node's processor causes the contents of this register to be updated with the lock address. The contents are invalid when the LOCK ADDRESS VALID bit is clear. Bit <31> always reads as 0.
<b>0</b>	<b>LOCK ADDRESS VALID</b> <i>[read/write 1 to clear]</i>

(continued on next page)

**Table 6–8 (Cont.) Address Lock Register Description**

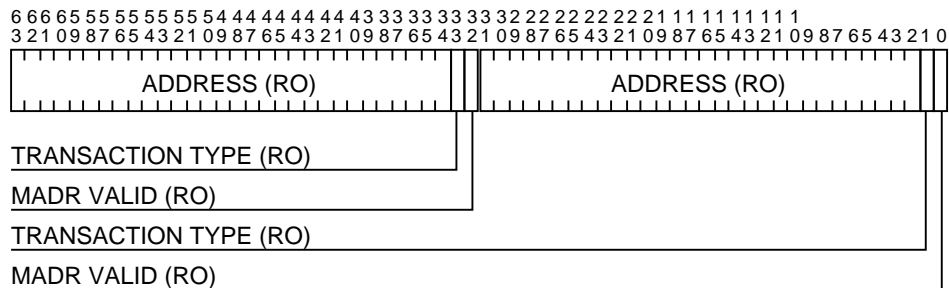
Field	Description
	When set, this bit indicates that the LOCK ADDRESS field is valid and a STxC succeeds. This bit is cleared on power-up, by a write from the system bus to the locked location, by a STxC from the processor to memory address space, or by writing a 1 to this bit.

## 6.5 Miss Address Register (CSR14)

The miss address register captures the system bus read or exchange address on every miss, and holds the 64th sample until a sample valid flag is cleared. The read or exchange may have resulted from a DECchip CPU read or write transaction. The latching strobe is skewed by 32 counts between the low and high longwords of CSR14.

The miss address register is located at addresses 2.0000.01C0<sub>16</sub> for CPU<sub>0</sub> and address 2.0800.01C0<sub>16</sub> for CPU<sub>1</sub>.

**Figure 6–9 Miss Address Register Low (MADRL)**



**Table 6–9 Miss Address Register Low Description**

Field	Description
<b>63:34</b>	<b>ADDRESS</b> <i>[read-only]</i> This field is undefined on power-up. Address field <33:4>, from CAD<63:34>.
<b>33</b>	<b>TRANSACTION TYPE</b> <i>[read-only]</i> This field is undefined on power-up. When set, indicates read cleared indicates exchange.
<b>32</b>	<b>MADR VALID</b> <i>[read-only]</i> Clear on power-up. When set, this bit indicates that this sample is valid. This bit is a read-only copy of CSR7 bit 31.
<b>31:2</b>	<b>ADDRESS</b> <i>[read-only]</i> This field is undefined on power-up. Address field <33:4>, from CAD<31:2>.
<b>1</b>	<b>TRANSACTION TYPE</b> <i>[read-only]</i> This field is undefined on power-up. When set, indicates read cleared indicates exchange.
<b>0</b>	<b>MADR VALID</b> <i>[read-only]</i>

(continued on next page)

## 6.5 Miss Address Register (CSR14)

Table 6–9 (Cont.) Miss Address Register Low Description

Field	Description
	This field is cleared on power-up. When set, this bit indicates that this sample is valid. This bit is a read-only copy of CSR7 bit 31.

## 6.6 C<sup>3</sup> Revision Register (CSR15)

Figure 6–10 C<sup>3</sup> Revision Register (CRR)

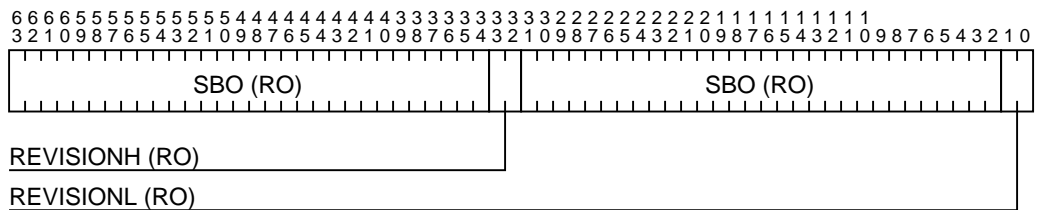


Table 6–10 C<sup>3</sup> Revision Register Description

Field	Description
<b>63:34</b>	<b>SBO</b> <i>[read-only]</i> A read always returns a zero value.
<b>33:32</b>	<b>REVISIONH</b> <i>[read-only]</i> Indicates the revision level of the bus interface (C <sup>3</sup> ) device that handles longwords 3 and 1. <ul style="list-style-type: none"> <li>• 0 - rev 1</li> <li>• 0 - rev 2 (if the bits 46 and 45 of the CBCTL Register can be set and cleared and the revision in the CRR field indicate a 0 then the bus interface slice is rev 2.)</li> <li>• 3 - rev 3</li> </ul>
<b>31:2</b>	<b>SBO</b> <i>[read-only]</i> A read always returns a zero value.
<b>1:0</b>	<b>REVISIONL</b> <i>[read-only]</i>

**Note**

The revision indicated in bits 33,32 and 1,0 must agree otherwise the CPU will not function.

(continued on next page)

## 6.6 C<sup>3</sup> Revision Register (CSR15)

Table 6–10 (Cont.) C<sup>3</sup> Revision Register Description

Field	Description
	Indicates the revision level of the bus interface (C <sup>3</sup> ) device that handles longwords 2 and 0. <ul style="list-style-type: none"><li>• 0 - rev 1</li><li>• 0 - rev 2 (if the bits 14 and 13 of the CBCTL Register can be set and cleared and the revision in the CRR field indicate a 0 then the bus interface slice is rev 2.)</li><li>• 3 - rev 3</li></ul>

**Note**

The revision indicated in bits 33,32 and 1,0 must agree otherwise the CPU will not function.

## 6.7 D-bus

A 16 KB serial ROM is located on the CPU module and is used to supply the processor with power-up code. This ROM pattern is loaded into the 21064 processor's internal instruction cache under the control of the processor after power-up reset *only*.

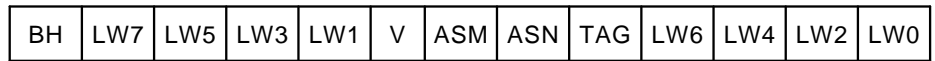
The loaded data consists of:

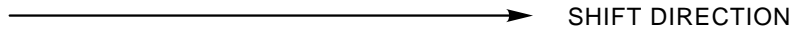
- Each cache block's tag
- ASN
- ASM
- Valid
- Branch history
- 8 longwords of data

This data is loaded in sequential order, starting with block 0 and ending with block 255. The order in which bits within each block are serially loaded is shown in Figure 6–11.

## 6.7 D-bus

**Figure 6–11 21064 Serial Load Data Format**





Bits within each field are arranged with high–order bit on the left.

An extension serial ROM of up to 16 KB is automatically multiplexed on to the D-bus after the initial 16 KB is read into the processor.

The software system is able to load this ROM into the data cache of the processor (or the backup cache to execute it) by alternately toggling the TMT bit in the SL\_XMIT register of the processor, and reading the RCV bit of the processors SL\_RCV register.

It is unnecessary to read the unused portion of the initial serial ROM under program control. The extension serial ROM can be read immediately. The first extension ROM data bit is present in the SL\_RCV register 500 ns after the completion of the initial serial ROM load and 500 ns after every subsequent 0 to 1 transition of the TMT bit in the SL\_XMIT register.

The TMT bit should not be toggled from 1 to 0 nor 0 to 1 with a delay smaller than 500 ns between transitions. This ROM can be read only once, after a hard system reset.

As initialization proceeds, the D-bus microcontroller (87C652) monitors the progress of the 21064 processor as it reads each serial ROM by observing the state of the serial ROM read status signals. When all ROMs are read, or an error is detected, the micro has the ability to communicate with 21064 over the 21064 Serial Line Interface.

To monitor the bit by bit progress of 21064's reading the serial ROMs, the D-bus micro drives a 1 on EV/SERIAL LINE OUTPUT and enables communication with 21064 by clearing the DISABLE COMMUNICATIONS WITH EV bit. By monitoring the EV/SERIAL LINE input, the D-bus micro detects each clock sent to the serial ROM devices.

See Table 6–11 for mapping of D-bus microcontroller ports to module functions.

**Table 6–11 D-bus Micro Port Mapping**

Port	Port Type	Init State	Functional Description
P3.5	I	—	Interval timer interrupt request
P3.4	I	—	Regular clock input EV 6.6 - 4.7344MHz EV 10.0 - 3.1250MHz
P3.3	I	—	Interval timer interrupt request

(continued on next page)



Table 6–11 (Cont.) D-bus Micro Port Mapping

Port	Port Type	Init State	Functional Description
P3.2	I	—	Serial ROM read status; when set DECchip's instruction cache initial serial ROM load not complete
P3.1	O	1	DECchip/Serial line output - active only when P2.7 is cleared
P3.0	I	—	DECchip/Serial line input - active only when P2.7 is cleared
P2.7	O	1	Disable communication with 21064
P2.6	I	—	CPU ID - CPU <sub>0</sub> = 1, CPU <sub>1</sub> = 0
P2.5	I	—	Serial ROM read status; when set supplemental serial ROM 3 not read
P2.4	I	—	Serial ROM read status; when set supplemental serial ROM 2 not read
P2.3	I	—	Serial ROM read status; when set supplemental serial ROM 1 not read; as the processor never completely reads this ROM this is never cleared.
P2.2	I	—	Serial ROM read status. When set, 21064 instruction cache initial serial ROM load not completed.
P2.1	I	—	Clock/Voltage detect - When clear, indicates that oscillator is not running or 3.3 V power is below regulation. This is useful only when CPU ID is 0; if CPU ID is 1 CRESET L would assert when either condition above occurred, which in turn would cause the D-bus micro to reset as well.
P2.0	I	—	21064 processor resting due to 3.3V under voltage or oscillator not running. As above, useful only if CPU ID is 0.
P1.7	B	—	Serial command bus data line
P1.6	B	—	Serial command bus clock line

The D-bus microcontroller's input oscillator frequency is as follows:

Table 6–12 D-bus Microcontroller Clock Frequency

System Bus OSC (MHz)	system bus Period (ns)	Micro OSC (MHz)
333	24.0	10.41667
320	25.0	10.00000
303	26.4	9.46970
200	40.0	6.25

## 6.7 D-bus

**Table 6–13 D-bus Microcontroller System Control Bus Address**

Transaction	CPU <sub>0</sub> Address <sub>16</sub>	CPU <sub>1</sub> Address <sub>16</sub>
Read	B1	B3
Write	B0	B2

## 6.8 System Bus Arbiter

The system bus arbiter supports three arbitrated nodes: two CPU modules, and one I/O module. The arbiter logic is built into the CPU<sub>0</sub> module. Arbitration is executed in a prioritized round-robin scheme. I/O is the highest priority, and both CPU's a lesser but equal priority. This provides the arbitration characteristic of CPU<sub>0</sub>, I/O, CPU<sub>1</sub>, I/O, CPU<sub>0</sub>, I/O, etc. If the system contains only 1 processor module, the arbitration algorithm becomes a standard round-robin. These arbitration characteristics occur only when the system bus is saturated.

When the system bus is saturated with write traffic, a single dead system bus cycle (nominal 24 ns) is inserted every third write transaction. This is done to avoid starving the 21064 from its backup cache.

To minimize arbitration overhead, arbitration cycles occur in parallel with data transfer cycles.

The I/O module requests and is granted the system bus using the IOREQ\_L and IOG\_L signals, and CPU module<sub>1</sub> using the CPUREQ\_L and CPUG\_L signals.

## 6.9 System Bus Clock Generator/Distributor

The system bus clock generator/distributor is located on the CPU<sub>0</sub> processor module. The system bus clock is asynchronous to the CPU clock. However, the frequency of the system bus clock is not more than the processors clock frequency (nominally 166.67 MHz) divided by 4 minus 2.4 ns. This clock consists of two phases where phase 3 is 270 degrees behind phase 1. It is distributed as an ECL differential pair on tightly controlled/matched 50  $\Omega$  lines. Each system bus module receives its own copy of the system bus clock.

If the system bus clock oscillator stops, CRESET L will be asserted.

## 6.10 System Bus CRESET L Generation

SYSTEM BUS CRESET L is generated on CPU<sub>0</sub>. This signal asserts on the following conditions:

- ASYNCHRESET from the power subsystem is asserted.
- The oscillator on the system bus or 21064 on CPU<sub>0</sub> stop.
- The 3.3 V supply on CPU<sub>0</sub> goes under voltage.

The clock detect circuits on the CPU module guarantee that the oscillator generating the clocks for the module have been running for at least 20 ms. This is to allow enough time for the oscillators to stabilize.

System bus CRESET L deassertion is synchronous with the system bus PHI1 H clock.

ASYNCHRESET from the power subsystem must remain asserted for at least 40 ms.

### 6.11 OCP Halt Request Buffer

The halt request line from the operator control panel is driven to CPU<sub>0</sub> module and buffered there, and re-driven out the SYSTEM EVENT INTERRUPT LINE.

### 6.12 Nonvolatile EEPROM

The EEPROM is a nonvolatile 256 x 8 byte RAM that resides on the serial control bus. This RAM is used to store the CPU module's serial number as well as maintain error logging information. The last four bytes of this RAM also contain the following information:

RAM Address <sub>16</sub>	Description
FC	<p>First nibble contains the values to program SELECT DRACK, and 2nd QW SELECT in the CBCTL register. Refer to Section 6.1 for the definitions and expected values for these bits.</p> <p>The second nibble contains the cache size of the module as programmed in the BIU_CTL register. Refer to Table 4-16 for the definition of the actual values.</p>
FD	This byte contains a number indicating how many serial ROM's are used in this module's implementation. Nominally this value is 2. (Format is binary.)
FE	Bits <7:1> are reserved, bit <0> indicates that the DEC 4000-xx Console should be unloaded automatically.
FF	This location contains the byte checksum of the last four bytes of the RAM. Thus by adding locations FC through FF ignoring overflows, a 0 sum indicates clean data.

The serial control bus addresses for these RAMs are shown below:

**Table 6-14 Nonvolatile EEPROM System Control Bus Address**

Transaction	CPU <sub>0</sub> Address <sub>16</sub>	CPU <sub>1</sub> Address <sub>16</sub>
Read	A9	AB
Write	A8	AA

---

## CPU Module Transactions

The CPU module has two transaction initiators: the processor and the system bus interface controller.

The processor initiates a transaction as a result of a program operation that requires access to memory or I/O devices. When a local resource can not service the processor request (as a result of a cache miss or I/O space access), a request is made to the system bus interface controller to obtain the information from the system bus.

System bus transactions occur as a result of system bus resource requests from any of the possible system bus Commanders. Transactions are monitored by every member of the bus for the following reasons:

- Any module on the system bus could be a responder to a specific system bus transaction.
- The system bus implements a “snooping” protocol to guarantee cache coherence.

See Chapter 19 for details.

### 7.1 Processor Transactions

The DECchip CPU requests an external cycle when it determines that the cycle it wants to perform requires module level action.

The cycle types are as follows:

- A BARRIER cycle is generated by the MB instruction. As no external write buffer exists between the processor and an error detection point in the system the DEC 4000 CPU module acknowledges the cycle.
- The FETCH and FETCHM cycles are generated by the FETCH and FETCHM instructions respectively. The backup cache controller acknowledges the instruction and no other action takes place.
- The READ\_BLOCK cycle is generated on read misses. The backup cache controller reads the addressed block from memory and supplies it, 128 bits at a time, to 21064 via the data bus. The backup cache location that missed is victimized and updated with the new cache entry.
- The WRITE\_BLOCK cycle is generated on write misses, and on writes to shared blocks. The backup cache controller pulls the write data, 128 bits at a time, from 21064 via the data bus, and writes the valid longwords to memory.

## 7.1 Processor Transactions

- The LDxL cycle is generated by the LDLL and LDQL instructions. The cycle works just like a READ\_BLOCK, although the backup cache is not probed by the processor. The backup cache controller performs the backup cache probe and if the reference is to cacheable address space, the address is latched into the Address Lock Register.
- The STxC cycle is generated by the STLC and STQC instructions. The cycle works just like a WRITE\_BLOCK, although the backup cache is not probed by the processor. The backup cache controller performs the backup cache probe and the cycle is acknowledged with the completion status.

**Table 7–1 Processor Initiated Transactions**

TRANSACTION	Activity
P-Cache Read	Not visible outside processor
P-Cache Write	Backup cache written if hit Write Block Generated if Miss
P-Cache Masked Write	Backup cache written if hit Write Block Generated if Miss
Fast backup cache Read Hit	Backup cache data read
Fast backup cache Write Hit	Data written to backup cache data store, DIRTY bit set
Fast backup cache Masked Write Hit	Data written to backup cache data store, DIRTY bit set
Read Block <sup>1</sup>	System bus read or exchange cycle generated
Write Block <sup>2</sup>	System bus read or exchange and possibly write or nut cycles generated
LDxL - Load Lock	System bus read, exchange, or nut cycle generated, IF (cacheable address space reference) address latched and Lock bit set  IF (non-cacheable address space) no change to address lock or LOCK bit.
STxC - Store Conditional	System bus read, exchange and possibly nut or write cycles generated, IF (cacheable address space reference) LOCK bit cleared (it was set) store completes  IF (non-cacheable address space) no change to LOCK bit, store failed if responder asserts UC_ERR L during cycle.
Barrier <sup>3</sup>	All data buffers flushed to system coherence point.
FETCH/FETCHM <sup>4</sup>	Acknowledge request, no other module level activity.

<sup>1</sup>Generated as a result of a Fast backup cache Read Miss.

<sup>2</sup>Generated as a result of a Fast backup cache Write Miss.

<sup>3</sup>Generated as a result of the execution of a Memory Barrier Instruction.

<sup>4</sup>Generated as a result of the execution of a FETCH or FETCHM Instruction.

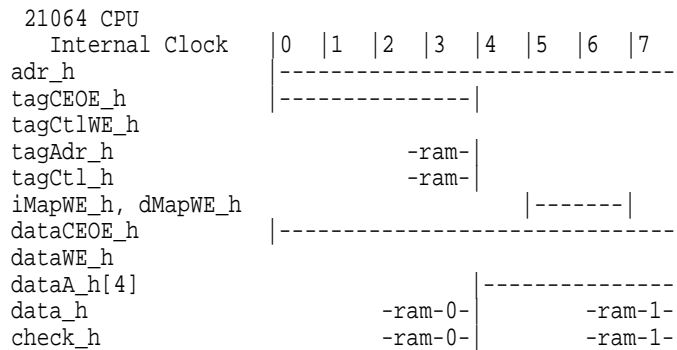
### 7.1.1 21064 CPU Chip Transactions

## 7.1 Processor Transactions

### 7.1.1.1 Fast External Cache Read Hit

A fast external cache read consists of a probe read overlapped with the first data read, followed by the second data read if the probe hits. The following diagram illustrates the DEC 4000 CPU fast external cache read which selects 4 CPU cycle reads (BC\_RD\_SPD = 3), 4 CPU cycle writes (BC\_WR\_SPD = 3), chip enable control (OE = L).

If the probe misses, the cycle aborts at the end of clock cycle 3. If the probe hits and the miss address had bit 4 set the two data reads would have been swapped (dataA\_h[4] would have been true in cycles 0, 1, 2, 3, and would have been false in cycles 4, 5, 6, 7).

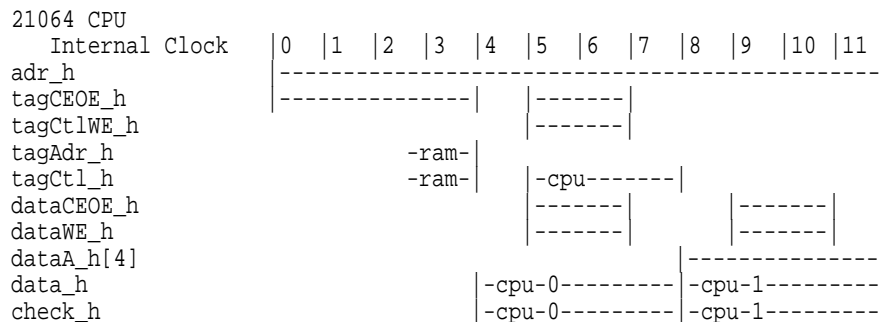


### 7.1.1.2 Fast External Cache Write Hit

A fast external cache write consists of a probe read, followed by 1 or 2 data writes. The following diagram shows that the DEC 4000 CPU external cache transaction is using 4 CPU cycle reads (BC\_RD\_SPD = 3), 4 CPU cycle writes (BC\_WR\_SPD = 3), chip enable control (OE = L), and a 2 cycle write pulse centered in the 4 cycle write (BC\_WE\_CTL[15..1] = LLLLLLLLLLLLLLHH).

Note that the 21064 CPU drives the TAGCTL\_H pins one CPU cycle later than it drives the data\_h and check\_h pins relative to the start of the write cycle. This is because, unlike DATA\_H and CHECK\_H, the TAGCTL\_H field must be read during the tag probe which precedes the write cycle.

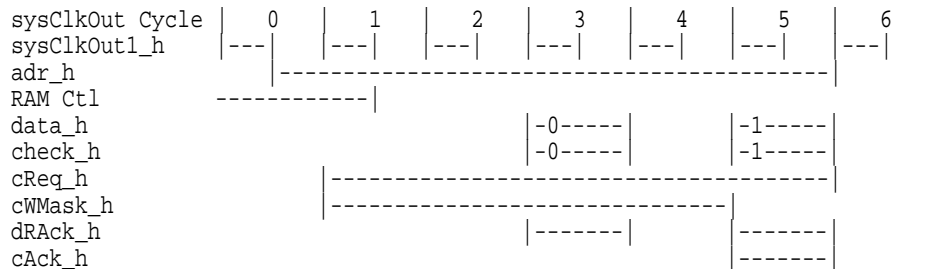
Because the 21064 can switch its pins to a low impedance state much more quickly than most RAMs can switch their pins to a high impedance state, the 21064 CPU waits one CPU cycle before driving the TAGCTL\_H pins in order to minimize tristate driver overlap. If the probe misses, the cycle aborts at the end of clock cycle 3.



## 7.1 Processor Transactions

### 7.1.1.3 READ\_BLOCK

A READ\_BLOCK transaction appears at the external interface on external cache read misses, either because it really was a miss, or because the external cache has not been enabled.



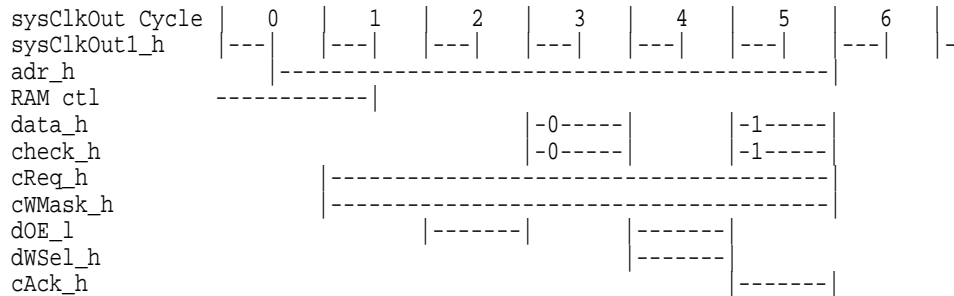
1. The CREQ\_H pins are always idle in the system clock cycle immediately before the beginning of an external transaction. The ADR\_H pins always change to their final value (with respect to a particular external transaction) at least one CPU cycle before the start of the transaction.
2. The READ\_BLOCK transaction begins. The 21064 CPU has already placed the address of the block containing the miss on adr\_h. The 21064 CPU places the quadword-within-block and the I/D indication on cWMask\_h. The 21064 CPU places a READ\_BLOCK command code on CREQ\_H. The 21064 CPU clears the RAM control pins DATAA\_H[4..3], DATACEOE\_H[3..0] and TAGCEOE\_H no later than one CPU cycle after the system clock edge where the transaction begins.
3. The external logic obtains the first 16 bytes of data. Although a single stall cycle has been shown here, there could be no stall cycles, or many stall cycles.
4. The external logic has the first 16 bytes of data. It places it on the DATA\_H and CHECK\_H buses. It asserts DRACK\_H to tell the 21064 that the data and check bit buses are valid. The 21064 detects DRACK\_H at the end of this cycle, and reads in the first 16 bytes of data at the same time.
5. The external logic obtains the second 16 bytes of data. Although a single stall cycle has been shown here, there could be no stall cycles, or many stall cycles.
6. The external logic has the second 16 bytes of data. It places it on the DATA\_H and CHECK\_H buses. It asserts DRACK\_H to tell the 21064 CPU that the data and check bit buses are valid. The 21064 CPU detects DRACK\_H at the end of this cycle, and reads in the second 16 bytes of data at the same time. In addition, the external logic places an acknowledge code on CACK\_H to tell the 21064 that the READ\_BLOCK cycle is completed. The 21064 detects the acknowledge at the end of this cycle, and may change the address.
7. Everything is idle. The 21064 can start a new external cache cycle at this time.

Because external logic owns the RAMs by virtue of 21064 having deasserted its RAM control signals at the beginning of the transaction, external logic may cache the data by asserting its write pulses on the external cache during cycles 3 and 5.

The 21064 CPU performs EDC checking on the data supplied to it through the data and check buses if so requested by the acknowledge code. It is not necessary to place data in the external cache to get checking.

### 7.1.1.4 Write\_block

A WRITE\_BLOCK transaction appears at the external interface on external cache write misses (either because it really was a miss, or because the external cache has not been enabled), or on external cache write hits to shared blocks.



1. The CREQ\_H pins are always idle in the system clock cycle immediately before the beginning of an external transaction. The ADR\_H pins always change to their final value (with respect to a particular external transaction) at least one CPU cycle before the start of the transaction.
2. The WRITE\_BLOCK cycle begins. The 21064 has placed the address of the block on ADR\_H. The 21064 places the longword valid masks on CWMASK\_H and a WRITE\_BLOCK command code on CREQ\_H. The 21064 clears DATA\_H[4..3] and TAGCEOE\_H no later than one CPU cycle after the system clock edge at which the transaction begins. The 21064 clears DATACEOE\_H[3..0] at least one CPU cycle before the system clock edge where the transaction begins.
3. The external logic detects the command, and asserts DOE\_L to tell the 21064 to drive the first 16 bytes of the block onto the data bus.
4. The 21064 drives the first 16 bytes of write data on to the DATA\_H and CHECK\_H buses, and the external logic writes it into the destination. Although a single stall cycle has been shown here, there could be no stall cycles, or many stall cycles.
5. The external logic asserts DOE\_L and DWSEL\_H to tell the 21064 to drive the second 16 bytes of data onto the data bus.
6. The 21064 drives the second 16 bytes of write data onto the DATA\_H and CHECK\_H buses, and the external logic writes it into the destination. Although a single stall cycle has been shown here, there could be no stall cycles, or many stall cycles. In addition, the external logic places an acknowledge code on CACK\_H to tell the 21064 that the WRITE\_BLOCK cycle is completed. The 21064 detects the acknowledge at the end of this cycle, and changes the address and command to the next values.
7. Everything is idle. Because external logic owns the RAMs by virtue of 21064 having deasserted its RAM control signals at the beginning of the transaction, external logic may cache the data by asserting its write pulses on the external cache during cycles 3 and 5.

The 21064 CPU performs EDC generation on data it drives onto the data bus. DEC 4000's 21064 CPU interface performs EDC checking and correction on this data.



## 7.1 Processor Transactions

Although in the above diagram external logic cycles through both 128-bit chunks of potential write data, this need not always be the case. External logic must pull from the 21064 chip only those 128-bit chunks of data that contain valid longwords as specified by the CWMASK\_H signals. The only requirement is that if both halves are pulled from the 21064, the lower half must be pulled before the upper half.

### 7.1.1.5 LDxL

An LDxL transaction appears at the external interface as a result of an LDQL or LDL instruction being executed. The external cache is not probed. With the exception of the command code output on the CREQ pins, the LDxL transaction is exactly the same as a READ\_BLOCK transaction. (See Section 7.1.1.3.)

### 7.1.1.6 STxC

An STxC transaction appears at the external interface as a result of STLC and STQC instructions. The external cache is not probed.

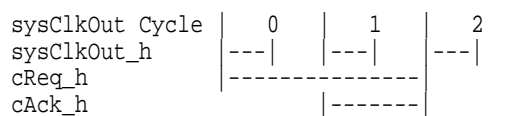
The STxC transaction is the same as the WRITE\_BLOCK transaction, with the following exceptions:

1. The code placed on the CREQ pins is different.
2. The CWMASK field never validates more than a single longword or quadword of data.
3. External logic has the option of making the transaction fail by using the cAck code of STxC\_FAIL. It may do so without asserting either DOE\_L or DWSEL\_H.

See Section 7.1.1.4.

### 7.1.1.7 BARRIER

The BARRIER transaction appears on the external interface as a result of an MB instruction. The acknowledgment of the BARRIER transaction tells the 21064 CPU that all invalidates have been supplied to it, and that any external write buffers have been pushed out to the coherence point. Any errors detected during these operations can be reported to the 21064 when the BARRIER transaction is acknowledged. The DEC 4000 CPU immediately acknowledges this transaction because it does not buffer block\_write transactions.

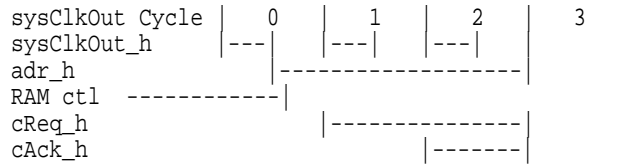


1. The BARRIER transaction begins. The 21064 places the command code for BARRIER onto the CREQ\_H outputs.
2. The external logic notices the BARRIER command, and because it has completed processing the command (it is not going to do anything), it places an acknowledge code on the CACK\_H inputs.
3. The 21064 detects the acknowledge on CACK\_H, and removes the command. The external logic removes the acknowledge code from CACK\_H. The cycle is finished.

## 7.1 Processor Transactions

### 7.1.1.8 FETCH

A FETCH transaction appears on the external interface as a result of a FETCH instruction. The transaction supplies an address to the external logic that DEC 4000 chooses to ignore and responds with an immediate acknowledge.



1. The CREQ\_H pins are always idle in the system clock cycle immediately before the beginning of an external transaction. The ADR\_H pins always change to their final value (with respect to a particular external transaction) at least one CPU cycle before the start of the transaction.
2. The FETCH transaction begins. The 21064 has placed the effective address of the FETCH on the address outputs. The 21064 places the command code for FETCH on the CREQ\_H outputs. The 21064 clears the RAM control pins (DATAA\_H[4..3], DATACEO\_H[3..0] and TAGCEO\_H) no later than one CPU cycle after the system clock edge begins the transaction.
3. The external logic notices the FETCH command, and because it has completed processing the command (it is not going to do anything), it places an acknowledge code on the CACK\_H inputs.
4. The 21064 detects the acknowledge on CACK\_H, and removes the address and the command. The external logic removes the acknowledge code from CACK\_H. The cycle is finished.

### 7.1.1.9 FETCHM

A FETCHM transaction appears on the external interface as a result of a FETCHM instruction. The transaction supplies an address to the external logic, which DEC 4000 chooses to ignore and responds with an immediate acknowledge. With the exception of the command code placed on CREQ\_H, the FETCHM transaction is the same as the FETCH transaction. Refer to Section 7.1.1.8.

## 7.1.2 Cacheable versus Non-Cacheable, and Allocate-Invalid

### Cacheable

Only memory-like locations are cached. Memory-like locations are defined as locations where address bit <33> is equal to 0. These locations are placed in the backup cache when it is enabled as a side effect of the processor issuing a READ\_BLOCK. They are also placed in the primary cache as the read data is acknowledged with OK.

### Non-Cacheable

Non-memory-like locations are *not* cached. These are defined as locations where address bit <33> is equal to 1. These locations are not placed in the backup cache, and the read data is acknowledged with OK\_NCACHE, or OK\_NCACHE\_NCHCK. Writes to non-cacheable space are restricted to aligned quadword accesses only. The quadword write data is presented to the system bus in the proper quadword associated with the address of the access. The data presented in the

## 7.1 Processor Transactions

other quadwords of the cache block (during that system bus cycle) is undefined. However the correct system bus parity is driven. No read merge occurs. All I/O locations are non-cacheable locations.

### Allocate-Invalid

When the ENB BACKUP CACHE INIT bit in the BCC register is clear, memory-like locations with address bit <33> equal to 0 and address bit <32> equal to 1, are treated as cacheable locations to be allocated in the backup cache as invalid. Read references to locations in this address space are intercepted by the C<sup>3</sup> and cause either an EXCHANGE, or a READ request on the system bus (depending on the state of the DIRTY bit of the cache block in question). Address bit <32> is masked off before the address is presented on the system bus to provide a legal DEC 4000 system bus address.

When the read data is returned, the cache block in the backup cache is marked invalid.

When the ENB BACKUP CACHE INIT bit in the BCC register is set, memory-like locations with address bit <33> equal to 0 and address bit <32> equal to 1, is treated as Cacheable locations to be allocated in the backup cache. Read references to locations in this address space are intercepted by the C<sup>3</sup> and cause the data found in the local CPU modules CSR's to be returned as the read data and subsequently filled into the backup cache. Address bit <32> is passed through the C<sup>3</sup> and presented to the system bus. The C<sup>3</sup> acknowledges the transaction and the other CPU module will not probe its backup cache. See the definition of the backup cache control and status register, and Section 11.2 for details.

This feature can be used to flush dirty entries from the cache, and/or to guarantee that all cache entries are marked invalid. Simple flushing of dirty entries is accomplished by setting the FORCE SHARED bit in the CBCTL register, and walking the Allocate-Invalid address space to flush the cache (a process at low IPL). When the full cache address space (nominally 1-4 MB) has been accessed, the cache is guaranteed to be clean; however not necessarily invalid. If the routine that flushes the cache operates at an IPL higher than any other processes can execute, when complete, the cache is clean and marked completely invalid.

---

### Note

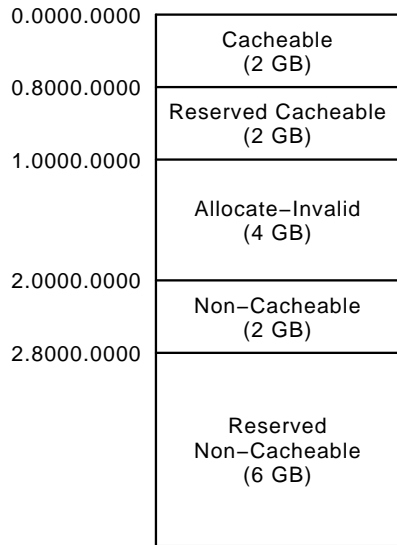
---

As the bus interface re-maps all references through allocate-invalid space by clearing address bit <32> and presenting the resulting address to the memory subsystem. Only privileged processes should be allowed to map Allocate-invalid space into their virtual address space.

---

Figure 7-1 shows the address map.

Figure 7–1 Address Space Map



## 7.2 System Bus Transactions

There are four system bus transaction types: READ, WRITE, EXCHANGE, and NUT. READ and WRITE transactions consist of a command/address cycle followed by two data cycles. The EXCHANGE transaction is used for dirty victim processing and consists of a command/address cycle followed by four data cycles, two write and two read. The NUT transaction consists of a command/address cycle only.

Because the system bus is based on a “snooping” protocol, every system bus transaction is monitored by all system bus participants. The memory modules can act only as responders or bystanders, and as such respond only to requests from bus commanders. The processor and I/O modules can act as either commanders, responders, or bystanders.

Table 7–2 System Bus Initiated Transactions

Transaction	Abbrv	System Bus Cycles	Activity
Write	WR	7	backup cache probe IF HIT update/invalidate IF update pull system bus SHARED, cache block SHARED = prev SHARED
Read	RD	7	Backup cache probe; Provide data if HIT dirty
Exchange	XD	7	Backup cache probe (READ); Provide data if HIT dirty
No-op Transaction	NUT	7	No operation

See Table 7–4 for a detailed description of the control flows for system bus initiated cycles.

## 7.2 System Bus Transactions

### 7.2.1 CPU as Commander

When one of the CPU module's processors requests data that does not reside on the CPU module, a request is passed to the system bus interface controller, and a system bus transaction is initiated.

### 7.2.2 CPU as Bystander

The CPU module is a bystander when some other system bus commander has requested data from a resource on the system bus that does not reside exclusively on the CPU module.

As a bystander, each backup cache controller may have to update or invalidate a stale datum, or provide dirty data to the system bus thereby preempting a memory module from returning stale data (for that transaction only).

### 7.2.3 CPU as Responder

The CPU module is a responder when some other system bus commander requests data from a system bus visible CPU module register.

## 7.3 Control Flow of CPU Module Transactions

The following sections detail the flow of the processor module transactions.

### 7.3.1 Processor Initiated Transactions

The CPU module backup cache controller provides backup cache control under the following circumstances:

- A backup cache miss occurs
- A LDxC execution
- A STxC execution
- A FETCH/FETCHM execution
- MB execution
- When a write to a shared cache block is detected

The behavior of the backup cache controller is based on the current processor cycle type, the state of the backup cache control bits, and the state of the system bus. The Table 7-3 illustrates the control flows for these cycles.

## 7.3 Control Flow of CPU Module Transactions

**Table 7–3 Processor Initiated Transactions Control Flow**

Cycle	Datum/Cache		Activity
	Status	Size	
Read HIT†	X‡	X	<ul style="list-style-type: none"> <li>Block read - Processor managed<sup>3, 2</sup></li> <li>Update duplicate tag store</li> </ul>
Write HIT	NOT Shared†	X	<ul style="list-style-type: none"> <li>Block written (DIRTY = TRUE) - processor managed<sup>3</sup></li> </ul>
	Shared	32 bytes	<ul style="list-style-type: none"> <li>Request system bus<sup>3, 4, 5</sup></li> <li>system bus WRITE cycle<sup>1, 5</sup></li> <li>Update Cache block (DIRTY = FALSE, SHARED = system bus shared during write)</li> <li>Relinquish system bus</li> <li>Acknowledge write</li> </ul>
	Shared	< 32 bytes	<ul style="list-style-type: none"> <li>Request system bus<sup>3, 4, 5</sup></li> <li>System bus WRITE cycle<sup>1, 5</sup> with merge</li> <li>Update cache block (DIRTY = FALSE, SHARED = system bus shared during write)</li> <li>Relinquish system bus</li> <li>Acknowledge write</li> </ul>

<sup>1</sup>See Section 5.5 for details.

<sup>2</sup>Processor checks data EDC.

<sup>3</sup>Processor checks tag store and control store parity.

<sup>4</sup>backup cache controller checks tag store and control store parity.

<sup>5</sup>backup cache controller checks data EDC.

†“Fast Cache” cycles, external logic does not intervene

‡X - Don't care

(continued on next page)

## 7.3 Control Flow of CPU Module Transactions

**Table 7–3 (Cont.) Processor Initiated Transactions Control Flow**

Cycle	Datum/Cache		Activity
	Status	Size	
Read MISS	No victim / Clean victim	X	<ul style="list-style-type: none"> <li>Request system bus<sup>3, 4</sup></li> <li>System bus READ cycle<sup>6</sup></li> <li>Cache block allocated<sup>2</sup> (SHARED = system bus share during read, DIRTY = FALSE)</li> <li>Relinquish system bus</li> <li>Acknowledge read</li> </ul>
	Dirty Victim	X	<ul style="list-style-type: none"> <li>Request system bus<sup>3, 4</sup></li> <li>System bus EXCHANGE cycle<sup>5, 6</sup></li> <li>Cache block allocated<sup>2</sup> (SHARED = system bus share during read, DIRTY = FALSE)</li> <li>Relinquish system bus</li> <li>Acknowledge read</li> </ul>
	Enable allocate disabled (BCC bit 0 clear)	X	<ul style="list-style-type: none"> <li>Request system bus</li> <li>If backup cache probe dirty system bus EXCHANGE cycle<sup>6</sup></li> <li>If backup cache probe not dirty system bus READ cycle<sup>6</sup></li> <li>Update cache block (VALID = NOT VALID)</li> <li>Relinquish system bus</li> <li>Acknowledge Read no cache</li> </ul>

<sup>2</sup>Processor checks data EDC.

<sup>3</sup>Processor checks tag store and control store parity.

<sup>4</sup>backup cache controller checks tag store and control store parity.

<sup>5</sup>backup cache controller checks data EDC.

<sup>6</sup>system bus interface controller checks system bus parity.

(continued on next page)

## 7.3 Control Flow of CPU Module Transactions

**Table 7–3 (Cont.) Processor Initiated Transactions Control Flow**

Cycle	Datum/Cache		Activity
	Status	Size	
Write MISS	No victim / Clean victim	< 32 bytes	<p>“Multiple-Arb operation”</p> <ul style="list-style-type: none"> <li>Request system bus<sup>3, 4</sup></li> <li>System bus READ cycle<sup>6</sup></li> <li>Cache block allocated (SHARED = system bus share during read, DIRTY = FALSE)</li> <li>Continue to request system bus</li> <li>Write data merged with cache block (SHARED = SHARED, DIRTY = FALSE)</li> <li>System bus WRITE cycle<sup>1, 5</sup></li> <li>Update backup cache</li> <li>Relinquish system bus</li> <li>Acknowledge write</li> </ul>
	Dirty victim	< 32 bytes	<p>“Multiple-Arb Operation”</p> <ul style="list-style-type: none"> <li>Request system bus<sup>3, 4</sup></li> <li>System bus EXCHANGE cycle<sup>5, 6</sup></li> <li>Cache block allocated (SHARED = system bus share during read, DIRTY = FALSE)</li> <li>Continue to request system bus</li> <li>Write data merged with cache block (SHARED = SHARED, DIRTY = FALSE)</li> <li>System bus WRITE cycle<sup>1, 5</sup></li> <li>Update backup cache</li> <li>Relinquish system bus</li> <li>Acknowledge write</li> </ul>

<sup>1</sup>See Section 5.5 for details.

<sup>3</sup>Processor checks tag store and control store parity.

<sup>4</sup>backup cache controller checks tag store and control store parity.

<sup>5</sup>backup cache controller checks data EDC.

<sup>6</sup>system bus interface controller checks system bus parity.

(continued on next page)



## 7.3 Control Flow of CPU Module Transactions

**Table 7–3 (Cont.) Processor Initiated Transactions Control Flow**

Cycle	Datum/Cache		Activity
	Status	Size	
	Enable Allocate Disabled (BCC bit 0 clear)	< 32 bytes	<p>“Multiple-Arb operation”</p> <ul style="list-style-type: none"> <li>Request system bus</li> <li>If backup cache probe dirty system bus EXCHANGE cycle<sup>6</sup></li> <li>If backup cache probe not dirty system bus READ cycle<sup>6</sup></li> <li>Update Cache block (VALID = NOT VALID)</li> <li>Continue to request system bus</li> <li>System bus WRITE cycle from merge buffer</li> <li>Relinquish system bus</li> <li>Acknowledge write</li> </ul>
	X	32 bytes	<ul style="list-style-type: none"> <li>Request system bus<sup>3,4</sup></li> <li>System bus WRITE cycle<sup>5</sup></li> <li>Relinquish system bus</li> <li>Acknowledge write</li> </ul>
LOAD LOCK	X	X	<ul style="list-style-type: none"> <li>Request system bus</li> <li>Probe backup cache<sup>4</sup></li> <li>Same as generic read except if reference to cacheable address space lock bit set after arbitrating for the system bus; otherwise no change to lock bit.<sup>6</sup> (lock bit VALID = TRUE, lock address = access address) (IF (backup cache hit) change system bus read to NUT.)<sup>2</sup></li> <li>Relinquish system bus</li> <li>Acknowledge read</li> </ul>

<sup>2</sup>Processor checks data EDC.

<sup>3</sup>Processor checks tag store and control store parity.

<sup>4</sup>backup cache controller checks tag store and control store parity.

<sup>5</sup>backup cache controller checks data EDC.

<sup>6</sup>system bus interface controller checks system bus parity.

(continued on next page)

## 7.3 Control Flow of CPU Module Transactions

Table 7–3 (Cont.) Processor Initiated Transactions Control Flow

Cycle	Datum/Cache		Activity
	Status	Size	
STORE COND - Cacheable	X	X	<ul style="list-style-type: none"> <li>Request system bus</li> <li>Probe backup cache <sup>4</sup></li> <li>Same as generic write to except fails IF (Lock bit VALID = FALSE). (IF (write failed due to lock bit not set) change system bus write to NUT.)</li> <li>Relinquish system bus</li> <li>Acknowledge write (conditionally)</li> </ul>
STORE COND. I/O	X	X	<ul style="list-style-type: none"> <li>Request system bus</li> <li>If the CUCERR_L signal is asserted by the responder module during the first cycle 4 of a system bus transaction when the DECchip CPU is performing a store conditional to non-cacheable address space, the write does not complete, and the store conditional failed indicators are signaled to the processor.</li> <li>Relinquish system bus</li> <li>Acknowledge write (conditionally)</li> </ul>

<sup>4</sup>backup cache controller checks tag store and control store parity.

### 7.3.2 System Bus Initiated Transactions

The CPU module system bus interface controller provides backup cache control when the system bus is active.

The behavior of the system bus interface controller is determined by the system bus transaction, and the state of the backup cache control bits. Table 7–4 illustrates the control flows for these cycles. The activity column shows the activity that occurs after the processor has relinquished its ownership of the backup cache and the backup cache TAG probe results are available.

## 7.3 Control Flow of CPU Module Transactions

**Table 7–4 System Bus Initiated Transactions Control Flow**

Cycle	Probe Result	
	Status	Activity
Read HIT backup cache	Dirty	Assert system bus DIRTY and SHARED set SHARED bit in tag control store if not already set, cache block remains DIRTY. Provide Data to system bus <sup>1,2</sup>
	Clean	Assert system bus SHARED <sup>1</sup> . Set SHARED bit in tag control store if not already set.
Read HIT duplicate tag store, or lock address (if valid)	Clean	Assert system bus SHARED.
Write HIT	X‡	IF (location found in P-Cache or Commander ID is I/O module or in ARB) Update backup cache (Clean, shared = no change) and assert system bus SHARED ELSE Invalidate backup cache location (clean, not shared) <sup>1,3,4</sup> . IF (HIT address lock register) clear address lock.
		or  IF (I/O conditional update set) IF (location found in P-Cache) Update backup cache (Clean, shared = no change) and assert system bus SHARED ELSE Invalidate backup cache location (clean, not shared) <sup>1,3,4</sup> . IF (HIT address lock register) clear address lock.
Exchange HIT	Dirty	Assert system bus DIRTY and SHARED. Address Lock not changed. Provide Read data to system bus <sup>123</sup> .
	Clean	Assert system bus SHARED <sup>13</sup> . Address Lock not changed.
Read MISS	NA†	NOP <sup>13</sup> . IF (HIT Address Lock register) assert system bus SHARED.
Write MISS	NA	NOP <sup>134</sup> IF (WRITE address hits the address lock) clear address lock.

<sup>1</sup>Backup cache controller checks tag store and control store parity.

<sup>2</sup>Backup cache controller checks data EDC.

<sup>3</sup>System bus interface controller checks system bus parity.

<sup>4</sup>System bus interface controller checks duplicate tag store parity.

†Not applicable

‡ Don't care

(continued on next page)

## 7.3 Control Flow of CPU Module Transactions

**Table 7–4 (Cont.) System Bus Initiated Transactions Control Flow**

Cycle	Probe Result	
	Status	Activity
Exchange MISS	NA	NOP <sup>13</sup> . IF (READ address HIT Address Lock register ) assert system bus SHARED. IF (EXCHANGE write address hits the address lock) clear address lock.
NUT	X	NOP <sup>3</sup>

<sup>1</sup>Backup cache controller checks tag store and control store parity.

<sup>3</sup>System bus interface controller checks system bus parity.

**Note**

The backup cache TAG probe during an exchange cycle is for the Read data ONLY.

---

## Cache Invalidate Management

As memory locations in the system are updated, it may become necessary to invalidate cached copies of these locations to maintain cache coherence. The reasons for invalidation are divided into two major categories. Those due to processor, and those due to system bus activity.

The primary instruction stream cache is a virtual cache and as such all invalidate activity are completely under software control.

### 8.1 Processor Caused Invalidates

The primary instruction stream cache is a virtual cache and its coherence is managed by the system software.

The primary data stream cache is a physical cache and is managed as a subset of the backup cache. The backup cache controller may victimize a location in the backup cache as a result of an instruction stream read miss, or a masked write read-merge operation. Thus to maintain the subset rule, the primary data cache is invalidated whenever an instruction stream (read allocate) or read-merge victim address hits the duplicate tag store.

System software should disable the primary data stream cache while flushing the backup cache. When flushing is complete, the primary instruction stream cache should also be flushed. This is done to guarantee that the primary caches remain a strict subset of the backup cache.

### 8.2 System Bus Caused Invalidates

Table 8-1 indicates the cache invalidate policies for system bus generated cycles.

## 8.2 System Bus Caused Invalidates

**Table 8–1 Invalidate Management System Bus Caused**

Cycle	Backup Cache	P-cache
Read	NOP	NOP
Write	If (found in backup cache and (location NOT found in P-Cache and commander ID is not I/O subsystem )) invalidate backup cache location. †	IF (found in primary D-cache) invalidate D-cache.
	IF (conditional invalidates for I/O enabled) IF (found in backup cache and location NOT found in P-Cache) invalidate backup cache location.†	IF (found in Primary D-cache) invalidate D-cache.
	IF (ENB COND INVALIDATE cleared) Invalidate backup cache location.†	IF (found in Primary D-cache) invalidate D-cache.
Exchange	NOP	NOP

†See Section 5.6 which discusses the P-Cache duplicate tag store.

## Exceptions and Interrupts

When an interrupt or exception occurs, the processor drains the pipeline, loads the PC into the EXC\_ADDR IPR, and dispatches to one of the PAL exception routines. If multiple exceptions occur, the 21064 CPU dispatches to the highest priority PAL entry point. Refer to *The Alpha Architecture Reference Manual* for details.

### 9.1 Processor-Generated Interrupts

There are two types of processor generated interrupts: general exceptions and machine check exceptions.

#### General Exceptions

General exceptions are caused by user malfeasance (that is, arithmetic traps or attempted illegal opcode execution), or by normal system operation (for example, translation buffer miss). The list of general exceptions is given in Table 9-1.

**Table 9-1 General Exception Isolation Matrix**

PAL entry	Cause	Cause isolation
External Signal RESET L asserted	RESET	NR†
ARITH	Arithmetic exception (divide by 0 etc.)	EXC_SUM IPR
DTB_MISS	Data translation buffer miss	NR†
UNALIGN	D-Stream unaligned reference	NR†
DTB_FAULT	Remaining D-Stream memory management errors.	NR†
ITB_MISS	Instruction translation buffer miss	NR†
ITB_ACV	Instruction stream access violation	NR†
CALLPAL	CALLPAL instruction executed	Entry based on EXC_ADDR- >instruction[7..0]
OPDEC	Attempted execution of a reserved or privileged opcode	NR† EXC_ ADDR points to instruction
FEN	Floating-point operation attempted with floating point unit disabled, under or overflows, inexact errors, div 0, invalid ops	IPR EXC_SUM

†Isolation not required as PALcode entry identifies cause.

## 9.1 Processor-Generated Interrupts

### Machine Check Exceptions

Machine-check exceptions are special exceptions that are caused by errors in the hardware system. They all dispatch to the general MCHK entry. Refer to Table 9–2 for the machine check isolation matrix.

**Table 9–2 Machine Check Isolation Matrix**

PAL entry	Cause	Cause Isolation
MCHK	BIU detects backup cache tag store parity error	BIU_STAT IPR
MCHK	BIU detects backup cache tag control store parity error	BIU_STAT IPR
MCHK	BIU detects backup cache EDC data store error	BIU_STAT IPR
MCHK	System external transaction terminated with CACK_HERR (Hard Error)	BIU_STAT IPR

#### Note

When a system error occurs, all caches in the system should be examined to make sure that a location has not been purposely marked with a “bad” EDC code.

### 9.1.1 Exception Handling

Most exceptions have unique entries through which control flows. This allows easy identification, and fast dispatch to the appropriate system code. Exceptions that fall into this category are listed below:

- Reset
- Arithmetic
- DTB Miss
- Unaligned Reference
- Data Access Fault
- ITB Miss
- ITB Access Violation
- Reserved Opcode Fault
- Floating Point Operation

They can occur relatively frequently as part of normal system operation.

The class of exceptions that occur as a result of hardware system errors are called machine checks. These result when an uncorrectable system error is detected during the processing of a data request.

Generally, exceptions are handled as follows by PALcode. First the PALcode determines the cause of the exception. If possible, it corrects the problem and returns the system to normal operation. If a problem is not correctable, or



## 9.1 Processor-Generated Interrupts

error logging is required, control is passed through the SCB to the appropriate exception handler.

### 9.1.1.1 PAL Priority Level

Table 9–3 is the prioritized list of the exceptions that can occur on a DEC 4000 system. This list goes from the highest to the lowest priority.

**Table 9–3 Exception Priority/PAL Offset/SCB Offset/IPL**

Priority	Description	PAL Offset (h)	SCB Offset (h)	IPL (h)
1	HALT	0000	NA	NA
2	Machine Check	0020	0660	31
3		0020	0670	31
4	DTB Miss PAL	09E0	NA	NA
5	DTB Miss NATIVE	08E0	NA	NA
6	ITB Miss	03E0	NA	NA
7	ITB Access Violate	07E0	0080-00C0	X
8	Data Access Fault	01E0	0080-00C0	X
9	Unaligned Data	11E0	0300-03F0	X
10	Arithmetic NFPU	17E0	0010	X
11	Arithmetic ARTH	17E0	0200	X
12	Reserve Opcode Fault	13E0	0420	X

### 9.1.1.2 PALcode 0020 Entry Characteristics

Exceptions occur as a direct † result of the detection of errors during the execution of the current instruction.

The PALcode found at the PALentry 0020<sub>16</sub> must sift through the system error information and determine the severity of the error. In some cases the PALcode at this entry may correct the error and allow the machine to continue execution without any higher level software intervention.

### 9.1.1.3 Parse Tree PALcode Entry 0020<sub>16</sub>

Due to the nature of backup cache errors, the PALcode executed upon entry is restricted to “read-only” behavior shown as broken lines in Figure 9–1. The following procedure should be followed. Once it has been determined that a backup cache error has not occurred, the restrictions are lifted.

Whenever a backup cache error occurs, the state of the backup cache is effectively frozen. However memory system coherence is still maintained. This is done by having cache allocation suspended whenever an error is detected. See Section 5.4.1. Backup cache probing by the processor should also be disabled by PAL code by writing a 0 to the BC\_ENA bit of the BIU\_CTL register.

† Except during “disconnected” write operations which occur as a result of masked write operations causing two consecutive system bus transactions. If an error is detected between the completion of the first and second system bus transactions, (due to an unrelated intervening system bus transaction) a machine check occurs and the information relating to the actual error is logged in the C<sup>3</sup>-CSRs. Software can determine when this occurs by comparing the address in the 21064-EXC\_ADDR register to the address locked in the C<sup>3</sup>-CSRs.

## 9.1 Processor-Generated Interrupts

---

### Note

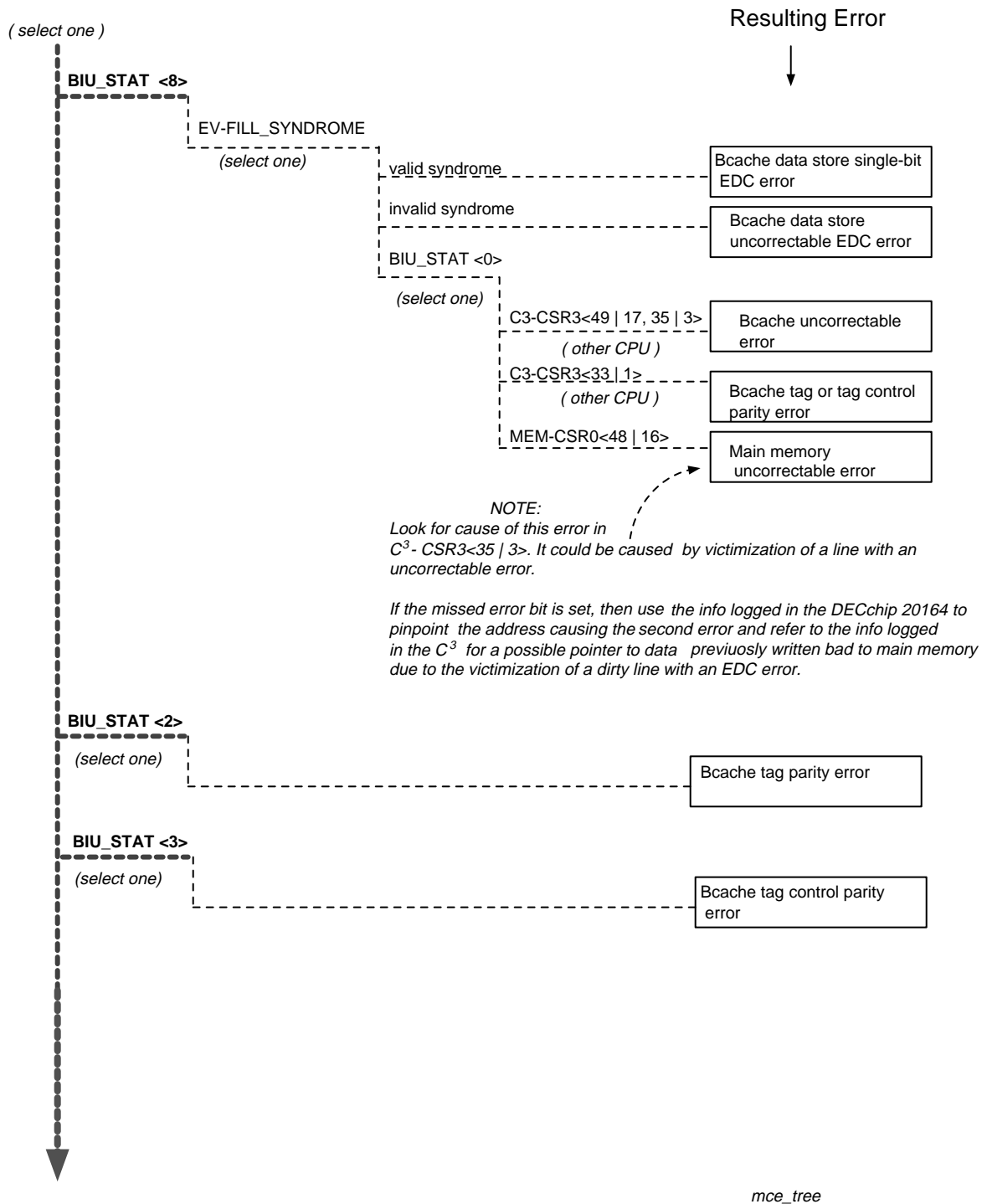
---

Reading of modifiable memory like data areas should be avoided as these locations could be dirty in the disabled cache and thus produce an incoherent access.

---

## 9.1 Processor-Generated Interrupts

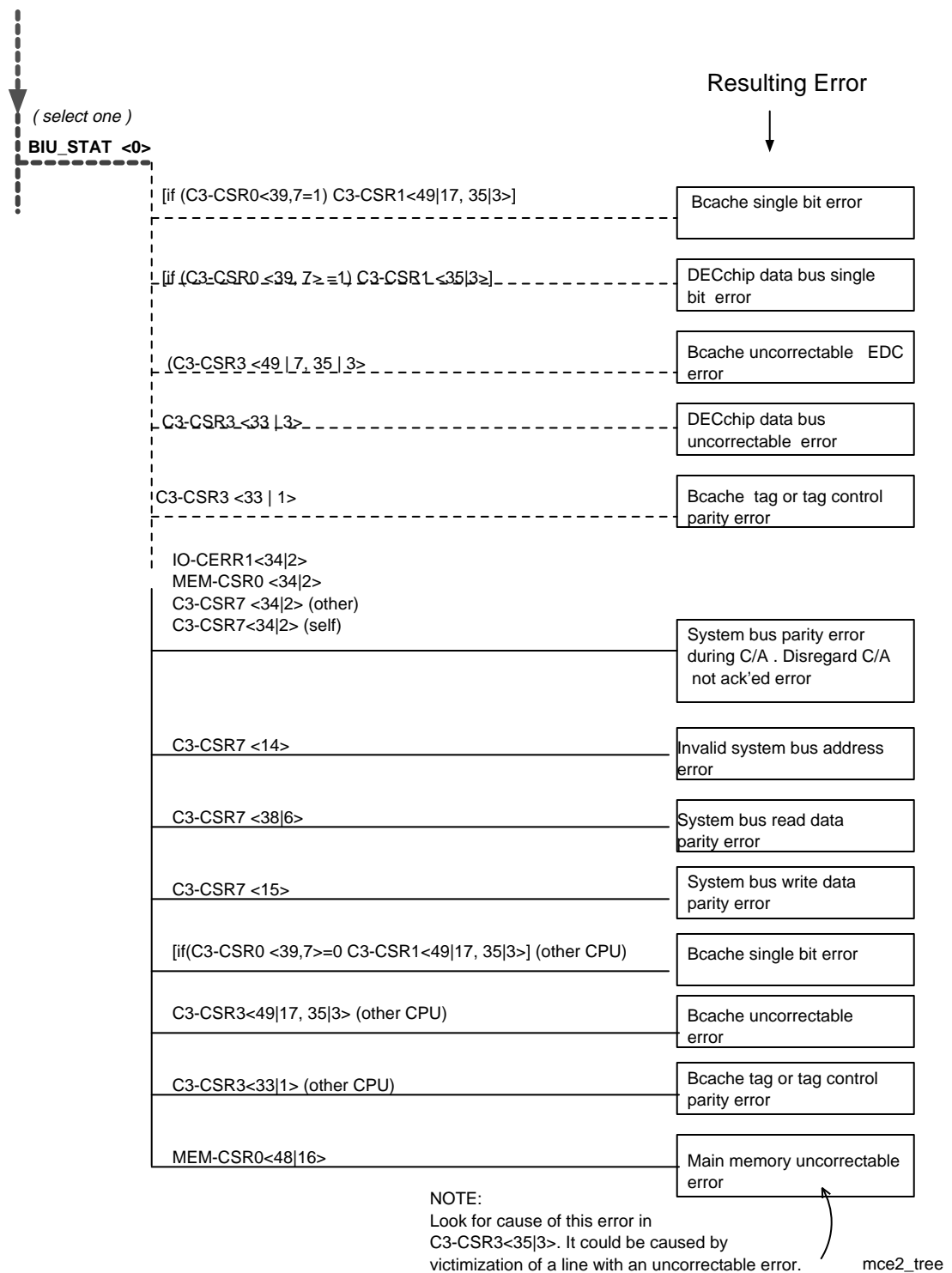
Figure 9–1 Machine-check Exception Parse Tree



(continued on next page)

## 9.1 Processor-Generated Interrupts

Figure 9–1 (Cont.) Machine-check Exception Parse Tree



### 9.1.1.4 PAL Routine Behavior

#### 9.1.1.4.1 Backup Cache Tag Parity Error

##### **Current CPU Backup Cache Tag Parity Error Processor Detected**

This error is restricted to reads only. PAL must scrub the parity error from the tag store using the standard backup cache initialization procedure, and if the DIRTY bit on the cache block in question is set, a SYSTEM FATAL Error should be signaled to the system software. Otherwise, only error logging is required.

See Chapter 10, Section 10.2.1 for details.

#### 9.1.1.4.2 Backup Cache Tag Control Parity Error

##### **Current CPU Backup Cache Tag Control Parity Error Processor Detected**

This error is restricted to reads. PAL must scrub the parity error from the Tag Control store using the standard backup cache initialization procedure. This error is FATAL to the referenced cached location context.

See Chapter 10, Section 10.2.1 for details.

#### 9.1.1.4.3 Backup Cache Data Single Bit EDC Error

##### **Current CPU Backup Cache EDC Error Processor Detected**

This error is restricted to reads. PAL must scrub the EDC error from the data store using the 21064 backup cache force hit mode. The instruction that encounters the error should be restarted. A CRD interrupt should be dispatched.

There is no way to determine if the failure was due to a backup cache SRAM fault or a fault on the bus between the C<sup>3</sup> and the DECchip CPU chip. If the location found in the backup cache does not have the same address as the address found in the CPU chip's FILL\_ADDR register, then the error occurred on the data bus between the CPU chip and the C<sup>3</sup> chip. Generally, this type of error indicates a hard fault. However if a simple test of the interface passes, the error may be recoverable after scrubbing affected locations and restarting the failing instruction.

See Chapter 10, Section 10.2.2 for details.

#### 9.1.1.4.4 Backup Cache Data Uncorrectable EDC Error

##### **Current CPU Backup Cache EDC Error Processor Detected**

This error occurs on reads only. PALcode must scrub the EDC error from the Data store using the 21064 backup cache force Hit mode. If the DIRTY bit on the cache block is set, the error is FATAL to the context the cached location is referenced in. Otherwise the error should be scrubbed from the cache and the failing instruction restarted.

If the location found in the backup cache does not have the same address as the address found in the EV-FILL\_ADDR register, this indicates the error occurred on the data bus between 21064 CPU chip and the C<sup>3</sup> chip. Generally, this type of error is indicative of a hard fault. However, if a simple test of the interface passes, the error may be recoverable after scrubbing affected locations and restarting the failing instruction.

## 9.1 Processor-Generated Interrupts

---

### Note

---

If the EDC error has a syndrome of 1F indicating an uncorrectable error it means that the data store of the cache was written with an intentionally bad EDC pattern as a result of the CUCERR L signal being asserted during a backup cache read fill or write update.

---

See Chapter 10 and Section 10.2.2 for details.

#### 9.1.1.4.5 Backup Cache Data Single Bit EDC Error

##### Current CPU Backup Cache EDC Error C<sup>3</sup> Detected

See Section 9.1.1.4.3.

#### 9.1.1.4.6 Backup Cache Data Uncorrectable EDC Error

##### Current CPU backup cache EDC Error C<sup>3</sup> Detected

See Section 9.1.1.4.4.

**9.1.1.4.7 21064 Data Bus Single-bit EDC Error** EDC error occurred on the bus as the data was being driven from the DECchip 21064 CPU chip to the C<sup>3</sup> chip. Generally this type of error is indicative of a hard fault. However if a simple test of the interface passes, this indicates the error may be recoverable after scrubbing affected locations and restarting the failing instruction.

See Section 9.1.1.4.3.

**9.1.1.4.8 21064 Data Bus Uncorrectable EDC Error** EDC error occurred on the bus as the data was being driven from the 21064 CPU chip to the C<sup>3</sup> chip. Generally this type of error is indicative of a hard fault. However, if a simple test of the interface passes, the error may be recoverable after scrubbing affected locations and restarting the failing instruction.

See Section 9.1.1.4.4.

#### 9.1.1.4.9 Backup Cache Tag or Tag Control Parity Error

##### Current CPU Backup Cache Parity Error C<sup>3</sup> Detected

First, the actual cause of the error should be determined. This is done by calculating the expected Tag Control and Tag parity based on the data latched in the backup cache Uncorrectable Error Register, and the backup cache Uncorrectable Error Address Register. The result indicates whether the error was a Tag Control Store Error, a Tag Store error, or both.

##### Tag Control Store Errors

Restricted to reads only, PALcode must scrub the parity error from the tag control store using the standard backup cache initialization procedure. This error is FATAL to the context the cached location is referenced in.

##### Tag Store Errors

Restricted to reads only, PAL must scrub the parity error from the Tag store using the standard backup cache initialization procedure, and if the DIRTY bit on the cache block in question was set, a SYSTEM FATAL error should be signaled to the system software. Otherwise only error logging is required.

See Chapter 10 and Section 10.2.1 for details.

## 9.1 Processor-Generated Interrupts

If a tag parity error occurs where an even number of tag bits change state, during victimization of a dirty cache block, it is possible to generate a WRITE DATA not acknowledged error (causing a machine check). When this occurs, a SYSTEM FATAL error should be signaled to the system software. See Section 10.4.3 for details.

**9.1.1.4.10 System Bus Parity Error** This error is based on logged information and simple R/W to devices on system bus determine nature of failure. Disable failing module if possible and restart execution otherwise system FATAL.

See Table 10–6 for details.

**9.1.1.4.11 Invalid System Bus Address** This error occurs when a process has mapped physical addresses that do not exist in the system, or if an error on an address bus or a “double” bit error has occurred in a backup cache Tag Store. PALcode should verify the error was a legitimately invalid address as a result of incorrect mapping and pass control to the access violation exception handler. If the address logged is a legitimate physical address, and no system bus parity errors have been logged, it is SYSTEM FATAL. (Unprotected address bus or double bit parity protected address bus error)

A C/A Parity Error may result in the C/A NOT ACK'ED bit being set in one of the commander CSR's. This is because a responder does not acknowledge a C/A that has bad parity.

See Table 10–6 for details.

**9.1.1.4.12 Other CPU Errors** When a Machine Check is initiated as a result of errors on the other CPU module, that module must correct the error(s). Backup cache locations can be scrubbed only by using standard backup cache initialization mechanisms, and these require residence on the CPU local to the failing backup cache. The other CPU is notified in the case of these errors automatically via the HARDWARE ERROR INTERRUPT. Refer to Section 9.2.1 for details.

**9.1.1.4.13 Main Memory Uncorrectable EDC Errors** Main memory uncorrectable EDC errors are detected only when a memory module responds to a system bus read. The severity of this error depends on the context of the processes that reference it. In USER space it's PROCESS FATAL, in SYSTEM space it's SYSTEM FATAL.

## 9.2 Non-processor Generated Interrupts

Hardware interrupts are caused by hardware activity that requests the attention of the processor. Every processor in the DEC 4000 system is configured as shown in Table 9–4.

**Table 9–4 Hardware Interrupt Configuration**

HIR†	Vector	Description
5	None	System Event Interrupt
4	None	Interval Timer Interrupt - 976.5625 microseconds

†Hardware interrupt request - found in the 21064 HIRR IPR

(continued on next page)

## 9.2 Non-processor Generated Interrupts

**Table 9–4 (Cont.) Hardware Interrupt Configuration**

HIR†	Vector	Description
3	None	Inter-processor Interrupt
2	Fbus+ interrupt register‡	Futurebus+ interrupt
1	Local interrupt register‡	Local I/O interrupt
0	None	Hardware error

†Hardware interrupt request - found in the 21064 HIRR IPR

‡Both the Fbus+ and local interrupt request registers are accessed over the system bus and reside on the I/O module. Refer to the I/O Module Chapter for further details.

### 9.2.1 Interrupt Handling

All system interrupts are funneled through the “Interrupt” PAL entry defined as 21064 IPR PAL\_BASE + “00E0<sub>16</sub>”. From there the appropriate IPL is set and the system is interrogated to determine the cause of the interrupt. When the cause has been determined, the associated SCB offset is added to the SCB base address, and control is passed to that Interrupt Service Routine.

#### 9.2.1.1 PAL Priority Level

Table 9–5 shows the prioritized list of the interrupts that can occur on a DEC 4000 System. This list goes from the highest to the lowest priority interrupts.

**Table 9–5 Interrupt Priority/SCB Offset/IPL**

PAL Priority	Description	SCB Offset (h)	SRM IPL (h)	PAL IPL (d)
1	Hardware 5 -HLT	0000	31	20
2	Hardware 5 -PWF	0640	30	20
3	Hardware 0 -HRD	0660-0670	31	20
4	Hardware 0 -CRD	0620-0630	20	20
5	Hardware 4	0600	22	20
6	Hardware 3	0610	22	20
7	Hardware 1	800-1FF0	14	20
8	Hardware 2	800-1FF0	14	20
9	Serial Line	800-1FF0	14	20
10	Performance Counter 0	0650	14	20
11	Performance Counter 1	0650	14	20
12	Software 1-15	0500-05F0	1-0F	20
13	Asynchronous System Trap	0240-0270	2	20

#### 9.2.1.2 PALcode 00E0 Entry Characteristics

Various system status and error conditions are reported using one of the many interrupts that cause entry into the PAL interrupt entry point. Due to the nature of backup cache errors, the PALcode executed upon entry is restricted to read-only behavior.

Once it is determined that a backup cache error has not occurred, the restrictions are lifted.



## 9.2 Non-processor Generated Interrupts

Whenever a backup cache error occurs, the state of the backup cache is effectively frozen. However, memory system coherence is still maintained. This is done by having cache allocation suspended whenever an error is detected. See Section 5.4.1. PALcode should also disable the backup cache probing by the processor by writing a "0" to BC\_ENA bit of the BIU\_CTL register.

---

### Note

---

Reading of modifiable memory like data areas should be avoided because these locations could be dirty in the disabled cache and produce an incoherent access.

---

### 9.2.1.3 Parse Tree PALcode Entry 00E0<sub>16</sub>

---

### Parse Diagram Note

---

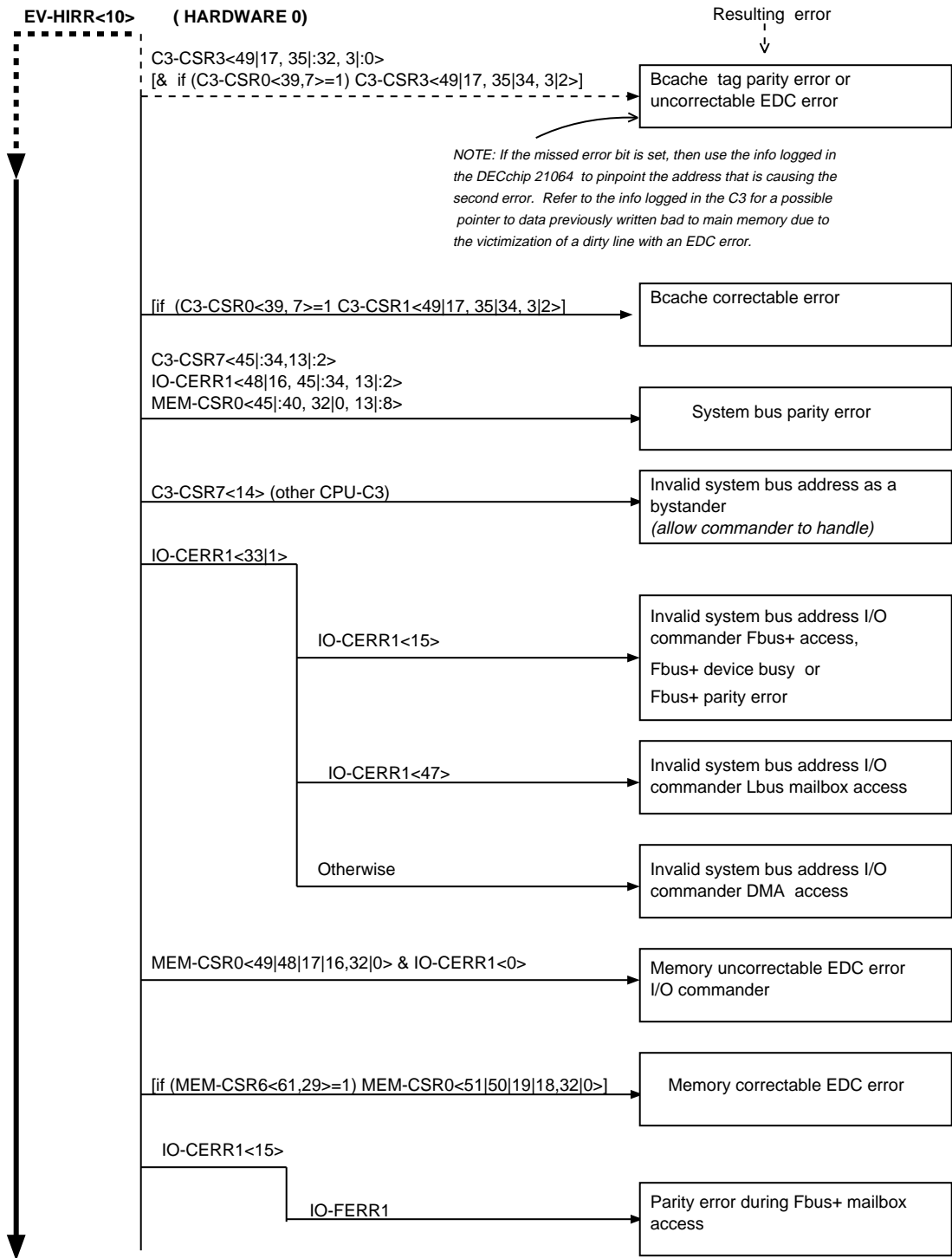
Flow identified with broken line indicates "Read-Only" behavior restriction.

---

C<sup>3</sup> CSR accesses are local except where specified, MEM CSR accesses are to all memory modules in the system. Figure 9–2 shows the interrupt parse tree.

## 9.2 Non-processor Generated Interrupts

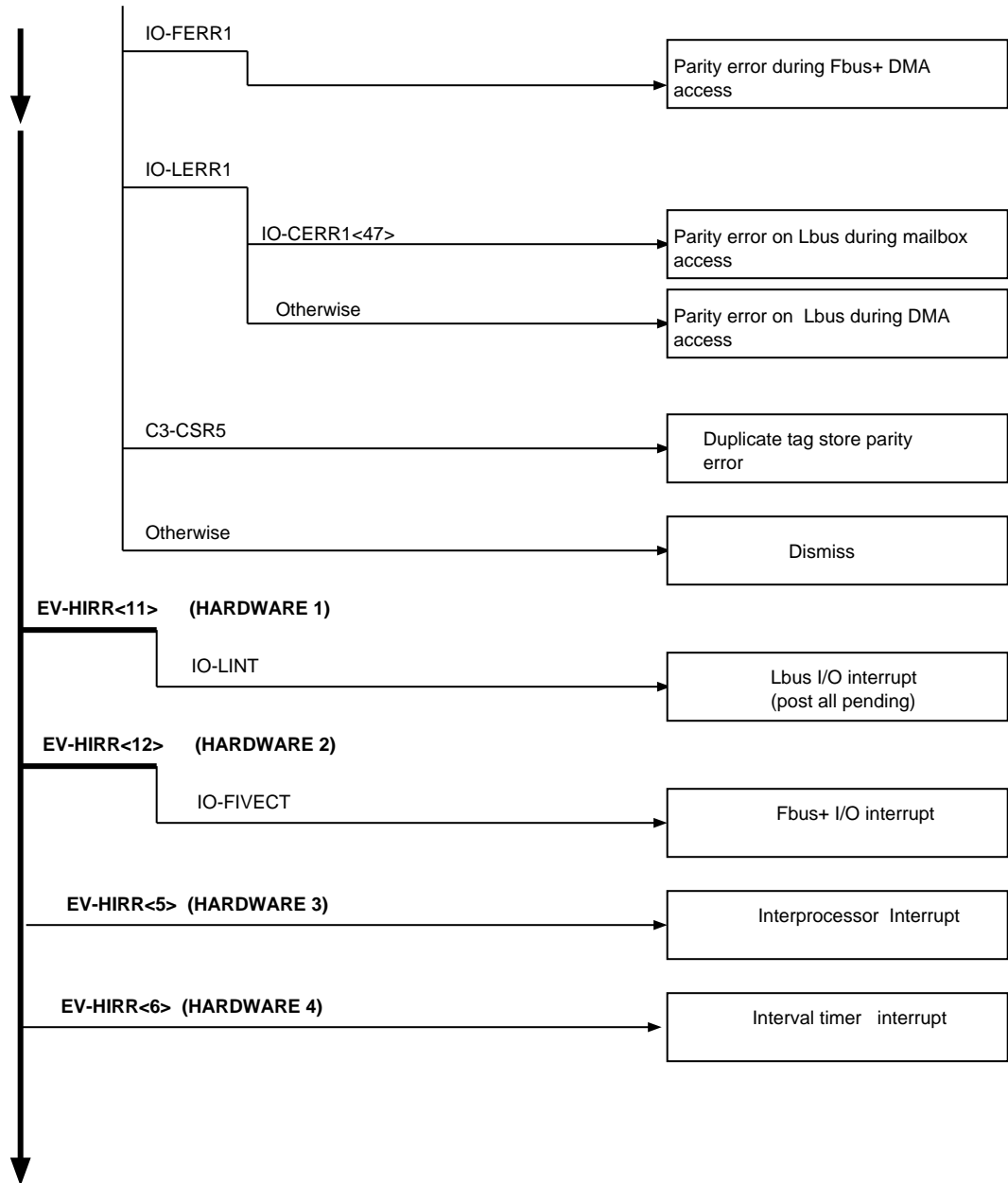
Figure 9–2 Interrupt Parse Tree



(continued on next page)

## 9.2 Non-processor Generated Interrupts

Figure 9–2 (Cont.) Interrupt Parse Tree

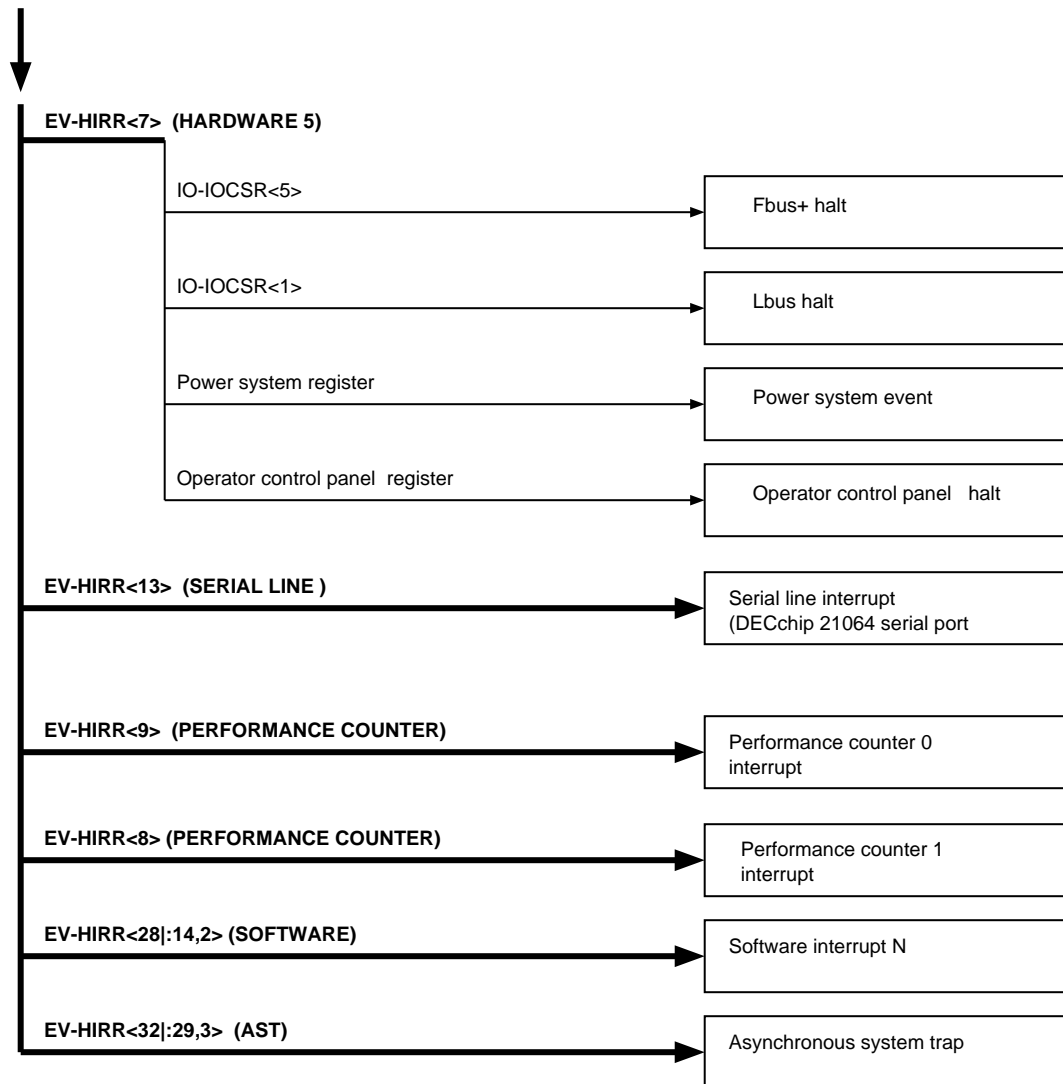


int1\_tree

(continued on next page)

## 9.2 Non-processor Generated Interrupts

Figure 9–2 (Cont.) Interrupt Parse Tree



int2\_tree

## 9.2 Non-processor Generated Interrupts

### 9.2.1.4 Hardware 0 Hardware Error

Interrupts generating a Hardware Interrupt 0 are caused by the detection of hardware errors on the CPU, I/O, Memory modules, and/or the system bus. These errors consist of RAM array correctable and uncorrectable errors as well as bus transport and protocol errors. Correctable error interrupts are individually maskable at each module's detection point. Before servicing the hardware error interrupt, it should be cleared in the local System Interrupt Clear register.

#### **Backup Cache Tag Parity or Uncorrectable EDC Error**

A given CPU module must scrub its own backup cache Tag Parity and Uncorrectable EDC errors. If the error causing this interrupt is a result of this module initiating a transaction, it also causes a machine check exception and the processing of the error is left up to the machine check handler. If however this node was not the transaction initiator (system bus probe), the interrupt should initiate the scrubbing/logging process.

Parity errors in the tag or tag control stores of a backup cache can be scrubbed only by using the FORCE EDC/CONTROL and SDV bits of the BCC CSR.

If a tag, or tag control parity error is detected while scrubbing dirty entries from the backup cache using the Allocate Invalid Address Space, the expected location is not scrubbed. If an uncorrectable data error is encountered, the data is written to the system bus coincident with the assertion of the CUCERR L signal. This causes the location to be written into Main Memory with a bad EDC code. The tag control store of the backup cache location in question is not updated.

#### **Backup Cache Single Bit EDC Error**

A given CPU module must scrub its own backup cache single bit EDC errors. If the error causing this interrupt is a result of this module initiating a transaction, and if EDC correction is disabled in the backup cache control register, it causes a machine check exception and the processing of the error is left to the machine check handler. If however this node was not the transaction initiator, (system bus probe) and EDC correction was disabled, the interrupt should initiate the scrubbing/logging process.

When scrubbing dirty entries from the backup cache using the allocate invalid address space, if a tag, or tag control parity error is detected, the expected location is not scrubbed. If an uncorrectable data error is encountered, the data is written to the system bus coincident with the assertion of the CUCERR L signal. This causes the location to be written into main memory with a bad EDC code. The tag control store of the backup cache location in question is not updated.

When EDC correction is enabled, no machine checks occur for this error, so the interrupt handler is responsible for scrubbing/logging errors that occur in its own cache.

As compared with hardware error correction, this approach is vulnerable to single-bit errors that may occur during Instruction stream reads of the PALcode machine check handler, to single-bit errors that occur in multiple quadwords of a cache fill block, and to single-bit errors that occur as a result of multiple silo'ed load misses.

## 9.2 Non-processor Generated Interrupts

### System Bus Parity Error

System bus parity errors indicate that a node on the system bus has a bad driver or receiver, or a problem exists in the physical interconnect. It is unlikely that this is a correctable error so the handler should attempt to discover which node is bad, and if possible disable it so it won't interfere with normal bus operation.

If no Uncorrectable error can be detected, retrying the operation causing the error maybe help to isolate it.

Generally when an uncorrectable error of this type is detected it is system fatal.

A C/A Parity Error may result in the C/A NOT ACK'ED bit being set in one of the commander CSR's. This is because responders do not acknowledge a C/A that has bad parity. See Section 9.1.1.4.10 for details.

### Invalid System Bus Address Bystander

When an invalid system bus address is broadcast by a CPU node, and not acknowledged, the system bus C\_ERR L signal is asserted. The CPU that initiated the transaction reports a machine check and receives a Hardware Error Interrupt. The bystander CPU receives only the Interrupt. The transaction initiator should handle the error logging and recovery for this error. This results in an access violation. See Section 9.1.1.4.11.

### Invalid System Bus Address I/O Commander Futurebus+/Local MBX, DMA

When an invalid system bus address is broadcast by the I/O node and not acknowledged, the system bus C\_ERR L signal is asserted. Both CPU's receive a hardware error interrupt and one should be designated to handle the error logging and recovery.

Information found in the IO-CERR1 register indicates whether this error occurred as a result of mailbox operations or DMA accesses and the logging and recovery can be handled differently for each.

### Memory Uncorrectable EDC Error I/O Commander

When an uncorrectable EDC error is detected while the I/O module is the system bus commander, the I/O module asserts the C\_ERR L signal causing the CPU modules to receive a hardware error interrupt. When this occurs, one CPU should be designated to handle the error logging and recovery. This may or may not be SYSTEM FATAL.

### Memory Correctable EDC Error

When a correctable EDC error is detected, the memory module asserts the C\_ERR L signal if the ENABLE CRD reporting bit is set in memory module CSR6. This, in turn causes the CPU module(s) to receive a hardware error interrupt. When this occurs, one CPU should be designated to handle the scrubbing and logging activity.

### Parity Errors Futurebus+ (Fbus+)

Read IO-FERR2 for address, IO-FMBPR for mailbox address, read contents at mailbox address for error status if the IO-CERR1 bit 15 is set, otherwise read FERR2 for failing DMA address.

A parity error on the Fbus+ most likely indicates a hard failure of either an Fbus+ adapter or the I/O module's Futurebus+ transceivers. When this error occurs, the system software should attempt to determine which adapter is failing and disable it from interfering with normal bus operation. Both CPU modules receive a hardware error interrupt, so one should be designated to handle the logging/recovery activity.

## 9.2 Non-processor Generated Interrupts

### Parity Errors on Local I/O bus (L-bus)

A parity error on the L-bus most likely indicates a hard failure of either an L-bus device or the I/O module's L-bus buffers. When this error occurs, the system software should attempt to determine which device is failing and disable it from interfering with normal bus operation. Both CPU modules receive a hardware error interrupt, so one should be designated to handle the logging/recovery activity.

### Duplicate Tag Store Parity Error

Duplicate tag store parity errors are treated in hardware as uncorrectable errors. However the system always recovers from a parity error of this sort without any loss of data and/or memory coherence.

### 9.2.1.5 Hardware 1 Local I/O

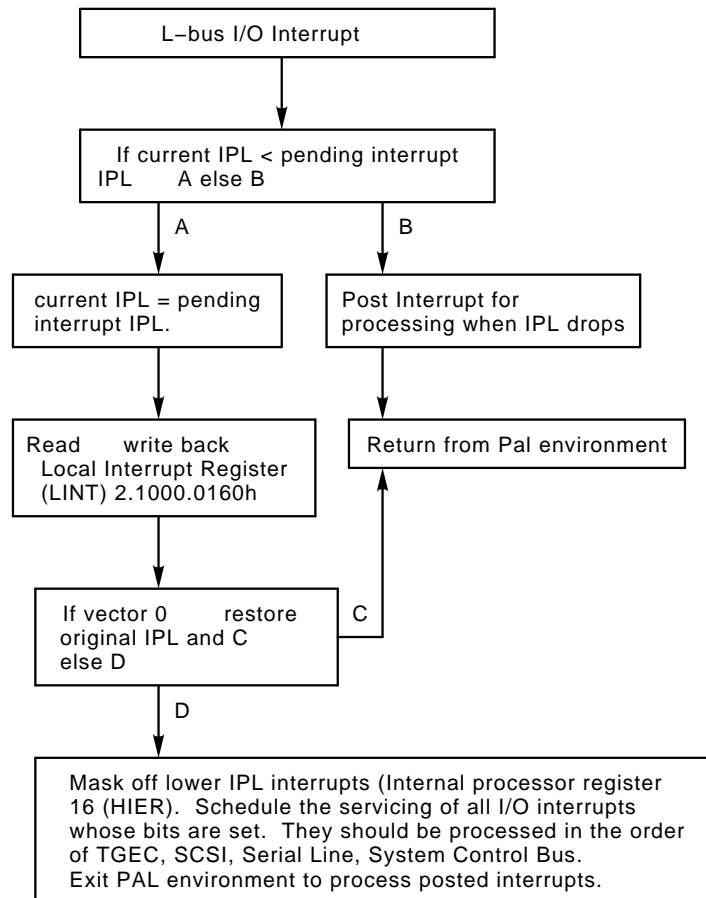
Hardware 1 interrupts are generated as a result of "Local I/O" device interrupt requests. They occur as a result of normal device operation, and when device errors are detected. For a complete description of the unique device interrupt request characteristics, refer to the component specifications for the device in question.

When a local I/O interrupt is posted, the PALcode entered reads the local interrupt register located on the I/O module. This register contains a bit field indicating which local I/O device posted the interrupt. (Several device interrupts could be pending.) PALcode dispatches to the appropriate interrupt service routine based on a priority scheme resident in the PALcode environment.

Local I/O interrupts are restricted to being serviced only by a dedicated processor (identified as the primary processor). The primary processor designation can be made at system power-up, or moved from one processor to the other over time. This requires that at any time, the processor not designated the primary must mask off the local I/O INTERRUPT REQUEST line bit <1> in the 21064 CPU HIER internal processor register.

## 9.2 Non-processor Generated Interrupts

Figure 9-3 Local I/O Interrupt Flow



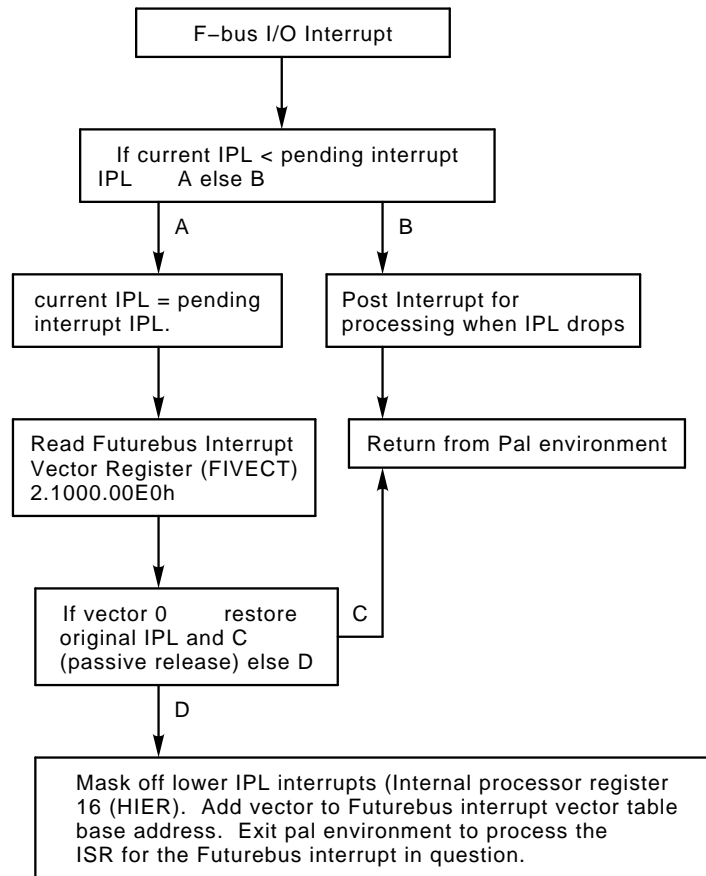
### 9.2.1.6 Hardware 2 Futurebus+

Hardware 2 interrupts are generated as a result of Futurebus+ adapter interrupt requests. These occur as a result of normal adapter operation, and when adapter errors are detected.

When an Futurebus+ interrupt is posted, the PALcode invoked reads the Futurebus+ interrupt register located on the I/O module. This register contains a vector offset which allows the PALcode to dispatch to the appropriate interrupt service routine. If all pending vectors have been read from the Futurebus+ interrupt register, a value of 0 is returned to indicate "passive release" of the interrupt.



Figure 9-4 Futurebus+ Interrupt Flow



### 9.2.1.7 Hardware 3 Interprocessor

Hardware 3 interrupts are requested when a CPU requires the attention of itself or another CPU.

The interprocessor interrupt request register is used to post an interrupt request to a specific processor. Note that, like software interrupts, no indication is given as to whether there is already an interprocessor interrupt pending when one is requested. Therefore, the interprocessor interrupt service routine must not assume there is a one-to-one correspondence between interrupts requested and interrupts generated. See the *Alpha Architecture Reference Manual* for the proper usage and definition of interprocessor interrupts.

### 9.2.1.8 Hardware 4 Interval Timer

The interval timer interrupt occurs at a regular interval allowing the processor to effectively schedule processing time to each process requiring attention.

The interval timer interrupt regularly interrupts the processor every 976.5625 microseconds. The PALcode handling this interrupt must update its copy of the absolute time, copy it to register R4, clear the interrupt in the local system interrupt clear register, and pass control to the interval timer interrupt routine.

## 9.2 Non-processor Generated Interrupts

### 9.2.1.9 Hardware 5 System Events

System events such as power system status changes, halt requests from the operator control panel, Futurebus+ adapter, or an SGEN on the I/O module are signaled via the Hardware 5 interrupt.

Actual power system status, and operator control panel halt request status must be requested from each subsystem over the serial control bus. Halt requests from network interfaces and Futurebus+ adapters are identified in the I/O module's IOCSR.

There is no passive-release mechanism associated with this interrupt so software must make sure that only one processor services the interrupt.

The pending interrupt must be cleared by explicitly writing 1 to the SYSTEM EVENT CLEAR bit in the system interrupt clear register (CSR12). This bit must be cleared in both CPU's registers regardless of which processor services this interrupt.

### 9.2.1.10 Software X Interrupt

Software interrupts provide a mechanism to allow a process to force the flow of control back into the system domain. Their use is determined by the software environment.

### 9.2.1.11 Serial Line Interrupt

The Serial Line Interrupt occurs when a change in state on the 21064 CPU serial data receive line changes. See Section 6.7 for details on the use of the 21064 serial line.

This interrupt should be masked off under normal operation.

### 9.2.1.12 Asynchronous System Trap Interrupt

Asynchronous system traps provide a way of notifying a process of events that are not synchronized with its execution, but which must be dealt with in the context of the process.

---

## Fault Management and Error Recovery

The following sections describe how the DEC 4000 system handles system faults.

### 10.1 Processor Errors

The general error handling mechanisms send an interrupt to the processor for uncorrectable and correctable errors. Uncorrectable errors detected on data accesses force machine check exceptions to occur as the data is being returned to the processor. Correctable errors and latent hardware errors are signaled using the `HARDWARE ERROR INTERRUPT`.

See Section 10.2, Section 10.3, and Section 10.4 for the complete DEC 4000 CPU module error matrix, and Chapter 9 for the error parse trees.

### 10.2 Backup Cache Errors

#### 10.2.1 Tag and Tag Control Store Parity Errors

Backup cache tag and tag control parity errors are detected only when the tag and tag control stores of the backup cache are read (probed). The error handling mechanism used when an error is encountered is based not only on whether the backup cache controller or 21064 detected the error, but also on what cycle was being performed.

Parity Errors in the Tag and Tag Control Stores may or may not be fatal errors to the system, or for that matter to non-system processes. See Table 10–1 for the complete error severity matrix.

**Table 10–1 Tag/Tag Control Error Severity Matrix**

Error	Address Context	Dirty	Severity
Tag Parity	Unknown	Yes	System Fatal
	Unknown	No	Scrub location - recoverable
Tag Control Parity	User†	Unknown	Process Fatal - Scrub location
	System	Unknown	System Fatal

---

†Virtual page mapping could have been changed between the time the error occurred and the time it was examined, therefore a deterministic address context can not be established. In lieu of a deterministic means for identifying address context, system software should consider any tag control parity error, `SYSTEM FATAL`.

---

The 21064 processor probes the backup cache tag and tag control stores whenever a load, load lock, store, or store conditional cycle is requested to a cacheable

## 10.2 Backup Cache Errors

location. If the 21064 processor discovers a parity error during this probe the following sequence of events occurs:

1. The transaction is forced to miss the external cache.
2. The BC\_TAG (processor register) holds the results of the external cache tag probe.
3. The machine check PALcode is invoked.
4. BIU\_STAT: BIU\_TCPERR (if tag control parity error) and/or BIU\_TPERR (if tag parity error) are set
5. The BIU\_ADDR register holds the physical address of the probe where the parity error was detected.

the backup cache controller detects tag and tag control parity errors under several different circumstances. These include the origin of the cycle as well as the cycle type. Table 10–2 shows the complete error handling matrix for backup cache tag and tag control store parity errors.

### Cycle requests

A cycle can originate from either the system bus or the 21064 CPU.

when a tag or tag control store parity error is detected during a backup cache probe due to the system bus initiated transaction, the probe results in a fail condition. When this occurs, the backup cache controller logs the error in the BCUE register, and the corresponding address in the BCUEA register, and asserts the C\_ERR L signal during the next  $t_0$  or idle system bus cycle. Refer to the Chapter 19 for the details of the C\_ERR L signal. The backup cache controller does *not* do the following:

- Accept data from the system bus if the system bus transaction is a write
- Supply data to the system bus even if the address probes HIT, DIRTY
- Modify the contents of the backup cache, Tag, or Tag Control Stores

Once this error is detected, and the BCUE error bits are set, the allocation of the backup cache is disabled and remains disabled until all of the BCUE error bits have been cleared, and the BCC ENB\_ALLOCATE bit set.

When a tag or tag control store parity error is detected during a transaction requested by the 21064, such as read block, load lock, write block, or store conditional, the resultant system bus transaction is converted to a NUT transaction. The BCUE and BCUEA registers log the error, and allocation in the backup cache is disabled. Data is neither read from the system bus (for read type transactions) or written on the system bus (for write type transactions). The state of the backup cache remains unchanged.

The cycle requested by the 21064 CPU is acknowledged with the HARD\_ERROR response invoking the Machine Check PALcode, and the C\_ERR L signal is asserted during the next  $t_0$  or idle system bus cycle.

Table 10–2 shows the backup cache tag control or tag store errors, the transaction that caused them, and the recovery procedure.

**Table 10–2 Backup Cache Tag Control or Tag Store Errors**

Transaction Causing Probe	Error Recovery Activity
System Bus Address Probe	<ul style="list-style-type: none"> <li>• Set PAR ERROR BIT in backup cache uncorrectable error (BCUE).</li> <li>• Latch system bus address in backup cache uncorrectable error address (BCUEA) register.</li> <li>• Disable the allocation of the cache.</li> <li>• Log subsequent address probe related errors as missed errors only, until the BCUE error bits have been cleared.</li> <li>• If the system bus address probe was due to a read or exchange cycle the CUCERR_L is asserted as the read data is returned to the transaction commander. This is done even if the module discovering the error was a bystander in the system bus transaction.</li> </ul>
21064 read block, write block, load lock, store conditional	<ul style="list-style-type: none"> <li>• Force a NUT transaction on the system bus.</li> <li>• Assert system bus C_ERR L signal</li> <li>• Log the error in BCUE and BCUEA registers and freeze them</li> <li>• Return cycle acknowledgment to 21064 of HARD_ERROR</li> <li>• Disable the allocation of the cache.</li> <li>• Log subsequent address probe related errors as missed errors only, until the BCUE error bits have been cleared.</li> </ul>

### 10.2.2 Data Store EDC Errors

This section describes the error detection and correction errors.

#### 10.2.2.1 Correctable

Correctable data store errors can be detected only on the following conditions:

- The backup cache controller or the 21064 processor read the backup cache data store.
- When an EDC error occurs on the bus between the C<sup>3</sup> chip and the 21064 processor.

The mechanism used to report the error is different depending on which subsystem detected it. All correctable errors are fully recoverable. Those detected by the backup cache controller are corrected “on the fly” and the normal flow of the transaction requested completes. Those detected by the 21064 processor cause a machine check to allow software to correct the corrupted data.

## 10.2 Backup Cache Errors

The 21064 detects correctable EDC errors only when executing a Load, or Load Lock instruction. When the error is detected, the following sequence of events occurs:

1. Data put into the instruction or data cache (appropriately) unchanged, block gets validated.
2. Machine check
3. BIU\_STAT: FILL\_EDC set, and FILL\_IRD set for instruction stream reference cleared for D-Stream. (FILL\_SEQ set if multiple errors occur.)
4. FILL\_ADDR<33:5> & BIU\_STAT[FILL\_QW] gives bad quadword's address
5. If data astream FILL\_ADDR<4:2> contain PA bits <4:2> of location which the failing load instruction attempted to read.
6. FILL\_SYNDROME contains syndrome bits associated with the failing quadword.
7. BIU\_ADDR, BIU\_STAT<6:0> locked and the contents are UNPREDICTABLE
8. If instruction stream DC\_STAT locked and the contents are UNPREDICTABLE, If data stream DC\_STAT locked, RA identifies register which holds the bad data, LW, LOCK, INT, VAX\_FP identify type of load instruction
9. BC\_TAG holds the results of the external cache tag probe if the external cache was enabled for this transaction.

If the physical address of the location with the correctable error is not resident in the backup cache, the error was a bus error and the transaction should be replayed. Otherwise, the backup cache location should be scrubbed and the transaction replayed.

The backup cache controller can detect the error under several different circumstances, and the way it is handled depends on how it was discovered. Table 10-3 shows the complete error handling matrix for backup cache data correctable errors detected by the backup cache controller. Correctable errors are detected and reported when a system bus reads HIT dirty Locations, a backup cache entry is victimized, or a masked write to a shared location.

All correctable errors detected by the backup cache controller are corrected to allow the normal operation to proceed as if no error occurred. If the physical address of the location with the correctable error is not resident in the backup cache the error was bus error and no scrubbing is required, otherwise the backup cache location should be scrubbed.

## 10.2 Backup Cache Errors

**Table 10–3 Backup Cache Data Correctable Errors**

Transaction Causing Probe	Access Reason	Error Recovery Activity
System Bus READ (Dirty) or Exchange (Read Dirty)	System bus READ/ EXCHANGE Dirty	The backup cache Correctable Error (BCCE) and backup cache Correctable Error Address Registers (BCCEA) are frozen and the CORRECTABLE ERROR bit set, if the BCC ENB BACKUP CACHE COR ERR INTERRUPT bit is set, the system bus C_ERR L signal is asserted.
System bus Write	—	—
21064 Read Block	Victim processing	The BCCE and BCCEA registers are frozen and the CORRECTABLEERROR bit set if the BCC ENB BACKUP CACHE COR ERR INTERRUPT bit is set, the system bus C_ERR L signal is asserted.
21064 write block	Victim processing / Shared masked write / Unmasked Write read allocate	The BCCE and BCCEA registers are frozen and the CORRECTABLE ERROR bit set. If the BCC ENB BACKUP CACHE COR ERR INTERRUPT bit is set, the system bus C_ERR L signal is asserted.
21064 load lock	Victim processing	The BCCE and BCCEA registers are frozen and the CORRECTABLE ERROR bit set. If the BCC ENB BACKUP CACHE COR ERR INTERRUPT bit is set, the system bus C_ERR L signal is asserted.
21064 store conditional	Victim processing / Shared masked write	The BCCE and BCCEA registers are frozen and the CORRECTABLE ERROR bit set. If the BCC ENB BACKUP CACHE COR ERR INTERRUPT bit is set, the system bus C_ERR L signal is asserted.

## 10.2 Backup Cache Errors

### 10.2.2.2 Uncorrectable Errors

Uncorrectable data store errors can be detected only when either the backup cache controller or the 21064 processor read the backup cache data store. Or when an EDC error occurs on the bus between the C<sup>3</sup> chip and the 21064 processor. The mechanism used to report the error is different depending on which subsystem detected it.

The severity of the error to the system's integrity depends on the address space the error occurred in as well as the state of the location in the backup cache.

**Table 10–4 Uncorrectable Data Store Error Severity Matrix**

Address Context	Dirty	Severity
User	Yes	Process Fatal if corrupted data used - Scrub
	No	Scrub location - recoverable
System	Yes	System Fatal if corrupted data used
	No	Scrub location - recoverable

The 21064 CPU detects uncorrectable EDC errors only while executing a load, or load lock instruction. When the error is detected the following sequence of events occurs:

1. Data put into instruction or data cache (appropriately) unchanged, block gets validated.
2. Machine check.
3. BIU\_STAT: FILL\_EDC set, and FILL\_IRD set for instruction stream reference cleared for data stream. (FILL\_SEQ set if multiple errors occur.)
4. FILL\_ADDR<33:5> & BIU\_STAT[FILL\_QW] gives bad quadword's address.
5. If data-stream FILL\_ADDR<4:2> contain PA bits <4:2> of location which the failing load instruction attempted to read.
6. FILL\_SYNDROME contains syndrome bits associated with the failing quadword.
7. BIU\_ADDR, BIU\_STAT<6:0> locked, contents are UNPREDICTABLE.
8. If instruction stream DC\_STAT locked, contents are UNPREDICTABLE. If data stream DC\_STAT locked, RA identifies register holding the bad data, LW, LOCK, INT, VAX\_FP identify type of load instruction.
9. BC\_TAG holds the results of the external cache tag probe if the external cache was enabled for this transaction.

When detected by the 21064 (as part of the machine check handler), if the physical address of the location with the uncorrectable error is not resident in the backup cache, the error was a bus error and should be re-tried.

The backup cache controller can detect the error under several different circumstances, and the way it is handled depends on how it was discovered. Table 10–5 shows the complete error handling matrix for backup cache data uncorrectable errors detected by the backup cache controller. uncorrectable errors are detected/reported when a system bus reads HIT Dirty Locations, a backup cache entry is victimized, or a masked write to a shared location occurs.



## 10.2 Backup Cache Errors

When detected by the backup cache controller, if the physical address of the location with the uncorrectable error is not resident in the backup cache the error was a bus error and severity of the error is the same as if a DIRTY backup cache location was found with an uncorrectable error - this location should be flushed to main memory where it is written with bad EDC.

**Table 10–5 Backup Cache Uncorrectable Errors**

Transaction Causing Probe	Access Reason	Error Recovery Activity
System bus read (dirty) or exchange (read dirty)	System bus READ /EXCHANGE dirty	Assert system bus CUCERR L with the data returned to the system bus.  The backup cache uncorrectable error (BCUE) and backup cache uncorrectable error address registers (BCUEA) are frozen and the UNCORRECTABLE ERROR bit set.
System bus write probe	If hit, system bus write data has either bad parity or has CUCERR L	The backup cache is written with bad EDC.
21064 read block	Victim Processing	The system bus C_ERR L signal is asserted, the victim written to memory coincident with the assertion of CUCERR L, and the read data is returned from the system bus to the processor.  The backup cache is not updated, and backup cache allocation is disabled.  The BCUE and BCUEA registers are frozen and the UNCORRECTABLE ERROR bit set.
	System bus returned data with bad system bus parity or CUCERR L asserted with read data	The system bus C_ERR L signal is asserted, bad "second" read data from the system bus is returned to the processor, bad "second" data from the system bus is written with "bad EDC" in the backup cache.  The BCUE and BCUEA registers are frozen and the UNCORRECTABLE ERROR bit set, processor is acknowledged with HARD_ERROR.

(continued on next page)

## 10.2 Backup Cache Errors

**Table 10–5 (Cont.) Backup Cache Uncorrectable Errors**

Transaction Causing Probe	Access Reason	Error Recovery Activity
21064 Write Block	Victim processing / Shared masked write	<p>The system bus C_ERR L signal is asserted.</p> <p>The victim/shared write is written to memory coincident with the assertion of CUCERR L, and the backup cache is updated with the new data.</p> <p>In the case of a shared masked write, the backup cache longwords with failing EDC that are not updated by the processor write, are not rewritten.</p> <p>The BCUE and BCUEA registers are frozen and the UNCORRECTABLE ERROR bit set.</p>
21064 load lock	Victim processing	<p>The system bus C_ERR L signal is asserted,</p> <p>The victim is written to memory coincident with the assertion of CUCERR L, and the read data is returned from the system bus to the processor.</p> <p>The BCUE and BCUEA registers are frozen and the UNCORRECTABLE ERROR bit set.</p>
21064 store conditional	Victim processing	<p>The system bus C_ERR L signal is asserted.</p> <p>The victim is written to memory coincident with the assertion of CUCERR L, and the backup cache is updated with the new data.</p> <p>The BCUE and BCUEA registers are frozen and the UNCORRECTABLE ERROR bit set.</p> <p>If store fails no data cycles occur; however C_ERR L will still be asserted.</p>

## 10.3 Duplicate Primary Cache Tag Store Parity Errors

Duplicate P-Cache Tag Store Parity Errors can be detected only by the backup cache controller when a system bus write occurs.

When an error is detected, the DEC 4000 system bus C\_ERR L signal is asserted, and the backup cache Duplicate Tag Store Error Register (DTSER) frozen, and the Error bit set. Detection of the parity error forces the invalidation of the associated Primary data cache location, and if HIT, the invalidation of the associated backup cache location regardless of system bus transaction commander ID.

## 10.3 Duplicate Primary Cache Tag Store Parity Errors

Thus single bit errors in the Duplicate Tag Store are not system fatal; however a **HARDWARE ERROR INTERRUPT** occurs every time one is encountered.

### 10.4 System Bus Errors

System bus errors may not be reportable as the error handling routines are most likely located in main memory. Only when the system bus is still operable, and main memory has not been corrupted, are system bus errors reportable.

#### 10.4.1 C/A Parity Error

When a system bus node detects a C/A parity error, it logs the error, signals the system by asserting the **C\_ERR L** signal, and ignores the rest of the system bus transaction. Table 10–6 shows the C/A parity errors.

**Table 10–6 C/A Parity Errors**

Commander	Responder	Bystander
<ul style="list-style-type: none"><li>• C/A cycle not acknowledged</li><li>• Reply with a <b>HARD_ERROR</b> failure status to the processor</li><li>• Freeze system bus Error Address Registers,</li><li>• set the appropriate bits in the system bus Error Register, and</li><li>• assert the system bus <b>C_ERR L</b> signal</li></ul>	<ul style="list-style-type: none"><li>• N/A (No responder)</li><li>• Data cache must be flushed as duplicate tag store is incoherent.</li></ul>	<ul style="list-style-type: none"><li>• Freeze system bus error address registers</li><li>• Set the appropriate bits in the system bus error register</li><li>• Assert the system bus <b>C_ERR L</b> signal</li></ul>

#### 10.4.2 Data Parity Error

A system bus data parity error is detected under two circumstances: as a responder and as a bystander.

A responder does not acknowledge the data transfer indicating to the commander that the transaction failed because a parity error. The commander asserts the **C\_ERR L** signal to notify the system of the error.

A bystander checks system bus data parity when a system bus write probe indicates a backup cache hit, and the system bus write data is used to update the backup cache. If this bystander detects bad parity, the cache location in question is marked invalid and the **C\_ERR L** signal is asserted to notify the system of the error.

## 10.4 System Bus Errors

**Table 10–7 Data Parity Errors**

Commander	Responder	Bystander
<ul style="list-style-type: none"> <li>Read data, respond with HARD_ERROR to the processor.</li> <li>Write data, not acknowledged by responder, respond with HARD_ERROR to the processor, assert C_ERR L.</li> </ul>	<ul style="list-style-type: none"> <li>Write data, do not acknowledge the data</li> <li>Freeze system bus error address registers</li> <li>Set the appropriate bits in the system bus error register.</li> </ul>	<ul style="list-style-type: none"> <li>If accepting data on a system bus write update, mark the backup cache location invalid.</li> <li>Freeze system bus error address registers.</li> <li>Set the appropriate bits in the system bus error register.</li> <li>Assert the system bus C_ERR L signal.</li> </ul>

### 10.4.3 Invalid Address - Bus Time-out

An invalid address is identified as an address that is not acknowledged by any responder. See Chapter 19.

When a commander detects an invalid address on the system bus, does the following:

- It freezes the system bus error address registers.
- It sets the appropriate bits in the system bus error register.
- It asserts the system bus C\_ERR L SIGNAL.
- It responds to the processor with HARD\_ERROR which initiates a machine check.

An invalid address is detected when a C/A is not acknowledged and no system bus node indicates that a parity error has occurred or when a dirty victim with a bad tag value (possible double-bit error) is written to memory with an EXCHANGE address that is outside currently configured memory space. This has occurred when the WRITE DATA NOT ACK'ED bit is asserted, and the EXCHANGE field logged in the CBEAH register indicates an invalid address. .

## 10.5 I/O Subsystem Errors

All errors occurring in the I/O subsystem are signaled using the Futurebus+ and Local I/O interrupts and all system bus errors detected by the I/O subsystem are signaled using the C\_ERR L signal. See the I/O Module Chapter.

## 10.6 C\_ERR L Assertion

When the backup cache controller detects the assertion of the C\_ERR L system bus signal it asserts the CPU module HARDWARE ERROR INTERRUPT.

---

## CPU Power-Up and Initialization

This section describes the behavior of the CPU module when system bus RESET\_L deasserts. The initial states of the processor and module registers are described in Chapter 21.

### 11.1 Internal Processor Registers and the Internal JSR Stack

See Appendix A for a description of the processors internal registers power-up state and a discussion regarding the initialization of the JSR stack via PAL code.

### 11.2 Backup Cache Initialization

Backup cache initialization is performed by the 21064 processor. This is accomplished as follows:

1. The cache size set to the appropriate value in BCC.
2. Guarantee that other system bus commanders do not drive “Cacheable address space” addresses.
3. Clear BC\_EN in the 21064 BIU\_CTL Register.
4. Clear FILL WRONG PARITY and disable tag and tag control store parity checking by clearing the ENB TAG & DUP TAG PAR CHK bits in the BCC register.
5. Set SELECT DRACK, 2ND QW SELECT in BCBTL to desired values.
6. Set EDC H, EDC L, and SHARED, DIRTY, VALID (BCC) to desired values.
7. Set FORCE EDC/CONTROL H in the BCC Register
8. Set ENABLE BACKUP CACHE INIT H in the BCC Register
9. Set ENABLE FILL BACKUP CACHE H in the BCC Register
10. Perform LDQ from memory locations starting at the appropriate address range.
  - CPU<sub>0</sub> = 1.0000.0000 - 1.7FFF.FFFF
  - CPU<sub>1</sub> = 1.8000.0000 - 1.FFFF.FFFF and continue up in 32-byte increments until an address range equal to the size of the cache in the system has been exhausted.

The backup cache control store contains the values for shared, dirty, and valid provided in the BCC register before initialization. The backup cache tag store contains a tag equivalent to the high-order address bits specified in the 9th step above. Each quadword in the backup cache data store of a particular cache block contains data identical to the data returned during the read (format indicated in Section 11.2.1) and the EDC store contains the values indicated in step 4 above.

## 11.2 Backup Cache Initialization

Once the backup cache RAMS are initialized, the backup cache control and status register should be set as follows:

FORCE EDC/CONTROL	Cleared
ENB BACKUP CACHE INIT	Cleared
FILL WRONG DUP TAG STORE PAR	Cleared
ENB DUP TAG STORE PAR CHK	Set
ENB BACKUP CACHE COND I/O UPDATES	Cleared
ENB BACKUP CACHE EDC CHK	Set
ENB BACKUP CACHE CORRECTION	Set
ENB BACKUP CACHE COR ERR INTERRUPT	Set
FILL WRONG CONTROL PAR	Cleared
ENB CONTROL BIT PAR CHK	Set
FILL WRONG TAG PAR	Cleared
ENB TAG PAR CHK	Set
FORCE FILL SHARED	Cleared
ENB FILL BACKUP CACHE	Set

All error bits in the backup cache correctable, uncorrectable, and duplicate tag store error registers should be cleared.

### 11.2.1 LDQ Data Format, BCC ENABLE BACKUP CACHE INIT Set

The LDC data format is described below.

Figure 11–1 LDQ Data Format (LDF)

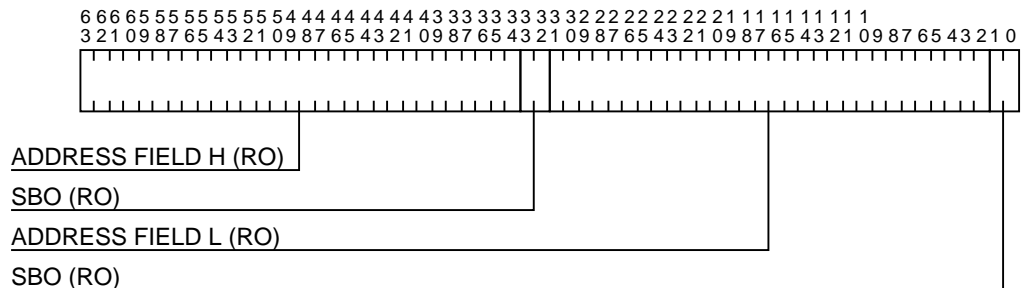


Table 11–1 LDQ Data Format Description

Field	Description
<b>63:34</b>	<b>ADDRESS FIELD H</b> <i>[read-only]</i> It contains the C/A address field for this cycle. Which is equal to the address of the data reference of the LDQ.
<b>33:32</b>	<b>SBO</b> <i>[read-only]</i> These bits should be 0.
<b>31:2</b>	<b>ADDRESS FIELD L</b> <i>[read-only]</i>

(continued on next page)

**Table 11–1 (Cont.) LDQ Data Format Description**

Field	Description
	It contains the C/A address field for this cycle. Which is equal to the address of the data reference of the LDQ.
<b>1:0</b>	<b>SBO</b> <i>[read-only]</i> These bits should be 0.

### 11.3 Duplicate Tag Store Initialization

The Duplicate Tag Store is reset to all 0's and correct parity on power-up. No user intervention is required.

### 11.4 System Bus Interface Initialization

Several things should be done on a system bus initialization as listed below.

1. The system bus control register should be initialized so that system bus parity checking is enabled and all other writeable bits cleared. and SELECT DRACK and 2ND QW SELECT bits are set to the appropriate values.
2. All error bits in the system bus error register should be cleared.
3. All functioning system bus commanders should be enabled for system bus ARB in the system bus Control Register.

### 11.5 CPU Clocks and Reset

System bus reset asserts asynchronously with respect to the system bus clocks, and deasserts synchronously with PHI1. The CPU module is designed to expect the clocks to be free running during reset. See Chapter 19 and Section 6.10 for details.

### 11.6 Power-Up Sequence

The system power supply is required to bring 3.3V up before 5V. If 3.3V goes below regulation, the DCOK signal to 21064 is deasserted.

### 11.7 Powering Up with Bad Main Memory

Because the processor's backup cache update policy is designed to accept the data of an I/O module write (when the location is in the backup cache), it is possible to simultaneously load the backup cache with the console image as it is written to a broken memory module.

This is done by initializing the backup cache tag store with the tag values of the memory region being written, and the tag control store valid, not shared, not dirty and by creating a mailbox structure in the backup cache and initializing its tag control store to valid, not shared, dirty the processor responds with the mailbox data when the I/O module requests it. The only function required by the memory module is to acknowledge the system bus transactions as they occur. This means the memory module should be initialized as though it is functioning normally.

The console image can be started as if it was located on a memory module, and just moved into the backup cache via the normal cache allocation processes handled in hardware.

# Part III

---

## The I/O Module

This part contains a detailed functional description of the DEC 4000 I/O module.



---

## The KFA40 I/O Module

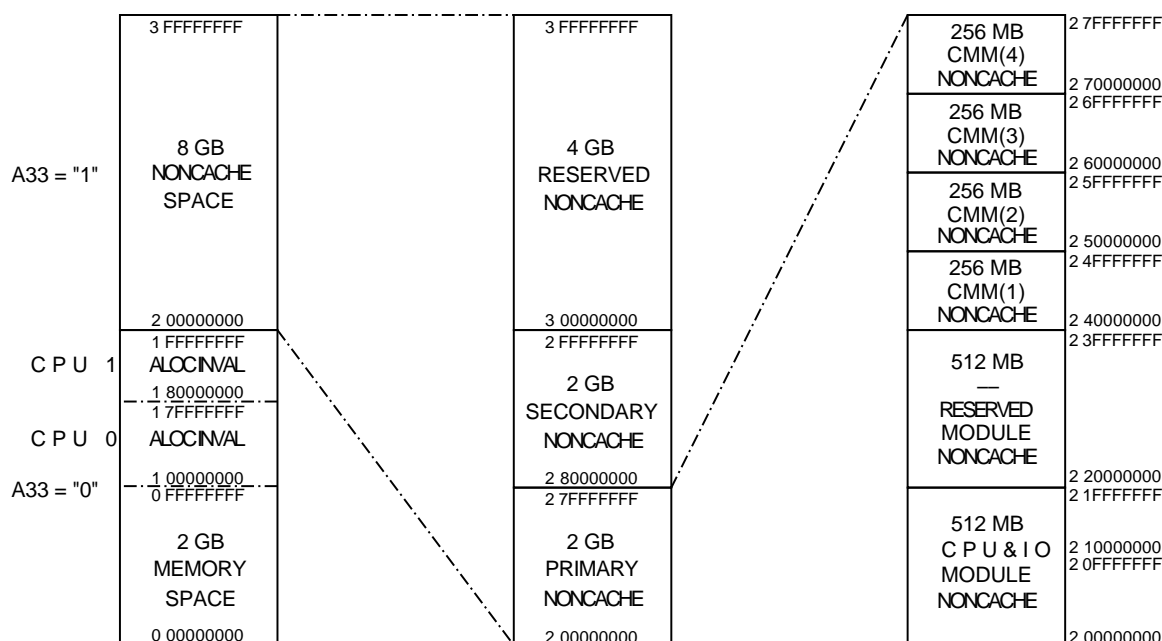
The KFA40 I/O module contains the complete I/O subsystem for a DEC 4000 AXP system including the following:

- A Futurebus+ Profile B interface allowing 32- and 64-bit data transfers
- Two Ethernet interfaces
- Five SCSI/DSSI channels
- A time-of-year (TOY) clock
- 8 Kilobytes of EEROM for console use
- Two serial line units
- 512 Kilobytes of flash erasable PROM (FEPRM) for console code
- A serial control bus controller (based on I<sup>2</sup>C)

## Address Mapping

The 34-bit physical address space of the system bus is split equally between memory space and I/O space as shown in Figure 13-1.

Figure 13-1 System Bus Address Map



NOTES: 1. ALL ADDRESSES ARE PHYSICAL (PA<33:0>).

ADDMAP

### 13.1 System Memory Space

The first 8-Gbyte region ( $2^{33}$ ) of address space is defined as system memory space. Two Gbytes of this region are used for the maximum physical memory of the DEC 4000 AXP platform (up to four 512 Mbyte memory modules). Memory modules are configured by the console software at power-up to provide a contiguous physical memory range within this space. The remainder of system memory space is reserved for future use. The I/O module does not respond to any addresses in system memory space.

## 13.2 Primary I/O Space

### 13.2 Primary I/O Space

The second 8 Gbyte region is defined as I/O space. Two Gbytes are set aside for the DEC 4000 AXP platform's primary I/O space (registers existing directly on the system bus). The remaining 6 Gbytes are reserved for future use.

The I/O module is allocated 256 Mbyte ( $2^{28}$ ) of primary I/O space. The CPU modules share a 256 Mbyte ( $2^{28}$ ) region while each of four memory module slots is assigned a 256 Mbyte ( $2^{28}$ ) region. Modules are free to decode their regions as needed. While the size of these address ranges may seem excessive, they allow the hardware to implement very simple and fast address decoders.

The I/O module decodes its 256 Mbyte space as shown in Figure 13–2. Accesses to undefined regions of the I/O module's address space result in either a machine check or the access of a mirrored register location.

Registers are accessible as quadword registers aligned on hexaword boundaries. Access to other quadwords within the aligned hexaword results not in a machine check, but in unpredictable I/O module operation.

Because of hardware partitioning, the individual longwords of a quadword register are divided between two gate array devices. Therefore, many control and status bits are duplicated in the register descriptions to eliminate communication paths between the devices.

**Figure 13–2 I/O Module Register Map**

IOCSR	2 1000 0000
CERR1	2 1000 0020
CERR2	2 1000 0040
CERR3	2 1000 0060
LMBPR	2 1000 0080
FMBPR	2 1000 00A0
DIAGCSR	2 1000 00C0
FIVECT	2 1000 00E0
FHVECT	2 1000 0100
FERR1	2 1000 0120
FERR2	2 1000 0140
LINT	2 1000 0160
LERR1	2 1000 0180
LERR2	2 1000 01A0

Figure 13–3 Lbus Diagnostic Mode Register Map

DLCA0	2 1000 0200
DLCA1	2 1000 0220
DLCA2	2 1000 0240
DLCA3	2 1000 0260
DLCB0	2 1000 0280
DLCB1	2 1000 02A0
DLCB2	2 1000 02C0
DLCB3	2 1000 02E0
DLMA0	2 1000 0300
DLMA1	2 1000 0320
DLMA2	2 1000 0340
DLMA3	2 1000 0360
DLMB0	2 1000 0380
DLMB1	2 1000 03A0
DLMB2	2 1000 03C0
DLMB3	2 1000 03E0

## 13.2 Primary I/O Space

Figure 13–4 Futurebus+ Diagnostic Mode Register Map

DFCA0	2 1000 0400
DFCA1	2 1000 0420
DFCA2	2 1000 0440
DFCA3	2 1000 0460
DFCB0	2 1000 0480
DFCB1	2 1000 04A0
DFCB2	2 1000 04C0
DFCB3	2 1000 04E0
DFMA0	2 1000 0500
DFMA1	2 1000 0520
DFMA2	2 1000 0540
DFMA3	2 1000 0560
DFMB0	2 1000 0580
DFMB1	2 1000 05A0
DFMB2	2 1000 05C0
DFMB3	2 1000 05E0

The diagnostic mode data registers allow direct write and read control of the cache line buffers for the Futurebus+ and Local I/O bus interfaces during power-up diagnostics. When bits <0> and <32> in the DIAGCSR register are set, the diagnostic mode data registers are enabled. When bits <0> and <32> are cleared the registers still respond to system bus transactions, but the data returned is unpredictable. The diagnostic mode registers are used with the MERGE SELECT bits in the DIAGCSR register to steer the data through the internal data paths. This provides complete diagnostic coverage of the cache buffer data paths. Switching from normal mode to diagnostic mode may destroy data currently in the cache line buffers. Diagnostic mode must be restricted to power-up diagnostics and cannot be used on an active system.

The registers are named to specify their function as follows:

**D<sub>wxy</sub>n**

Where:

*w* F = Futurebus+, L = Local I/O bus  
*x* C = Cache line data location, M = mailbox data location  
*y* A = first cache line buffer, B = second cache line buffer  
*n* Quadword within cache line

1. QW0, bytes [7..0]
2. QW1, bytes [15..8]
3. QW2, bytes [23..16]

## 4. QW3, bytes [32..24]

Each of the diagnostic mode data registers writes or reads a quadword of data from its respective cache line buffer. When the Merge Select bits in the DIAGCSR are cleared, the *DLCyn*, and *DFCyn* registers can be written and read to check the operation of the cache line read buffers and mailbox data structure buffers. When the *DwCxy* (cache line buffer) or *DwMyn* (mailbox buffer) is written, the storage location used for DMA write operations (merge buffer) is also written. With the MERGE SELECT bits in the DIAGCSR set, this merge buffer is read when either the *DwCyn* or *DwMyn* register is accessed. The data returned during a read from the *DwCyn* or *DwMyn* register with the Merge Select bit set are the last data value written to either the *DwCyn* or *DwMyn* register.

### 13.3 Diagnostic Mode Address Registers

The diagnostic mode address registers allow access to the address pointers used by the Futurebus+ and Local I/O bus interfaces. Figure 13–5 shows a map of these registers.

Figure 13–5 Diagnostic Address Register Map

DLAA	2 1000 0600
DLAB	2 1000 0680
DFAA	2 1000 0700
DFAB	2 1000 0780

When bits <0> and <32> in the DIAGCSR register are set, the diagnostic mode address registers are enabled. When bits <0> and <32> are cleared, the registers still respond to system bus transactions, but the data returned is unpredictable. These registers are for diagnostic testing of the address pointers and should not be accessed when Futurebus+ or Local I/O bus DMA activity is present.

The diagnostic mode address registers are written as a side effect of writing selected diagnostic mode data registers. Therefore, these registers are read-only registers. Direct writes to the DLAA, DLAB, DFAA or DFAB registers have no effect. These registers can be updated by accessing the diagnostic mode data registers as described in Table 13–1.

Data read from the diagnostic mode address registers is not read back as written but reflects the normal processing of an address for use on the Cobra bus. The following algorithm is used to determine the read data:

```
Write_Data, Read_Data: quadword;
Read_Data := (((Write_Data DIV 4) OR 000000030000000316)
AND 1FFFFFFF1FFFFFFF16);
```

## 13.3 Diagnostic Mode Address Registers

**Table 13–1 Diagnostic Mode Address Register Access**

<b>Write Register</b>	<b>Read Register</b>
DLCA0 (2 1000 0200)	DLAA (2 1000 0600)
DLMA0 (2 1000 0300)	DLAA (2 1000 0600)
DLCB0 (2 1000 0280)	DLAB (2 1000 0680)
DLMB0 (2 1000 0380)	DLAB (2 1000 0680)
DFCA0 (2 1000 0400)	DFAA (2 1000 0700)
DFMA0 (2 1000 0500)	DFAA (2 1000 0700)
DFCB0 (2 1000 0480)	DFAB (2 1000 0780)
DFMB0 (2 1000 0580)	DFAB (2 1000 0780)

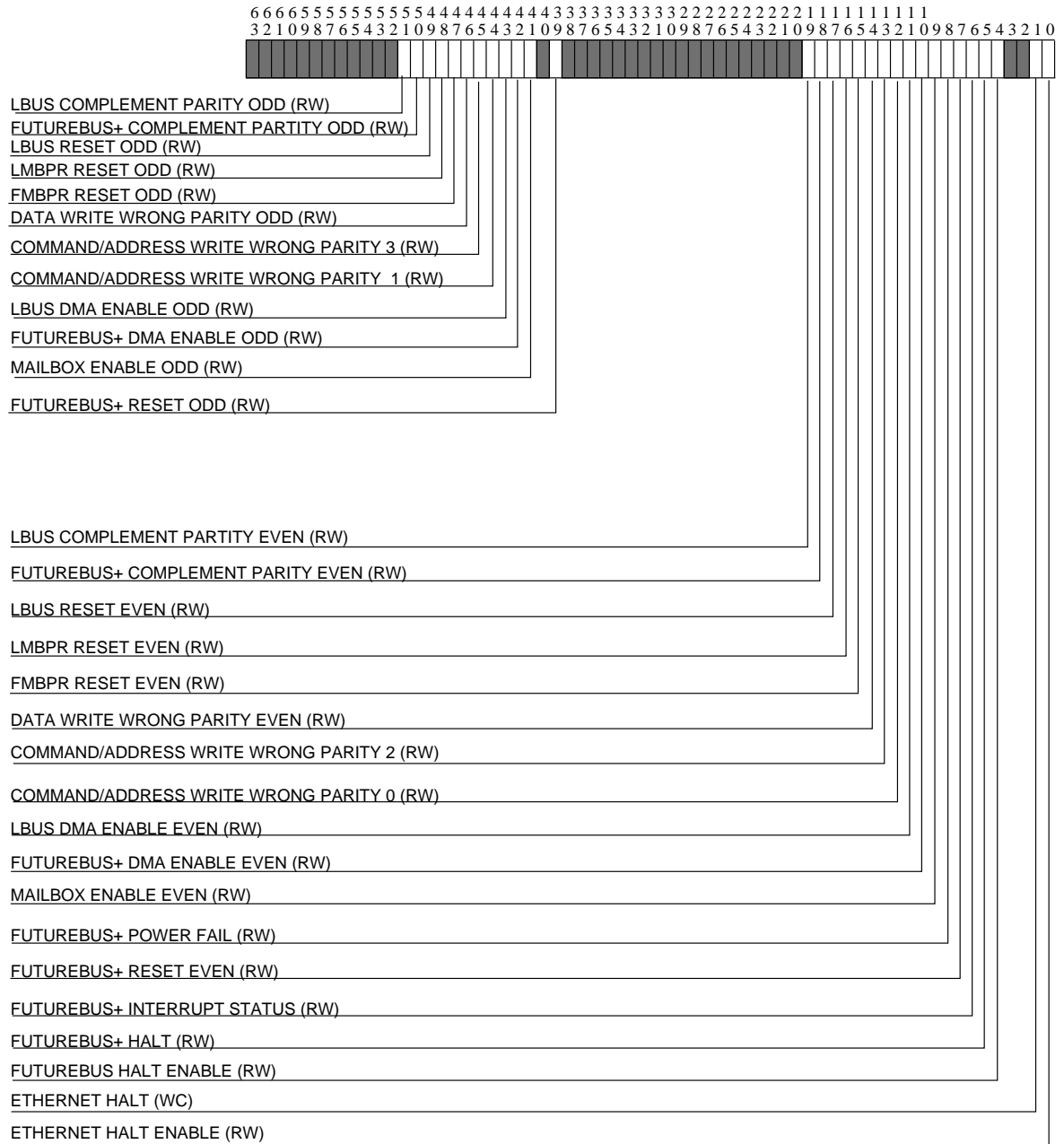
## 13.4 I/O Control/Status Register (IOCSR)

This register stores information related to the I/O operations.



## 13.4 I/O Control/Status Register (IOCSR)

Figure 13–6 I/O Control/Status Register (IOCSR)



## 13.4 I/O Control/Status Register (IOCSR)

Table 13–2 I/O Control and Status Register Description

Field	Description
51	<b>LBUS COMPLEMENT PARITY ODD</b> <i>[read/write]</i> Setting this bit causes the Lbus parity generation and check logic to complement its outputs. This causes errors on data parity generation and checking for diagnostic purposes.
50	<b>FUTUREBUS+ COMPLEMENT PARITY ODD</b> <i>[read/write]</i> Setting this bit causes the Futurebus+ parity generation and check logic to complement its outputs. This causes errors on both address and data parity generation and checking for diagnostic purposes.
49	<b>LBUS RESET ODD</b> <i>[read/write]</i> Setting this bit causes the assertion of the Lbus reset signal and resets the Lbus logic in the odd interface gate array. This causes all local I/O devices and controllers to their reset state. The processor must assert this signal for a minimum of 200 ms to guarantee a complete reset of all the local I/O devices. The Lbus Reset even bit must be set in conjunction with this bit to completely reset the Lbus interface. Failure to do so produces unpredictable results.
48	<b>LMBPR RESET ODD</b> <i>[read/write]</i> Setting this bit clears the full status of the Lbus mailbox pointer register in the odd interface gate array and allows a subsequent STQ_C to the FMPR to complete with the low bit set. The Lbus reset odd and even and LMBPR reset even bits must also be set in order to clear out any mailbox operation that is in progress or hung because of an error.
47	<b>FMBPR RESET ODD</b> <i>[read/write]</i> Setting this bit clears the full status of the Futurebus+ mailbox pointer register in the odd interface gate array device and allows a subsequent STQ_C to the FMPR to complete with the low bit set. The Futurebus+ reset odd and even (bits 39 and 7 of this register) must be set and cleared after any error on Futurebus+ MBX (8000 in CERR1) to clear out any mailbox operation in progress or hung due to an error.
46	<b>DATA WRITE WRONG PARITY ODD</b> <i>[read/write]</i> When this bit is set, the I/O module generates wrong data parity for (1) all odd longwords during the next read of an I/O module primary register or (2) during the next DMA write cycle when the I/O module is the commander, whichever occurs first. This bit remains asserted for only one data transfer. This bit is cleared on power-up, after a system bus reset, or after the data cycle for which its action is taken.
45	<b>COMMAND/ADDRESS WRITE WRONG PARITY 3</b> <i>[read/write]</i> When a 1 is written to this bit field, wrong parity is generated for longword 3 during the next system bus command/address transfer phase when the I/O module is the commander. This bit is active for only one system bus command/address transfer. This bit is cleared on power-up or after a system bus reset.
44	<b>COMMAND/ADDRESS WRITE WRONG PARITY 1</b> <i>[read/write]</i> When a 1 is written to this bit field, wrong parity is generated for longword 1 during the next system bus command/address transfer phase when the I/O module is the commander. This bit is active for only one system bus command/address transfer. This bit is cleared on power-up or after a system bus reset.
43	<b>LBUS DMA ENABLE ODD</b> <i>[read/write]</i>

(continued on next page)

## 13.4 I/O Control/Status Register (IOCSR)

Table 13–2 (Cont.) I/O Control and Status Register Description

Field	Description
	When a 1 is written to this bit field, DMA read and write operations from local I/O devices are enabled. Both LBUS DMA ENABLE EVEN and LBUS DMA ENABLE ODD must be set to the same value. This bit is cleared on power-up or after a system bus reset.
<b>42</b>	<b>FUTUREBUS+ DMA ENABLE ODD</b> <i>[read/write]</i> When a 1 is written to this bit field, DMA read and write operations from Futurebus+ devices are enabled. Both FUTUREBUS+ DMA ENABLE EVEN and FUTUREBUS+ DMA ENABLE ODD must be set to the same value. This bit is cleared on power-up or after a system bus reset.
<b>41</b>	<b>MAILBOX ENABLE ODD</b> <i>[read/write]</i> When a 1 is written to this bit field, mailbox operations to remote buses are enabled. If this bit is cleared writes to the LMBPR or FMBPR complete without error, but the registers are not updated and no fetch of a mailbox data structure from memory takes place. Both MAILBOX ENABLE EVEN and MAILBOX ENABLE ODD must be set to the same value. This bit is cleared on power-up or after a system bus reset.
<b>39</b>	<b>FUTUREBUS+ RESET ODD</b> <i>[read/write]</i> Setting this bit resets the Futurebus+ interface logic in the odd interface gate array. The Futurebus+ reset signal is not asserted when this bit is set. This bit must be asserted in conjunction with the FUTUREBUS+ RESET EVEN bit to correctly reset the Futurebus+ interface. Failure to do so produces unpredictable results.
<b>19</b>	<b>LBUS COMPLEMENT PARITY EVEN</b> <i>[read/write]</i> Setting this bit causes the Lbus parity generation and check logic to complement its outputs. This causes errors on data parity generation and checking for diagnostic purposes.
<b>18</b>	<b>FUTUREBUS+ COMPLEMENT PARITY EVEN</b> <i>[read/write]</i> Setting this bit causes the Futurebus+ parity generation and check logic to complement its outputs. This causes errors on both address and data parity generation and checking for diagnostic purposes.
<b>17</b>	<b>LBUS RESET EVEN</b> <i>[read/write]</i> Setting this bit resets the Lbus interface logic in the even interface gate array. The Lbus reset signal is not asserted when this bit is set. This bit must be asserted in conjunction with the LBUS RESET ODD bit to correctly reset the Lbus interface. Failure to do so produces unpredictable results.
<b>16</b>	<b>LMBPR RESET EVEN</b> <i>[read/write]</i> Setting this bit clears the full status of the Lbus mailbox pointer register in the even interface gate array and allows a subsequent STQ_C to the FMPR to complete with the low bit set. The LBUS RESET ODD AND EVEN and LMBPR ODD bits must also be set to clear out any mailbox operation that is in progress or hung due to an error.
<b>15</b>	<b>FMBPR RESET EVEN</b> <i>[read/write]</i> Setting this bit clears the full status of the Futurebus+ mailbox pointer register in the even interface gate array and allows a subsequent STQ_C to the FMPR to complete with the low bit set. The FUTUREBUS+ RESET ODD AND EVEN (bits <39> and <7> of this register) must be set and cleared after any error on Futurebus+ MBX (8000 in CERR1) to clear out any mailbox operation that is in progress or hung due to an error.
<b>14</b>	<b>DATA WRITE WRONG PARITY EVEN</b> <i>[read/write]</i>

(continued on next page)

## 13.4 I/O Control/Status Register (IOCSR)

**Table 13–2 (Cont.) I/O Control and Status Register Description**

Field	Description
	When this bit is set, wrong data parity is generated by the I/O module for (1) all even longwords during the next read of an I/O module primary register or (2) during the next DMA write cycle when the I/O module is the commander, whichever occurs first. This bit remains asserted for only one data transfer. This bit is cleared on power-up, after a system bus reset, or after the data cycle for which its action is taken.
<b>13</b>	<b>COMMAND/ADDRESS WRITE WRONG PARITY 2</b> <i>[read/write]</i> When a 1 is written to this bit field, wrong parity is generated for longword 2 during the next system bus command/address transfer phase when the I/O module is the commander. This bit is active for only one system bus command/address transfer. This bit is cleared on power-up or after a system bus reset.
<b>12</b>	<b>COMMAND/ADDRESS WRITE WRONG PARITY 0</b> <i>[read/write]</i> When a 1 is written to this bit field, wrong parity is generated for longword 0 during the next system bus command/address transfer phase when the I/O module is the commander. This bit is active for only one system bus command/address transfer. This bit is cleared on power-up or after a system bus reset.
<b>11</b>	<b>LBUS DMA ENABLE EVEN</b> <i>[read/write]</i> When a 1 is written to this bit field, DMA read and write operations from local I/O devices are enabled. Both LBUS DMA ENABLE EVEN and LBUS DMA ENABLE ODD must be set to the same value. This bit is cleared on power-up or after a system bus reset.
<b>10</b>	<b>FUTUREBUS+ DMA ENABLE EVEN</b> <i>[read/write]</i> When a 1 is written to this bit field, DMA read and write operations from Futurebus+ devices are enabled. Both FUTUREBUS+ DMA ENABLE EVEN and FUTUREBUS+ DMA ENABLE ODD must be set to the same value. This bit is cleared on power-up or after a system bus reset.
<b>9</b>	<b>MAILBOX ENABLE EVEN</b> <i>[read/write]</i> When a 1 is written to this bit field, mailbox operations to remote buses are enabled. If this bit is cleared, writes to the LMBPR or FMBPR complete without error, but the registers are not updated and no fetch of a mailbox data structure from memory takes place. Both the MAILBOX ENABLE EVEN and MAILBOX ENABLE ODD must be set to the same value. This bit is cleared on power-up or after a system bus reset.
<b>8</b>	<b>FUTUREBUS+ POWER FAIL</b> <i>[read/write]</i> Setting this bit causes a Futurebus+ power fail message to be sent.
<b>7</b>	<b>FUTUREBUS+ RESET EVEN</b> <i>[read/write]</i> Setting this bit causes the Futurebus+ reset signal (RE_L) to assert. The processor must time the assertion of this bit to cause the correct operation on the Futurebus+. The processor is required to assert this bit for a minimum of 100 ms, and a maximum of 200 ms. For a full reset, this bit must be set for a minimum of 4 ms and a maximum of 12 ms for BUS_INIT.
<b>6</b>	<b>FUTUREBUS+ INTERRUPT STATUS</b> <i>[read/write]</i> This bit is asserted if the Futurebus+ interrupt queue is not empty. It is cleared on power-up or after a system bus reset or Futurebus+ reset.
<b>5</b>	<b>FUTUREBUS+ HALT</b> <i>[read/write]</i>

(continued on next page)

## 13.4 I/O Control/Status Register (IOCSR)

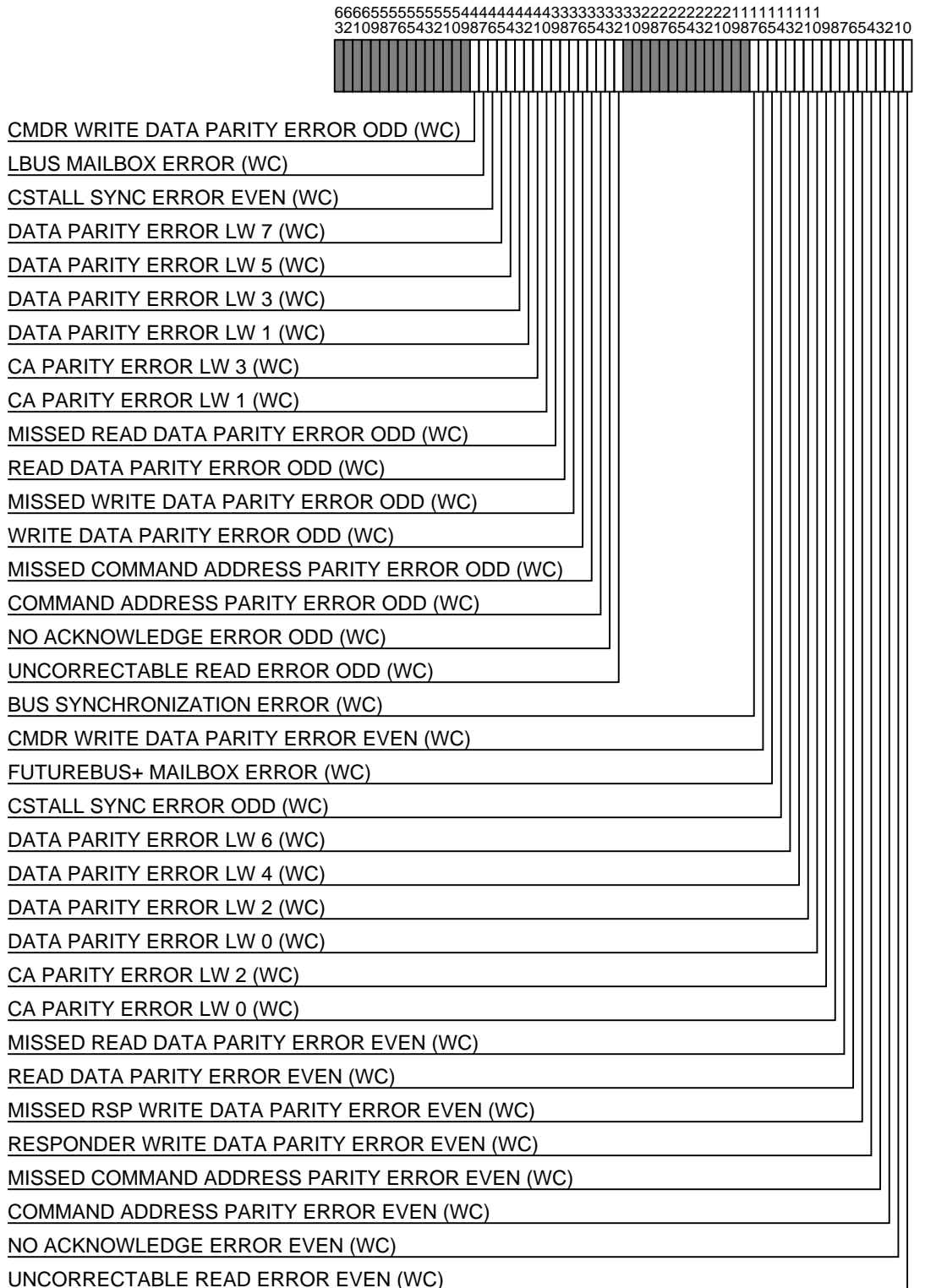
Table 13–2 (Cont.) I/O Control and Status Register Description

Field	Description
	This bit is asserted if a Futurebus+ device is requesting a CPU halt. Reading the Futurebus halt vector provides the vector for the requesting device and clears this bit.
<b>4</b>	<b>FUTUREBUS HALT ENABLE</b> <i>[read/write]</i> When set, this bit allows writes by a Futurebus+ I/O device to the Futurebus+ halt request register to assert the system event interrupt on the system bus. This bit is cleared on power-up or after a system bus reset.
<b>1</b>	<b>ETHERNET HALT</b> <i>[read/write 1 to clear]</i> This bit is asserted if either local Ethernet controllers is requesting a CPU halt. The port requesting the halt can be determined by examining the TGEC device registers. Write one to clear.
<b>0</b>	<b>ETHERNET HALT ENABLE</b> <i>[read/write]</i> When set, this bit allows remote trigger messages from the local Ethernet interfaces to assert the system event interrupt on the system bus. The halt for an individual TGEC device can be enabled or disabled through the TGEC device registers. This bit is cleared on power-up or after a system bus reset.

## 13.5 System Bus Error Register 1 (CERR1)

## 13.5 System Bus Error Register 1 (CERR1)

Figure 13–7 System Bus Error Register 1 (CERR1)



## 13.5 System Bus Error Register 1 (CERR1)

Table 13–3 System Bus Error Register 1 Description

Field	Description
<b>48</b>	<p><b>CMDR WRITE DATA PARITY ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set if an odd longword write data parity error is detected while the I/O module is acting as the system bus commander. When this bit is set, the failing command/address is saved in CERR2 and CERR3. Write one to clear. This bit is cleared on power-up or after a system bus reset.</p>
<b>47</b>	<p><b>LBUS MAILBOX ERROR</b> <i>[read/write 1 to clear]</i></p> <p>This bit is asserted when an error is detected during the processing of a mailbox operation directed to the Lbus. The contents of the LMBPR is frozen until this bit is cleared. Write one to clear. Cleared on power-up or after a system bus reset. (Note: this error condition is not currently generated by the I/O module and all Lbus mailbox errors are indicated within the status field of the mailbox data structure.)</p>
<b>46</b>	<p><b>CSTALL SYNC ERROR EVEN</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set if a CSTALL out of sync error is detected by the even slice of the system bus interface. Write one to clear. Cleared on power-up or after a system bus reset.</p>
<b>45</b>	<p><b>DATA PARITY ERROR LW 7</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error is detected on longword 7 during the data portion of a system bus transaction. This longword corresponds to bytes 31..28 of the hexaword transferred.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>44</b>	<p><b>DATA PARITY ERROR LW 5</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error is detected on longword 5 during the data portion of a system bus transaction. This longword corresponds to bytes 23..20 of the hexaword transferred.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>43</b>	<p><b>DATA PARITY ERROR LW 3</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error is detected on longword 3 during the data portion of a system bus transaction. This longword corresponds to bytes 15..12 of the hexaword transferred.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>42</b>	<p><b>DATA PARITY ERROR LW 1</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error is detected on longword 1 during the data portion of a system bus transaction. This longword corresponds to bytes 7..4 of the hexaword transferred.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>41</b>	<p><b>CA PARITY ERROR LW 3</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error on longword 3 of a system bus command /address transfer is detected. Please reference the Chapter 19 for the mapping of the command/address field to the system bus longwords.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>40</b>	<p><b>CA PARITY ERROR LW 1</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error on longword 1 of a system bus command /address transfer is detected. Please reference the Chapter 19 for the mapping of the command/address field to the system bus longwords.</p> <p>Write one to clear. Undefined on power-up.</p>

(continued on next page)

## 13.5 System Bus Error Register 1 (CERR1)

Table 13–3 (Cont.) System Bus Error Register 1 Description

Field	Description
<b>39</b>	<p><b>MISSED READ DATA PARITY ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when an odd longword parity error is detected during the transfer of read data from a responder node when the I/O module is the commander, <i>after</i> some other error has been detected by the error logic. This indicates that the full error status (for example, the FAILING COMMAND/ADDRESS, and DATA PARITY ERROR bits) is not available for this error (and therefore, the error has been "missed").</p> <p>Write one to clear. Cleared on power-up.</p>
<b>38</b>	<p><b>READ DATA PARITY ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when an odd longword parity error is detected during the transfer of read data from a responder node when the I/O module is the commander. If this indicator is set, the full error status (including the Failing Command /Address, and Data Parity Error bits) is available to help identify the source of this error.</p> <p>Write one to clear. Cleared on power-up.</p>
<b>37</b>	<p><b>MISSED WRITE DATA PARITY ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when an odd longword parity error is detected during the transfer of write data from a commander node to an I/O module primary I/O space register, <i>after</i> some other error has been detected by the error logic. This indicates that the full error status (for example, the Failing Command/Address, and Data Parity Error bits) is not available for this error (and therefore, the error has been "missed").</p> <p>Write one to clear. Cleared on power-up.</p>
<b>36</b>	<p><b>WRITE DATA PARITY ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when an odd longword parity error is detected during the transfer of write data from a commander node on the system bus to an I/O module primary I/O space register. If this indicator is set, the full error status (including the Failing Command/Address, and Data Parity Error bits) is available to help identify the source of this error.</p> <p>Write one to clear. Cleared on power-up.</p>
<b>35</b>	<p><b>MISSED COMMAND ADDRESS PARITY ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when an odd longword parity error is detected during the transfer of Command/Address from a commander node to a selected responder node, <i>after</i> some other error has been detected by the error logic. This indicates that the full error status (for example, the Failing Command/Address, and Data Parity Error bits) is not available for this error and therefore, the error has been "missed".</p> <p>This bit is set by the I/O module when a parity error on any <i>odd</i> Longword of the Command/Address transfer occurs, regardless of whether this module is the intended responder.</p> <p>Write one to clear. Cleared on power-up.</p>
<b>34</b>	<p><b>COMMAND ADDRESS PARITY ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error is detected on the odd longwords of a system bus Command/Address transfer regardless of whether the transfer is directed to this module or not. When this bit is set, the error status in the CERR2 and CERR3 registers and in CERR1 [41:40] is available to help locate the source of the error.</p> <p>Write one to clear. Cleared on power-up.</p>

(continued on next page)



## 13.5 System Bus Error Register 1 (CERR1)

Table 13–3 (Cont.) System Bus Error Register 1 Description

Field	Description
<b>33</b>	<p><b>NO ACKNOWLEDGE ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when, while acting as the system bus master, the I/O module fails to receive an acknowledge for the command/address that it has placed on the bus. This error indicates that the selected device does not exist, the selected device is broken, the I/O module is broken, or a command/address parity error has occurred during the cycle. The failing command/address is saved in CERR2 and CERR3. Write one to clear. Cleared at power up or after a system bus reset.</p>
<b>32</b>	<p><b>UNCORRECTABLE READ ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when, while acting as the system bus master, the I/O module receives an uncorrectable read data error from the selected device. The failing command/address is saved in CERR2 and CERR3. Write one to clear. Cleared at power up or after a system bus reset.</p>
<b>17</b>	<p><b>BUS SYNCHRONIZATION ERROR</b> <i>[read/write 1 to clear]</i></p> <p>The bit is set when the I/O module detects command fields on the upper and lower quadwords of the system bus that do not match. This error is an indication that the gate array devices that make up the interface for the current bus master are out of sync. This error condition is checked for all system bus transactions whether or not the I/O module is the system bus master. When this bit is set, the failing command/address is saved in CERR2 and CERR3. Write one to clear. Cleared on power-up or after a system bus reset.</p>
<b>16</b>	<p><b>CMDR WRITE DATA PARITY ERROR EVEN</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set if an even longword write data parity error is detected while the I/O module is acting as the system bus commander. When this bit is set, the failing command/address is saved in CERR2 and CERR3. Write one to clear. Cleared on power-up or after a system bus reset.</p>
<b>15</b>	<p><b>FUTUREBUS+ MAILBOX ERROR</b> <i>[read/write 1 to clear]</i></p> <p>This bit is asserted when an error is detected during the processing of a mailbox operation directed to the Futurebus+. The contents of the FMBPR are frozen until this bit is cleared. Write one to clear. Cleared on power up or after a system bus reset.</p>
<b>14</b>	<p><b>CSTALL SYNC ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set if a CSTALL out of sync error is detected by the odd slice of the system bus interface. Write one to clear. Cleared on power-up or after a system bus reset.</p>
<b>13</b>	<p><b>DATA PARITY ERROR LW 6</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error is detected on longword 6 during the data portion of a system bus transaction. This longword corresponds to bytes 27..24 of the hexaword transferred.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>12</b>	<p><b>DATA PARITY ERROR LW 4</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error is detected on longword 4 during the data portion of a system bus transaction. This longword corresponds to bytes 19..16 of the hexaword transferred.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>11</b>	<p><b>DATA PARITY ERROR LW 2</b> <i>[read/write 1 to clear]</i></p>

(continued on next page)

## 13.5 System Bus Error Register 1 (CERR1)

**Table 13–3 (Cont.) System Bus Error Register 1 Description**

Field	Description
	<p>This bit is set when a parity error is detected on longword 2 during the data portion of a system bus transaction. This longword corresponds to bytes 11..8 of the hexaword transferred.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>10</b>	<p><b>DATA PARITY ERROR LW 0</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error is detected on longword 0 during the data portion of a system bus transaction. This longword corresponds to bytes 3..0 of the hexaword transferred.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>9</b>	<p><b>CA PARITY ERROR LW 2</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error on longword 2 of a system bus command /address transfer is detected. Please reference the Please reference the Chapter 19 for the mapping of the command/address field to the system bus longwords.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>8</b>	<p><b>CA PARITY ERROR LW 0</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error on longword 0 of a system bus command /address transfer is detected. Please reference the C- bus specification for the mapping of the command/address field to the system bus longwords.</p> <p>Write one to clear. Undefined on power-up.</p>
<b>7</b>	<p><b>MISSED READ DATA PARITY ERROR EVEN</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when an even longword parity error is detected during the transfer of read data from a responder node when the I/O module is the commander, <i>after</i> some other error has been detected by the error logic. This indicates that the full error status (for example, the FAILING COMMAND/ADDRESS , and DATA PARITY ERROR bits) is not available for this error (and therefore, the error has been "missed").</p> <p>Write one to clear. Cleared on power-up.</p>
<b>6</b>	<p><b>READ DATA PARITY ERROR EVEN</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when an even longword parity error is detected during the transfer of read data from a responder node when the I/O module is the commander. If this indicator is set, the full error status (including the Failing Command /Address, and Data Parity Error bits) is available to help identify the source of this error.</p> <p>Write one to clear. Cleared on power-up.</p>
<b>5</b>	<p><b>MISSED RSP WRITE DATA PARITY ERROR EVEN</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when an even longword parity error is detected during the transfer of write data from a commander node to an I/O module primary I/O space register (I/O module is the system bus responder), <i>after</i> some other error has been detected by the error logic. This indicates that the full error status (for example, the Failing Command/Address, and Data Parity Error bits) is not available for this error (and therefore, the error has been "missed").</p> <p>Write one to clear. Cleared on power-up.</p>
<b>4</b>	<p><b>RESPONDER WRITE DATA PARITY ERROR EVEN</b> <i>[read/write 1 to clear]</i></p>

(continued on next page)

## 13.5 System Bus Error Register 1 (CERR1)

Table 13–3 (Cont.) System Bus Error Register 1 Description

Field	Description
	<p>This bit is set when an even longword parity error is detected during the transfer of write data from a commander node on the system bus to an I/O module primary I/O space register (I/O module is the system bus responder). If this indicator is set, the full error status (including the Failing Command/Address, and Data Parity Error bits) is available to help identify the source of this error.</p> <p>Write one to clear. Cleared on power-up.</p>
<b>3</b>	<p><b>MISSED COMMAND ADDRESS PARITY ERROR EVEN</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when an even longword parity error is detected during the transfer of Command/Address from a commander node to a selected responder node, <i>after</i> some other error has been detected by the error logic. This indicates that the full error status (for example, the Failing Command/Address, and Data Parity Error bits) is not available for this error and therefore, the error has been "missed".</p> <p>This bit is set by the I/O module when a parity error on any <i>even</i> Longword of the Command/Address transfer occurs, regardless of whether this module is the intended responder.</p> <p>Write one to clear. Cleared on power-up.</p>
<b>2</b>	<p><b>COMMAND ADDRESS PARITY ERROR EVEN</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when a parity error is detected on the even longwords of a system bus Command/Address transfer regardless of whether the transfer is directed to this module or not. When this bit is set, the error status in the CERR2 and CERR3 registers and in CERR1 [9:8] is available to help locate the source of the error.</p> <p>Write one to clear. Cleared on power-up.</p>
<b>1</b>	<p><b>NO ACKNOWLEDGE ERROR EVEN</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when, while acting as the system bus master, the I/O module fails to receive an acknowledge for the command/address that it has placed on the bus. This error indicates that the selected device does not exist, the selected device is broken, the I/O module is broken, or a command/address parity error has occurred during the cycle. The failing command/address is saved in CERR2 and CERR3. Write one to clear. Cleared at power up or after a system bus reset.</p>
<b>0</b>	<p><b>UNCORRECTABLE READ ERROR EVEN</b> <i>[read/write 1 to clear]</i></p> <p>This bit is set when, while acting as the system bus master, the I/O module receives an uncorrectable read data error from the selected device. The failing command/address is saved in CERR2 and CERR3. Write one to clear. Cleared at power up or after a system bus reset.</p>

## 13.6 System Bus Error Register 2 (CERR2)

This register stores information related to a system bus errors.

## 13.6 System Bus Error Register 2 (CERR2)

Figure 13–8 System Bus Error Register 2 (CERR2)

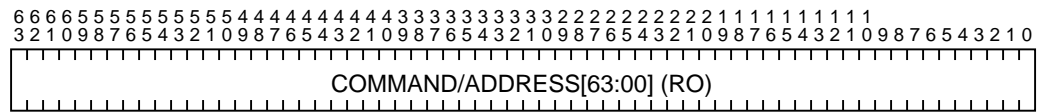


Table 13–4 System Bus Error Register 2 Description

Field	Description
<b>63:0</b>	<b>COMMAND/ADDRESS[63:00]</b> <i>[read-only]</i> These bits correspond to system bus CAD [63:00] during the command/address transfer of the failing cycle.

## 13.7 System Bus Error Register 3 (CERR3)

This register stores information related to a system bus errors.

Figure 13–9 System Bus Error Register 3 (CERR3)

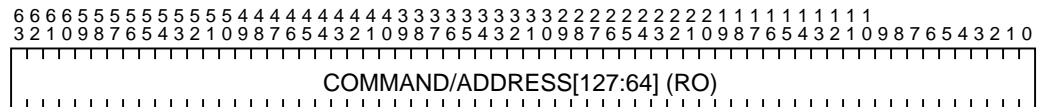


Table 13–5 System Bus Error Register 3 Description

Field	Description
<b>63:0</b>	<b>COMMAND/ADDRESS[127:64]</b> <i>[read-only]</i> These bits correspond to system bus CAD [127:64] during the command/address transfer of the failing cycle.

## 13.8 Lbus Mailbox Pointer Register (LMBPR)

This register is written with the address of the mailbox data structure in order to perform a read or write operation to the secondary I/O space of the Lbus. A STQ\_C instruction must be used to write this register. The STQ\_C fails (low bit clear) if the Lbus mailbox mechanism is currently in use. Writing this register causes the I/O module to fetch the mailbox data structure in memory and perform the requested operation. If an error occurs during an Lbus mailbox operation, the contents of the LMBPR are frozen until the LBUS MAILBOX ERROR bit in CERR1 is cleared. This allows the failing mailbox data structure to be located for error recovery purposes.

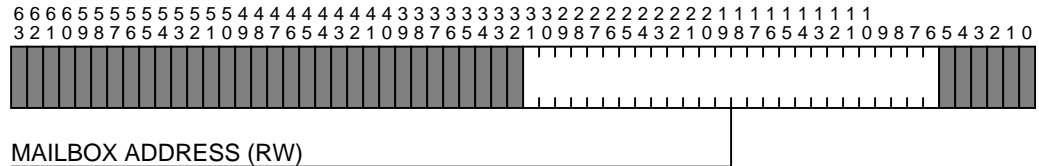
The MAILBOX ENABLE bits in the IOCSR must be set in order to write this register and have the mailbox data structure fetched from memory. The LBUS DMA ENABLE bits in the IOCSR must be set to allow the mailbox response and

## 13.8 Lbus Mailbox Pointer Register (LMBPR)

any read data returned to the mailbox data structure in memory to free up this register for further writes.

The assertion of the LBUS RESET bits in the IO\_CSR or a system bus reset causes any mailbox operation in progress to be aborted. The LMBPR is freed. That is, the STQ\_C no longer fails. The contents of the mailbox data structure in memory is not updated to reflect the aborted operation.

**Figure 13–10 Lbus Mailbox Pointer Register (LMBPR)**



**Table 13–6 Lbus Mailbox Pointer Register Description**

Field	Description
<b>31:6</b>	<b>MAILBOX ADDRESS</b> <i>[read/write]</i> This field contains the Lbus mailbox address.

## 13.9 Futurebus+ Mailbox Pointer Register (FMBPR)

This register is written with the address of the mailbox data structure in order to perform a read or write operation to the secondary I/O space of the Futurebus+. A STQ\_C instruction must be used to write this register. The STQ\_C fails (low bit clear) if the Futurebus+ mailbox mechanism is currently in use. Writing this register causes the I/O module to fetch the mailbox data structure in memory and perform the requested operation. If an error occurs during an Futurebus+ mailbox operation, the contents of the FMBPR are frozen until the FUTUREBUS+ MAILBOX ERROR bit in CERR1 is cleared. This allows the failing mailbox data structure to be located for error recovery purposes.

The MAILBOX ENABLE bits in the IOCSR must be set in order to write this register and have the mailbox data structure fetched from memory. The FUTUREBUS+ DMA ENABLE bits in the IOCSR must be set in order to allow the mailbox response and any read data returned to the mailbox data structure in memory and to free up this register for further writes.

The assertion of the FUTUREBUS+ RESET bit in the IO\_CSR or a system bus reset causes any mailbox operation in progress to be aborted. The FMBPR becomes free, that is, the STQ\_C no longer fails. The contents of the mailbox data structure in memory is not updated to reflect the aborted operation.

## 13.9 Futurebus+ Mailbox Pointer Register (FMBPR)

Figure 13–11 Futurebus+ Mailbox Pointer Register (FMBPR)

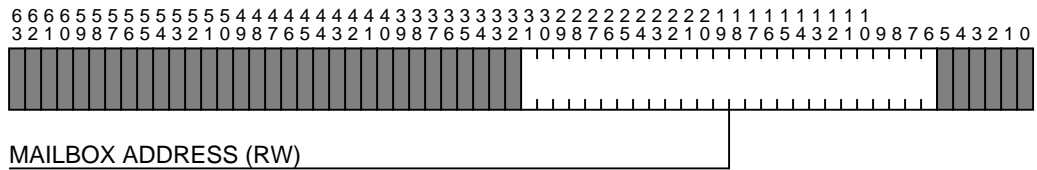


Table 13–7 Futurebus+ Mailbox Pointer Register Description

Field	Description
<b>31:6</b>	<b>MAILBOX ADDRESS</b> <i>[read/write]</i> This field holds the address of the Futurebus+ mailbox.

## 13.10 Diagnostic Control/Status Register (DIAGCSR)

Figure 13–12 Diagnostic Control/Status Register (DIAGCSR)

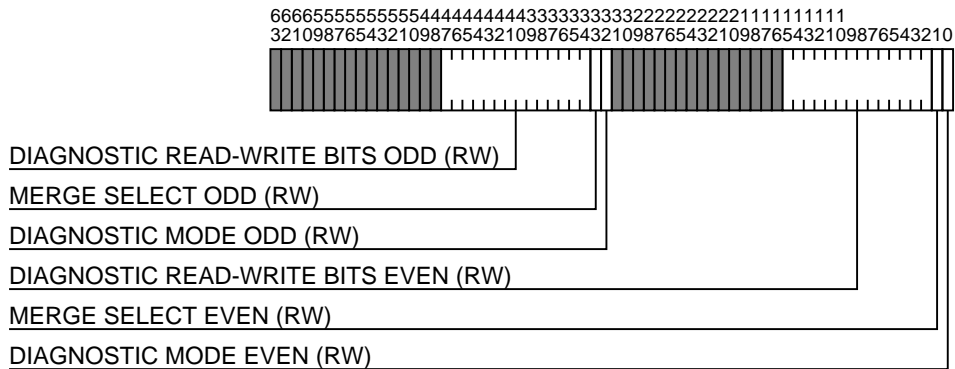


Table 13–8 Diagnostic Control/Status Register Description

Field	Description
<b>47:34</b>	<b>DIAGNOSTIC READ-WRITE BITS ODD</b> <i>[read/write]</i> These are general-purpose data bits that can be used by diagnostic routines to test read and write operations to the primary register space. Cleared on power-up.
<b>33</b>	<b>MERGE SELECT ODD</b> <i>[read/write]</i>

(continued on next page)

## 13.10 Diagnostic Control/Status Register (DIAGCSR)

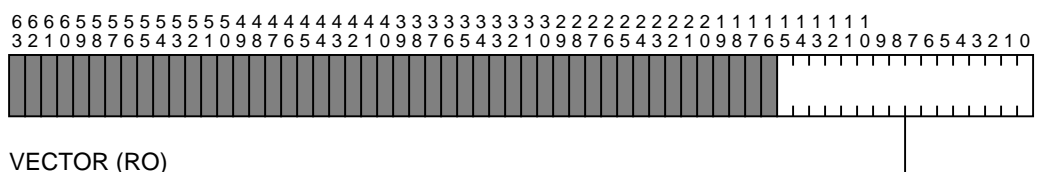
Table 13–8 (Cont.) Diagnostic Control/Status Register Description

Field	Description
	This bit controls the read data path of the diagnostic registers. When clear, the diagnostic registers $DLC_{xy}$ , $DLM_{xy}$ , $DFC_{xy}$ , and $DFM_{xy}$ in the odd interface gate array are read back from the cache line or mailbox buffers that are used during Cobra Bus read operations. When set, the registers are read back from the write buffer used during DMA write operations and mailbox status writes. Cleared on power-up.
<b>32</b>	<b>DIAGNOSTIC MODE ODD</b> <i>[read/write]</i> This bit places the data path in the odd longword slice gate array into diagnostic mode. DMA operation from the local I/O devices and the Futurebus+ is disabled. This bit enables the remaining DIAGCSR register bits for control of the gate array data paths. This bit also enables the diagnostic data path registers ( $DF_{xxx}$ and $DL_{xxx}$ registers) for direct read and write of the storage elements in the gate array cache line buffers.
<b>15:2</b>	<b>DIAGNOSTIC READ-WRITE BITS EVEN</b> <i>[read/write]</i> These are general-purpose data bits that can be used by diagnostic routines to test read and write operations to the primary register space. Cleared on power-up.
<b>1</b>	<b>MERGE SELECT EVEN</b> <i>[read/write]</i> This bit controls the read data path of the diagnostic registers. When clear, the diagnostic registers $DLC_{xy}$ , $DLM_{xy}$ , $DFC_{xy}$ , and $DFM_{xy}$ in the even interface gate array are read back from the cache line or mailbox buffers that are used during system bus read operations. When set, the registers are read back from the write buffer used during DMA write operations and mailbox status writes. Cleared on power-up.
<b>0</b>	<b>DIAGNOSTIC MODE EVEN</b> <i>[read/write]</i> This bit places the data path in the even longword slice gate array in diagnostic mode. DMA operation from the local I/O devices and the Futurebus+ is disabled. This bit enables the remaining DIAGCSR register bits for control of the gate array data paths. This bit also enables the diagnostic data path registers ( $DF_{xxx}$ and $DL_{xxx}$ registers) for direct reading and writing of the storage elements in the gate array cache line buffers. Cleared on power-up.

## 13.11 Futurebus+ Interrupt Vector Register (FIVECT)

This register stores vector information related to Futurebus+ interrupts.

Figure 13–13 Futurebus Interrupt Vector Register (FIVECT)



## 13.11 Futurebus+ Interrupt Vector Register (FIVECT)

Table 13–9 Futurebus Interrupt Vector Register Description

Field	Description
<b>15:0</b>	<b>VECTOR</b> <i>[read-only]</i> This field contains the vector for the interrupting Futurebus+ device. This register is the head of a FIFO queue for Futurebus+ interrupts. Reading this register removes one entry from the queue. If the queue is empty, reading this field returns zeros. These bits are cleared on power-up or by setting the FUTUREBUS+ RESET EVEN and FUTUREBUS+ RESET ODD (bits 39 and 7 respectively) bits in the IOCSR.

## 13.12 Futurebus Halt Vector Register (FHVECT)

Figure 13–14 Futurebus Halt Vector Register (FHVECT)

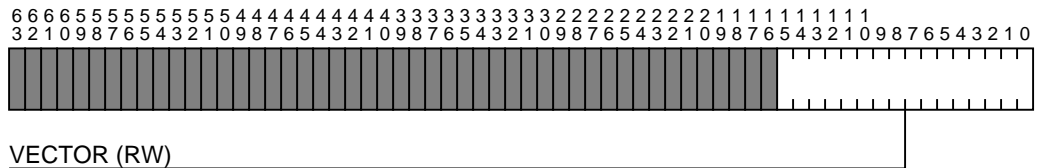


Table 13–10 Futurebus Halt Vector Register Description

Field	Description
<b>15:0</b>	<b>VECTOR</b> <i>[read/write]</i> This field contains the vector for the Futurebus+ device that has requested a CPU halt by writing a vector to the Futurebus+ adapter Futurebus+ Halt Request register. Reading this register removes the vector value and allows Futurebus+ Halt Request register to be written again by a Futurebus+ device. Subsequent reads of the register return zeros until a new halt vector is written by a Futurebus+ device. Cleared at power-up or by setting the FUTUREBUS+ RESET EVEN and FUTUREBUS+ RESET ODD bits in the IOCSR.

## 13.13 Futurebus Error Register 1 (FERR1)



## 13.13 Futurebus Error Register 1 (FERR1)

Figure 13–15 Futurebus Error Register 1 (FERR1)

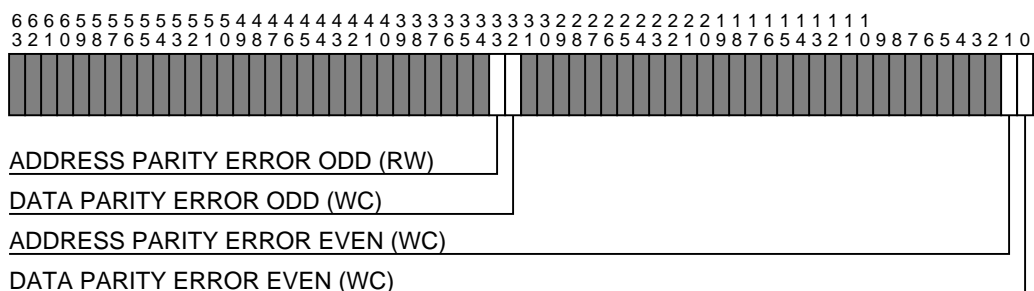


Table 13–11 Futurebus Error Register 1 Description

Field	Description
<b>33</b>	<b>ADDRESS PARITY ERROR ODD</b> <i>[read/write]</i>  This bit is set when an address parity error is detected by the even gate array slice during the connection phase of a Futurebus+ transaction. The I/O module operates as the potential Futurebus+ slave during this operation. When this bit is set, the C_ERR hardware error interrupt is asserted on the system bus. Because the address is latched in both the even and odd gate array slices, either one or both slices may detect the parity error. Write one to clear, cleared by the Futurebus+ Reset bit in the IO_CSR register or by a system bus reset.
<b>32</b>	<b>DATA PARITY ERROR ODD</b> <i>[read/write 1 to clear]</i>  This bit is set when (1) a parity error in an odd longword is detected during a Futurebus+ write transaction when the I/O module is operating as the Futurebus+ slave or (2) on a mailbox read operation. When this bit is set, the C_ERR hardware error interrupt is asserted on the system bus. Write one to clear, cleared by the Futurebus+ Reset bit in the IO_CSR register or by a system bus reset.
<b>1</b>	<b>ADDRESS PARITY ERROR EVEN</b> <i>[read/write 1 to clear]</i>  This bit is set when an address parity error is detected by the even gate array slice during the connection phase of a Futurebus+ transaction. The I/O module operates as the potential Futurebus+ slave during this operation. When this bit is set, the C_ERR hardware error interrupt is asserted on the system bus. Because the address is latched in both the even and odd gate array slices, either one or both slices may detect the parity error. Write one to clear, cleared by the FUTUREBUS+ RESET bit in the IO_CSR register or by a system bus reset.
<b>0</b>	<b>DATA PARITY ERROR EVEN</b> <i>[read/write 1 to clear]</i>  This bit is set when a parity error in an even longword is detected during a Futurebus+ write transaction when the I/O module is operating as the Futurebus+ slave or on a mailbox read operation. When this bit is set, the C_ERR hardware error interrupt is asserted on the system bus. Write one to clear. Cleared by the FUTUREBUS+ RESET bit in the IO_CSR register or by a system bus reset.

## 13.14 Futurebus Error Register 2 (FERR2)

## 13.14 Futurebus Error Register 2 (FERR2)

Figure 13–16 Futurebus Error Register 2 (FERR2)

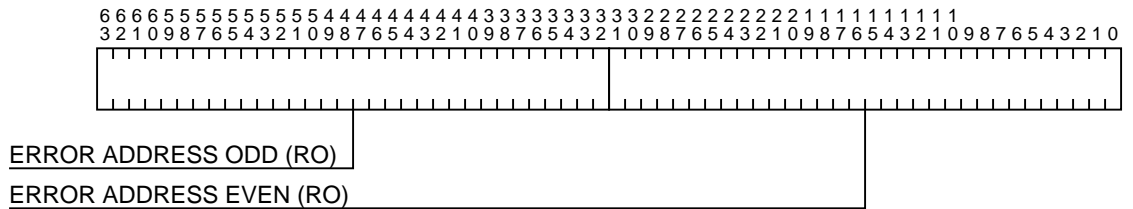


Table 13–12 Futurebus Error Register 2 Description

Field	Description
<b>63:32</b>	<b>ERROR ADDRESS ODD</b> <i>[read-only]</i> When the Address Parity Error odd bit in FERR1 is set, this field contains the failing address. If the bit is not set, the contents of this register are unpredictable. This register is not cleared by any reset or power-up condition but is constantly loaded with the address on the Futurebus+. The contents of the register are frozen by an address parity error. The loading of addresses resumes when the Address Parity Error odd bit in FERR1 is cleared.
<b>31:0</b>	<b>ERROR ADDRESS EVEN</b> <i>[read-only]</i> When the ADDRESS PARITY ERROR EVEN bit in FERR1 is set, this field contains the failing address. If the bit is not set, the contents of this register are unpredictable. This register is not cleared by any reset or power-up condition but is constantly loaded with the address on the Futurebus+. The contents of the register are frozen by an address parity error. The loading of addresses resumes when the ADDRESS PARITY ERROR EVEN bit in FERR1 is cleared.

## 13.15 Local Interrupt Register (LINT)

## 13.15 Local Interrupt Register (LINT)

Figure 13–17 Local Interrupt Register (LINT)

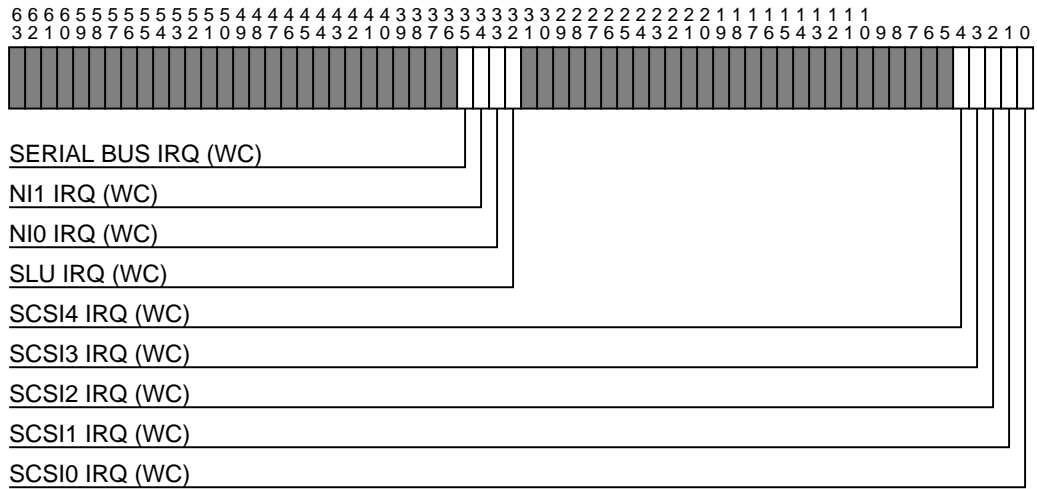


Table 13–13 Local Interrupt Register Description

Field	Description
<b>35</b>	<b>SERIAL BUS IRQ</b> <i>[read/write 1 to clear]</i> This bit is asserted if the I <sup>2</sup> C-bus controller is requesting an interrupt. Write one to clear, cleared on power-up or after an Lbus Reset.
<b>34</b>	<b>NI1 IRQ</b> <i>[read/write 1 to clear]</i> This bit is asserted if local Ethernet controller 1 is requesting and interrupt. Write one to clear, cleared on power up or after an Lbus reset.
<b>33</b>	<b>NI0 IRQ</b> <i>[read/write 1 to clear]</i> This bit is asserted if local Ethernet controller 0 is requesting and interrupt. Write one to clear, cleared on power up or after an Lbus reset.
<b>32</b>	<b>SLU IRQ</b> <i>[read/write 1 to clear]</i> This bit is asserted when the controller for the serial lines is requesting an interrupt. Write one to clear, cleared on power up or after an Lbus reset.
<b>4</b>	<b>SCSI4 IRQ</b> <i>[read/write 1 to clear]</i> This bit is asserted if SCSI/DSSI controller 4 is requesting an interrupt. Write one to clear, cleared on power-up or after an Lbus reset.
<b>3</b>	<b>SCSI3 IRQ</b> <i>[read/write 1 to clear]</i> This bit is asserted if SCSI/DSSI controller 3 is requesting an interrupt. Write one to clear, cleared on power-up or after an Lbus reset.
<b>2</b>	<b>SCSI2 IRQ</b> <i>[read/write 1 to clear]</i> This bit is asserted if SCSI/DSSI controller 2 is requesting an interrupt. Write one to clear, cleared on power-up or after an Lbus reset.
<b>1</b>	<b>SCSI1 IRQ</b> <i>[read/write 1 to clear]</i> This bit is asserted if SCSI/DSSI controller 1 is requesting an interrupt. Write one to clear, cleared on power-up or after an Lbus reset.
<b>0</b>	<b>SCSI0 IRQ</b> <i>[read/write 1 to clear]</i>

(continued on next page)

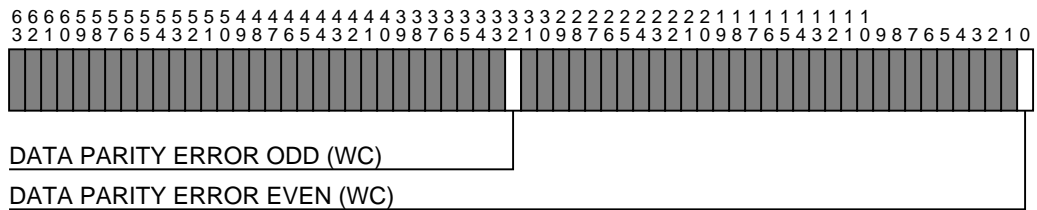
## 13.15 Local Interrupt Register (LINT)

**Table 13–13 (Cont.) Local Interrupt Register Description**

Field	Description
	This bit is asserted if SCSI/DSSI controller 0 is requesting an interrupt. Write one to clear, cleared on power-up or after an Lbus reset.

## 13.16 Lbus Error Register 1 (LERR1)

**Figure 13–18 Lbus Error Register 1 (LERR1)**



**Table 13–14 Lbus Error Register 1 Description**

Field	Description
<b>32</b>	<p><b>DATA PARITY ERROR ODD</b> <i>[read/write 1 to clear]</i></p> <p>This bit indicates that there was a data parity error on the local I/O bus detected by the odd gate array. Data parity errors are detected during the following transactions:</p> <ul style="list-style-type: none"> <li>• TGEC DMA write cycle</li> <li>• SCSI controller DMA write cycle</li> </ul>
<p><b>Note</b></p> <p>Data parity errors during a mailbox read of the SCSI Script RAM or TGEC CSR are reported in the status field of the mailbox data structure. Parity errors during a TGEC or SCSI controller DMA read cycle are reported by the specific device.</p>	
<b>0</b>	<p><b>DATA PARITY ERROR EVEN</b> <i>[read/write 1 to clear]</i></p> <p>When this bit is set, the C_ERR hardware error interrupt is asserted on the system bus. Write one to clear, cleared on power-up or by setting the Lbus Reset bits in the IOCSR register.</p>

(continued on next page)

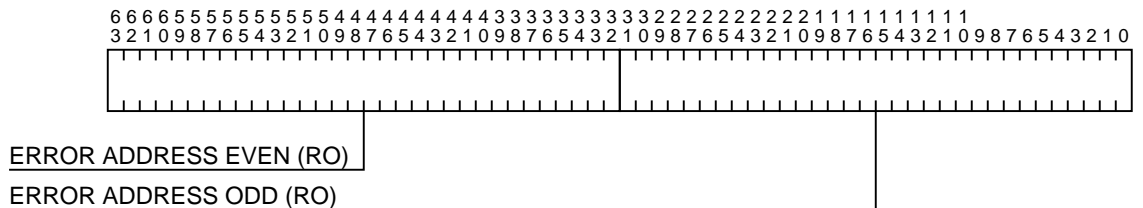
## 13.16 Lbus Error Register 1 (LERR1)

**Table 13–14 (Cont.) Lbus Error Register 1 Description**

Field	Description
	<p>This bit indicates that there was a data parity error on the local I/O bus detected by the even gate array. Data parity errors are indicated with this register during the following transactions:</p> <ul style="list-style-type: none"> <li>• TGEC DMA write cycle</li> <li>• SCSI controller DMA write cycle</li> </ul>
	<p><b>Note</b></p> <p>Data parity errors during a mailbox read of the SCSI Script RAM or TGEC CSR are reported in the status field of the mailbox data structure. Parity errors during a TGEC or SCSI controller DMA read cycle are reported by the specific device.</p>
	<p>When this bit is set, the C_ERR hardware error interrupt is asserted on the system bus. Write one to clear, cleared on power-up or by setting the Lbus Reset bits in the IOCSR register.</p>

## 13.17 Lbus Error Register 2 (LERR2)

**Figure 13–19 Lbus Error Register 2 (LERR2)**



**Table 13–15 Lbus Error Register 2 Description**

Field	Description
<b>31:0</b>	<b>ERROR ADDRESS ODD</b> <i>[read-only]</i>

(continued on next page)

## 13.17 Lbus Error Register 2 (LERR2)

Table 13–15 (Cont.) Lbus Error Register 2 Description

Field	Description
	<p>When the DATA Parity Error odd bit in LERR1 is set this register contains the failing address. If the error occurred during a mailbox read of the SCSI Script RAM this register contains the address of the mailbox data structure in memory. If the error occurred during a TGEC or SCSI controller DMA write cycle this register contains the memory address of the DMA write operation.</p> <p>This register is not cleared by any reset or power-up condition but is constantly loaded with a new address at the beginning of every Lbus cycle. The contents of the register are frozen by a detected data parity error. The loading of addresses resumes when the Data Parity Error odd bit in LERR1 is cleared.</p>
<b>63:32</b>	<p><b>ERROR ADDRESS EVEN</b> <i>[read-only]</i></p> <p>When the DATA Parity Error even bit in LERR1 is set this register contains the failing address. If the error occurred during a mailbox read of the SCSI Script RAM this register contains the address of the mailbox data structure in memory. If the error occurred during a TGEC or SCSI controller DMA write cycle this register contains the memory address of the DMA write operation.</p> <p>This register is not cleared by any reset or power-up condition but is constantly loaded with a new address at the beginning of every Lbus cycle. The contents of the register are frozen by a detected data parity error. The loading of addresses resumes when the Data Parity Error even bit in LERR1 is cleared.</p>

## 13.18 Secondary I/O Space

The main function of the I/O module's interface to the system bus is to provide an adapter between the system bus and two secondary buses. These secondary buses are:

- The Futurebus+ which is the open bus available for the user
- The Local I/O bus which is a 32-bit bus connecting the local I/O devices contained on the KFA40 I/O module.

Read operations directed at a secondary bus are very slow relative to CPU clock speed. A CSR read may take 1 to 10  $\mu$ s or even longer, while a processor clock cycle is under 10 ns.

Because the 21064 processor pin bus and the DEC 4000 AXP system bus are nonpended, a read operation that is allowed to address a node on a secondary bus can cause the system bus to stall while data is accessed. If the system bus is allowed to stall, multiprocessors stall and I/O devices can not access main memory. This can cause data to back up and potentially overflow, producing a deadlock situation if the system bus is stalled for a CSR read and the secondary bus is stalled trying to access the system bus. Because it is unacceptable for the entire system to stall for an indeterminate amount of time, (1) the system bus must be complicated by adding pended operations, or (2) device drivers must be complicated by "pending" CSR reads.

### 13.18.1 Mailbox Data Structure

In order to decouple the comparatively long access latencies of these buses from the system bus (which the CPU uses for access to main memory) these buses are not directly accessible by the processor. A mechanism called a mailbox has been provided for access to devices on either of the secondary buses. The processor builds a structure in main memory called the mailbox data structure that describes the operation to be performed. The processor then writes a pointer to this structure into a mailbox pointer register. The I/O module reads the mailbox

data structure, performs the operation specified (read or write) and returns status and any data to the structure in memory.

An additional problem that must be handled is that I/O space typically has restrictions on the size of transaction that is allowed. Sometimes byte operations are required. Often, longword operations are required. The 21064 processor does not provide byte address information. To get longword or byte addresses the mailbox data structure contains a byte mask field.

### 13.18.2 Software-visible Control Flow

To provide software-visible control flow, a store quad conditional (STQ\_C) instruction is used to access the mailbox pointer register. The STQ\_C to I/O space does not require a preceding load lock (LDx/L) instruction as with normal lock operations in memory space. The STQ\_C failure (low bit clear) indicates that the register being accessed is busy. The processor code can then reschedule the register write operation.

The atomic STQ\_C to the mailbox pointer registers alerts the I/O module to read the mailbox data structure. Ownership of the mailbox data structure passes between the CPU and I/O module so that at any given time only one is writing and the other is reading. Because the memory locations that contain the mailbox data structure are cached by the CPU module, bus traffic is minimized.

### 13.18.3 Mailbox Pointer Registers

There are two mailbox pointer registers, the Lbus mailbox pointer register (LMBPR) for operations to the local I/O devices and the Futurebus+ mailbox pointer register (FMBPR) for devices on the Futurebus. The DEC 4000 AXP I/O module implements only one pointer behind each mailbox pointer register (a one deep queue). If this pointer is in use, additional writes to the mailbox pointer register result in STQ\_C failures.

Software must manage the allocation of mailbox data structures. Subsequent operations must not overwrite data which is still in use.

New drivers and applications should use this mailbox mechanism directly. To allow existing drivers and some applications that touch I/O space to run without modification, operating systems could map secondary I/O locations to some virtual address region. An access to this virtual address range could then trap to a PAL routine or exception handler which is capable of using the mailbox mechanism. However, performance is not as good as a driver that uses the mailbox mechanism directly.

Figure 13–20 shows the format of the memory resident mailbox data structure.

## 13.18 Secondary I/O Space

**Figure 13–20 Mailbox Mechanism**

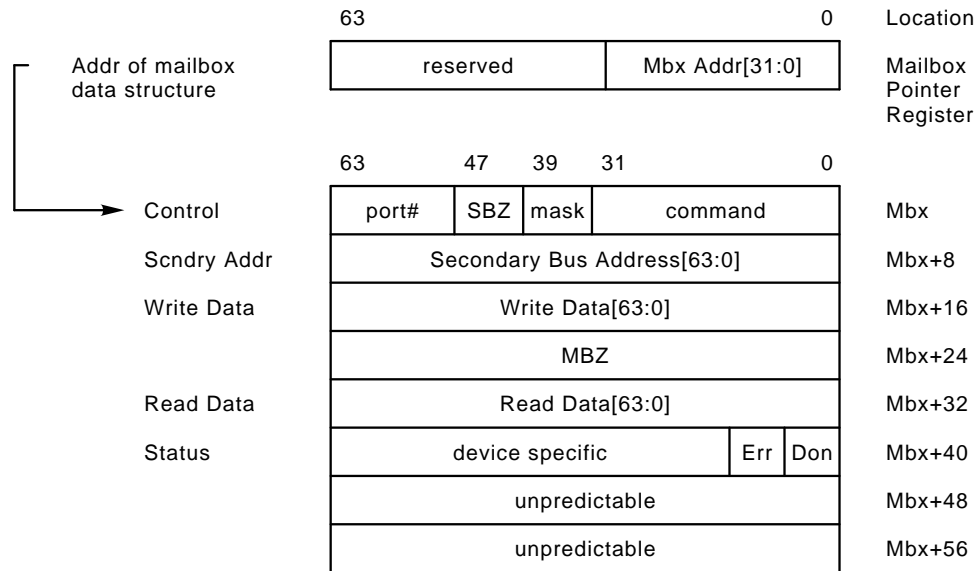


Table 13–16 describes the fields in the mailbox data structure.

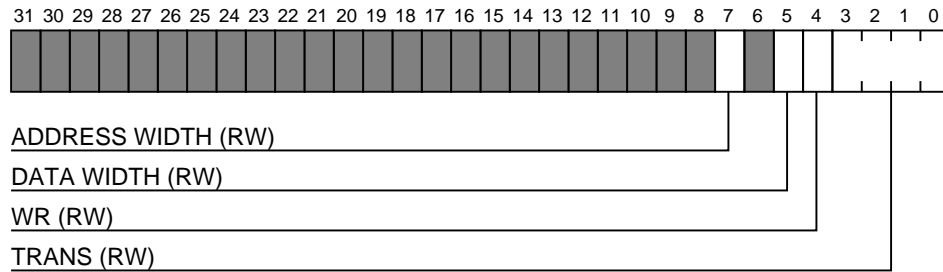
**Table 13–16 Mailbox Data Structure**

Field	Description
Control[31:0]	Command[31:0]: Bus specific command; the intention of this field is that these bits map exactly to command lines on the bus.
Control[39:32]	Mask[7:0]: Byte mask within the given naturally-aligned transfer. Set bit to 1 to prevent corresponding byte location from being written. The number of valid byte mask bits is determined by the data width field above. Byte mask is ignored unless partial operation is specified in transaction type.
Control[63:48]	Port[15:00]: Specifies the remote adapter for the operation. This field is not used by the DEC 4000 AXP I/O system because both the Lbus and Futurebus+ adapters are one piece designs and are not separated by a remote cable.
Write Data[63:00]	Contains the data to be written to the secondary bus device during a mailbox write operation. Data bits above the natural width of the device or within bytes specifically disabled with the mask field (for those devices that support masked writes) are not used during the write operation.
Read Data[63:00]	Contains data returned during a mailbox read operation. Data bits above the natural width of the device accessed are unpredictable.
Status[0]	Done: cleared by processor, set by I/O module when the mailbox operation is complete.
Status[1]	Error: This bit is not valid unless the done bit is set
Status[63:2]	Implementation-specific error status



The two secondary I/O buses (Futurebus+ and Local I/O bus) use different encodings of the command field. The following section describes the command field encoding for the two buses.

**Figure 13–21 Futurebus+ Mailbox Command Field (FCMD)**



**Table 13–17 Futurebus+ Mailbox Command Field Description**

Field	Description
7	<p><b>ADDRESS WIDTH</b> <i>[read/write]</i></p> <p>0 = 32-bit 1 = 64-bit</p>
5	<p><b>DATA WIDTH</b> <i>[read/write]</i></p> <p>0 = 32-bit 1 = 64-bit</p>
<p><b>Note</b></p> <p>The data width field takes precedence over the low order address bits in the Secondary Bus Address field.</p>	
4	<p><b>WR</b> <i>[read/write]</i></p> <p>0 = read 1 = write</p>
3:0	<p><b>TRANS</b> <i>[read/write]</i></p> <p>0 = unmasked 2 = partial - byte mask is valid</p>

## 13.18 Secondary I/O Space

Figure 13–22 Futurebus+ Mailbox Status Field (FSTAT)

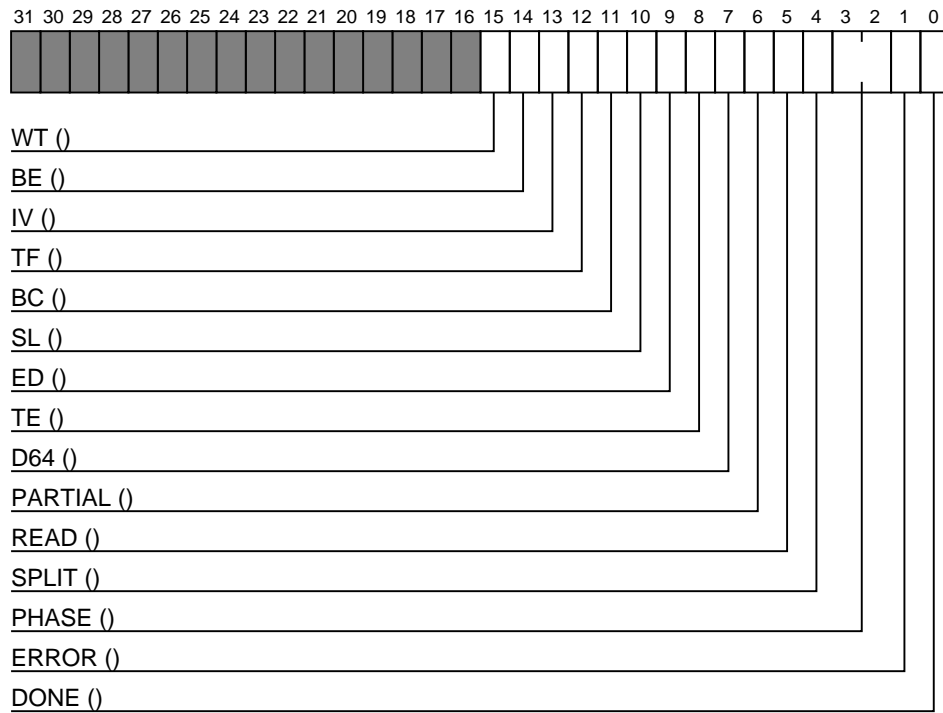


Table 13–18 Futurebus+ Mailbox Status Field Description

Field	Description
<b>15</b>	<b>WT</b> Futurebus+ WT status bit
<b>14</b>	<b>BE</b> Futurebus+ BE status bit
<b>13</b>	<b>IV</b> Futurebus+ IV status bit
<b>12</b>	<b>TF</b> Futurebus+ TF status bit
<b>11</b>	<b>BC</b> Futurebus+ BC status bit
<b>10</b>	<b>SL</b> Futurebus+ SL status bit
<b>9</b>	<b>ED</b> Futurebus+ ED status bit
<b>8</b>	<b>TE</b> Futurebus+ TE status bit
<b>7</b>	<b>D64</b> The Futurebus+ D64 command bit, echoed from the mailbox command field.
<b>6</b>	<b>PARTIAL</b> The Futurebus+ Partial command bit, echoed from the mailbox command field.
<b>5</b>	<b>READ</b> The Futurebus+ Read command bit, echoed from the mailbox command field.
<b>4</b>	<b>SPLIT</b> Indicates the mailbox operation was split by the device on the Futurebus+.
<b>3:2</b>	<b>PHASE</b> Bus phase that the error was captured in. Because of the pipelined nature of Futurebus+ error detection, this may be the phase after the error actually occurred. <ul style="list-style-type: none"> <li>• 11 = Idle</li> <li>• 10 = Connection</li> <li>• 01 = Disconnection</li> <li>• 00 = Data</li> </ul>
<b>1</b>	<b>ERROR</b> Indicates that an error was detected during the mailbox operation. Not valid unless the Done bit is also set.
<b>0</b>	<b>DONE</b> This bit is set by the I/O module when the mailbox operation has been completed and indicates that the remaining mailbox status is valid.

## 13.18 Secondary I/O Space

**Figure 13–23 Lbus Mailbox Command Field (LCMD)**

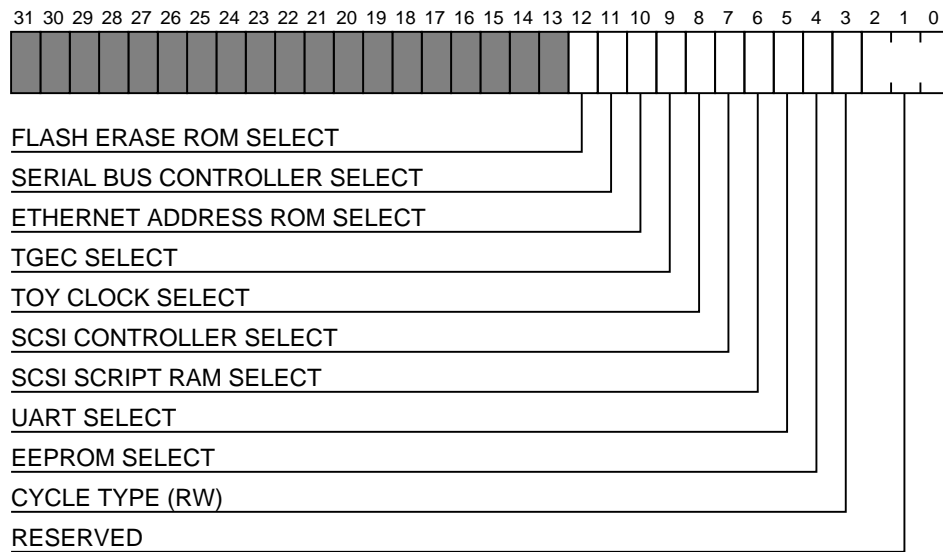
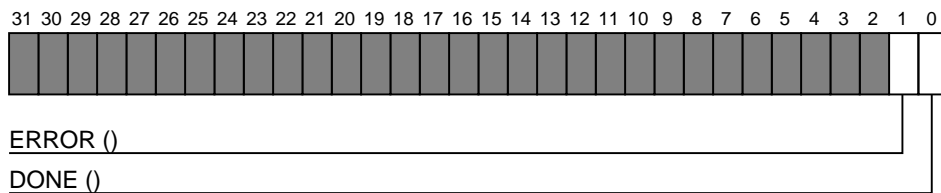


Table 13–19 Lbus Mailbox Command Field Description

Field	Description
<b>12</b>	<b>FLASH ERASE ROM SELECT</b> This bit, when set, selects the Flash Erase ROMs as the target for the mailbox operation. Bits <18:2> of the Secondary Bus Address select the longword location within the ROMs.
<b>11</b>	<b>SERIAL BUS CONTROLLER SELECT</b> This bit when set selects the $I^2C$ – bus controller (serial control bus controller) as the target for the mailbox operation. Bit 2 selects the register within the device.
<b>10</b>	<b>ETHERNET ADDRESS ROM SELECT</b> This bit when set selects the Ethernet station address ROM as the target for the mailbox operation. Bits <7:2> of the Secondary Bus Address selects the location within the ROM.
<b>9</b>	<b>TGEC SELECT</b> This bit, when set, selects the TGEC Ethernet interface devices as the target for the mailbox operation. Bit <7> of the Secondary Bus Address selects one of the two TGEC interfaces, bits <6:2> select the register within the device.
<b>8</b>	<b>TOY CLOCK SELECT</b> This bit when set selects the DS1287 device as the target for the mailbox operation. Bits <7:2> of the Secondary Bus Address select the register within the device.
<b>7</b>	<b>SCSI CONTROLLER SELECT</b> This bit, when set, selects the five SCSI controller (53C710) devices as the target for the mailbox operation. Bits <8:6> of the Secondary Bus Address select one of the five SCSI controllers, bits <5:2> of the Secondary Bus Address select the register within the device.
<b>6</b>	<b>SCSI SCRIPT RAM SELECT</b> This bit, when set, selects the Script RAM for the SCSI controllers as the target for the mailbox operation. Bits <16:2> of the Secondary Bus Address select the longword within the Script RAM buffer.
<b>5</b>	<b>UART SELECT</b> This bit, when set, selects the 85C30 dual UART device as the target for the mailbox operation. Bits <3:2> of the Secondary Bus Address select the register within the device.
<b>4</b>	<b>EEPROM SELECT</b> This bit, when set, selects the EEPROM device as the target for the mailbox operation. Bits <14:2> of the secondary bus address select the location within the 8Kx8 EEPROM device.
<b>3</b>	<b>CYCLE TYPE</b> <i>[read/write]</i> <ul style="list-style-type: none"> <li>• 0 = read</li> <li>• 1 = write</li> </ul>
<b>2:0</b>	<b>RESERVED</b> Reserved for future use.

## 13.18 Secondary I/O Space

**Figure 13–24 Lbus Mailbox Status Field (LSTAT)**



**Table 13–20 Lbus Mailbox Status Field Description**

Field	Description
<b>0</b>	<b>DONE</b> This bit is set by the I/O module when the mailbox operation has been completed and indicates that the remaining mailbox status is valid.
<b>1</b>	<b>ERROR</b> Indicates that an error was detected during the mailbox operation. Not valid unless the done bit is also set.

### 13.18.4 Mailbox Operation

The address of a mailbox data structure is a double-hexaword-aligned memory location. The I/O module writes into the second hexaword of the mailbox data structure only. It may duplicate the data/status fields in both halves of the second hexaword so that it can write a full cache line without a preceding read. The I/O module does not overwrite the processor's original command in the mailbox data structure so that a failed command can be debugged. Software may reuse mailbox data structures (for example, multiple reads from the same CSR), or it may maintain templates that are copied to the Mailbox Data Structure.

Byte mask bits may be needed by some hardware devices for correct operation of the read as well as the write. Setting the appropriate bit in the Control field of the Mailbox Data Structure disables the corresponding byte location.

The following is a simple description of use of the mailbox mechanism by software for both read and write operations. The *Alpha System Reference Manual* gives complete details of the mailbox operation.

#### Read Operation from a Secondary Bus Address

1. Check status word for mailbox data structure in use; Clear status word in mailbox data structure.
2. Write secondary bus address into mailbox data structure
3. Assemble control bits in CPU register (r/w, command, byte mask, port number); write to mailbox data structure.
4. STQ\_C of mailbox data structure address to mailbox pointer register
5. If STQ\_C failed, then mailbox mechanism on I/O module was busy; back off and try again. If STQ\_C succeeded, continue to do "useful work" until data is required.

6. When data is required, load status word from mailbox data structure, BLBC until data is returned (cache coherence scheme updates location when written by I/O device)
7. Check error bit. load read data from mailbox data structure if no error (note that data should already be in cache due to status word read. Additional bus transaction is not required)

Mailbox read operation overhead:

- Memory read by CPU if Mbx+32 is not in processor cache
- Memory read by CPU if Mbx is not in processor cache
- I/O write to Mailbox Pointer register
- Memory read by I/O module (data supplied by processor cache)
- Secondary bus read by I/O module.
- Memory write of data and status by I/O module (causes invalidate of processor cache)
- Memory read by CPU to get data and status

**Write Operation to a Secondary Bus Address**

1. Check status word for mailbox data structure in use. Clear status word in mailbox data structure.
2. Deposit write data into mailbox data structure
3. Write secondary bus address into mailbox data structure.
4. Assemble control bits in CPU register (r/w, command, byte mask, port number). Write to mailbox data structure.
5. STQ\_C of mailbox data structure address to mailbox pointer register.
6. If STQ\_C failed, the mailbox mechanism on the I/O module was busy. Back off and try again. If STQ\_C succeeded, write is "complete" in dump-and-run sense. The I/O module interprets mask and store data to desired location; the I/O module sets completion and any error status when done.)

Mailbox write operation overhead:

- Memory is read by CPU if Mbx+32 is not in processor cache.
- Memory is read by CPU if Mbx is not in processor cache.
- I/O write to mailbox pointer register is done.
- Memory is read by I/O module (data supplied by processor cache).
- Secondary bus write by I/O module is done.
- Memory write of status information by I/O module is done.

There are three sources of interrupts to the processor that exist on the I/O module as follows:

- Futurebus+ interrupts
- Local I/O device interrupts
- Hardware error interrupts (parity errors, timeouts, etc.).

The 21064 processor on the CPU module provides six asynchronous, level-sensitive interrupts that are individually maskable via PAL code. Prioritization of these inputs is handled in PAL code. The encoding of system interrupts onto the processor inputs is shown in Table 14–1.

**Table 14–1 DEC 4000 AXP System Interrupt Encoding**

IRQ(n)	Interrupt Source	System Bus Signal
5	System event (Power or Halt)	CSYS_EVENT_L
4	Interval timer	CINT_TIM
3	Interprocessor Interrupt	(CPU INTERNAL)
2	Futurebus+ interrupt	CIRQ_L[1]
1	Local I/O interrupt	CIRQ_L[0]
0	Hardware error	C_ERR_L

## 14.1 Futurebus+ Interrupts

Futurebus+ devices interrupt the system by issuing a write cycle to the Futurebus+ interrupt request register, a Futurebus visible CSR implemented in the I/O module's Futurebus+ adapter. At system configuration time, the processor loads the address of the interrupt CSR and the data value (vector) to be used when interrupting into the INT\_LOC and INT\_VAL registers on the Futurebus+ module. The location of these configuration registers within the Futurebus+ device is vendor specific.

The Futurebus+ adapter on the I/O module implements a simple 16 deep FIFO queue for capturing Futurebus+ interrupts. If the queue is full, the adapter issues a busy status on any further writes by a Futurebus+ device. This causes the device to release the Futurebus+ and retry the cycle.

Whenever there is data in the queue, the Futurebus+ interrupt line to the CPU module is asserted after any pending write operations in the Futurebus+ cache line buffers have been flushed out. Bit <6> in the IOCSR register is set to indicate that the Futurebus+ interrupt queue is not empty. Care must be taken when using a polled interrupt handler because the setting of Futurebus+



## 14.1 Futurebus+ Interrupts

interrupt status bit in the IOCSR does not guarantee the flushing of DMA writes. After detecting the interrupt condition, the processor reads the Futurebus+ interrupt vector register to take the first value off the queue. The CPU uses this data value to dispatch to the appropriate service routine. The interrupt line to the CPU module is deasserted after the read of the Futurebus+ interrupt vector register. The interrupt line is asserted again if more vectors remain in the FIFO after any pending writes have been flushed. This process continues until the queue is empty. A read of the Futurebus+ interrupt vector register when the queue is empty, returns a vector of all 0s.

## 14.2 Local I/O Interrupts

The interrupts from the local I/O devices (SCSI/DSSI ports, Ethernet port, and serial line units) are combined with a logical OR function into one processor interrupt. The local interrupt (LINT) register contains a bit for each interrupting device (Section 13.2).

The LINT register implements separate edge detectors for the device interrupts. If any of the bits in the LINT register are set, the CIRQ\_L[0] interrupt to the CPU is asserted after any pending write operations in the Lbus cache line buffers have been flushed. After detecting the interrupt, the processor reads the local interrupt (LINT) register to determine which device or devices are requesting service. It is the responsibility of PAL code to prioritize these inputs to determine the order in which devices are serviced.

The processor must write a 1 to the bit in the LINT register that it wishes to clear. This action causes the CIRQ\_L[0] interrupt to deassert. It is asserted again if there are more interrupt bits set in the LINT register after any writes in the cache line buffers have been flushed. The interrupt output from each device remains asserted until the processor clears the interrupting condition or disables interrupts within the particular device. The hardware depends on the device interrupt being cleared in order to guarantee another edge to set the appropriate LINT register bit.

## 14.3 Hardware Error Interrupts

The I/O module asserts the hardware error interrupt (C\_ERR) signal to the CPU module when it detects any of the conditions listed in Table 14–2. The processor interrogates the I/O control and status register (IO\_CSR), system bus error register (CERR1), Lbus error register (LERR1), and Futurebus error register (FERR1) to determine the cause. The I/O module makes no attempt to retry any failed operation for which it was the initiator.

For system bus operations, the I/O module detects parity errors on a received command/address transaction on the bus, whether the operation was directed at the I/O module or not. At other times, it is the responsibility of the transaction slave to fail any operation (no acknowledge) for which it detects an error. The transaction initiator then asserts the hardware error interrupt.

**Table 14–2 I/O Module Hardware Error Interrupt Conditions**

Error Condition	I/O Module State	Saved State
System bus command/address parity	System bus Bystander	CMD,ADR,CMD ID
System bus No Acknowledge	System bus Master	CMD,ADR, CMD ID
System bus Read Data Parity	System bus Master	CMD,ADR,CMD ID, LW error bits
System bus Uncorrectable Read	System bus Master	CMD,ADR,CMD ID
Mailbox Error	Lbus or Futurebus+ Master	Mailbox Pointer Register
Futurebus+ Write Data Parity	Futurebus+ Slave	Futurebus+ Address
Futurebus+ Address Parity	Futurebus+ Slave	Futurebus+ Address
Lbus Data Parity	Mailbox read of TGEC CSR	Mailbox Address
Lbus Data Parity	Mailbox read of SCSI CSR	Mailbox Address
Lbus Data Parity	Mailbox read of SCSI Script RAM	Mailbox Address
Lbus Data Parity	TGEC DMA write	DMA Address
Lbus Data Parity	SCSI DMA write	DMA Address

## 14.4 System Event Interrupt Requests

There are four sources of system event interrupt requests in the DEC 4000 AXP system as follows:

- Halt switch on the operator control panel
- Change in the power supply status
- Remote boot detection by the local Ethernet controller
- Futurebus+ I/O device halt interrupt request

Any of these conditions generate a `CSYS_EVENT` interrupt pulse on the system bus. This is an edge-sensitive interrupt on the processors. The processors are responsible for determining the source of the request and setting priorities for servicing multiple requests.

After detecting the system event interrupt, the processor first checks the power supply controller for a change in status and takes appropriate action to handle the new condition.

### Power supply status conditions:

- Power system shutdown
- AC power failure
- DC power failure
- Over temperature warning
- Over temperature shutdown
- Fan failure
- Disk or tape power supply (LDC) failure
- Reserve battery activated

## 14.4 System Event Interrupt Requests

The processor also reads the I/O control and status register (IOCSR) to determine if the local Ethernet controller or a Futurebus+ is requesting a CPU halt or reboot operation. If a Futurebus+ device is requesting a CPU halt, a 16-bit vector provided by the device is available in the Futurebus+ halt vector register. This vector value is software settable and is written to the device by the processor during the device initialization sequence. The halt requests from the local Ethernet controller and Futurebus+ device are individually maskable in the IOCSR.

The processor also interrogates the operator control panel to determine if the halt button has been pressed. The operator control panel and power system controller are accessed through a serial bus interface on the I/O module.

---

## Futurebus+ Adapter Architecture

The Futurebus+ adapter on the I/O module supports Futurebus+ devices compatible with Profile B as described in the IEEE 896.2 specification. The Futurebus+ is used on the DEC 4000 AXP system solely as an I/O bus. No provisions are made to support main memory on the Futurebus+. The cache coherency domain of the memory system does not extend to the Futurebus+.

While the I/O module is compatible with the electrical and protocol sections of Profile B, it is not compatible with the mechanical or CSR specifications of the profile. The module is mechanically incompatible because it is physically larger than the Futurebus+ Profile B modules and plugs into a special system backplane enclosure slot that provides connections to the internal I/O devices. The CSR incompatibility exists because the DEC 4000 AXP CPU always assumes it is the configuration processor (Monarch processor in Futurebus+ terms) on the Futurebus+; therefore it has no need to provide Futurebus+ accessible registers to allow another device on the bus to configure the system. Many of the bus protocol parameters that are settable through Futurebus+ registers are hardwired on the DEC 4000 AXP I/O module.

All Futurebus+ accesses by the CPU must be performed using the mailbox mechanism for secondary I/O space access, as described in Section 13.18.

### 15.1 Futurebus+ Address Space

#### Adapter as Futurebus+ Slave

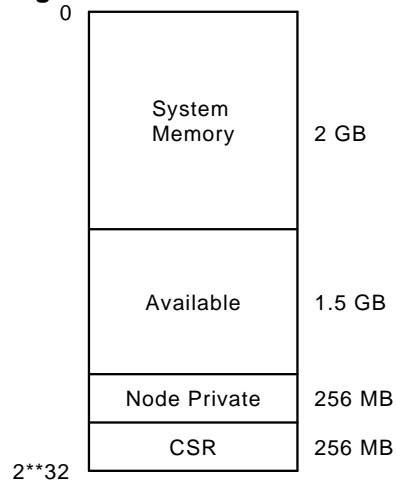
The Futurebus+ adapter on the I/O module makes the physical memory available for access by Futurebus+ modules. The adapter responds only to Futurebus+ transactions that use 32-bit addressing (A32 transactions). The adapter does not respond as a slave to Futurebus+ transactions that use 64-bit addressing (A64 transactions). Primary I/O and secondary I/O spaces are not accessible by Futurebus+ transactions.

Devices that can generate 64-bit addresses should always assume that they are in a system mixed with 32-bit nodes. CSRs must always be accessed with A32 transactions, and memory addresses that fit in 32-bit space (64-bit addresses with at least 32 leading zeros) must always use the A32 qualifier.

The Futurebus+ A32 address space is shown in Figure 15-1 and Table 15-1. A64 address space is shown in Figure 15-2.

## 15.1 Futurebus+ Address Space

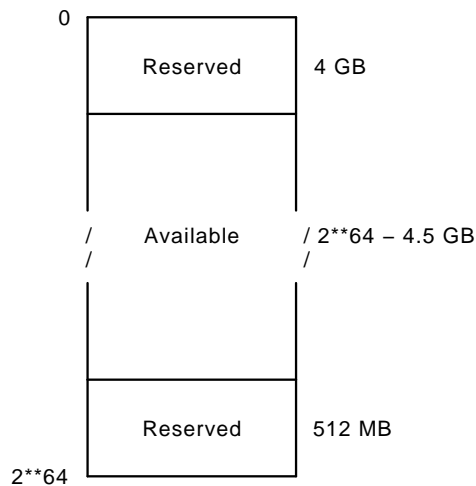
**Figure 15–1 A32 Futurebus+ Address Space**



**Table 15–1 Futurebus+ Address Mapping**

Region	Size	Address Range
System Memory	2 GB	0000 0000H - 7FFF FFFFH
Available Futurebus+ Memory Space	1.5 GB	8000 0000H - DFFF FFFFH
Node Private Space	256 MB	E000 0000H - EFFF FFFFH
Futurebus+ CSR Space	256 MB	F000 0000H - FFFF FFFFH

**Figure 15–2 A64 Futurebus+ Address Space**



### Adapter as Futurebus+ Master

The DEC 4000 AXP CPU can access Futurebus+ addresses only through the use of the mailbox mechanism (Section 13.18). This mechanism allows access to the Futurebus+ using either A32 or A64 addressing. Bit 7 in the Mailbox

## 15.1 Futurebus+ Address Space

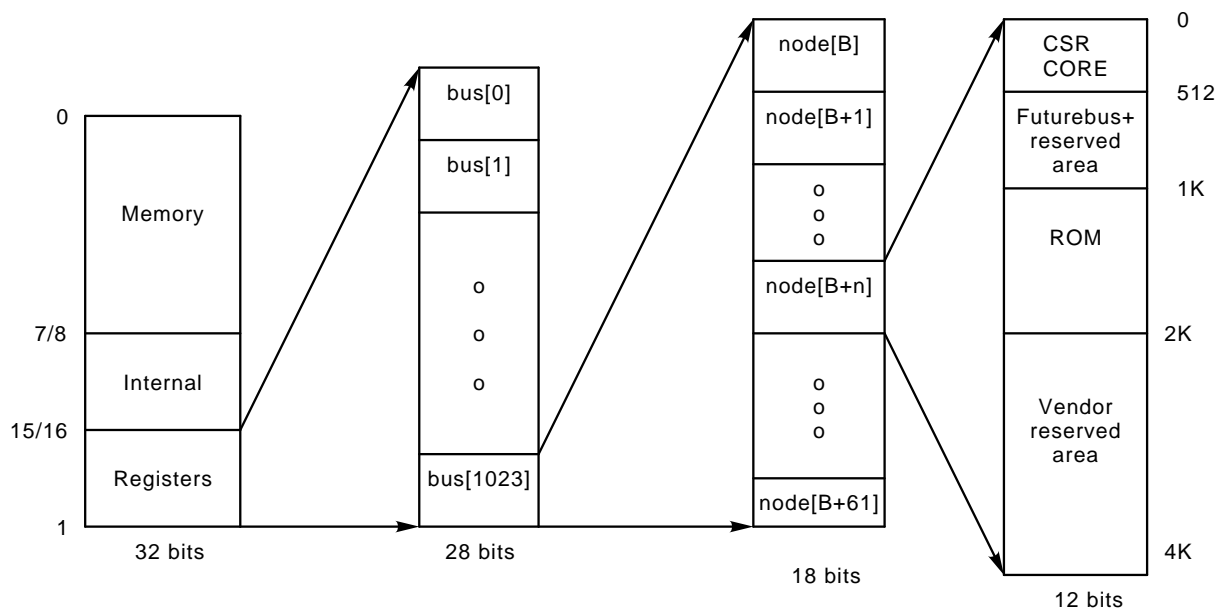
Data Structure control field specifies the address size during the Futurebus+ transaction by the adapter.

The adapter does not respond to master transactions that map back to system memory or adapter Futurebus+ registers.

### Futurebus+ Node CSR Addressing

Each Futurebus+ module is allocated 4 kbytes of address space for register space. The IEEE P1212 and IEEE 896.1 specifications describe these registers. The register space is divided into CORE CSRs, Futurebus+ specific CSRs, a node ROM window and a vendor reserved area (Figure 15–3).

Figure 15–3 Futurebus+ Node Addressing



## 15.2 Futurebus+ Adapter Registers

The I/O module provides five Futurebus+ visible registers called the Futurebus+ Interrupt Request [3..0] registers and Futurebus+ Halt Request register. These registers are located at the following Futurebus+ addresses (bus 1023, node 0):

- Futurebus+ Interrupt Request [0] - FFFC 0800
- Futurebus+ Interrupt Request [1] - FFFC 0804
- Futurebus+ Interrupt Request [2] - FFFC 0808
- Futurebus+ Interrupt Request [3] - FFFC 080C
- Futurebus+ Halt Request - FFFC 0810

The Futurebus+ Interrupt Request [3..0] registers are used by Futurebus+ modules to cause a CPU interrupt. These are write only registers that can be accessed only by using the 32-bit addressing mode of the Futurebus+. These registers do not respond to A64 writes or any read cycle.

## 15.2 Futurebus+ Adapter Registers

The DEC 4000 AXP system provides one interrupt request level to the CPU for Futurebus interrupts. The four Futurebus+ Interrupt Request registers feed into a single interrupt FIFO queue that is 16 entries deep.

Because this register is really the input to a 16-deep FIFO queue, the queue may be full when a module attempts to write a vector value to it. The I/O module responds with a busy status to the Futurebus+ master. The master must give up ownership of the bus and retry the cycle at a later time.

The Futurebus+ halt request register is used by Futurebus+ modules to cause a CPU halt interrupt. This is a write-only register that can be accessed only by using the 32-bit addressing mode of the Futurebus+. The register does not respond to A64 writes or any read cycle.

The writing of a 16-bit halt vector to this register by the Futurebus+ master causes a system event interrupt if the Halt Enable bit in the primary Control and Status register is set. Because this register may be full when a module attempts to write a halt vector value to it, the I/O module responds with a BUSY status to the Futurebus+ master. The master must give up ownership of the bus and retries the cycle at a later time.

## 15.3 Futurebus+ Interrupt Request Register (FIRQ)

Figure 15–4 Futurebus+ Interrupt Request Register (firq)

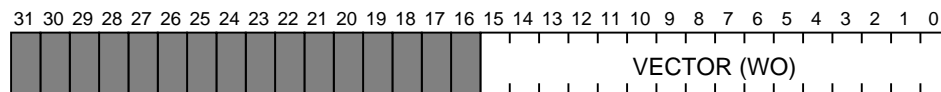
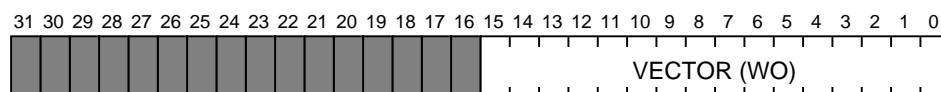


Table 15–2 Futurebus+ Interrupt Request Register Description

Field	Description
15:0	<b>VECTOR</b> <i>[write-only]</i> The sixteen bit data value to be used by the CPU as the interrupt vector.

## 15.4 Futurebus+ Halt Request Register (FHRQ)

Figure 15–5 Futurebus+ Halt Request Register (fhrq)



## 15.4 Futurebus+ Halt Request Register (FHRQ)

Table 15–3 Futurebus+ Halt Request Register Description

Field	Description
15:0	<b>VECTOR</b> <i>[write-only]</i> The sixteen bit data value to be used by the CPU as the halt vector.

## 15.5 Supported Data Transfer Protocols

### 15.5.1 Futurebus+ Slave

The I/O module responds to both 32-bit (D32) and 64-bit (D64) Futurebus+ transfers. The only supported transaction types are:

- Read
- Read Partial
- Write
- Write Partial
- Read Response

The CPU attempts to configure modules at system initialization to perform burst-data transfers in multiples of 32 bytes, which is the natural transfer size of the system bus. This avoids the need for the I/O module to perform a read block-merge data-write block operation which wastes system bandwidth. System performance degrades rapidly if excessive use of partial block transfers is made.

Data transfers should start on naturally aligned 32-byte boundaries. If the transfer is nonaligned, the I/O module uses the ED status line of the Futurebus+ cycle to end the transaction on a naturally aligned boundary. This should force the device to align further transfers in the same DMA operation.

The Futurebus+ device should declare the length of the transaction during the first data phase of the Futurebus+ cycle. This allows the I/O module to optimize the transaction. When the data length is declared, it implies that the transaction is aligned to the natural boundary of the transaction length.

### 15.5.2 Futurebus+ Master

The I/O module acts as a Futurebus+ master when performing operations through the mailbox mechanism. The cycles supported are:

- Read
- Read Partial
- Write
- Write Partial
- Read Response

The I/O module can perform both D64 and D32 transactions of the defined types. In all cases, the I/O module indicates a transfer size of 1, because there is only one data value specified in the mailbox mechanism.



## 15.6 Futurebus+ Error Handling

### 15.6 Futurebus+ Error Handling

As a Futurebus+ slave, the I/O module indicates any detected error conditions to the Futurebus+ master using the define transaction status indicators. It is the bus master's responsibility to either RE TRY the operation or inform the host CPU.

As a Futurebus+ master, the I/O module interrupts the host CPU on any error detected on a system bus transaction. Errors occurring on the Futurebus+ are reported to the CPU through the status field of the mailbox mechanism, the FERR1 and FERR2 registers, or both.

The local (Lbus) I/O bus (Lbus) is a 32-bit multiplexed, address and data bus that interconnects the local I/O devices. The local I/O devices are the SCSI controller chips, the EEPROM, and the serial line units.

### 16.1 Lbus Addressing

The Lbus address space is not a divided, flat address space as might be expected. Individual device type select bits are provided in the command field of the mailbox data structure, while the secondary bus address field is used to address an individual location within the selected device. This structure simplifies the logic implementation and reduces access latency.

With the exception of the SCSI controller, access to the SCSI script RAM, DMA activity between Lbus devices is not permitted. The SCSI controllers can perform the following operations:

- SCSI script RAM instruction fetches
- Main memory instruction fetches
- SCSI data transfers to or from SCSI script RAM
- SCSI data transfers to or from main memory
- SCSI script RAM to SCSI script RAM transfers
- Main memory to main memory transfers
- Main memory move transfers to or from SCSI Script RAM

Bit <31> of the 53C710 SCSI controller's DMA address is used to select the type of DMA activity to be performed. When bit <31> is 0, the 53C710 DMA cycle is directed to main memory, when bit <31> is a 1, the DMA cycle is directed to the SCSI controller private bus. Bit 30 of the address is used to select between a controller's own CSR space and the SCSI script RAM.

During a 53C710 DMA cycle to the SCSI script RAM (Bit 30 = 0), address bits <16:2> select the longword within the RAM. Address bits <30:17> have no effect during the cycle. During a SCSI controller DMA to its own register space (Bit 30 = 1), bits <5:2> select the register within the controller. Bits <29:6> have no effect. DMA accesses to a controller's own register space are used to update controller registers directly from a command script. Scripts can be generic because the controller number is not included in the CSR address decoding.

## 16.1 Lbus Addressing

Table 16–1 SCSI Controller Address Decoding

A[31]	A[30]	Source or Destination
0	x	System memory
1	0	Script RAM
1	1	Controller's CSRs

In the following descriptions of the local devices, a 32-bit register format is used to describe the individual device registers. Many of the devices use a byte-wide interface. Register data values for these devices are provided on bits <7:0> of the Write Data and Read Data fields of the mailbox data structure. Unused bits are shown in gray. The hardware does not check the values in the unused bit locations during write operations and it does not guarantee that the data in unused locations is of any predictable value during read operations.

## 16.2 EEPROM

The I/O module provides 8 kbytes of EEPROM for use by the console firmware as nonvolatile storage. Bit <4> in the Lbus mailbox data structure command field selects the EEPROM device for access. Bits <14:2> of the secondary bus address field select the byte within the device. Only single byte write and read operations are supported for this device. Bits <7:0> of the mailbox data structure write data and read data fields are used for write and read operations respectively. The mask field of the mailbox data structure has no effect.

## 16.3 Serial Line Ports

The I/O module contains two serial line ports: the console serial line port and the auxiliary serial line port. Both serial lines operate in asynchronous mode only. The console serial line does not provide modem control and uses a 6-pin DECconnect MMJ connector mounted on the I/O module handle. This line is compatible with the RS-232-C,D,E, EIA 423, CCITT V.28 and V.10 standards. The auxiliary serial line provides modem control and uses a 25-pin DIN connector mounted on the I/O module handle. This line is compatible with RS-232-C,D,E, EIA 423, CCITT V.28 and V.24 standards, as well as BELL 101, 103 and 112 modems.

Both serial lines are implemented using a single 85C30 device. Bit <5> in the Lbus mailbox data structure command field selects the 85C30 device for access, while bits <3:2> of the secondary bus address field select the register within the device. Only single-byte write and read operations are supported for this device. Bits <7:0> of the mailbox data structure write data and read data fields are used for write and read operations respectively. The mask field of the mailbox data structure has no effect.

Figure 16–1 shows the mapping of the mailbox data structure secondary bus address field to the 85C30 registers. The 85C30 provides two registers for each serial line unit.

Figure 16–1 Serial Line Unit Register Map

31	0	address
reserved	AUX_CTRL	0000 0000
reserved	AUX_DATA	0000 0004
reserved	CON_CTRL	0000 0008
reserved	CON_DATA	0000 000C

The 85C30 contains 10 internal read registers (RR0 - RR3, RR8, RR10 - RR13, RR15) and 16 internal write registers (WR0 - WR15) for each channel. The 85C30 contains only one WR2 register and WR9 register, but they can be accessed by either channel.

With the exception of RR0, WR0, RR8, and WR8 internal registers are accessed by a two-step process using a register pointer to perform the addressing.

1. The pointer bits must be set by writing to the AUX\_CTRL or CON\_CTRL registers which correspond to WR0 for each channel when the pointer bits are 0. The pointer bits may be written in either channel because only one set exists in the 85C30.
2. After the pointer bits are set, the next read or write cycle to the AUX\_CTRL or CON\_CTRL accesses the desired internal register. At the end of this read or write, the pointer bits are reset to zero so the next control write is to the register pointer. If for some reason the state of the pointer bits is unknown, they may be reset to 0 by performing a read to the AUX\_CTRL or CON\_CTRL registers.

Because the pointer bits are reset to zero unless explicitly set otherwise, WR0 and RR0 for each channel may be accessed in a single cycle. That is, it is not necessary to write the pointer bits with 0 before accessing WR0 and RR0 for either channel.

There are three pointer bits in WR0 that allow access to the registers with addresses 0 through 7. To access the registers with addresses 8 through 15, a special command must accompany the pointer bits. The command may be written to WR0 at the same time that the pointer bits are written.

There are three ways to read the receive data buffer (RR8) or write to the transmit data buffer (WR8) for each channel:

1. The two-step process
2. By accessing the 85C30 using the AUX\_DATA register for channel B or CON\_DATA register for channel A.
3. A read or write of the AUX\_DATA register or CON\_DATA register accesses the data registers directly, and independently, of the state of the pointer bits. This allows single-cycle access to the data registers and does not disturb the pointer bits.

## 16.4 AUX\_CTRL Register (ACR)

## 16.4 AUX\_CTRL Register (ACR)

Figure 16–2 AUX\_CTRL Register (acr)

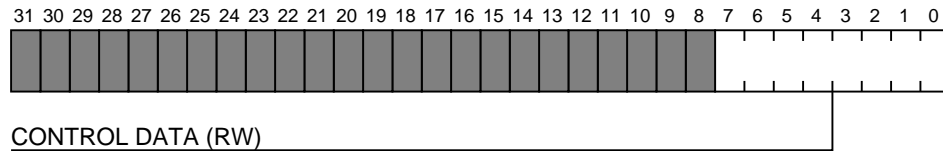


Table 16–2 AUX\_CTRL Register Description

Field	Description
7:0	<b>CONTROL DATA</b> <i>[read/write]</i> This 8-bit data field accesses the WR0 or RR0 registers of the auxiliary serial line channel when the register pointers are reset to 0. The data format of the WR0 register contains a register pointer that can be written to select one of the auxiliary serial line internal registers. The next read or write of the AUX_CTRL accesses the desired internal register. Any read or write of the AUX_CTRL register when the register pointer is not 0, resets the register pointer to 0.

## 16.5 AUX\_Data Register (AD)

Figure 16–3 AUX\_Data Register (ad)

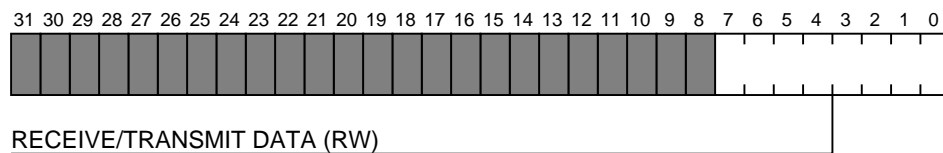


Table 16–3 AUX\_Data Register Description

Field	Description
7:0	<b>RECEIVE/TRANSMIT DATA</b> <i>[read/write]</i> This register provides direct access to the receive data register (RR8) and transmit data register (WR8) for the auxiliary serial line unit. This corresponds to channel B in the 85C30.

## 16.6 Console Control Register (CON\_CTRL)

Figure 16–4 Console Control Register (CCTRL)

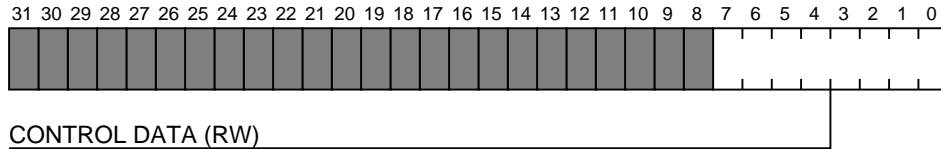


Table 16–4 Console Control Register Description

Field	Description
7:0	<p><b>CONTROL DATA</b> <i>[read/write]</i></p> <p>This 8-bit data field accesses the WR0 or RR0 registers of the console serial line channel when the register pointers are reset to 0. The data format of the WR0 register contains a register pointer that can be written to select one of the console serial line internal registers. The next read or write of the CON_CTRL accesses the desired internal register. Any read or write of the CON_CTRL register when the register pointer is nonzero resets the register pointer to 0.</p>

## 16.7 Console Data Register (CD)

Figure 16–5 Console Data Register (CD)

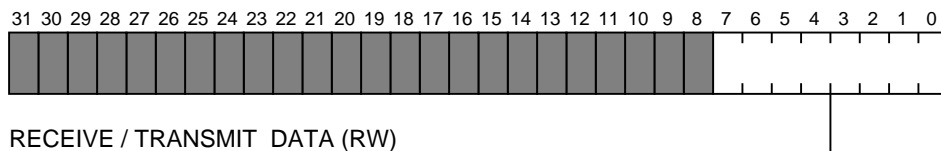


Table 16–5 Console Data Register Description

Field	Description
7:0	<p><b>RECEIVE / TRANSMIT DATA</b> <i>[read/write]</i></p> <p>This register provides direct access to the receive data register (RR8) and transmit data register (WR8) for the console serial line unit. This corresponds to channel A in the 85C30.</p>

Because the 85C30 does not provide enough modem control signals within an individual channel, the modem control signals for the auxiliary serial line are split between both channels. The following table is a guide to using the modem control signals provided for the auxiliary serial line:

## 16.7 Console Data Register (CD)

**Table 16–6 Auxiliary Serial Line Modem Control**

Signal	Digital Abbreviation	CCITT V.24 Number	Channel	Register	Bit
Data Signaling Rate Selector	SPDMI	112	A (console)	RR0	5
Calling Indicator	RI	125	A	RR0	3
Data Channel Received Line Signal Detector	CD	109	B (auxiliary)	RR0	3
Data Set Ready	DSR	107	A	RR0	4
Clear to Send	CTS	106	B	RR0	5
Data Signaling Rate Detector (DTE)	DSRS	111	B	WR5	1
Data Terminal Ready	DTR	108/2	A	WR5	7
Request To Send	RTS	105	A	WR5	1

## 16.8 Time-of-Year Clock (TOY)

The Time-of-Year (TOY) clock is implemented on the DEC 4000 AXP I/O module using the Dallas Semiconductor DS1287 real time clock device. This device provides the battery backed up TOY clock, 50 bytes of nonvolatile RAM, and a programmable square wave output. The square wave output is used to drive the system bus timer interrupt (CINT\_TIM) signal.

The interrupt output of the DS1287 is not connected to any processor interrupt. Therefore the timers and alarms, if desired, are only available using polled software routines.

Bit 8 in the Lbus mailbox data structure command field selects the DS1287 device for access. Bits <7:2> of the Secondary Bus Address field select the register within the device. Only single byte write and read operations are supported for this device. Bits <7:0> of the mailbox data structure write data and read data fields are used for write and read operations respectively. The mask field of the mailbox data structure has no effect. The following sections describe the TOY clock registers:

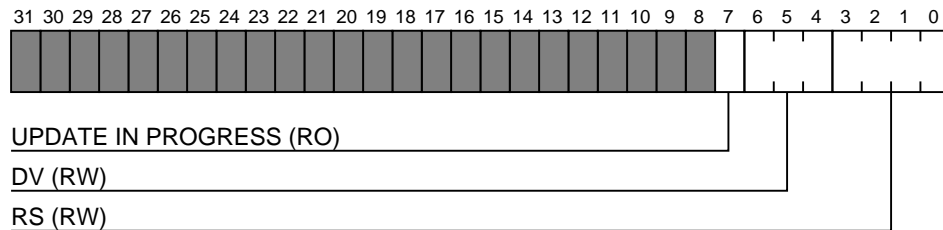
## 16.8 Time-of-Year Clock (TOY)

**Figure 16–6 TOY Clock Register Map**

31	0	address
reserved	seconds	0000 0000
reserved	sec alarm	0000 0004
reserved	minutes	0000 0008
reserved	min alarm	0000 000C
reserved	hours	0000 0010
reserved	hrs alarm	0000 0014
reserved	day of wk	0000 0018
reserved	day of mon	0000 001C
reserved	month	0000 0020
reserved	year	0000 0024
reserved	reg A	0000 0028
reserved	reg B	0000 002C
reserved	reg C	0000 0030
reserved	reg D	0000 0034
reserved	TOY RAM[0]	0000 0038
reserved	TOY RAM[49]	0000 00FC

## 16.9 TOY Register A (TOYA)

**Figure 16–7 TOY Register A (toyA)**





## 16.9 TOY Register A (TOYA)

**Table 16–7 TOY Register A Description**

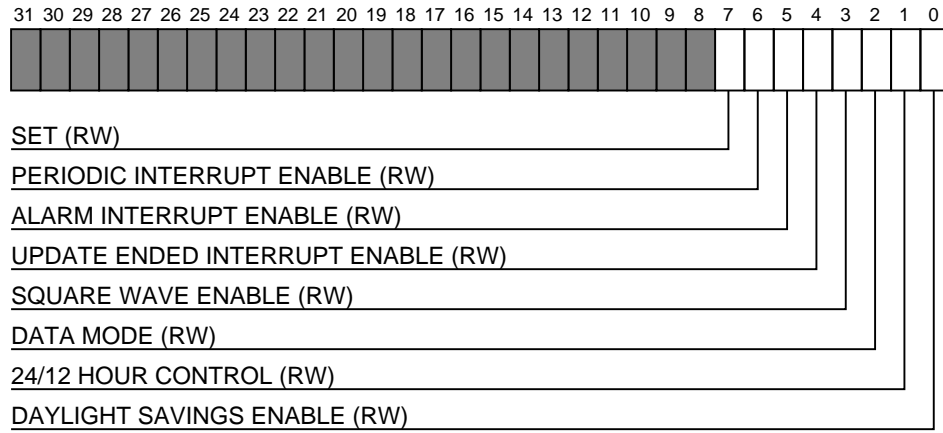
Field	Description
7	<p><b>UPDATE IN PROGRESS</b> <i>[read-only]</i></p> <p>The update in progress bit is a status flag that can be monitored. When the UIP bit is a one, the update transfer occurs. When UIP is 0, the update transfer does not occur for at least 244 <math>\mu</math>s. The time, calendar, and alarm information is fully available for access when the UIP bit is zero. The UIP bit is read only and is not affected by an Lbus reset. Writing the SET bit (in TOY Register B) to a 1 inhibits any update transfer and clears the UIP status bit.</p>
6:4	<p><b>DV</b> <i>[read/write]</i></p> <p>These three bits are used to turn the oscillator on or off and to reset the countdown chain. A pattern of 010 is the only combination of bits that turns the oscillator on and allows the TOY clock to keep time. A pattern of 11X enables the oscillator, but holds the countdown chain in reset. The next update occurs 500 ms after a pattern of 010 is written to this field.</p>
3:0	<p><b>RS</b> <i>[read/write]</i></p> <p>These four rate-selection bits select one of the 13 taps on the 15-stage divider or disable the divider output. The tap selected may be used to generate an output square wave and/or periodic interrupt. The user may do one of the following:</p> <ol style="list-style-type: none"> <li>1. Enable the interrupt with the PIE bit</li> <li>2. Enable the square wave output with the SQWE bit</li> <li>3. Enable both at the same time and the same rate</li> <li>4. Enable neither</li> </ol> <p>The available square wave frequencies are shown in Table 16–8.</p>

**Table 16–8 TOY Clock Square Wave Frequencies**

RS bits <3:0>	Frequency (Hz)	RS bits <3:0>	Frequency (Hz)	RS bits <3:0>	Frequency (Hz)
0	off	5	2048	10	64
1	256	6	1024	11	32
2	128	7	512	12	16
3	8192	8	256	13	8
4	4096	9	128	14	4
				15	2

## 16.10 TOY Register B (TOYB)

Figure 16–8 TOY Register B (toyB)



## 16.10 TOY Register B (TOYB)

Table 16–9 TOY Register B Description

Field	Description
7	<b>SET</b> <i>[read/write]</i> When the SET bit is a 0, the update transfer works normally by advancing the counts once per second. When a 1 is written to the SET bit, any update transfer is inhibited, and the program may initialize the time and calendar bytes without an update occurring. Read cycles can be executed in a similar manner. SET is a [read/write] bit which is not modified by an Lbus reset or internal functions of the TOY clock.
6	<b>PERIODIC INTERRUPT ENABLE</b> <i>[read/write]</i> The periodic interrupt enable (PIE) bit is a [read/write] bit that allows the periodic interrupt flag (PF) bit in TOY Register C to cause the device IRQ pin to be asserted. The device interrupt is not used in this application.
5	<b>ALARM INTERRUPT ENABLE</b> <i>[read/write]</i> The alarm interrupt enable AIE bit is a [read/write] bit that allows the alarm flag (AF) bit in TOY Register C to cause the device IRQ pin to be asserted. The device interrupt is not used in this application.
4	<b>UPDATE ENDED INTERRUPT ENABLE</b> <i>[read/write]</i> The updated ended interrupt enable UIE bit is a [read/write] bit that allows the Update End Flag (UF) bit in TOY Register C to cause the device IRQ pin to be asserted. The device interrupt is not used in this application.
3	<b>SQUARE WAVE ENABLE</b> <i>[read/write]</i> When the Square Wave Enable (SQWE) bit is set to a one, a square wave signal at the frequency set by the rate-selection bits RS[3:0] is driven out on the C-bus CINT_TIM signal. When the SQWE bit is zero, the CINT_TIM signal is held deasserted; the state of SQWE is cleared by an Lbus reset.
2	<b>DATA MODE</b> <i>[read/write]</i> The Data Mode (DM) bit indicates whether time and calendar information are in binary or BCD format. The DM bit is set by software to the appropriate format and can be read as required. This bit is not modified by internal functions of the TOY clock or by an Lbus reset. A 1 in this field signifies binary data while a zero in DM specifies BCD data.
1	<b>24/12 HOUR CONTROL</b> <i>[read/write]</i> The 24/12 control bit establishes the format of the hours byte. A 1 indicates the 24-hour mode and a 0 indicates the 12-hour mode. This bit is not affected by internal TOY clock functions or Lbus reset.
0	<b>DAYLIGHT SAVINGS ENABLE</b> <i>[read/write]</i> The Daylight Savings Enable (DSE) bit is a [read/write] bit which enables two special updates when DSE is set to one. On the first Sunday in April the time increments from 1:59:59 AM to 3:00:00 AM, on the last Sunday in October when the time first reaches 1:59:59 AM it changes to 1:00:00 AM. These special updates do not occur when the DSE bit is zero. This bit is not affected by internal functions or Lbus reset.

## 16.11 TOY Register C (TOYC)

Figure 16–9 TOY Register C (toyC)

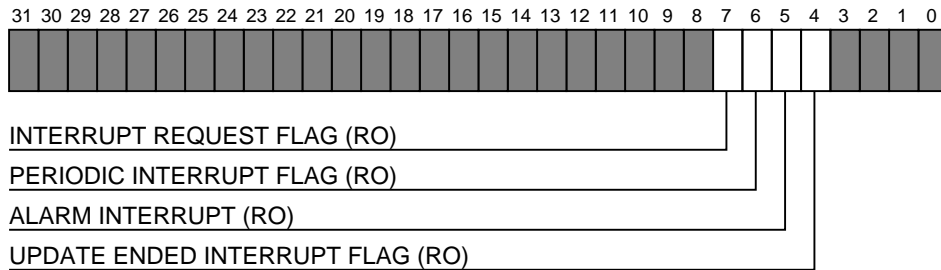


Table 16–10 TOY Register C Description

Field	Description
<b>7</b>	<p><b>INTERRUPT REQUEST FLAG</b> <i>[read-only]</i></p> <p>The Interrupt Request Flag (IRQF) bit is set to a 1 when one or more of the following are true:</p> <ul style="list-style-type: none"> <li>• PF = PIE = 1</li> <li>• AF = AIE = 1</li> <li>• UF = UIE = 1</li> <li>• <math>IRQF = PF * PIE + AF * AIE + UF * UIE</math></li> </ul> <p>All flag bits are cleared after TOY Register C is read or after an Lbus reset.</p>
<b>6</b>	<p><b>PERIODIC INTERRUPT FLAG</b> <i>[read-only]</i></p> <p>The Periodic Interrupt Flag (PF) is a read-only bit which is set to a one when an edge is detected on the selected tap of the divider chain. The RS&lt;3:0&gt; bits establish the periodic rate. PF is set to a one independent of the state of the PIE bit. When both PF and PIE are ones, the IRQ signal is active and sets the IRQF bit. The PF bit is cleared by an Lbus reset or a read of TOY Register C.</p>
<b>5</b>	<p><b>ALARM INTERRUPT</b> <i>[read-only]</i></p> <p>A one in the Alarm Interrupt Flag (AF) bit indicates that the current time has matched the alarm time. If the AIE bit is also a one, the IRQF bit is asserted. The AF bit is cleared by an Lbus reset or a read of TOY Register C.</p>
<b>4</b>	<p><b>UPDATE ENDED INTERRUPT FLAG</b> <i>[read-only]</i></p> <p>The Update Ended Interrupt Flag (UF) bit is set after each update cycle. If the UIE bit is also a one, the IRQF bit is asserted. The UF bit is cleared by an Lbus reset or a read of TOY Register C.</p>

## 16.12 TOY Register D (TOYD)

## 16.12 TOY Register D (TOYD)

Figure 16–10 TOY Register D (TOYD)

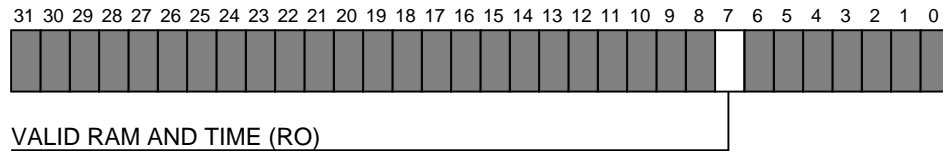


Table 16–11 TOY Register D Description

Field	Description
7	<b>VALID RAM AND TIME</b> <i>[read-only]</i> The Valid RAM and Time (VRT) bit is set to the one state by the device manufacturer prior to shipment. This bit is not writeable and should always be a one when read. If a zero is ever present, an exhausted Lithium energy source is indicated and both the contents of the TOY clock data and RAM data are questionable. This bit is unaffected by an Lbus reset.

---

## Ethernet Adapters (TGEC)

The DEC 4000 AXP I/O module uses the TGEC (DC253) as the interface to the Ethernet ports.

Thinwire and standard Ethernet connectors are available on the I/O module handle for each port. Only one connector can be used at a time for each port.

Bit <9> in the Lbus mailbox data structure command field selects the TGEC devices for access. Bit 7 of the secondary bus address selects one of the two TGEC devices while bits <6:2> select the register within the device. Bits <31:0> of the mailbox data structure write data and read data fields are used for write and read operations respectively. Registers within the TGEC are accessible only as longwords, the Mask field in the mailbox data structure has no effect during access to the TGEC.

The TGEC asserts the Nix\_IRQ bit (bits 6 or 7 depending upon the device) in the LINT register to indicate an interrupting condition. The assertion of any bits in the LINT register asserts the CIRQ\_L[0] system bus interrupt line. The TGEC interrupt remains asserted until the interrupting condition is cleared in the TGEC control/status registers.

The TGEC provides 16 control/status registers which may be accessed by the host. The address map for these registers is shown in Figure 17-1. Bit <7> of the Secondary Bus Address field in the mailbox data structure selects one of the two Ethernet ports. The base address of each interface is shown in Table 17-1.

**Table 17-1 Ethernet Interface Selection**

Addr [7]	Controller	Base Address
0	NI(0)	0000 0000 <sub>16</sub>
1	NI(1)	0000 0080 <sub>16</sub>

**Figure 17–1 TGEC Register Map**

NI(n)_CSR0	NI(n)+00H
NI(n)_CSR1	NI(n)+08H
NI(n)_CSR2	NI(n)+10H
NI(n)_CSR3	NI(n)+18H
NI(n)_CSR4	NI(n)+20H
NI(n)_CSR5	NI(n)+28H
NI(n)_CSR6	NI(n)+30H
NI(n)_CSR7	NI(n)+38H
NI(n)_CSR8	NI(n)+40H
NI(n)_CSR9	NI(n)+48H
NI(n)_CSR10	NI(n)+50H
NI(n)_CSR11	NI(n)+58H
NI(n)_CSR12	NI(n)+60H
NI(n)_CSR13	NI(n)+68H
NI(n)_CSR14	NI(n)+70H
NI(n)_CSR15	NI(n)+78H

## 17.1 TGEC Programming

The operation of the TGEC is controlled by a program in host memory called the port driver. The TGEC and the port driver communicate through two data structures: Command and Status Registers (CSRs) located in the TGEC and mapped in the host I/O address space, and through descriptor lists and data buffers, collectively called Host Communication Area , in host memory.

The CSRs are used for initialization, global pointers, commands and global errors reporting, while the host memory resident structures handle the actions and statuses related to buffer management.

### 17.1.1 Programming Overview

The TGEC can be viewed as two independent, concurrently executing processes: reception and transmission. After the TGEC completes its Initialization sequence, these two processes alternate between three states: STOPPED, RUNNING or SUSPENDED. State transitions occur as a result of port driver commands (writing to a CSR) or various external events occurrences. Some of the port driver commands require the referenced process to be in a specific state.

A simple programming sequence of the chip may be summarized as:

1. After power on (or reset), check the device type (SGEC or TGEC) by running the following commands:
  - Write '00000000' to the CSR is address  $20008018_{16}$  (it can be CSR3 if it is a TGEC device, or CSR6 if it is an SGEC device).

- Read from the same CSR.
  - If the data is '00000000' it is a TGEC device.
  - If the data is '01E0F000' it is a SGEC device.
- 2. Verifying the self test completed successfully.
- 3. Writing CSRs to set major parameters such as System Base Register, Interrupt Vector, Address Filtering mode and so on.
- 4. Creating the transmit and receive lists in memory and writing the CSRs to identify them to the TGEC.
- 5. Placing a setup frame in the transmit list, to load the internal reception address filtering table.
- 6. Starting the Reception and Transmission processes placing them in the RUNNING state.
- 7. Waiting for TGEC interrupts. CSR5 contains all the global interrupt status bits.
- 8. Remediating the suspension cause, if either of the Reception or Transmission processes enter the SUSPENDED state.
- 9. Issuing a Tx poll demand command, to return the transmission process to the RUNNING state. A Rx Poll Demand can be issued to remedy the reception process suspension cause, and to return the reception process to the RUNNING state.

If the Rx poll demand is not issued, the Reception process will return to the RUNNING state when the TGEC receives the next recognized incoming frame.

The following sections contain detailed programming and state transitions information.

### 17.1.2 TGEC Command and Status Registers

The TGEC contains 16 command and status registers which may be accessed by the host.

### 17.1.3 Host Access to CSRs

The TGEC's CSRs are located in I/O address space.

The CSRs must be quadword aligned and can be accessed only with longword instructions. The address of CSR $x$  is the base address plus  $8x$  bytes. For example, if the base address is 2000 8000, then the address of CSR2 is 2000 8010.

---

#### Note

---

Accessing to an odd-address register will cause unpredictable results.

---

In order to save chip real estate, yet not tie up the host bus for extended periods of time, the 16 CSRs are subdivided into two groups:

1. Physical CSRs (CSR0 through CSR7, and CSR15)
2. Virtual CSRs (CSR8 through CSR14)

The group the CSR is part of, determines the way the host will access it.



## 17.1 TGEC Programming

### 17.1.3.1 TGEC Physical CSRs

These registers are physically present in the chip. Host access to these CSRs is by a single instruction (for example, MOVL). There is no host perceivable delay and the instruction completes immediately. Most commonly used TGEC features are contained in the physical CSRs.

### 17.1.3.2 TGEC Virtual CSRs

These registers are not physically present in the TGEC and are incarnated by the on-chip processor. Accesses to TGEC functions implied by these registers may take up to 20  $\mu$ seconds. So as not to tie up the host bus, virtual CSR access requires several steps by the host.

CSR5<DN> is used to synchronize access to the virtual CSRs: after the first virtual CSR access, the TGEC deasserted CSR5<DN> until it will complete the action.

---

#### Note

---

Accessing the virtual CSRs, without polling first on the CSR5<DN> reassertion will cause unpredictable results.

---

**17.1.3.2.1 CSR Write** To write to a virtual CSR the host takes the following actions:

1. It issues a write CSR instruction. Instruction completes immediately, but the data is not yet copied by the TGEC.
2. It wait for CSR5<DN>. No TGEC virtual CSR may be accessed before CSR5<DN> asserts.

**17.1.3.2.2 CSR Read** To read a virtual CSR the host takes the following actions:

1. It issues a read CSR instruction. Instruction completes immediately, but no valid data is sent to the host.
2. It waits for CSR5<DN>. No TGEC virtual CSR may be accessed before CSR5<DN> asserts.
3. It reissues a read CSR instruction, to the same CSR as in step 1. The host receives valid data.

17.1.4 Vector Address, IPL, Sync/Asynch (CSR0)

Because the TGEC may generate an interrupt on parity errors during host writes to CSRs, this register must be the first one written by the host.

**CSR0 address is: 20008000<sub>16</sub>**

**Note**

A parity error during CSR0 host write may cause a host system crash due to an erroneous Interrupt Vector. To protect against such an eventuality, CSR0 must be written as follows while the IPL - to which the TGEC's IRQ\_L is assigned - is disabled:

1. Write CSR.
2. Read CSR.
3. Compare value read to value written. If values mismatch, repeat from step 1.
4. Read CSR5 and examine CSR5<ME> for pending parity interrupt. Should an interrupt be pending, write CSR5 to clear it.

Figure 17–2 Vector Address, IPL, Sync/Asynch (CSR0)

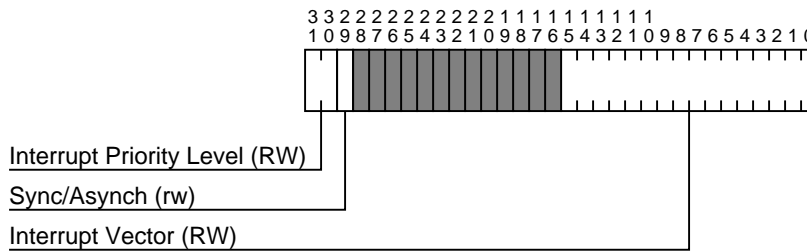


Table 17–2 Vector Address, IPL, Sync/Asynch Description

Field	Description
<b>31:30</b>	<b>INTERRUPT PRIORITY LEVEL</b> <i>[read/write]</i> This is the interrupt priority level that the TGEC will respond to when IRQ_L is asserted. • IP
<b>29</b>	<b>SYNC/ASYNCH</b> <i>[read/write]</i> This bit determines the TGEC operating mode when it is the bus master. When set, the TGEC will operate as a synchronous device and when clear, the TGEC will operate as an asynchronous device.
<b>15:00</b>	<b>INTERRUPT VECTOR</b> <i>[read/write]</i>

(continued on next page)

## 17.1 TGEC Programming

**Table 17–2 (Cont.) Vector Address, IPL, Sync/Asynch Description**

Field	Description
	During an Interrupt Acknowledge cycle for a TGEC interrupt on IRQ_L, this is the value that the TGEC will drive on the host bus CDAL<31:0> pins (CDAL pins <1:0> and <31:16> are set to “0”). Bits <1:0> are ignored when CSR0 is written, and set to “1” when read.

**Table 17–3 CSR0 access**

Value after <b>RESET</b> :	1FFF0003 <sub>16</sub>
<b>Read</b> access rules:	None
<b>Write</b> access rules:	The IPL to which the TGEC IRQ_L is assigned - must be DISABLED

17.1.5 Transmit Polling Demand (CSR1)

CSR1 address is: 20008008<sub>16</sub>

Figure 17–3 Transmit Polling Demand (CSR1)

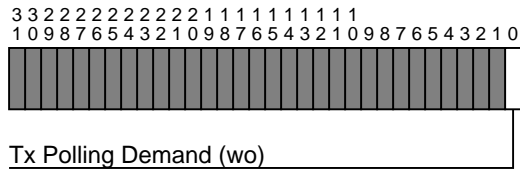


Table 17–4 Transmit Polling Demand Description

Field	Description
<b>0</b>	<b>TX POLLING DEMAND</b> <i>[write-only]</i> Checks the transmit list for frames to be transmitted. The PD value is meaningless.

Table 17–5 CSR1 Access

Value after <b>RESET</b> :	Not applicable
<b>Read</b> access rules:	None
<b>Write</b> access rules:	Tx process SUSPENDED

## 17.1 TGEC Programming

### 17.1.6 Receive Polling Demand (CSR2)

CSR2 address is: 20008010<sub>16</sub>

Figure 17–4 Receive Polling Demand (CSR2)

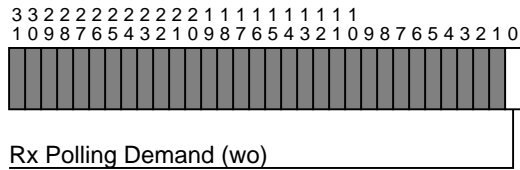


Table 17–6 Receive Polling Demand Description

Field	Description
<b>0</b>	<b>RX POLLING DEMAND</b> <i>[write-only]</i> Checks the receive list for receive descriptors to be acquired. The PD value is meaningless.

Table 17–7 CSR2 Access

Value after <b>RESET</b> :	Not applicable
<b>Read</b> access rules:	None
<b>Write</b> access rules:	Rx process SUSPENDED

### 17.1.7 Descriptor List Addresses (CSR3, CSR4)

The two descriptor list head address registers are identical in function, one being used for the transmit buffer descriptors and one being used for the receive buffer descriptors. In both cases, the registers are used to point the TGEC to the start of the appropriate buffer descriptor list.

The descriptor lists reside in VAX **physical** memory space and must be **longword** aligned.

---

**Note**

---

For best performance, it is recommended that the descriptor lists be OCTAWORD aligned.

---

**CSR3 address is: 20008018<sub>16</sub>**

**CSR4 address is: 20008020<sub>16</sub>**

---

**Note**

---

If the Transmit descriptor list is built as a ring (the chain descriptor points at the first descriptor of the list), the ring must contain, at least, TWO descriptors in addition to the chain descriptor.

---

Initially, these registers must be written before the respective Start command is given (see Section 17.1.9), else the respective process will remain in the STOPPED state. New descriptor list head addresses are only acceptable while the respective process is in the STOPPED or SUSPENDED states. Addresses written while the respective process is in the RUNNING state, are ignored and discarded.

If the host attempts to read any of these registers before ever writing to them, the TGEC responds with unpredictable values.

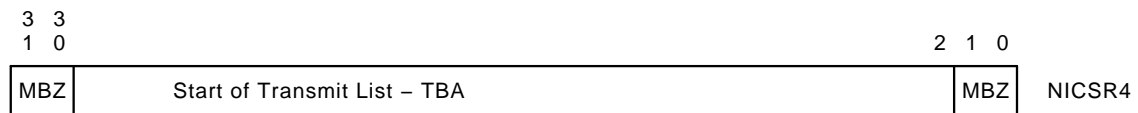
## 17.1 TGEC Programming

Figure 17–5 Descriptor List Addresses Format



I/O Address: 2000 800C  
(16)

Longword Read/Write Access



I/O Address: 2000 8010  
(16)

Longword Read/Write Access

ESB90P0055

Table 17–8 lists the descriptor list addresses.

Table 17–8 Descriptor List Addresses Bits

Bit	Name	Access	Description
29:00	RBA	R/W	Address of the start of the receive list. This is a 30-bit VAX physical address.
29:00	TBA	R/W	Address of the start of the transmit list. This is a 30-bit VAX physical address.

**Note**

The descriptor lists must be longword aligned.

Table 17–9 lists the CSR3 access modes.

Table 17–9 CSR3 Access

Value after <b>RESET</b> :	unpredictable
<b>Read</b> access rules:	None
<b>Write</b> access rules:	Rx process STOPPED or SUSPENDED

**Table 17–10 CSR4 access**

---

Value after <b>RESET</b> :	unpredictable
<b>Read</b> access rules:	None
<b>Write</b> access rules:	Tx process STOPPED or SUSPENDED

---

After either CSR3 or CSR4 are written, the new address is readable from the written CSR. However, if the TGEC status did not match the related write access rules, the new address does not take effect and the written information is (lost, EVEN if the TGEC later matches the right condition.



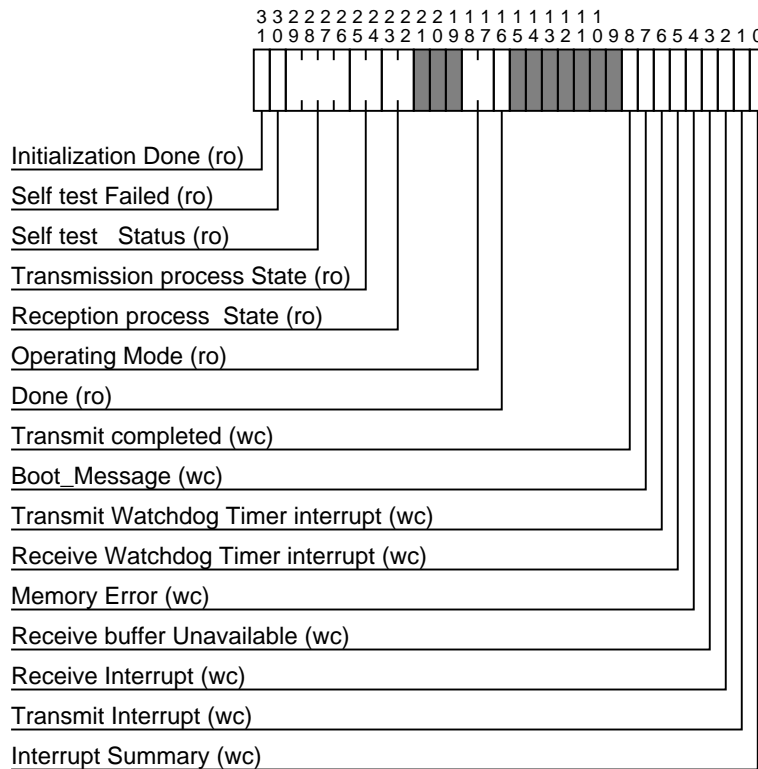
## 17.1 TGEC Programming

### 17.1.8 Status Register (CSR5)

This register contains all the status bits the TGEC reports to the host.

**CSR5 address is: 20008028<sub>16</sub>**

**Figure 17–6 Status Register 5 (CSR5)**



**Table 17–11 Status Register 5 Description**

Field	Description
<b>31</b>	<b>INITIALIZATION DONE</b> <i>[read-only]</i> When set, indicates the TGEC has completed the Initialization (reset and self test) sequences, and is ready for further commands. When clear, indicates the TGEC is performing the Initialization sequence and ignores all commands. After the initialization sequence completes, the transmission and reception processes are in the STOPPED state.
<b>30</b>	<b>SELF TEST FAILED</b> <i>[read-only]</i> When set, indicates the TGEC self test has failed. The self test completion code bits indicate the failure type.
<b>29:26</b>	<b>SELF TEST STATUS</b> <i>[read-only]</i>

(continued on next page)

**Table 17–11 (Cont.) Status Register 5 Description**

Field	Description														
	The self test completion code according to the following table. Only valid if SF is set.														
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0001</td> <td>ROM error</td> </tr> <tr> <td>0010</td> <td>RAM error</td> </tr> <tr> <td>0011</td> <td>Address filter RAM error</td> </tr> <tr> <td>0100</td> <td>Transmit FIFO error</td> </tr> <tr> <td>0101</td> <td>Receive FIFO error</td> </tr> <tr> <td>0110</td> <td>Self_test loopback error</td> </tr> </tbody> </table>	Value	Meaning	0001	ROM error	0010	RAM error	0011	Address filter RAM error	0100	Transmit FIFO error	0101	Receive FIFO error	0110	Self_test loopback error
Value	Meaning														
0001	ROM error														
0010	RAM error														
0011	Address filter RAM error														
0100	Transmit FIFO error														
0101	Receive FIFO error														
0110	Self_test loopback error														
	Self test takes 25ms to complete after Hardware or Software RESET.														
<b>25:24</b>	<b>TRANSMISSION PROCESS STATE</b> <i>[read-only]</i>														
	Indicates the current state of the Transmission process, as follows:														
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>STOPPED</td> </tr> <tr> <td>01</td> <td>RUNNING</td> </tr> <tr> <td>10</td> <td>SUSPENDED</td> </tr> </tbody> </table>	Value	Meaning	00	STOPPED	01	RUNNING	10	SUSPENDED						
Value	Meaning														
00	STOPPED														
01	RUNNING														
10	SUSPENDED														
<b>23:22</b>	<b>RECEPTION PROCESS STATE</b> <i>[read-only]</i>														
	This field indicates the current state of the Reception process, as follows:														
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>STOPPED</td> </tr> <tr> <td>01</td> <td>RUNNING</td> </tr> <tr> <td>10</td> <td>SUSPENDED</td> </tr> </tbody> </table>	Value	Meaning	00	STOPPED	01	RUNNING	10	SUSPENDED						
Value	Meaning														
00	STOPPED														
01	RUNNING														
10	SUSPENDED														
<b>18:17</b>	<b>OPERATING MODE</b> <i>[read-only]</i>														
	These bits indicate the current TGEC operating mode as in the following table:														
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>Normal operating mode.</td> </tr> <tr> <td>01</td> <td>Internal Loopback - Indicates the TGEC is disengaged from the Ethernet wire. Frames from the transmit list are looped back to the receive list, subject to address filtering. Section 17.5.2 explains this mode of operation.</td> </tr> <tr> <td>10</td> <td>External Loopback - Indicates the TGEC is working in full duplex mode. Frames from the transmit list are transmitted on the Ethernet wire and also looped back to the receive list, subject to address filtering.</td> </tr> <tr> <td>11</td> <td>Reserved for Diagnostics.</td> </tr> </tbody> </table>	Value	Meaning	00	Normal operating mode.	01	Internal Loopback - Indicates the TGEC is disengaged from the Ethernet wire. Frames from the transmit list are looped back to the receive list, subject to address filtering. Section 17.5.2 explains this mode of operation.	10	External Loopback - Indicates the TGEC is working in full duplex mode. Frames from the transmit list are transmitted on the Ethernet wire and also looped back to the receive list, subject to address filtering.	11	Reserved for Diagnostics.				
Value	Meaning														
00	Normal operating mode.														
01	Internal Loopback - Indicates the TGEC is disengaged from the Ethernet wire. Frames from the transmit list are looped back to the receive list, subject to address filtering. Section 17.5.2 explains this mode of operation.														
10	External Loopback - Indicates the TGEC is working in full duplex mode. Frames from the transmit list are transmitted on the Ethernet wire and also looped back to the receive list, subject to address filtering.														
11	Reserved for Diagnostics.														
<b>16</b>	<b>DONE</b> <i>[read-only]</i>														
	When set, indicates the TGEC has completed a requested virtual CSR access. After a reset, this bit is set.														
<b>8</b>	<b>TRANSMIT COMPLETED</b> <i>[read/write 1 to clear]</i>														
	This bit is set together with CSR5<TI> when the transmit process enters either the SUSPENDED or STOPPED state at the completion of the current frame transmission.														
<b>7</b>	<b>BOOT_MESSAGE</b> <i>[read/write 1 to clear]</i>														

(continued on next page)

## 17.1 TGEC Programming

**Table 17–11 (Cont.) Status Register 5 Description**

Field	Description
	When set, indicates that the TGEC has detected a boot_message on the serial line and has set the external pin BOOT_L.
<b>6</b>	<b>TRANSMIT WATCHDOG TIMER INTERRUPT</b> <i>[read/write 1 to clear]</i> When set, indicates the transmit watchdog timer has timed out, indicating the TGEC transmitter was babbling. The Transmission process is <i>aborted</i> and placed in the STOPPED state. (Also reported into the Tx descriptor status TDES0<TO> flag).
<b>5</b>	<b>RECEIVE WATCHDOG TIMER INTERRUPT</b> <i>[read/write 1 to clear]</i> When set, indicates the Receive Watchdog Timer has timed out, indicating that some other node is babbling on the network. Current frame reception is aborted and RDES0<LE> and RDES0<LS> will be set. Bit CSR5<RI> will also set. The Reception process remains in the RUNNING state.
<b>4</b>	<b>MEMORY ERROR</b> <i>[read/write 1 to clear]</i> Is set when any of the following occur: <ul style="list-style-type: none"> <li>• TGEC is the CP-BUS Master and the ERR_L pin is asserted by external logic (generally indicative of a memory problem).</li> <li>• Parity error detected on a host to TGEC CSR write or TGEC read from memory.</li> </ul> <p>When a memory error is set, the reception and transmission processes are aborted and placed in the STOPPED state.</p>
<b>Note</b>	
At this point, it is mandatory that the port driver issue a reset command and rewrite all CSRs.	
<b>3</b>	<b>RECEIVE BUFFER UNAVAILABLE</b> <i>[read/write 1 to clear]</i> When set, indicates that the next descriptor on the receive list is owned by the host and could not be acquired by the TGEC. The Reception process is placed in the SUSPENDED state. Section 17.5 explains the Reception process state transitions. Once set by the TGEC, this bit will not be set again until the TGEC encounters a descriptor it can not acquire. To resume processing receive descriptors, the host must flip the ownership bit of the descriptor and can issue the Rx Poll Demand command. If no Rx poll demand is issued, the Reception process resumes when the next recognized incoming frame will is received.
<b>2</b>	<b>RECEIVE INTERRUPT</b> <i>[read/write 1 to clear]</i> When set, indicates that a frame has been placed on the receive list. Frame specific status information was posted in the descriptor. The Reception process remains in the RUNNING state.
<b>1</b>	<b>TRANSMIT INTERRUPT</b> <i>[read/write 1 to clear]</i>

(continued on next page)

**Table 17–11 (Cont.) Status Register 5 Description**

Field	Description
	When set, indicates one of the following: <ul style="list-style-type: none"> <li>• Either all the frames in the transmit list have been transmitted (next descriptor owned by the host), or a frame transmission was aborted due to a locally induced error. The port driver must scan down the list of descriptors to determine the exact cause. The Transmission process is placed in the SUSPENDED state. Section 17.5.1 explains the Transmission process state transitions. To resume processing transmit descriptors, the port driver must issue the Poll Demand command.</li> <li>• A frame transmission completed, and TDES1&lt;IC&gt; was set. The Transmission process remains in the RUNNING state, unless the next descriptor is owned by the host or the frame transmission aborted due to an error. In the latter cases, the Transmission process is placed in the SUSPENDED state.</li> </ul>
<b>0</b>	<b>INTERRUPT SUMMARY</b> <i>[read/write 1 to clear]</i> The logical “OR” of CSR5 bits 1 through 6.

Table 17–12 shows the bit access modes for CSR5.

**Table 17–12 CSR5 Access**

Value after <b>RESET</b> :	0039FE00 <sub>16</sub>
<b>Read</b> access rules:	None
<b>Write</b> access rules:	CSR5<07:01> bits cleared by 1, others bits not writeable

### 17.1.8.1 CSR5 Status Report

The Status register CSR5 is split into two words:

The high word which contains the global status of the TGEC, as the initialization status, the DMA and operation mode and the Receive and Transmit process states.

The low word which contains the status related to the Receive and Transmit frames.

Any change of the CSR5 bits <ID>, <SF>, <OM> or <DN> - which is always the result of a host command - is reported without an interrupt.

Any process state change initiated by a host command CSR6<ST> or CSR6<SR>, is reported without an interrupt.

In the above two cases, the driver must poll on CSR5 to get the acknowledge of its command (i.e. polling on <ID, SF> after Reset or polling on <TS> after a START\_TX command).

Any process state change initiated by the TGEC activity is immediately followed by at least one of the CSR5<6:1> interrupts and the interrupt\_summary CSR5<IS>.

## 17.1 TGEC Programming

The TGEC 16-bit internal processor updates the 32-bit CSR5 register in two phases: the high word is modified first, then the low word is written, generating an interrupt to the host. In this case, the driver must scan first the CSR5 low word to get the interrupt status, then the CSR high\_word to get the related process state. (that is, <TC, TI> interrupt with <TS> = SUSPENDED reports an end of transmission due to a Tx descriptor unavailable.)

If the host polls on the process state change, it may detect a change without interrupt, due to the small time window separating the CSR5 high\_word and low\_word updates.

Maximum time window is  $4 \cdot T$  cycles of the host clock.

17.1.9 Command and Mode Register (CSR6)

This register is used to establish operating modes and for port driver commands.

CSR6 address is: 20008030<sub>16</sub>

Figure 17–7 Command and Mode Register (CSR6)

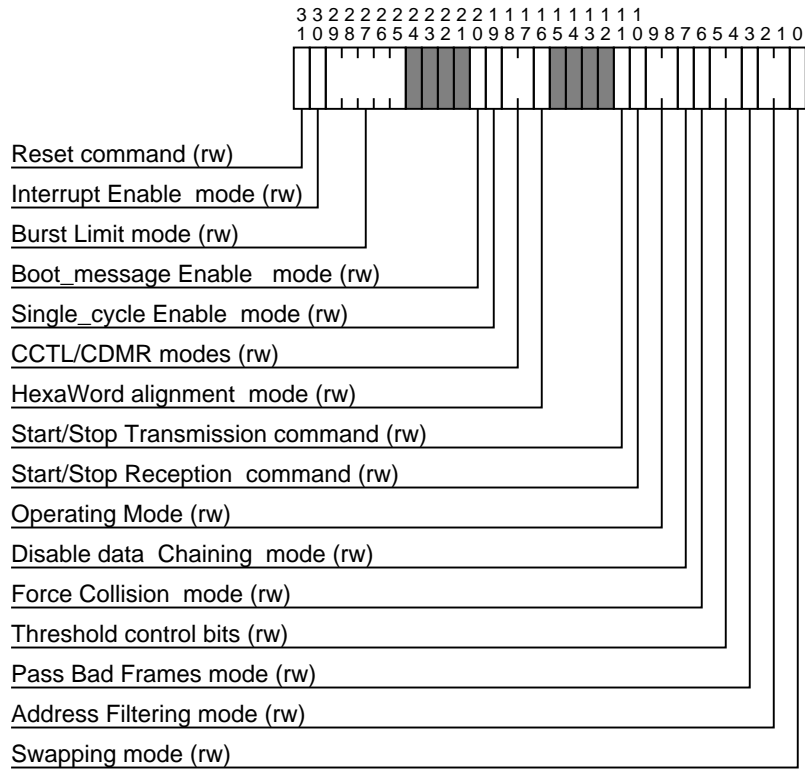


Table 17–13 Command and Mode Register Description

Field	Description
31	<b>RESET COMMAND</b> <i>[read/write]</i> Upon being set, the TGEC will abort all processes and start the reset sequence. After completing the reset and self test sequence, the TGEC will set bit CSR5<ID>. Clearing this bit has no effect. The CSR6<RE> value is unpredictable on read after HARDWARE reset.
30	<b>INTERRUPT ENABLE MODE</b> <i>[read/write]</i> When set, setting of CSR5 bits 1 through 6 will cause an interrupt to be generated.
29:25	<b>BURST LIMIT MODE</b> <i>[read/write]</i>

(continued on next page)

## 17.1 TGEC Programming

**Table 17–13 (Cont.) Command and Mode Register Description**

Field	Description										
	<p>Specifies the maximum number of longwords to be transferred in a single DMA burst on the host bus.</p> <p>When CSR6&lt;SE&gt; is cleared, permissible values are 1,2,4,8 and 16. When SE is set, the only permissible values are 1 and 4: a value of 2 and 8 or 16 is respectively forced to 1 and 4.</p>										
<b>20</b>	<p><b>BOOT_MESSAGE ENABLE MODE</b> <i>[read/write]</i></p> <p>When set, enables the boot_message recognition. When the TGEC recognizes an incoming boot message on the serial line, CSR5&lt;BO&gt; is set and the external pin BOOT_L is asserted for a duration of 6*Tcycles (of the host clock).</p> <p>Takes effect only if CSRs 11,12,13 have been initialized and a setup frame has been proceeded.</p> <p>The boot message recognition is disabled regardless of the &lt;BE&gt; bit, when either the PROMISCUOUS filtering or the INVERSE filtering mode is enabled (CSR6&lt;AF&gt;=01 or SDES1&lt;IF&gt;=1).</p>										
<b>19</b>	<p><b>SINGLE_CYCLE ENABLE MODE</b> <i>[read/write]</i></p> <p>When set, the TGEC transfers only a single longword or an octaword in a single DMA burst on the host bus.</p>										
<b>18:17</b>	<p><b>CCTL/CDMR MODES</b> <i>[read/write]</i></p> <p>These bits determine the functionality of CCTL_L/CDMR_L pin:</p> <table border="1" style="margin-left: 40px;"> <thead> <tr> <th style="text-align: left;">Value</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>Used as CCTL pin</td> </tr> <tr> <td>01</td> <td>Used as CDMR pin for the control transfers</td> </tr> <tr> <td>10</td> <td>Used as CDMR pin for RX/TX transfers</td> </tr> <tr> <td>11</td> <td>Used as CDMR pin for control and RX/TX transfers</td> </tr> </tbody> </table>	Value	Meaning	00	Used as CCTL pin	01	Used as CDMR pin for the control transfers	10	Used as CDMR pin for RX/TX transfers	11	Used as CDMR pin for control and RX/TX transfers
Value	Meaning										
00	Used as CCTL pin										
01	Used as CDMR pin for the control transfers										
10	Used as CDMR pin for RX/TX transfers										
11	Used as CDMR pin for control and RX/TX transfers										
<b>16</b>	<p><b>HEXAWORD ALIGNMENT MODE</b> <i>[read/write]</i></p> <p>When set the receive and transmit DMA channels perform a hexaword alignment: if the buffer is not hexaword aligned, the DMA executes a first transfer up to the next hexaword boundary before starting the second transfer according to the selected burst limit. Meaningless if Single_cycle is enabled CSR6&lt;SE&gt; = 1.</p>										
<hr/> <p><b>Note</b></p> <p>For Burst_limit = 8 , the Receive and Transmit DMA transfers never cross a hexaword boundary within a burst.</p> <hr/>											
<b>11</b>	<p><b>START/STOP TRANSMISSION COMMAND</b> <i>[read/write]</i></p>										

(continued on next page)

Table 17–13 (Cont.) Command and Mode Register Description

Field	Description
	<p>When set, the transmission process is placed the RUNNING state. The TGEC checks the transmit list at the current position for a frame to transmit. The address set by CSR4 or the position retained when the Tx process was previously stopped. If it does not find a frame to transmit, the transmission process enters the SUSPENDED state. The start transmission command is honored only when the transmission process is in the STOPPED state. The first time this command is issued, an additional requirement is that CSR4 has already been written to, else the transmission process will remain in the STOPPED state.</p> <p>When cleared, the transmission process is placed in the STOPPED state after completing transmission of the current frame. The next descriptor position in the transmit list is saved, and becomes the current position after transmission is restarted.</p> <p>The stop transmission command is honored only when the transmission process is in the RUNNING or SUSPENDED states.</p>
<b>10</b>	<p><b>START/STOP RECEPTION COMMAND</b> <i>[read/write]</i></p> <p>When set, the Reception process is placed in the RUNNING state, the TGEC attempts to acquire a descriptor from the receive list and process incoming frames. Descriptor acquisition is attempted from the <i>current</i> position in the list. The address set by CSR3 or the position retained when the Rx process was previously stopped. If no descriptor can be acquired, the Reception process enters the SUSPENDED state.</p> <p>The start reception command is honored only when the reception process is in the STOPPED state. The first time this command is issued, CSR3 must already have been written to, or else the Reception process will remain in the STOPPED state.</p> <p>When cleared, the reception process is placed in the STOPPED state, after completing reception of the current frame. The next descriptor position in the receive list is saved, and becomes the current position after reception is restarted. The stop reception command is honored only when the reception process is in the RUNNING or SUSPENDED states.</p>
<b>9:8</b>	<p><b>OPERATING MODE</b> <i>[read/write]</i></p> <p>These bits determine the TGEC main operating mode.</p> <ul style="list-style-type: none"> <li><b>00</b> — Normal operating mode.</li> <li><b>01</b> — Internal Loopback - The TGEC will loopback buffers from the transmit list. The data will be passed from the transmit logic back to the receive logic. The receive logic will treat the looped frame as it would any other frame, and subject it to the address filtering and validity check process.</li> <li><b>10</b> — External Loopback - The TGEC transmits normally and in addition, will enable its receive logic to its own transmissions. The receive logic will treat the looped frame as it would any other frame, and subject it to the address filtering and validity check process.</li> <li><b>11</b> — Reserved for Diagnostic</li> </ul>
<b>7</b>	<p><b>DISABLE DATA CHAINING MODE</b> <i>[read/write]</i></p> <p>When set, no data chaining will occur in reception. Frames, longer than the current receive buffer, will be truncated. RDES0&lt;FS,LS&gt; will always be set. The frame length returned in RDES0&lt;FL&gt; will be the true length of the nontruncated frame while RDES0&lt;BO&gt; will indicate that the frame has been truncated due to buffer overflow.</p> <p>When clear, frames too long for the current receive buffer, will be transferred to the next buffer(s) in the receive list.</p>

(continued on next page)



## 17.1 TGEC Programming

**Table 17–13 (Cont.) Command and Mode Register Description**

Field	Description										
<b>6</b>	<p><b>FORCE COLLISION MODE</b> <i>[read/write]</i></p> <p>This bit allows the collision logic to be tested. The chip must be in internal loopback mode for FC to be valid. If FC is set, a collision will be forced during the next transmission attempt. This will result in 16 transmission attempts with Excessive Collision reported in the transmit descriptor.</p>										
<b>5:4</b>	<p><b>THRESHOLD CONTROL BITS</b> <i>[read/write]</i></p> <p>These bits control the selected threshold level for the TGEC Transmit FIFO. Four threshold levels are allowed: 72 bytes, 96 bytes 128, bytes and 160 bytes.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Threshold</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>72 bytes</td> </tr> <tr> <td>01</td> <td>96 bytes</td> </tr> <tr> <td>10</td> <td>128 bytes</td> </tr> <tr> <td>11</td> <td>160 bytes</td> </tr> </tbody> </table>	Value	Threshold	00	72 bytes	01	96 bytes	10	128 bytes	11	160 bytes
Value	Threshold										
00	72 bytes										
01	96 bytes										
10	128 bytes										
11	160 bytes										
<b>3</b>	<p><b>PASS BAD FRAMES MODE</b> <i>[read/write]</i></p> <p>When this bit is set, the TGEC will pass frames that have been damaged by collisions or are too short due to premature reception termination. Both events should have occurred within the collision window (64 bytes), else other errors will be reported.</p> <p>When clear, these frames will be discarded and never show up in the host receive buffers.</p>										
<hr/> <p><b>Note</b></p> <hr/> <p>The swapping mode bit should be selected once after reset. Any modification will cause unexpected results.</p> <hr/>											
<b>2:1</b>	<p><b>ADDRESS FILTERING MODE</b> <i>[read/write]</i></p> <p>These bits define the way incoming frames will be address filtered:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>Normal - Incoming frames will be filtered according to the values of the &lt;HP&gt; and &lt;IF&gt; bits of the setup frame descriptor.</td> </tr> <tr> <td>01</td> <td>Promiscuous - All incoming frames will be passed to the host, regardless of the &lt;HP&gt; bit value.</td> </tr> <tr> <td>10</td> <td>All Multicast - All incoming frames with multicast address destinations will be passed to the host. Incoming frames with physical address destinations will be filtered according to the &lt;HP&gt; bit value.</td> </tr> </tbody> </table>	Value	Meaning	00	Normal - Incoming frames will be filtered according to the values of the <HP> and <IF> bits of the setup frame descriptor.	01	Promiscuous - All incoming frames will be passed to the host, regardless of the <HP> bit value.	10	All Multicast - All incoming frames with multicast address destinations will be passed to the host. Incoming frames with physical address destinations will be filtered according to the <HP> bit value.		
Value	Meaning										
00	Normal - Incoming frames will be filtered according to the values of the <HP> and <IF> bits of the setup frame descriptor.										
01	Promiscuous - All incoming frames will be passed to the host, regardless of the <HP> bit value.										
10	All Multicast - All incoming frames with multicast address destinations will be passed to the host. Incoming frames with physical address destinations will be filtered according to the <HP> bit value.										
<b>0</b>	<p><b>SWAPPING MODE</b> <i>[read/write]</i></p> <p>When clear the TGEC works with data in DEC™ standard byte ordering Endian. When set the TGEC swaps the data bytes to MOTOROLA™ standard byte ordering Endian.</p>										

Table 17–14 lists the CSR6 bit access modes.

**Table 17–14 CSR6 Access**

Value after <b>RESET</b> :	81E0F000 hex or 01E0F000 hex
<b>Read</b> access rules:	None
<b>Write</b> access rules:	
* <RE, IE, CD, SW>	Unconditional
* <BE>	Setup_frame proceeded, Password (CSR11, CSR12 and CSR13) initialized, Promiscuous and Inverse filtering modes disabled
* <BL, SE, OM>	Rx and Tx processes STOPPED
* <HW>	Rx and Tx processes STOPPED, Single_Cycle mode <SE> disabled
* <FC>	Rx and Tx processes STOPPED, Internal_Loopback mode
* <DC, PB, AF>	Rx process STOPPED
* <TR>	Tx process STOPPED
* Start_Receive <SR>=1	Rx STOPPED and CSR3 Initialized
* Start_Transmit <ST>=1	Tx STOPPED and CSR4 Initialized
* Stop_Receive <SR>=0	Rx RUNNING or SUSPENDED
* Stop_Transmit <ST>=0	Tx RUNNING or SUSPENDED

After CSR6 is written, the new value is readable from CSR6. However, if the TGEC status does not match the related write access rules, the new mode setting and command do not take effect and the written information is lost, even if the TGEC later matches the right condition.

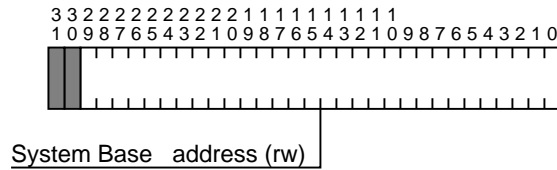
## 17.1 TGEC Programming

### 17.1.10 System Base Register (CSR7)

This CSR contains the physical starting address of the VAX System Page Table. This register must be loaded by host software before any address translation occurs so that memory will not be corrupted.

**CSR7 address is: 20008038<sub>16</sub>**

**Figure 17–8 System Base Register (CSR7)**



**Table 17–15 System Base Register Description**

Field	Description
<b>29:0</b>	<b>SYSTEM BASE ADDRESS</b> <i>[read/write]</i> The physical starting address of the VAX System Page Table. Not used if VA (Virtual Addressing) is cleared in all descriptors. This register should be loaded only once after a reset. Subsequent modifications of this register at any other time may cause unpredictable results.

Table 17–16 shows the bit access modes.

**Table 17–16 CSR7 access**

Value after <b>RESET</b> :	Unpredictable
<b>Read</b> access rules:	None
<b>Write</b> access rules:	Writing once after initialization

17.1.11 Reserved Register (CSR8)

This entire register is reserved.

17.1.12 Watchdog Timers (CSR9)

The TGEC has two timers that restrict the length of time in which the chip can receive or transmit.

**CSR9 address is: 20008048<sub>16</sub>**

Figure 17–9 Watchdog Timer (CSR9)

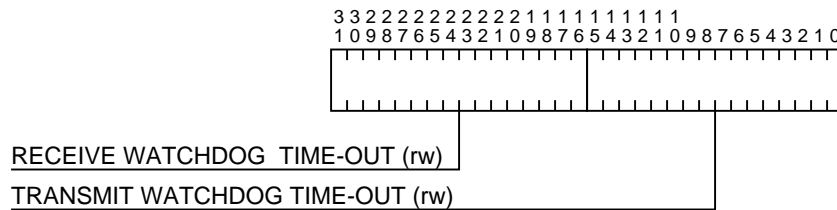


Table 17–17 Watchdog Timer Description

Field	Description
<b>31:16</b>	<p><b>RECEIVE WATCHDOG TIME-OUT</b> <i>[read/write]</i></p> <p>The Receive Watchdog Timer protects the host cpu against babbling transmitters on the network. If the receiver stays on for <math>RT * 16</math> cycles of the serial clock, the TGEC will cut off reception and set the CSR5&lt;RW&gt; bit. If the timer is set to zero, it will never time-out. The value of RT is an unsigned integer. With a 10 MHz serial clock, this provides a range of 72µs to 100 ms. The default value is 1250 corresponding to 2 ms.</p> <p>The Rx watchdog timer is programmed only while the reception process is in the STOPPED state.</p> <div style="text-align: center; margin-top: 20px;"> <p>_____ <b>Note</b> _____</p> <p>A Rx watchdog value between 1 and 44 is forced to the minimum time_out value of 45 (72µs).</p> <p>_____</p> </div>
<b>15:0</b>	<p><b>TRANSMIT WATCHDOG TIME-OUT</b> <i>[read/write]</i></p>

(continued on next page)

## 17.1 TGEC Programming

**Table 17–17 (Cont.) Watchdog Timer Description**

Field	Description
	<p>The Transmit Watchdog Timer protects the network against babbling TGEC transmissions, on top of any such circuitry present in transceivers. If the transmitter stays on for <math>TT * 16</math> cycles of the serial clock, the TGEC will cut off the transmitter and set the CSR5&lt;TW&gt; bit. If the timer is set to zero, it will never time-out. The value of TT is an unsigned integer. With a 10 MHz serial clock, this provides a range of 72µs to 100ms. The default value is 1250 corresponding to 2ms.</p> <p>The Tx watchdog timer is programmed only while the Transmission process is in the STOPPED state.</p>
<p><b>Note</b></p> <p>A Tx watchdog value between 1 and 44 is forced to the minimum time_out value of 45 (72µs).</p>	

Table 17–18 shows the bit access modes.

**Table 17–18 CSR9 Access**

Value after <b>RESET</b> :	00000000 <sub>16</sub>
<b>Read</b> access rules:	None
<b>Write</b> access rules:	
* Rx watchdog timer	Rx process STOPPED
* Tx watchdog timer	Tx process STOPPED

These watchdog timers are enabled by default. These timers will assume the default values after hardware or software resets.

17.1.13 TGEC Identification and Missed Frame Count (CSR10)

This register contains a missed-frame counter and TGEC identification information.

**CSR10 address is: 20008050<sub>16</sub>**

Figure 17–10 TGEC Identification and Missed Frame Count (CSR10)

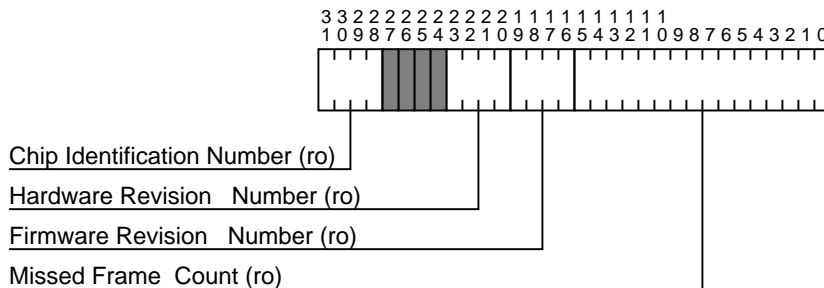


Table 17–19 TGEC Identification and Missed Frame Count Description

Field	Description
<b>31:28</b>	<b>CHIP IDENTIFICATION NUMBER</b> <i>[read-only]</i> This field allows to to determine whether the plugged in device is a TGEC or another TGEC compatible device (for example, SGEC or LC_SGEC). The TGEC device identification number is 2.
<b>23:20</b>	<b>HARDWARE REVISION NUMBER</b> <i>[read-only]</i> Revision number for this particular TGEC.
<b>19:16</b>	<b>FIRMWARE REVISION NUMBER</b> <i>[read-only]</i> Internal firmware revision number for this particular TGEC.
<b>15:0</b>	<b>MISSED FRAME COUNT</b> <i>[read-only]</i> Counter for the number of frames that were discarded and lost because host receive buffers were unavailable. The counter is cleared when read by the host.

Pass 1.0

DIN = 2 ; HRN = 1 ; FRN = 1

Table 17–20 shows the bit access modes.

Table 17–20 CSR10 access

Value after <b>RESET</b> :	20110000 <sub>16</sub>
<b>Read</b> access rules:	Missed_frame counter cleared by read
<b>Write</b> access rules:	Not applicable



Table 17–22 CSR11,12,13 access

---

Value after <b>RESET</b> :	00000000 <sub>16</sub> for each of CSR11,CSR12,CSR13
<b>Read</b> access rules:	None
<b>Write</b> access rules:	Boot message DISABLED (CSR6<BE> = 0)

---



## 17.1 TGEC Programming

### 17.1.15 Diagnostic Registers (CSR14, 15)

These registers are reserved for diagnostic features.

**CSR14 address is: 20008070<sub>16</sub>**

**CSR15 address is: 20008078<sub>16</sub>**

## 17.2 Descriptors and Buffers Format

The TGEC transfers frame data to and from receive and transmit buffers in host memory. These buffers are pointed to by descriptors which are also resident in host memory.

There are two descriptor lists: one for receive and one for transmit. The starting address of each list is written into CSRs 3 and 4 respectively. A descriptor list is a forward-linked (either implicitly or explicitly) list of descriptors, the last of which may point back to the first entry, thus creating a ring structure. Explicit chaining of descriptors, through setting xDES1<CA> is called Descriptor Chaining. The descriptor lists reside in VAX *physical* memory address space.

The TGEC first reads the descriptors, ignoring all unused bits regardless of their state. The only word the TGEC writes back, is the first word (xDES0) of each descriptor. Unused bits in xDES0 will be written as "0". Unused bits in xDES1 - xDES3 may be used by the port driver and the TGEC will never disturb them.

A data buffer can contain an entire frame or part of a frame, but it cannot contain more than a single frame. Buffers contain only data; buffer status is contained in the descriptor. The term Data Chaining is used to refer to frames spanning multiple data buffers. Data Chaining can be enabled or disabled, in reception, through CSR6<DC>. Data buffers reside in VAX memory space, either physical or virtual.

The virtual to physical address translation is based on the assumption that PTEs are locked in the host memory the time the TGEC owns the related buffer.

For best performance in virtual addressing mode, PPTE vectors must not cross a page of the PPTE table.

### 17.2.1 Receive Descriptors

The receive descriptor format is shown in Figure 17–12, and described in the following paragraphs.



## 17.2 Descriptors and Buffers Format

Table 17–23 (Cont.) RDES0 Bits

Bit	Name	Description															
02	DB	<p>Dribbling Bits - When set, indicates the frame contained a non-integer multiple of eight bits. This error will be reported only if the number of dribbling bits in the last byte is greater than two. Meaningless if RDES0&lt;CS&gt; or RDES0&lt;RF&gt; are set.</p> <p>The CRC check is performed independent of this error, however, only whole bytes are run through the CRC logic. Consequently, received frames with up to six dribbling bits will have this bit set, but if &lt;CE&gt; (or another error indicator) is not set, these frames should be considered valid:</p> <table border="1"> <thead> <tr> <th>CE</th> <th>DB</th> <th>Error</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>None</td> </tr> <tr> <td>0</td> <td>1</td> <td>None</td> </tr> <tr> <td>1</td> <td>0</td> <td>CRC error</td> </tr> <tr> <td>1</td> <td>1</td> <td>Alignment error</td> </tr> </tbody> </table>	CE	DB	Error	0	0	None	0	1	None	1	0	CRC error	1	1	Alignment error
CE	DB	Error															
0	0	None															
0	1	None															
1	0	CRC error															
1	1	Alignment error															
03	TN	Translation Not Valid - When set, indicates that a translation error occurred when the TGEC was translating a VAX virtual buffer address. It will only set if RDES1<VA> was set. The Reception process remains in the RUNNING state and attempts to acquire the next descriptor.															
05	FT	Frame Type - When set, indicates the frame is an Ethernet type frame (Frame Length_Field > 1500). When clear, indicates the frame is an IEEE 802.3 type frame. Meaningless for Runt frames < 14 bytes.															
06	CS	Collision Seen - When set, indicates the frame was damaged by a collision that occurred after the 64 bytes following the SFD.															
07	TL	Frame Too Long - When set, indicates the frame length exceeds the maximum Ethernet specified size of 1518 bytes.															
<p><b>Note</b></p> <p>Frame Too Long is only a frame length indication and does not cause any frame truncation.</p>																	
08	LS	Last Segment - When set, indicates this buffer contains the last segment of a frame and status information is valid.															
09	FS	First Segment - When set, indicates this buffer contains the first segment of a frame.															
10	BO	Buffer overflow - When set, indicates that the frame has been truncated due to a buffer too small to fit the frame size. This bit may only be set if Data Chaining is disabled (CSR6<DC> = 1).															
11	RF	Runt Frame - When set, indicates this frame was damaged by a collision or premature termination before the collision window had passed. Runt frames will only be passed on to the host if (CSR6<PB>) is set. Meaningless if RDES0<OF> is set.															

(continued on next page)

## 17.2 Descriptors and Buffers Format

**Table 17–23 (Cont.) RDES0 Bits**

Bit	Name	Description								
13:12	DT	Data Type - Indicates the type of frame the buffer contains, according to the following table: <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>Serial received frame.</td> </tr> <tr> <td>01</td> <td>Internally looped back frame.</td> </tr> <tr> <td>10</td> <td>Externally looped back frame, Serial received frame. (The TGEC does not differentiate between looped back and serial received frames. Therefore this information is global and reflects only CSR6&lt;OM&gt;).</td> </tr> </tbody> </table>	Value	Meaning	00	Serial received frame.	01	Internally looped back frame.	10	Externally looped back frame, Serial received frame. (The TGEC does not differentiate between looped back and serial received frames. Therefore this information is global and reflects only CSR6<OM>).
Value	Meaning									
00	Serial received frame.									
01	Internally looped back frame.									
10	Externally looped back frame, Serial received frame. (The TGEC does not differentiate between looped back and serial received frames. Therefore this information is global and reflects only CSR6<OM>).									
14	LE	Length Error - When set, indicates a frame truncation caused by one of the following: <ul style="list-style-type: none"> <li>• The frame segment does not fit within the current buffer and the TGEC does not own the next descriptor. The frame is truncated.</li> <li>• The Receive Watchdog timer expired. CSR5&lt;RW&gt; is also set.</li> <li>• Invalid Page Table Entry of one of the frame's buffer virtual address.</li> </ul>								
15	ES	Error Summary - The logical "OR" of RDES0 bits OF,CE,TN,CS,TL,LE,RF.								
30:16	FL	Frame Length - The length in bytes of the received frame. Meaningless if RDES0<LE> is set.								
31	OW	Own bit - When set, indicates the descriptor is owned by the TGEC. When cleared, indicates the descriptor is owned by the host. The TGEC clears this bit upon completing processing of the descriptor and its associated buffer.								

### 17.2.1.2 RDES1 Word

Table 17–24 describes the receive descriptor 1 bits.

**Table 17–24 RDES1 Bits**

Bit	Name	Descriptor
29	VT	Virtual Type - In case of virtual addressing (RDES1<VA> = 1), indicates the type of virtual address translation. When clear, the buffer address RDES3 is interpreted as a SVAPTE (System Virtual Address of the Page Table Entry). When set, the buffer address is interpreted as a PAPTE (Physical Address of the Page Table Entry). Meaningful only if RDES1<VA> is set.

(continued on next page)

## 17.2 Descriptors and Buffers Format

Table 17–24 (Cont.) RDES1 Bits

Bit	Name	Descriptor												
30	VA	<p>Virtual Addressing - When set, RDES3 is interpreted as a virtual address. The type of virtual address translation is determined by the RDES1&lt;VT&gt; bit. The TGEC uses RDES3 and RDES2&lt;Page Offset&gt; to perform a VAX virtual address translation process to obtain the physical address of the buffer. When clear, RDES3 is interpreted as the actual physical address of the buffer:</p> <table border="1"> <thead> <tr> <th>VA</th> <th>VT</th> <th>Addressing mode</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>x</td> <td>Physical</td> </tr> <tr> <td>1</td> <td>0</td> <td>Virtual - SVAPTE</td> </tr> <tr> <td>1</td> <td>1</td> <td>Virtual - PAPTE</td> </tr> </tbody> </table>	VA	VT	Addressing mode	0	x	Physical	1	0	Virtual - SVAPTE	1	1	Virtual - PAPTE
VA	VT	Addressing mode												
0	x	Physical												
1	0	Virtual - SVAPTE												
1	1	Virtual - PAPTE												
31	CA	<p>Chain Address - When set, RDES3 is interpreted as another descriptor's VAX physical address. This allows the TGEC to process multiple, non-contiguous descriptor lists and explicitly "chain" the lists. Note that contiguous descriptors are implicitly chained.</p> <p>In contrast to what is done for a Rx buffer descriptor, the TGEC clears neither the ownership bit RDES0&lt;OW&gt; nor one of the other bits of RDES0 of the chain descriptor after processing.</p> <p>To protect against infinite loop, a chain descriptor pointing back to itself is appears as owned by the host, regardless of the ownership bit state.</p>												

### 17.2.1.3 RDES2 Word

Table 17–25 describes the receive descriptor 2 bits.

Table 17–25 RDES2 Bits

Bit	Name	Descriptor
08:00	PO	Page Offset - The byte offset of the buffer within the page. Only meaningful if RDES1<VA> is set. Receive buffers must be word aligned.
30:16	BS	Buffer Size - The size, in bytes, of the data buffer. Receive buffers size must be an even number of bytes, not shorter than 16 bytes.

### 17.2.1.4 RDES3 Word

Table 17–26 describes the receive descriptor 3 bits.

**Table 17–26 RDES3 Bits**

Bit	Name	Descriptor
31:00	SV/PV/PA	SVAPTE/PAPTE/Physical Address - When RDES1<VA> is set, RDES3 is interpreted as the address of the Page Table Entry and used in the virtual address translation process. The type of the address System Virtual address (SVAPTE) or Physical Address (PAPTE) is determined by RDES1<VT>. When RDES1<VA> is clear, RDES3 is interpreted as the physical address of the buffer. When RDES1<CA> is set, RDES3 is interpreted as the VAX physical address of another descriptor. Receive buffers must be word aligned.

### 17.2.1.5 Receive Descriptor Status Validity

Table 17–27 summarizes the validity of the receive descriptor status bits regarding the reception completion status:

**Table 17–27 Receive Descriptor Status Validity**

Reception status	Rx Status report						
	RF	TL	CS	FT	DB	CE	(ES,LE,BO,DT,FS,LS,FL,TN,OF)
Overflow	X	V	X	V	X	X	V
Collision after 512 bits	V	V	V	V	X	X	V
Runt frame	V	V	V	V	X	X	V
Runt frame < 14 bytes	V	V	V	X	X	X	V
Watchdog timeout	V	V	X	V	X	X	V

V - Valid  
X - Meaningless

### 17.2.2 Transmit Descriptors

The transmit descriptor format is shown in Figure 17–13 and described in the following paragraphs.



## 17.2 Descriptors and Buffers Format

Table 17–28 (Cont.) TDES0 bits

Bit	Name	Description
09	LC	Late Collision - When set, indicates frame transmission was aborted due to a late collision. Meaningless if TDES0<UF>.
10	NC	No Carrier - When set, indicates the carrier signal from the transceiver was not present during transmission (possible problem in the transceiver or transceiver cable). Meaningless in internal loopback mode (CSR5<OM>=1).
11	LO	Loss of Carrier - When set, indicates loss of carrier during transmission (possible short circuit in the Ethernet cable). Meaningless in internal loopback mode (CSR5<OM>=1).
12	LE	Length Error - When set, indicates one of the following: <ul style="list-style-type: none"> <li>• Descriptor unavailable (owned by the host) in the middle of data chained descriptors.</li> <li>• Zero length buffer in the middle of data chained descriptors.</li> <li>• Setup or Diagnostic descriptors (Data type TDES1&lt;DT&gt; &lt;&gt; 0) in the middle of data chained descriptors.</li> <li>• Incorrect order of first_segment TDES1&lt;FS&gt; and last_segment TDES1&lt;LS&gt; descriptors in the descriptor list.</li> <li>• Invalid Page Table Entry of one of the frame's buffer virtual address.</li> </ul> <p>The Transmission process enters the SUSPENDED state and sets CSR5&lt;TC, TI&gt;.</p>
14	TO	Transmit Watchdog Timeout - When set, indicates the transmit watchdog timer has timed out, indicating the TGEC transmitter was babbling. The interrupt CSR5<TW> is set and the Transmission process is <i>aborted</i> and placed in the STOPPED state.
15	ES	Error Summary - The logical “OR” of UF, TN, EC, LC, NC, LO, LE and TO.
29:16	TDR	Time Domain Reflectometer - This is a count of bit time and is useful for locating a fault on the cable using the velocity of propagation on the cable. Only valid if TDES0<EC> is also set. Two Excessive Collisions in a row and with the same or similar (within 20) TDR values indicate a possible cable open.
31	OW	Own bit - When set, indicates the descriptor is owned by the TGEC. When cleared, indicates the descriptor is owned by the host. The TGEC clears this bit upon completing processing of the descriptor and its associated buffer.

### 17.2.2.2 TDES1 word

Table 17–29 describes the transmit descriptor 1 bits.



## 17.2 Descriptors and Buffers Format

**Table 17–29 TDES1 bits**

Bit	Name	Descriptor												
23	VT	Virtual Type - In case of virtual addressing (TDES1<VA> = 1), indicates the type of virtual address translation. When clear, the buffer address TDES3 is interpreted as a SVAPTE (System Virtual Address of the Page Table Entry). When set, the buffer address is interpreted as a PAPTE (Physical Address of the Page Table Entry). Meaningful only if TDES1<VA> is set.												
24	IC	Interrupt on Completion - When set, the TGEC will set CSR5<TI> after this frame has been transmitted. To take effect, this bit must be set in the descriptor where LS is set.												
25	LS	Last Segment - When set, indicates the buffer contains the last segment of a frame.												
26	FS	First Segment - When set, indicates the buffer contains the first segment of a frame.												
27	AC	Add CRC disable - When set, the TGEC will not append the CRC to the end of the transmitted frame. To take effect, this bit must be set in the descriptor where FS is set. If the transmitted frame is shorter than 64 bytes, the TGEC will add the padding field and the CRC regardless of the <AC> flag.												
29:28	DT	Data Type - Indicates the type of data the buffer contains, according to the following table: <table border="1" data-bbox="669 940 1442 1108"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>Normal transmit frame data</td> </tr> <tr> <td>10</td> <td>Setup frame - Explained in Section 17.2.3.</td> </tr> <tr> <td>11</td> <td>Diagnostic frame -</td> </tr> </tbody> </table>	Value	Meaning	00	Normal transmit frame data	10	Setup frame - Explained in Section 17.2.3.	11	Diagnostic frame -				
Value	Meaning													
00	Normal transmit frame data													
10	Setup frame - Explained in Section 17.2.3.													
11	Diagnostic frame -													
30	VA	Virtual Addressing - When set, TDES3 is interpreted as a virtual address. The type of virtual address translation is determined by the TDES1<VT> bit. The TGEC uses TDES3 and TDES2<Page Offset> to perform a VAX virtual address translation process to obtain the physical address of the buffer. When clear, TDES3 is interpreted as the actual physical address of the buffer: <table border="1" data-bbox="669 1331 1442 1499"> <thead> <tr> <th>VA</th> <th>VT</th> <th>Addressing mode</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>x</td> <td>Physical</td> </tr> <tr> <td>1</td> <td>0</td> <td>Virtual - SVAPTE</td> </tr> <tr> <td>1</td> <td>1</td> <td>Virtual - PAPTE</td> </tr> </tbody> </table>	VA	VT	Addressing mode	0	x	Physical	1	0	Virtual - SVAPTE	1	1	Virtual - PAPTE
VA	VT	Addressing mode												
0	x	Physical												
1	0	Virtual - SVAPTE												
1	1	Virtual - PAPTE												
31	CA	Chain Address - When set, TDES3 is interpreted as another descriptor's VAX physical address. This allows the TGEC to process multiple, non-contiguous descriptor lists and explicitly "chain" the lists. Note that contiguous descriptors are implicitly chained.  In contrast to what is done for a Tx buffer descriptor, the TGEC clears neither the ownership bit TDES0<OW> nor one of the other bits of TDES0 of the chain descriptor after processing.  To protect against infinite loop, a chain descriptor pointing back itself, is seen as owned by the host, regardless of the ownership bit state.												

## 17.2 Descriptors and Buffers Format

### 17.2.2.3 TDES2 Word

Table 17–30 describes the transmit descriptor 2 bits.

**Table 17–30 TDES2 Bits**

Bit	Name	Descriptor
08:00	PO	Page Offset - The byte offset of the buffer within the page. Only meaningful if TDES1<VA> is set. Transmit buffers may start on arbitrary byte boundaries.
30:16	BS	Buffer Size - The size, in bytes, of the data buffer. If this field is 0, the TGEC will skip over this buffer and ignore it. The frame size is the sum of all BS fields of the frame segments between and including the descriptors having TDES1<FS> and TDES1<LS> set. If the port driver wishes to suppress transmission of a frame, this field must be set to 0 in all descriptors comprising the frame and prior to the TGEC acquiring them. If this rule is not adhered to, corrupted frames might be transmitted.

### 17.2.2.4 TDES3 word

Table 17–31 describes the transmit descriptor 3 bits.

**Table 17–31 TDES3 Bits**

Bit	Name	Descriptor
31:00	SV/PV/PA	SVAPTE/PAPTE/Physical Address - When TDES1<VA> is set, TDES3 is interpreted as the address of the Page Table Entry and used in the virtual address translation process. The type of the address System Virtual address (SVAPTE) or Physical Address (PAPTE) is determined by TDES1<VT>. When TDES1<VA> is clear, TDES3 is interpreted as the physical address of the buffer. When TDES1<CA> is set, TDES3 is interpreted as the VAX physical address of another descriptor. Transmit buffers may start on arbitrary byte boundaries.

### 17.2.2.5 Transmit Descriptor Status Validity

Table 17–32 summarizes the validity of the Transmit descriptor status bits regarding the Transmission completion status:

**Table 17–32 Transmit descriptor status validity**

Transmission Status	Tx Status Report						
	LO	NC	LC	EC	HF	CC	(ES,TO,LE,TN,UF,DE)
Underflow	X	X	V	V	X	V	V
Excessive collisions	V	V	V	V	V	X	V
Watchdog timeout	X	V	X	X	X	V	V
Internal Loopback	X	X	V	V	X	V	V

V - Valid  
X - Meaningless



## 17.2 Descriptors and Buffers Format

Table 17–33 describes the setup frame descriptor bits.

**Table 17–33 Setup frame descriptor bits**

Word	Bit	Name	Description
SDES0	13	SE	Setup Error - When set, indicates the setup frame buffer size is incorrect regarding the address filtering mode: <ul style="list-style-type: none"> <li>• In perfect filtering, if the buffer size is not a multiple of 8 bytes or bigger than 128 bytes.</li> <li>• In hash filtering, if the buffer is shorter than 72 bytes.</li> </ul>
	15	ES	Error Summary - Set when SE is set.
	31	OW	Own bit - When set, indicates the descriptor is owned by the TGEC. When cleared, indicates the descriptor is owned by the host. The TGEC clears this bit upon completing processing of the descriptor and its associated buffer.
SDES1	24	IC	Interrupt on Completion - When set, the TGEC will set CSR5<TI> after this setup frame has been processed.
	25	HP	Hash/Perfect filtering mode - When set, the TGEC will interpret the setup frame as a hash table, and do an imperfect address filtering. The imperfect mode is useful when there are more than 16 multicast addresses to listen to. When clear, the TGEC will do a perfect address filter of incoming frames according to the addresses specified in the setup frame.
	26	IF	Inverse filtering - When set, the TGEC will do an inverse filtering: the TGEC will receive the incoming frames with destination address not matching the perfect addresses and will reject the frames with destination address matching one of the perfect addresses. Meaningful only for Perfect_filtering (SDES1<HP>=0), while promiscuous and All_Multicast modes are not selected (CSR6<AF>=0).
	29:28	DT	Data Type - Must be 2 to indicate setup frame.

(continued on next page)

## 17.2 Descriptors and Buffers Format

Table 17–33 (Cont.) Setup frame descriptor bits

Word	Bit	Name	Description
SDES2	30:16	BS	<p>Buffer Size - In perfect mode the buffer size indicates the number of perfect addresses stored into the setup buffer. If this number is smaller than 16, the remaining addresses are internally filled with one of the setup buffer addresses.</p> <p>The buffer size must be consistent with the filtering mode:</p> <ul style="list-style-type: none"><li>• In perfect filtering, the buffer size must be a multiple of 8 bytes from 8 up to 128 bytes.</li><li>• In hash filtering, the buffer size must be equal to or bigger than 72 bytes.</li></ul> <p>If the buffer size is incorrect, the setup frame is not loaded and the setup error bit SDES0&lt;SE&gt; is set.</p>
SDES3	29:1	PA	<p>Physical Address - Physical address of setup buffer. Setup buffer must be word aligned.</p>

### 17.2.3.4 Perfect Filtering Setup Frame Buffer

This section describes how the TGEC interprets a setup frame buffer when SDES1<HP> is clear.

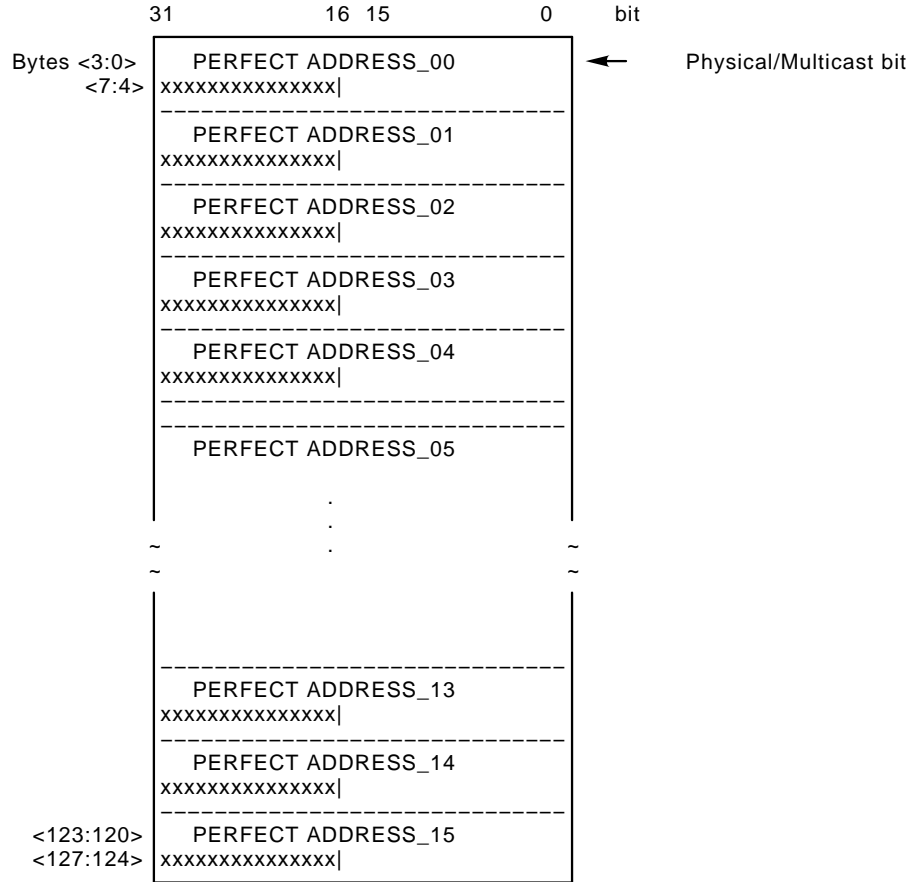
The TGEC can store 16 (full 48-bits Ethernet) destination addresses. It will compare the addresses of any incoming frame to these, and regarding the status of Inverse\_Filtering flag SDES1<IF>, will reject those which

- do not match, if SDES1<IF> = 0;
- match, if SDES1<IF> = 1.

The setup frame must always supply all 16 addresses. Any mix of physical and multicast addresses can be used. Unused addresses should be duplicates of one of the valid addresses. The addresses are formatted as shown in Figure 17–15.

## 17.2 Descriptors and Buffers Format

**Figure 17–15 Perfect Filtering setup frame buffer format**



xxxxxx = don't care

ESB90P0067

The low-order bit of the low-order bytes is the address's multicast bit.  
Example 17–1 illustrates a Perfect Filtering Setup buffer (fragment).

## 17.2 Descriptors and Buffers Format

### Example 17–1 Perfect Filtering Buffer

Ethernet addresses to be filtered:

- ① A8-09-65-12-34-76  
09-BC-87-DE-03-15  
.  
.  
.

Setup frame buffer fragment:

- ② 126509A8  
00007634  
DE87BC09  
00001503  
.  
.  
.

- ① Two Ethernet addresses written according to the DEC STD 134 specification for address display.
- ② Those two addresses as they would appear in the buffer.

### 17.2.3.5 Imperfect Filtering Setup Frame Buffer

This section describes how the TGEC interprets a setup frame buffer when SDES1<HP> is set.

The TGEC can store 512 bits, serving as hash bucket heads, and one physical 48-bit Ethernet address. Incoming frames with multicast destination addresses will be subjected to the imperfect filtering. Frames with physical destination addresses will be checked against the single physical address.

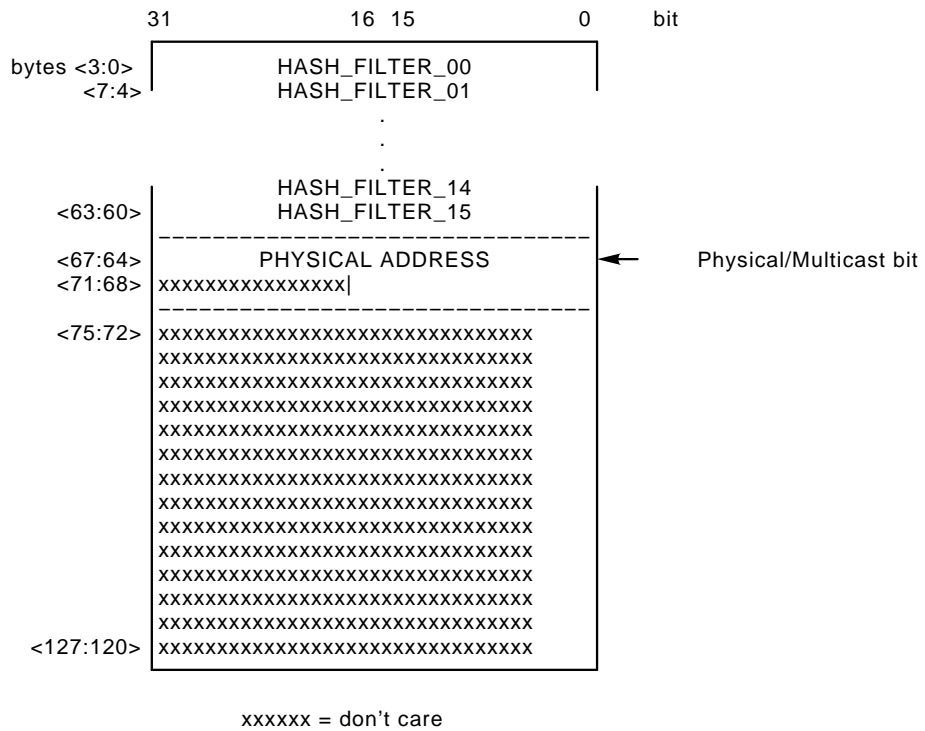
For any incoming frame with a multicast destination address, the TGEC applies the standard Ethernet CRC function the first six bytes containing the destination address, then uses the most significant nine bits of the result, as a bit index into the table. If the indexed bit is set, the frame is accepted. If it is cleared, the frame is rejected.

This filtering mode is called imperfect, because multicast frames not addressed to this station may slip through, but it will still cut down the number of frames the host will be presented with.

The format for the hash table and the physical address is shown in Figure 17–16.

## 17.2 Descriptors and Buffers Format

Figure 17–16 Imperfect Filtering Setup Frame Format



ESB90P0068

Bits are sequentially numbered from right to left and down the table. For example, if  $\text{CRC}(\text{destination address})\langle 8:0 \rangle = 33$ , the TGEC will examine bit number 1 in the second longword.

Example 17–2 illustrates an imperfect filtering setup frame buffer.



## 17.2 Descriptors and Buffers Format

### Example 17–2 Imperfect Filtering Buffer

Ethernet addresses to be filtered:

- ① 25-00-25-00-27-00  
A3-C5-62-3F-25-87  
D9-C2-C0-99-0B-82  
7D-48-4D-FD-CC-0A  
E7-C1-96-36-89-DD  
61-CC-28-55-D3-C7  
6B-46-0A-55-2D-7E
- ② A8-12-34-35-76-08

Setup frame buffer:

- ③ 00000000  
10000000  
00000000  
00000000  
00000000  
40000000  
00000080  
00100000  
00000000  
10000000  
00000000  
00000000  
00000000  
00010000  
00000000  
00400000
- ④ 353412A8  
00000876

- ① Ethernet multicast addresses written according to the DEC STD 134 specification for address display.
- ② An Ethernet physical address.
- ③ The first part of an Imperfect filter Setup frame buffer with set bits for the multicast addresses ①.
- ④ The second part of the buffer with the physical address ②.

## 17.3 TGEC Hardware and Software Reset

The TGEC responds to two types of reset commands: a hardware reset through the RESET\_L pin, and a software reset command triggered by setting CSR6<RE>. In both cases, the TGEC aborts all ongoing processing and starts the Reset sequence. The TGEC restarts and reinitializes all internal states and registers. No internal states are retained, no descriptors are owned and all the host visible registers are set to “0”, except where otherwise noted. The TGEC does not explicitly disown any owned descriptor; so descriptors OWNED bits might be left in a state indicating TGEC ownership.

The following table indicates the CSR fields which are not set to “0” after reset:

Field	Value
CSR3	Unpredictable
CSR4	Unpredictable

## 17.3 TGEC Hardware and Software Reset

Field	Value
CSR5<DN>	1
CSR6<BL>	1
CSR6<RE>	Unpredictable after HARDWARE reset 1 after SOFTWARE reset
CSR7	Unpredictable
CSR9	RT = TT = 1250

After the reset sequence completes, the TGEC executes the self test procedure to do basic sanity checking.

If the self test completes successfully, the TGEC initializes the TGEC, then sets the Initialization Done flag CSR5<ID>.

At the first failure detected in one of the basic tests executed in the self\_test routine, the test is aborted and the self\_test failure CSR5<SF> is set together with the self\_test error status CSR5<SS>which indicates the failure reason. The self test takes 25ms to complete after Hardware or Software RESET.

If the initialization completes successfully, the TGEC is ready to accept further host commands. Both the Reception and Transmission processes are placed in the STOPPED state.

Successive reset commands (either hardware or software) may be issued. The only restriction is that TGEC CSRs should not be accessed during a 1 $\mu$ s period following the reset. Access during this period will result in a CP-BUS timeout error. Access to TGEC CSRs during the self test are permitted; however, only CSR5 reads should be performed.

## 17.4 TGEC Interrupts

Interrupts are generated as a result of various events. CSR5 contains all the status bits which may cause an interrupt, provided CSR6<IE> is set. The port driver must clear the interrupt bits (by writing a “1” to the bit position), to enable further interrupts from the same source.

Interrupts are not queued, and if the interrupting event reoccurs before the port driver has responded to it, no additional interrupts will be generated. For example, CSR5<RI> indicates one or more frames were delivered to host memory. The port driver should scan all descriptors, from its last recorded position up to the first TGEC owned one.

An interrupt will only be generated once for simultaneous, multiple interrupting events. It is the port driver responsibility to scan CSR5 for the interrupt cause(s). The interrupt will not be regenerated, unless a new interrupting event occurs after the host acknowledged the previous one, and provided the port driver cleared the appropriate CSR5 bit(s). For example, CSR5<TI> and CSR5<RI> may both set, the host acknowledges the interrupt and the port driver begins executing by reading CSR5. Now CSR5<RU> sets. The port driver writes back its copy of CSR5, clearing CSR5<TI> and CSR5<RI>. After the host IPL is lowered below the TGEC level, another interrupt will be delivered with the CSR5<RU> bit set.

Should the port driver clear all CSR5 set interrupt bits before the interrupt has been acknowledged, the interrupt will be suppressed.

## 17.4 TGEC Interrupts

### 17.4.1 Startup Procedure

A sequence of checks and commands must be performed by the port driver to prepare the TGEC for operation.

1. Wait for the TGEC to complete its Initialization sequence by polling on CSR5<ID> and CSR5<SF> (refer to Section 17.1.8 for details).
2. Examine CSR5<SF> to find out whether the TGEC passed its self test. If it did not, it should be replaced (refer to Section 17.1.8 for details).
3. Write CSR0 to establish system configuration dependent parameters (refer to Section 17.1.4 for details).
4. If the port driver intends to use VAX virtual addresses, CSR7 must be written to identify the System Page Table to the TGEC (refer to Section 17.1.10 for details).
5. If the port driver wishes to change the default settings of the watchdog timers, it must write to CSR9 (refer to Section 17.1.12 for details).
6. Port driver must create the transmit and receive descriptor lists, then write to CSR3 and CSR4 to provide the TGEC with the starting address of each list. The first descriptor on the transmit list will usually contain a setup frame (refer to Section 17.1.7 for details).
7. Write CSR6 to set global operating parameters and start the Transmission and Reception processes. The Reception and Transmission processes enter the RUNNING state and attempt to acquire descriptors from the respective descriptor lists and begin processing incoming and outgoing frames (refer to Section 17.1.9 for details). The Reception and Transmission processes are independent of each other and can be started and stopped separately.

---

**Note**

---

If address filtering (either perfect or imperfect) is desired, the Reception process should be started only after the Setup frame has been processed.

---

8. The port driver now waits for any TGEC interrupts. If either the Reception or Transmission processes were SUSPENDED, the port driver must issue the Poll Demand command after it has rectified the suspension cause.

## 17.5 Reception Process

While in the RUNNING state, the reception process polls the receive descriptor list, attempting to acquire free descriptors. Incoming frames are processed and placed in acquired descriptors' data buffers, while status information is written to the descriptor RDES0 words. The TGEC always tries to acquire an extra descriptor in anticipation of incoming frames. Descriptor acquisition is attempted under the following conditions:

- Immediately after being placed in the RUNNING state through setting of CSR6<SR>.
- The TGEC begins writing frame data to a data buffer pointed to by the current descriptor.
- The last acquired descriptor was chained (RDES1<CA> = 1) to another descriptor.

## 17.5 Reception Process

- A virtual translation error was encountered RDES0<TN> while the TGEC was translating the buffer base address of the acquired descriptor.

As incoming frames arrive, the TGEC strips the preamble bits and stores the frame data in the receive FIFO. Concurrently, it performs address filtering according to CSR6 fields AF, HP and its internal filtering table. If the frame fails the address filtering, it is ignored and purged from the FIFO. Frames which are shorter than 64 bytes, due to collision or premature termination, are also ignored and purged from the FIFO, unless CSR6<PB> is set.

After 64 bytes have been received, the TGEC begins transferring the frame data to the buffer pointed to by the current descriptor. If Data Chaining is enabled (CSR6<DC> clear), the TGEC will write frame data overflowing the current data buffer into successive buffer(s). The TGEC sets the RDES0<FS> and RDES0<LS> in the first and last descriptors, respectively, to delimit the frame. Descriptors are released (RDES0<OW> bit cleared) as their data buffers fill up or the last segment of a frame has been transferred to a buffer.

The TGEC sets RDES0<LS> and the RDES0 status bits in the last descriptor it releases for a frame. After the last descriptor of a frame is released, the TGEC sets CSR5<RI>.

This process is repeated until the TGEC encounters a descriptor flagged as owned by the host. After filling up all previously acquired buffers, the Reception sets CSR5<RU> and enters the SUSPENDED state. The position in the receive list is retained.

Any incoming frames while in this state will cause the TGEC to fetch the current descriptor in the host memory. If the descriptor is now owned by the TGEC, the Reception re-enters the RUNNING state and starts the frame reception.

If the descriptor is still owned by the host, the TGEC increments the Missed Frames Counter (CSR10<MFC>) and discards the frame.

Table 17–34 summarizes the Reception process state transitions and resulting actions:

**Table 17–34 Reception Process State Transitions**

From State	Event	To state	Action
STOPPED	Start Reception command	RUNNING	Receive polling begins from last list position or from the list head if this is the first Start command issued, or if the receive descriptor list address (CSR3) was modified by the port driver.
RUNNING	TGEC attempts acquisition of a descriptor owned by the host	SUSPENDED	CSR5<RU> is set when the last acquired descriptor buffer is consumed. Position in list retained.
RUNNING	Stop Reception command	STOPPED	Reception process is STOPPED after the current frame, if any, is completely transferred to data buffer(s). Position in list retained.

(continued on next page)

## 17.5 Reception Process

Table 17–34 (Cont.) Reception Process State Transitions

From State	Event	To state	Action
RUNNING	Memory or host bus parity error encountered	STOPPED	Reception is cut off and CSR5<ME> is set.
RUNNING	Reset command	STOPPED	Reception is cut off.
SUSPENDED	Rx Poll demand or Incoming frame and available descriptor	RUNNING	Receive polling resumes from last list position or from the list head if CSR3 was modified by the port driver.
SUSPENDED	Stop Reception command	STOPPED	None.
SUSPENDED	Reset command	STOPPED	None.

### 17.5.1 Transmission Process

While in the RUNNING state, the Transmission process polls the transmit descriptor list for any frames to transmit. Frames are built and transmitted on the Ethernet wire. Upon completing frame transmission (or giving up), status information is written to the TDES0 words. Once polling starts, it continues (in sequential or descriptor chained order) until the TGEC encounters a descriptor flagged as owned by the host, or an error condition. At this point, the Transmission process is placed in the SUSPENDED state and CSR5<TC, TI> is set.

CSR5<TI> will also be set after completing transmission of a frame which has TDES1<IC> set in its last descriptor. In this case, the Transmission process remains in the RUNNING state.

Frames may be data chained and span several buffers. Frames must be delimited by TDES1<FS> and TDES1<LS> in the first and last descriptors, respectively, containing the frame. While in the RUNNING state, as the Transmission process starts, it first expects a descriptor with TDES1<FS> set. Frame data transfer from the host buffer to the internal FIFO is initiated.

Concurrently, if the current frame had TDES1<LS> clear, the transmission process attempts to acquire the next descriptor, expecting TDES1<FS> and TDES1<LS> to be clear, indicating an intermediary buffer, or TDES1<LS> to be set, indicating the end of the frame.

After the last buffer of the frame has been transmitted, the TGEC writes back final status information to the TDES0 word of the descriptor having TDES1<LS> set, optionally sets CSR5<TI> if TDES1<IC> was set, and repeats the process with the next descriptor(s). Actual frame transmission begins *after at least 72 bytes* have been transferred to the internal FIFO, or *a full frame is contained* in the FIFO. Descriptors are released (TDES0<OW> bit cleared) as soon as the TGEC is through processing a descriptor.

Transmit polling suspends under the following conditions:

- The TGEC reaches a descriptor with TDES0<OW> clear. To resume, the port driver must give descriptor ownership to the TGEC and issue a Poll Demand command.
- The TDES1<FS> and TDES1<LS> are incorrectly paired or out of order. TDES0<LE> will be set.

## 17.5 Reception Process

- A frame transmission is given up due to a locally induced error. The appropriate TDES0 bit is set.

The transmission process enters the SUSPENDED state and sets CSR5<TC, TI>. Status information is written to the TDES0 word of the descriptor causing the suspension. The position in the transmit list, in all of the above cases, is retained. The retained position is that of the *descriptor following the last descriptor closed (set to host ownership) by the TGEC.*

---

### Note

---

The TGEC does not automatically poll the Tx descriptor list and the port driver must explicitly issue a Tx Poll Demand command after rectifying the suspension cause.

---

The following table summarizes the transmission process state transitions:

**Table 17–35 Transmission process state transitions**

From state	Event	To state	Action
STOPPED	Start Transmission command	RUNNING	Transmit polling begins from the last list position or from the head of the list if this is the first Start command issued, or if the transmit descriptor list address (CSR4) was modified by the port driver.
RUNNING	TGEC attempts acquisition of a descriptor owned by the host.	SUSPENDED	CSR5<TC, TI> is set. Position in list retained.
RUNNING	Out of order delimiting flag (TDES0<FS> or TDES0<LS>) encountered.	SUSPENDED	TDES0<LE> and CSR5<TC, TI> are set. Position in list retained.
RUNNING	Frame transmission aborts due to a locally induced error (refer to Table 17–28 for details).	SUSPENDED	Appropriate TDES0 and CSR5<TC, TI> bits are set. Position in list retained.
RUNNING	Stop Transmission command	STOPPED	Transmission process is STOPPED after the current frame, if any, is transmitted. Position in list retained.
RUNNING	Transmit watchdog expires	STOPPED	Transmission is cut off and CSR5<TW, TC, TI> , TDES0<TO> are set. Position in list retained.
RUNNING	Memory or host bus parity error encountered	STOPPED	Transmission is cut off and CSR5<ME> is set.
RUNNING	Reset command	STOPPED	Transmission is cut off.

(continued on next page)

## 17.5 Reception Process

Table 17–35 (Cont.) Transmission process state transitions

From state	Event	To state	Action
SUSPENDED	Tx Poll Demand command	RUNNING	Transmit polling resumes from last list position or from the list head if CSR4 was modified by the port driver.
SUSPENDED	Stop Transmission command	STOPPED	None.
SUSPENDED	Reset command	STOPPED	None.

### 17.5.2 Loopback Operations

The TGEC supports two loopback modes for test applications.

- Internal loopback

This mode is generally used to verify correct operations of the TGEC internal logic. While in this mode, the TGEC will take frames from the transmit list and loop them back, internally, to the receive list. The TGEC is disengaged from the Ethernet wire while in this mode.

- External loopback

This mode is generally used to verify correct operations up to the Ethernet cable. While in this mode, the TGEC will take frames from the transmit list and transmit them on the Ethernet wire. Concurrently, the TGEC listens to the line which carries its own transmissions and places incoming frames in the receive list.

---

**Note**

---

Caution should be exercised in this mode as transmitted frames are placed on the Ethernet wire. Furthermore, the TGEC does not check the origin of any incoming frames, consequently, frames not necessarily originating from the TGEC might make it to the receive buffers.

---

In either of these modes, all the address filtering and validity checking rules apply. The port driver needs to take the following actions:

1. Place the Reception and Transmission processes in the STOPPED state. The port driver must wait for any previously scheduled frame activity to cease. This is done by polling the TS and RS fields in CSR5.
2. Prepare appropriate transmit and receive descriptor lists in host memory. These may follow the existing lists at the point of suspension, or may be new lists which will have to be identified to the TGEC by appropriately writing CSR3 and CSR4.
3. Write to CSR6<OM> according to the desired loopback mode and place the Transmission and Reception processes in the RUNNING state through Start commands.
4. Respond and process any TGEC interrupts, as in normal processing.

To restore normal operations, the port driver must execute above step #1, then write the OM field in CSR6 with “00”.

### 17.5.3 DNA CSMA/CD Counters and Events Support

This section describes the TGEC features that support the port driver in implementing and reporting the specified counters and events. <sup>1</sup>

Table 17–36 lists the CSMA/CD counters.

**Table 17–36 CSMA/CD Counters**

Counter	TGEC Features
Time because counter creation	Supported by the host driver.
Bytes received	Port driver must add up the RDES0<FL> fields of all successfully received frames.
Bytes sent	Port driver must add up the TDES2<BS> fields of all successfully transmitted buffers.
Frames received	Port driver must count the successfully received frames in the receive descriptor list.
Frames sent	Port driver must count the successfully transmitted frames in the transmit descriptor list.
Multicast bytes received	Port driver must add up the RDES0<FL> fields of all successfully received frames with multicast address destinations.
Multicast frames received	Port driver must count the successfully received frames with multicast address destinations.
Frames sent, initially deferred	Port driver must count the successfully transmitted frames with TDES0<DE> set.
Frames sent, single collision	Port driver must count the successfully transmitted frames with TDES0<CC> equal to 1.
Frames sent, multiple collisions	Port driver must count the successfully transmitted frames with TDES0<CC> greater than 1.
Send failure (Excessive collisions)	Port driver must count the transmit descriptors having TDES0<EC> set.
Send failure ( Carrier check failed)	Port driver must count the transmit descriptors having TDES0<LC> set.
Send failure (Short circuit )	Two successive transmit descriptors with the No_carrier flag TDES0<NC> set, indicates a short circuit.
Send failure (Open circuit)	Two successive transmit descriptors with the Excessive_collisions flag TDES0<EC> set with the same Time domain reflectometer value TDES0<TDR>, indicate an open circuit.
Send failure (Remote Failure to Defer )	Flagged as a late collision TDES0<LC> in the transmit descriptors.
Receive failure (Block Check Error)	Port driver must count the receive descriptors having RDES0<CE> set with RDES0<DB> cleared.
Receive failure (Framing Error )	Port driver must count the receive descriptors having both RDES0<CE> and RDES0<DB> set.
Receive failure(Frame too long)	Port driver must count the receive descriptors having RDES0<TL> set.

(continued on next page)

<sup>1</sup> As specified in the DNA Maintenance Operations (MOP) Functional Specification, Version T.4.0.0, 28 January 1988.



## 17.5 Reception Process

**Table 17–36 (Cont.) CSMA/CD Counters**

<b>Counter</b>	<b>TGEC Features</b>
Unrecognized frame destination	Not applicable.
Data overrun	Port driver must count the receive descriptors having RDES0<OF> set.
System buffer unavailable	Reported in the Missed_frame counter CSR10<MFC> (refer to Section 17.1.13).
User buffer unavailable	not applicable.
Collision detect check failed	Port driver must count the transmit descriptors having TDES0<HF> set.

CSMA/CD specified events can be reported by the port driver based on Table 17–36. The initialization failed event is reported through CSR5<SF>.

---

## SCSI/DSSI Adapters

The I/O module contains five NCR 53C710 controllers, each capable of supporting DSSI, SCSI-1 or SCSI-2 protocols. Eight-bit SCSI/DSSI transfers are supported using single ended drivers and receivers. Controllers 3 through 0 provide true DSSI drivers and receivers and are connected by the DEC 4000 AXP enclosure to the fixed disk storage devices. Controller 4 provides only SCSI driver and receiver levels and is connected by the DEC 4000 AXP enclosure to the removable media devices. Each controller can be configured to support the following data transfer rates between the controller and the peripheral devices.

- Asynchronous SCSI -  $\leq 5$  MB/sec
- Synchronous SCSI
  - single ended -  $\leq 6.25$  MB/sec
- DSSI
  - single ended -  $\leq 6.25$  MB/sec

Each controller contains a 2-MIPS scripts processor that allows both DMA and SCSI/DSSI instructions to be fetched from a local scratch memory or from main memory. These instructions enable the device to select, reselect, disconnect, wait for disconnect, transfer information, change bus phases and in general, implement all aspects of the SCSI/DSSI protocol. Complex bus sequences are built by the DEC 4000 AXP processor and executed independently by the scripts processor. The DEC 4000 AXP processor is interrupted after successful completion or an error condition.

The controllers also allow low-level control of the SCSI/DSSI bus through register read and write operations by the DEC 4000 AXP processor. This gives the DEC 4000 AXP processor the ability to sample and/or assert signals on the SCSI/DSSI bus. The diagnostic capabilities of the controllers allow them to perform a self-selection so they can be tested as both an initiator and a target.

A description of the NCR 53C710 SCSI I/O processor controller registers and programming information follows.

### 18.1 NCR 53C710 SCSI Adapter Registers

SCSI adapter registers can be addresses as bytes or longwords only; other access sizes result in bus errors.

---

#### Warning

---

The only register that the host CPU can access while the 53C710 is executing scripts is the ISTAT register. Attempts to access other registers interferes with the operation of the chip. All registers are accessible via scripts.

---

## 18.1 NCR 53C710 SCSI Adapter Registers

### 18.1.1 SCSI Register Map

The register map in Figure 18–1 shows the register address assignments for a single SCSI/DSSI controller. There are five controllers included on the I/O module.

Bits <5:2> of the secondary bus address field in the mailbox data structure select one of sixteen longword registers within a SCSI/DSSI controller. Bits <8:6> of this field are used to select one of the five controllers according to the following mapping function shown in Table 18–1.

**Table 18–1 SCSI/DSSI Selection**

Addr [8:6]	Controller	Device Slot
0	SCSI(0)	Fixed Slot A
1	SCSI(1)	Fixed Slot B
2	SCSI(2)	Fixed Slot C
3	SCSI(3)	Fixed Slot D
4	SCSI(4)	Removable Media E
5	unused	
6	unused	
7	unused	

## 18.1 NCR 53C710 SCSI Adapter Registers

Figure 18–1 SCSI/DSSI Controller Register Map

31				0	ADDRESS
SIEN (r/w)	SDID (r/w)	SCNTL1(r/w)	SCNTL0(r/w)		SCSI(n)+00
SOCL (r/w)	SODL (r/w)	SXFER(r/w)	SCID (r/w)		SCSI(n)+04
SBCL (r/w)	SBDL (r)	SIDL (r)	SFBR (r)		SCSI(n)+08
SSTAT2 (r)	SSTAT1 (r)	SSTAT0 (r)	DSTAT (r)		SCSI(n)+0C
DSA (r/w)					SCSI(n)+10
CTEST3 (r)	CTEST2 (r)	CTEST1 (r)	CTEST0 (r/w)		SCSI(n)+14
CTEST7(r/w)	CTEST6(r/w)	CTEST5(r/w)	CTEST4(r/w)		SCSI(n)+18
TEMP (r/w)					SCSI(n)+1C
LCRC (r/w)	CTEST8(r/w)	ISTAT (r/w)	DFIFO (r/w)		SCSI(n)+20
DCMD (r/w)	DBC (r/w)				SCSI(n)+24
DNAD (r/w)					SCSI(n)+28
DSP (r/w)					SCSI(n)+2C
DSPS (r/w)					SCSI(n)+30
SCRATCH (r/w)					SCSI(n)+34
DCNTL (r/w)	DWT (r/w)	DIEN (r/w)	DMODE (r/w)		SCSI(n)+38
ADDER (r)					SCSI(n)+3C

SCSI\_REG

## 18.2 SCSI Control 0 (SCNTL0)

### 18.2 SCSI Control 0 (SCNTL0)

Figure 18–2 SCSI Control 0 (SCNTL0)

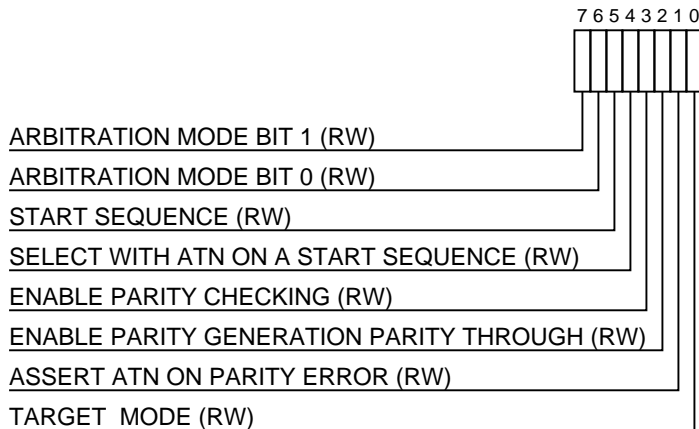


Table 18–2 SCSI Control 0 Description

Field	Description
7	<b>ARBITRATION MODE BIT 1</b> <i>[read/write]</i> (ARB1) Arbitration mode bit 1 Table 18–3 shows the interpretation of the arbitration mode bits.
6	<b>ARBITRATION MODE BIT 0</b> <i>[read/write]</i> (ARB0) Arbitration mode bit 0—See ARB1 text
5	<b>START SEQUENCE</b> <i>[read/write]</i> (START) When this bit is set, the 53C710 starts the arbitration sequence indicated by the Arbitration Mode bits. The Start Sequence bit is used in low level mode; when executing SCSI scripts, this bit is controlled by the scripts processor. An arbitration sequence should not be started if the connected bit in the SCNTL1 register indicates that 53C710 is already connected to the SCSI bus. This bit is automatically cleared when the arbitration sequence is complete. If a sequence is aborted, the connected bit in the SCNTL1 register should be checked to verify that the 53C710 did not connect to the SCSI bus.
4	<b>SELECT WITH ATN ON A START SEQUENCE</b> <i>[read/write]</i> (WATN) When this bit is set, the SCSI ATN signal is asserted during the selection phase (ATN is asserted at the same time BSY is deasserted while selecting a target). If a selection timeout occurs while attempting to select a target device, ATN is deasserted at the same time SEL is deasserted. When this bit is cleared, the ATN signal is not asserted during selection. When executing SCSI scripts, this bit is controlled by scripts processor, but it may be set manually in low level mode.
3	<b>ENABLE PARITY CHECKING</b> <i>[read/write]</i>

(continued on next page)

Table 18–2 (Cont.) SCSI Control 0 Description

Field	Description
	<p>(EPC) When this bit is set, the SCSI data bus is checked for odd parity when data is received from the SCSI bus in either initiator or target mode. The host data bus is checked for odd parity if bit 2, the Enable Parity Generation bit, is cleared. Host data bus parity is checked as data is loaded into the SODL register when sending SCSI data in either initiator or target mode. If a parity error is detected, bit 0 of the SSTAT0 register is set and an interrupt may be generated. If the 53C710 is operating in initiator mode and a parity error is detected, ATN can optionally be asserted, but the transfer continues until the target changes phase to Message Out.</p> <p>When this bit is cleared, parity errors are not reported.</p>
2	<p><b>ENABLE PARITY GENERATION PARITY THROUGH</b> <i>[read/write]</i></p> <p>(EPG) When this bit is set, the SCSI parity bit is generated by the 53C710. The host data bus parity lines DP&lt;3:0&gt; are ignored and should not be used as parity signals. When this bit is cleared, the parity present on the host data parity lines flows through the 53C710's internal FIFOs and is driven onto the SCSI bus when sending data (if the host bus is set to even parity, it is changed to odd before it is sent to the SCSI bus). This bit is set to enable the DP3_ABRT pin to function as an abort input (ABRT).</p>
1	<p><b>ASSERT ATN ON PARITY ERROR</b> <i>[read/write]</i></p> <p>(AAP) When this bit is set, the 53C710 automatically asserts the SCSI ATN signal upon detection of a parity error. ATN is asserted only in initiator mode. The ATN signal is asserted before deasserting ACK during the byte transfer with the parity error. The Enable Parity Checking bit must also be set for the 53C710 to assert ATN in this manner. The following parity errors can occur.</p> <ol style="list-style-type: none"> <li>1. A parity error detected on data received from the SCSI bus.</li> <li>2. A parity error detected on data transferred to the 53C710 from the host data bus.</li> </ol> <p>If the Assert ATN on Parity Error bit is cleared or the Enable Parity Checking bit is cleared, ATN is not asserted automatically on the SCSI bus when a parity error is received.</p>
0	<p><b>TARGET MODE</b> <i>[read/write]</i></p> <p>(TRG) This bit determines the default operating mode of the 53C710, though there are instances when the chip may act in a role other than the default. For example, a mostly initiator device may be selected as a target. An automatic mode change does affect the state of this bit. After completion of a mode change I/O operation, the 53C710 returns to the role defined by this bit.</p> <p>When this bit is set, the chip is a target device by default. When the target mode bit is cleared, the 53C710 is an initiator device by default.</p>

### Simple Arbitration

1. The 53C710 waits for a bus free condition to occur.
2. It asserts BSY and its SCSI ID (contained in the SCID register) onto the SCSI bus. If the SEL/signal is asserted by another SCSI device, the 53C710 deasserts BSY, deasserts its ID and sets the Lost Arbitration bit in the SSTATI register.
3. After an arbitration delay, the CPU should read the SBDL register to check if a higher priority SCSI ID is present. If no higher priority ID bit is set, and the LOST ARBITRATION bit is not set, the 53C710 has won arbitration.

## 18.2 SCSI Control 0 (SCNTL0)

4. Once the 53710 has won arbitration, SEL must be asserted via the SOCL for a bus clear plus a bus settle delay (1.2  $\mu$ S) before a low level selection can be performed.

### Full Arbitration, Selection and Reselection

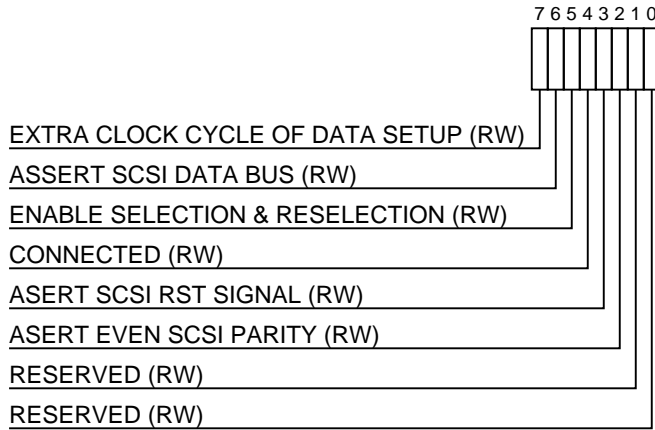
1. The 53C710 waits for a bus free condition.
2. It asserts BSY and its SCSI ID (the highest priority ID stored in the SCID register) onto the SCSI bus.
3. If the SEL signal is asserted by another SCSI device or if the 53C710 detects a higher priority ID, the 53C710 deasserts BSY deasserts its ID, and waits until the next bus free state to try arbitration again.
4. The 53C710 repeats arbitration until it wins control of the SCSI bus. When it has won, the Won Arbitration bit is set in the SSTATI register.
5. The 53C710 performs selection by asserting the following onto the SCSI bus SEL, the target's ID (stored in the SDID register) and the 53C710's ID (the highest priority ID stored in the SCID register)
6. After a selection is complete, the Function Complete bit is set in the SSTAT0 register.
7. If a selection timeout occurs, the SELECTION TIMEOUT BIT is set in the SSTAT0 register.

**Table 18–3 Arbitration Mode Bit Interpretations**

Arb1	Arb0	Arbitration Mode
0	0	Simple Arbitration
0	1	Reserved
1	0	Reserved
1	1	Full arbitration, selection, or reselection

## 18.3 SCSI Control 1 (SCNTL1)

**Figure 18–3 SCSI Control 1 (SCNTL1)**



**Table 18–4 SCSI Control 1 Description**

Field	Description
<b>7</b>	<p><b>EXTRA CLOCK CYCLE OF DATA SETUP</b> <i>[read/write]</i></p> <p>(EXC) When this bit is set, an extra clock period of data setup is added to each SCSI data transfer. The extra data setup time can provide additional system design flexibility, though it affects the SCSI transfer rates. Clearing this bit disables the extra clock cycle of data setup time.</p>
<b>6</b>	<p><b>ASSERT SCSI DATA BUS</b> <i>[read/write]</i></p> <p>(ADB) When this bit is set, the 53C710 drives the contents of the SCSI Output Data Latch (SODL) register onto the SCSI data bus. When the 53C710 is initiator, the SCSI I/O signal must be inactive to assert the SODL contents onto the SCSI bus. When the 53C710 is a target, the SCSI I/O signal must be active for the SODL contents to be asserted onto the SCSI bus. The contents of the SODL register can be asserted at any time, even before the 53C710 is connected to the SCSI bus. This bit should be cleared when executing SCSI scripts. It is normally used only for diagnostics testing or operation in low level mode.</p>
<b>5</b>	<p><b>ENABLE SELECTION &amp; RESELECTION</b> <i>[read/write]</i></p> <p>(ESR) When this bit is set, the 53C710 responds to bus initiated selections and reselections. The 53C710 can respond to selections and reselections in both initiator and target roles. If SCSI Disconnect/Reconnect is to be supported, this bit should be set as part of the initialization routine.</p>
<b>4</b>	<p><b>CONNECTED</b> <i>[read/write]</i></p> <p>(CON) This bit is automatically set any time the 53C710 is connected to the SCSI bus as an initiator or as a target. It is set after successfully completing arbitration or when the 53C710 has responded to a bus initiated selection or reselection. It also sets after successfully completing simple arbitration when operating in low level mode. When this bit is clear, the 53C710 is not connected to the SCSI bus.</p> <p>The CPU can force a connected or disconnected condition by setting or clearing this bit. This feature would be used primarily during loopback mode.</p>
<b>3</b>	<p><b>ASSERT SCSI RST SIGNAL</b> <i>[read/write]</i></p>

(continued on next page)



## 18.3 SCSI Control 1 (SCNTL1)

Table 18–4 (Cont.) SCSI Control 1 Description

Field	Description
	(RST) Setting this bit asserts the SCSI RST signal. The RST signal remains asserted until this bit is cleared. The 25 $\mu$ s minimum assertion time defined in the SCSI specification must be timed out by the controlling microprocessor or a script delay routine.
2	<b>ASERT EVEN SCSI PARITY</b> <i>[read/write]</i> (AESP) When this bit is set and the Enable Parity Generation bit is set in the SCNTL0 register, the 53C710 asserts even parity. It forces a SCSI parity error on each byte sent to the SCSI bus from the 53C710. If parity checking is enabled, then the 53C710 checks data received for odd parity. This bit is used for diagnostic testing and should be clear for normal operation. It can be used to generate parity errors to test error handling functions.
1	<b>RESERVED</b> <i>[read/write]</i>
0	<b>RESERVED</b> <i>[read/write]</i>

## 18.4 SCSI Destination ID (SDID)

This register sets the SCSI ID of the device to be selected when executing a select or reselect command. When executing SCSI scripts, the script processor writes the destination SCSI ID to this register. The SCSI ID is defined by the user in a SCSI scripts select or reselect instruction.

When using Table Indirect I/O commands, the destination ID is loaded from the data structure.

Figure 18–4 SCSI Destination ID (SDID)

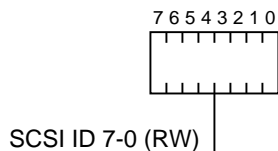


Table 18–5 SCSI Destination ID Description

Field	Description
7:0	<b>SCSI ID 7-0</b> <i>[read/write]</i> (ID7—ID0) Only one of these bits should be set for proper selection or reselection.

## 18.5 SCSI Interrupt Enable (SIEN)

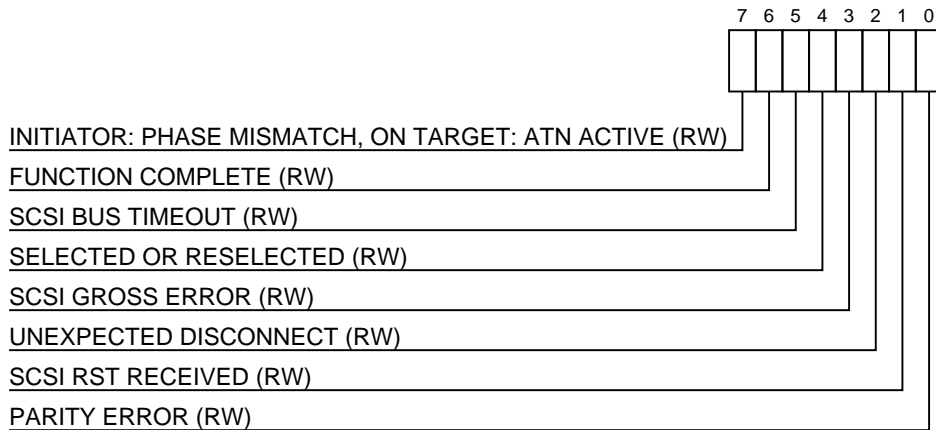
This register contains the interrupt mask bits corresponding to the interrupting conditions described in the SSTAT0 register. An interrupt is masked by clearing the appropriate mask bit. Masking an interrupt prevents IRQ from being asserted for the corresponding interrupt, but the status bit is still set in the SSTAT0 register. Masking an interrupt does not prevent the ISTAT SIP bit from being set, except in the case of nonfatal interrupts (SEL and FCMP). Setting a mask bit un masks the corresponding interrupt, enabling the assertion of IRQ for that interrupt.

## 18.5 SCSI Interrupt Enable (SIEN)

A masked nonfatal interrupt does not prevent unmasked or fatal interrupts from getting through; interrupt stacking does not begin until either the ISTAT SIP or DIP bit is set.

The 53C710 IRQ output is latched; once asserted, it remains asserted until the interrupt is cleared by reading the appropriate status register. Masking an interrupt after the IRQ output is asserted does not cause IRQ to be deasserted. In the case of nonfatal interrupts, masking an interrupt after it occurs causes the ISTAT SIP bit to clear allowing pending interrupts to fall through. (Interrupt stacking is disabled).

**Figure 18–5 SCSI Interrupt Enable (SIEN)**



**Table 18–6 SCSI Interrupt Enable Description**

Field	Description
7	<b>INITIATOR: PHASE MISMATCH, ON TARGET: ATN ACTIVE</b> <i>[read/write]</i>
6	<b>FUNCTION COMPLETE</b> <i>[read/write]</i>
5	<b>SCSI BUS TIMEOUT</b> <i>[read/write]</i>
4	<b>SELECTED OR RESELECTED</b> <i>[read/write]</i>
3	<b>SCSI GROSS ERROR</b> <i>[read/write]</i>
2	<b>UNEXPECTED DISCONNECT</b> <i>[read/write]</i>
1	<b>SCSI RST RECEIVED</b> <i>[read/write]</i>
0	<b>PARITY ERROR</b> <i>[read/write]</i>

## 18.6 SCSI Chip ID (SCID)

This register sets up the 53C710's SCSI ID. If more than one bit is set, the 53C710 responds to each corresponding SCSI ID. The 53C710 always uses the highest priority SCSI ID during arbitration. For example, if 84<sub>16</sub> were written to this register, the 53C710 would respond when another device selects ID 7 or ID 2. When arbitrating for SCSI bus, ID 7 would be used as the 53C710's SCSI ID.

## 18.6 SCSI Chip ID (SCID)

Figure 18–6 SCSI Chip ID (SCID)

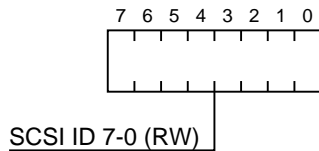


Table 18–7 SCSI Chip ID Description

Field	Description
7:0	SCSI ID 7-0 [read/write]

## 18.7 SCSI Transfer (SXFER)

When using Table Indirect I/O commands, bits <6:0> of this register are loaded from the I/O data structure.

Figure 18–7 SCSI Transfer (SXFER)

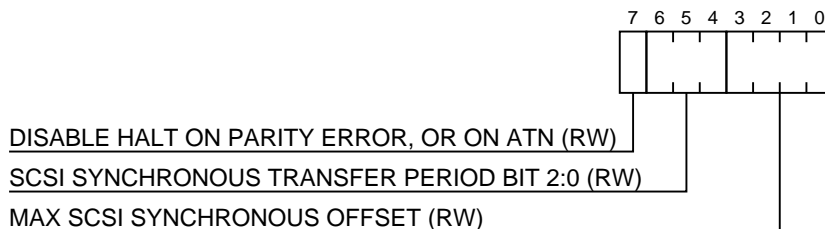


Table 18–8 SCSI Transfer Description

Field	Description
7	<p><b>DISABLE HALT ON PARITY ERROR, OR ON ATN</b> [read/write]</p> <p>(DHP) When this bit cleared, the 53C710 immediately halts the SCSI data transfer when a parity error is detected or when the ATN signal is asserted. If ATN or a parity error is received in the middle of data transfer, the 53C710 may transfer up to 3 additional bytes before halting to synchronize between internal core cells. During synchronous operation, the 53C710 transfers data until there are no outstanding synchronous offsets. If the 53C710 is receiving data, any data residing in the SCSI or DMA FIFOs is sent to memory before halting. While sending data in target mode with pass parity enabled, the byte with the parity error is not sent across the SCSI bus.</p> <p>When this bit is set, the 53C710 does not halt the SCSI transfer when ATN or a parity error is received until the end of a block move operation. When this bit is set and the initiator asserts ATN, the 53C710 completes the block move and then, depending on whether or not the ATN interrupt is enabled either generates an interrupt or continues fetching instructions (the instruction following should be a move, address IF ATN.)</p>

(continued on next page)

Table 18–8 (Cont.) SCSI Transfer Description

Field	Description
-------	-------------

**6:4 SCSI SYNCHRONOUS TRANSFER PERIOD BIT 2:0** *[read/write]*

(TP2—TP0) These bits determine the SCSI synchronous transfer period used by the 53C710 when sending synchronous SCSI data in either initiator or target mode. The table below describes the possible combinations and their relationship to the synchronous data transfer period used by the 53C710.

TP2	TP1	TP0	XFERP
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

The actual synchronous transfer period used by the 53C710 when transferring SCSI data is defined by the following equations.

- The minimum synchronous transfer period when sending SCSI data:

$$\text{Period} = \text{TCP} * (4 + \text{XFERP} + 1)$$

If Bit <7> in the SCNTL1 register is set

(one extra clock cycle of data setup)

$$\text{Period} = \text{TCP} * (4 + \text{XFERP})$$

If Bit <7> in the SCNTL1 register is clear

(no extra clock cycle of data setup)

- The minimum synchronous transfer period when receiving SCSI data:

$$\text{Period} = \text{TCP} * (4 + \text{XFERP})$$

Whether sending or receiving,  $\text{TCP} = 1 / \text{SCSI core clock frequency}$ . The SCSI core clock frequency is determined by the CF<1:0> bits in the DCNTL register and SSCF<1:0> bits in SBCL. The following table shows examples of synchronous transfer periods for SCSI-1 transfer rates. Table 18–9 and Table 18–10 shows example transfer periods for SCSI-1 and fast SCSI-2 transfer rates.

**3:0 MAX SCSI SYNCHRONOUS OFFSET** *[read/write]*

(MO3—MO0) These bits describe the maximum SCSI synchronous offset used by the 53C710 when transferring synchronous SCSI data in either initiator or target mode. Table 18–11 describes the possible combinations and their relationship to the synchronous data offset used by the 53C710. These bits determine the 53C710 method of transfer for Data In and Data Out phases only; all other information transfers occur asynchronously.

## 18.7 SCSI Transfer (SXFER)

**Table 18–9**

Clk (MHz)	SCSI Clk +DCNTL Bits 7,6	XFERP	Sync Transfer Period (ns)	Sync Transfer Rate (MB/s)
66.67	+3	0	180	5.55
66.67	+3	1	225	4.44
50	+2	0	160	6.25
50	+2	1	200	5
40	+2	0	200	5
37.50	+1.5	0	160	6.25
33.33	+1.5	0	180	5.55
25	+1	0	160	6.25
20	+1	0	200	5
16.67	+1	0	240	4.17

**Table 18–10**

Clk (MHz)	SCSI Clk +SBCL Bits 1,0	XFERP	Sync Transfer Period (ns)	Sync Transfer Rate (MB/s)
66.67	+15	0	90	11.11*
66.67	+15	1	1125	8.88
50	+1	0	80	12.5*
50	+1	1	100	10.0
40	+1	0	100	10.0
37.50	+1	0	106.67	9.375
33	+1	0	120	8.33
25	+1	0	160	6.25
20	+1	0	200	5
16.67	+1	0	240	4.17

**Table 18–11**

MO3	MO2	MO1	MO0	Synchronous Offset
0	0	0	0	0 Asynchronous operation

(continued on next page)

Table 18–11 (Cont.)

MO3	MO2	MO1	MO0	Synchronous Offset
0	0	0	1	1
0	0	1	1	3
0	0	1	1	3
0	1	1	1	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	X	X	1	Reserved
1	X	1	X	Reserved
1	1	X	X	Reserved

(\* Violates SCSI specifications)

## 18.8 SCSI Output Data Latch (SODL)

This register is used primarily for diagnostics testing or programmed I/O operations. Data written to this register is asserted onto the SCSI data bus by setting the Assert Data Bus bit in the SCNTL1 register. This register is used to send data via programmed I/O. Data flows through this register when sending data in any mode. It is also used to write to the synchronous data FIFO when testing the chip.

Figure 18–8 SCSI Output Data Latch ( SODL)

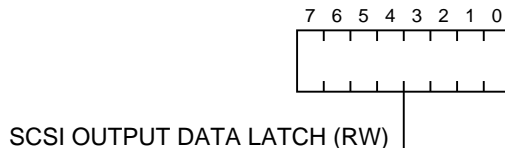


Table 18–12 SCSI Output Data Latch Description

Field	Description
7:0	SCSI OUTPUT DATA LATCH <i>[read/write]</i>

## 18.9 SCSI Output Control Latch (SOCL)

This register is used primarily for diagnostics testing or programmed I/O operation. It is controlled by the scripts processor when executing SCSI scripts. SOCL should be used only when transferring data via programmed I/O. Some bits are set or reset when executing SCSI scripts. Do not write to the register once the 53C710 becomes connected and starts executing SCSI scripts.

## 18.9 SCSI Output Control Latch (SOCL)

Figure 18–9 SCSI Output Control Latch (SOCL)

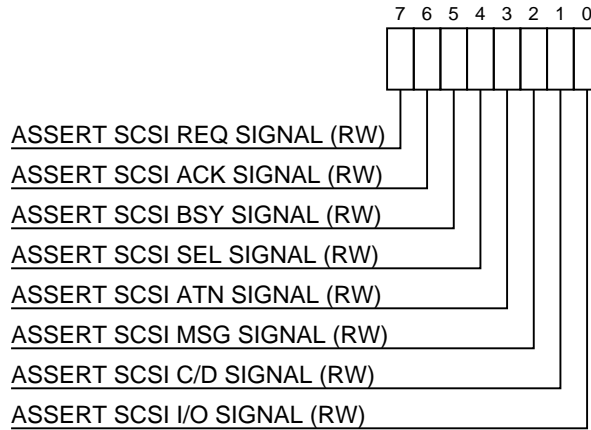


Table 18–13 SCSI Output Control Latch Description

Field	Description
7	<b>ASSERT SCSI REQ SIGNAL</b> <i>[read/write]</i>
6	<b>ASSERT SCSI ACK SIGNAL</b> <i>[read/write]</i>
5	<b>ASSERT SCSI BSY SIGNAL</b> <i>[read/write]</i>
4	<b>ASSERT SCSI SEL SIGNAL</b> <i>[read/write]</i>
3	<b>ASSERT SCSI ATN SIGNAL</b> <i>[read/write]</i>
2	<b>ASSERT SCSI MSG SIGNAL</b> <i>[read/write]</i>
1	<b>ASSERT SCSI C/D SIGNAL</b> <i>[read/write]</i>
0	<b>ASSERT SCSI I/O SIGNAL</b> <i>[read/write]</i>

## 18.10 SCSI First Byte Received (SFBR)

This register contains the first byte received in any asynchronous information transfer phase. For example, when the 53C710 is operating in initiator mode, this register contains the first byte received in Message In, Status Phase, Reserved In and Data In.

When a Block Move Instruction is executed for a particular phase, the first byte received is stored in this register even if the present phase is the same as the last phase. The first byte-received value for a particular input phase is not valid until after a MOVE instruction is executed.

This register is also the accumulator for register read-modify-writes with the SFBR as the destination allowin gbit testing after an operation.

Additionally, the SFBR register may be used to contain the device ID after a selection or reselection, if the COM bit is clear in the DCNTL register. However, for maximum flexibility it is strongly recommended that the ID byte be directed only to the LCRC register (COM bit set).

## 18.10 SCSI First Byte Received (SFBR)

The CPU cannot write the SFBR. Therefore SFBR cannot be written by a Memory Move instruction. However, it can be loaded via read/write operations. To load the SFBR with a byte stored in system memory, the byte must first be moved to an intermediate 53C710 register (such as the SCRATCH register), and then to the SFBR.

Figure 18–10 SCSI First Byte Received ( SFBR)

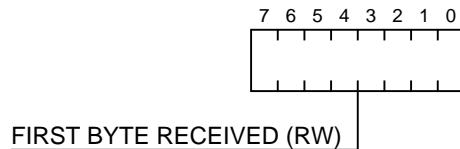


Table 18–14 SCSI First Byte Received Description

Field	Description
7:0	<b>FIRST BYTE RECEIVED</b> <i>[read/write]</i>

## 18.11 SCSI Input Data Latch (SIDL)

This register is used primarily for diagnostics testing, programmed I/O operation or error recovery. Data received from the SCSI bus can be read from this register. Data can be written to the SIDL register and then read back into the 53C710 by reading this register to provide loopback testing. When receiving SCSI data, the data flows into the register and out to the host FIFO. This register differs from the SBDL register in that this register contains latched data and the SBDL always contains exactly what is currently on the SCSI data bus. Reading this register causes the SCSI parity bit to be checked, and causes a parity error interrupt if the data is not valid.

Figure 18–11 SCSI Input Data Latch (SIDL)

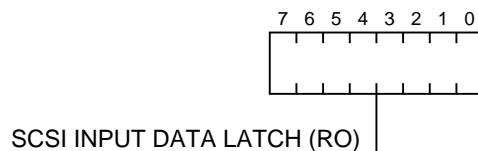


Table 18–15 SCSI Input Data Latch Description

Field	Description
7:0	<b>SCSI INPUT DATA LATCH</b> <i>[read-only]</i>



## 18.12 SCSI Bus Data Lines (SBDL)

### 18.12 SCSI Bus Data Lines (SBDL)

This register contains the SCSI data bus status. Even though the SCSI data bus is active low, these bits are active high. The signal status is not latched and is a true representation of exactly what is on the data bus at the time that the register is read. This register is used when receiving data via programmed I/O. This register can also be used for diagnostics testing or a low level mode.

Figure 18–12 SCSI Bus Data Lines (SBDL)

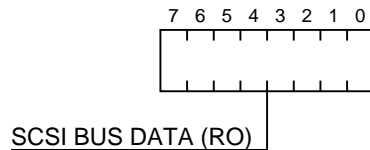


Table 18–16 SCSI Bus Data Lines Description

Field	Description
7:0	SCSI BUS DATA <i>[read-only]</i>

### 18.13 SCSI Bus Control Lines (SBCL)

When read, this register returns the SCSI control line status. A bit is set when the corresponding SCSI control line is asserted. These bits are not latched; they are a true representation of what is on the SCSI bus at the time the register is read. This register can be used for diagnostics testing or operation in low level mode. Writing to bits <7:2> has no effect. Table 18–17 explains the control bit interpretations.

Table 18–17 Bits <1:0> SSCF1-SSCFO (Synchronous SCSI Clock Control bits)

SSCF1	SSCf0	Synchronous CLK
0	0	Set by DCNTL
0	1	SCLK/1.0
1	0	SCLK/1.5
1	1	SCLK/2.0

When written, these bits determine the clock prescale factor used by the synchronous portion of the SCSI core. The default is to use the same clock prescale factor as the asynchronous logic (set by CF<1:0> in DCNTL). Setting one or both of these bits allows the synchronous logic to run at a different speed than the asynchronous logic; this is necessary for fast SCSI-2.

## 18.13 SCSI Bus Control Lines (SBCL)

Figure 18–13 SCSI Bus Control Lines (SBCL)

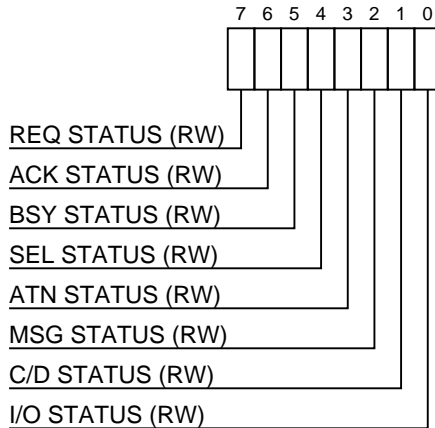


Table 18–18 SCSI Bus Control Lines Description

Field	Description
7	<b>REQ STATUS</b> <i>[read/write]</i>
6	<b>ACK STATUS</b> <i>[read/write]</i>
5	<b>BSY STATUS</b> <i>[read/write]</i>
4	<b>SEL STATUS</b> <i>[read/write]</i>
3	<b>ATN STATUS</b> <i>[read/write]</i>
2	<b>MSG STATUS</b> <i>[read/write]</i>
1	<b>C/D STATUS</b> <i>[read/write]</i>
0	<b>I/O STATUS</b> <i>[read/write]</i>

## 18.14 DMA Status (DSTAT)

Reading this register clears any bits that are set at the time the register is read, but does not necessarily clear the register because additional interrupts may be pending (the 53C710 stacks interrupts). DMA interrupt conditions may be individually masked through the DIEN register.

When performing consecutive 8-bit reads of both the DSTAT and SSTAT0 registers (in either order), insert a delay equivalent to 12 BCLK periods between the reads to ensure the interrupts clear properly. Also, if reading both registers when both the ISTAT SIP and DIP bits may not be set, the SSTAT0 register should be read before the DSTAT register to avoid missing a SCSI interrupt. Both concerns are avoided if the registers are read together as a 32-bit longword.

## 18.14 DMA Status (DSTAT)

Figure 18–14 DMA Status (DSTAT)

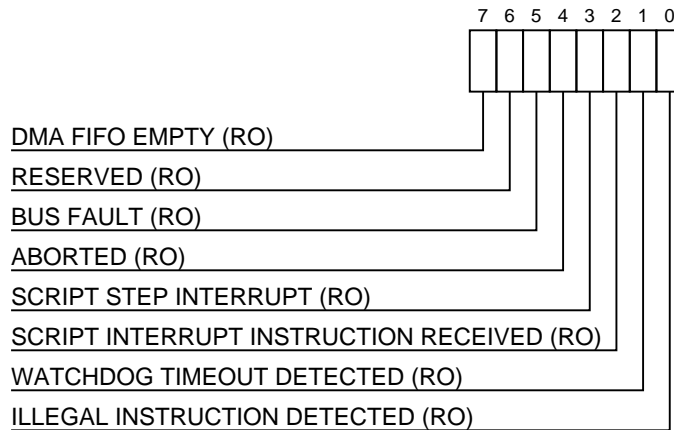


Table 18–19 DMA Status Description

Field	Description
7	<b>DMA FIFO EMPTY</b> <i>[read-only]</i> (DFE) This status bit is set when the DMA FIFO is empty. This bit may be changing at the time this register is read. It may be used to determine if any data resides in the FIFO when an error occurs and an interrupt is generated. This bit is a pure status bit and does not cause an interrupt.
6	<b>RESERVED</b> <i>[read-only]</i>
5	<b>BUS FAULT</b> <i>[read-only]</i> (BF) This bit is set when a host bus fault condition is detected. A host bus fault occurs when the 53C710 is bus master, and is defined as a memory cycle that is ended by the assertion of BERR (without HALT) or TEA (without TA ). A bus fault also occurs if RETRY is attempted after the first transfer of a cache-line burst.
4	<b>ABORTED</b> <i>[read-only]</i> (ABRT) This bit is set when an abort condition occurs. An abort condition occurs because of the following; the DP3-ABRT input signal is asserted by another device (parity generation mode) or a software abort command is issued by setting bit <7> of the ISTAT register.
3	<b>SCRIPT STEP INTERRUPT</b> <i>[read-only]</i> (SSI) If the Single-Step Mode bit in the DCNTL register is set, this bit is set and an interrupt is generated after executing each scripts instruction.
2	<b>SCRIPT INTERRUPT INSTRUCTION RECEIVED</b> <i>[read-only]</i> (SIR ) This status bit is set whenever an interrupt instruction is evaluated as true.
1	<b>WATCHDOG TIMEOUT DETECTED</b> <i>[read-only]</i> (WDT) This status bit is set when the watchdog timer decrements to zero. The watchdog timer is used only for the host memory interface. When the timer decrements to zero, it indicates that the memory system did not assert the acknowledge signal within the specified timeout period.

(continued on next page)

Table 18–19 (Cont.) DMA Status Description

Field	Description
0	<b>ILLEGAL INSTRUCTION DETECTED</b> <i>[read-only]</i> (IID) This status bit is set any time an illegal instruction is decoded, whether the 53C710 is operating in single-step mode or automatically executing SCSI scripts. This bit is also set if the 53C710 is executing a Wait Disconnect instruction and the SCSI REQ line is asserted without a disconnect occurring.

### 18.15 SCSI Status Zero (SSTAT0)

Reading this register clears any bits that are set at the time the register is read, but does not necessarily clear the register because additional interrupts may be pending (the 53C710 stacks interrupts). SCSI interrupt conditions may be individually masked through the SIEN register.

When performing consecutive 8-bit reads of both the DSTAT and SSTAT0 registers (in either order), insert a delay equivalent to 12 BCLK periods between the reads to ensure the interrupts clear properly. Also, if reading both registers when both the ISTAT SIP and DIP bits may not be set, the SSTAT0 register should be read before the DSTAT register to avoid missing a SCSI interrupt. Both concerns are avoided if the registers are read together as a 32-bit longword.

Figure 18–15 SCSI Status 0 (SSTAT0)

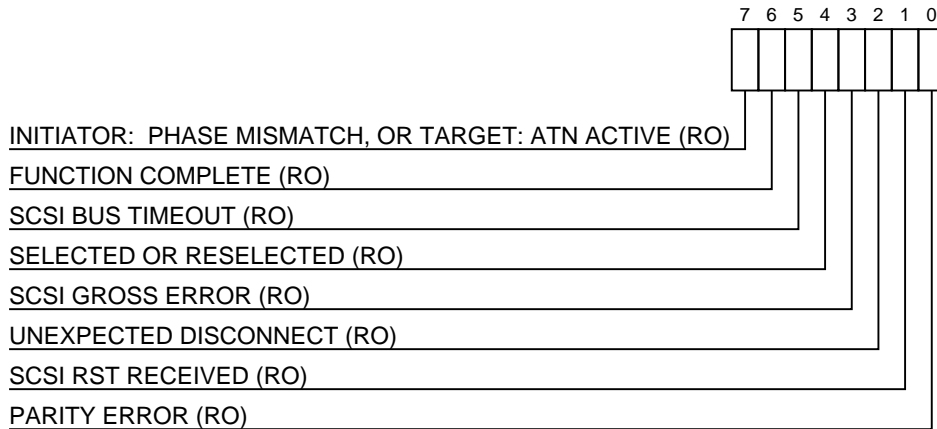


Table 18–20 SCSI Status 0 Description

Field	Description
7	<b>INITIATOR: PHASE MISMATCH, OR TARGET: ATN ACTIVE</b> <i>[read-only]</i> (M/A) In initiator mode, this bit is set if the SCSI phase asserted by the target does not match the SCSI phase defined in a Block Move instruction. The phase is sampled when REQ is asserted by the target. In target mode, this bit is set when the ATN signal is asserted by the initiator.

(continued on next page)

## 18.15 SCSI Status Zero (SSTAT0)

Table 18–20 (Cont.) SCSI Status 0 Description

Field	Description
<b>6</b>	<b>FUNCTION COMPLETE</b> <i>[read-only]</i> (FCMP) This bit is set when an arbitration-only or full-arbitration sequence has completed.
<b>5</b>	<b>SCSI BUS TIMEOUT</b> <i>[read-only]</i> (STO) This bit is set if one of the following conditions occur. <ol style="list-style-type: none"><li>1. There is a selection or reselection timeout. A selection/reselection timeout occurs if the device being selected or reselected does not respond within the 250 ms timeout period.</li><li>2. The Wait for Disconnect takes longer than 250 ms. The Wait for Disconnect instruction has a bus activity timer that is reset by the physical disconnect.</li><li>3. The Wait for Disconnect instruction has a bus activity timer that is reset by the physical disconnect.</li><li>4. No SCSI activity occurs for 250 ms while the 53C710 is connected to the bus. There is a timer on all bytes (in all phases) sent or received on the SCSI bus. The timer is a bus activity timer that is reset by a byte going over the SCSI bus. If 250 ms pass without a byte being moved, then a timeout occurs.</li></ol>
<b>4</b>	<b>SELECTED OR RESELECTED</b> <i>[read-only]</i> (SEL) This bit is set when the 53C710 is selected or reselected by another SCSI device. The Enable Selection and Reselection bit must be set in the SCNTL1 register for the 53C710 to respond to selection and reselection attempts.
<b>3</b>	<b>SCSI GROSS ERROR</b> <i>[read-only]</i> (SGE) This bit is set when the 53C710 encounters a SCSI Gross Error condition. The following conditions can cause a SCSI gross error condition. <ol style="list-style-type: none"><li>1. Data underflow: The SCSI FIFO register was read when no data was present.</li><li>2. Data overflow: Too many bytes were written to the SCSI FIFO or the synchronous offset caused the SCSI FIFO to be overwritten.</li><li>3. Offset underflow: When the 53C710 is operating in target mode and an ACK pulse is received when the outstanding offset is zero.</li><li>4. Offset overflow: The other SCSI device sent a REQ or ACK pulse with data that exceeded the maximum synchronous offset defined by the SXFER register.</li><li>5. Residual data in the synchronous data FIFO - A transfer other than synchronous data received was started with data left in the synchronous data FIFO.</li><li>6. A phase change occurred with an outstanding synchronous offset when the 53C710 was operating as an initiator.</li></ol>
<b>2</b>	<b>UNEXPECTED DISCONNECT</b> <i>[read-only]</i>

(continued on next page)

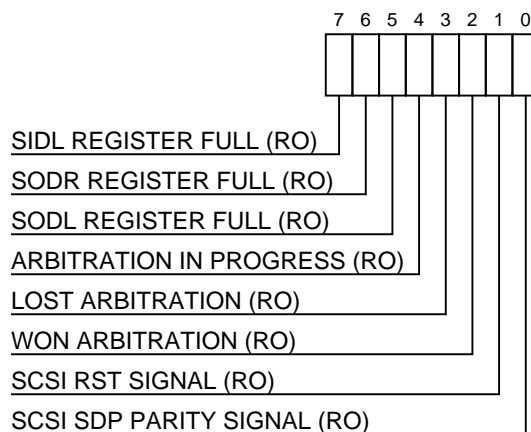
## 18.15 SCSI Status Zero (SSTAT0)

Table 18–20 (Cont.) SCSI Status 0 Description

Field	Description
	(UDC) This bit is valid only when the 53C710 is in initiator mode. It is set when the 53C710 is operating in initiator mode and the target device unexpectedly disconnects from the SCSI bus. When the 53C710 is executing SCSI scripts, an unexpected disconnect is defined to be a disconnect that does not occur after receiving either a Disconnect Message (04 <sub>16</sub> ) or a Command Complete Message (00 <sub>16</sub> ). When the 53C710 operates in low-level mode, any disconnect can cause an interrupt, even a valid SCSI disconnect.
<b>1</b>	<b>SCSI RST RECEIVED</b> <i>[read-only]</i>  (RST) This bit is set when the 53C710 detects an active RST signal, whether the reset was generated outside the chip or caused by the Assert RST bit in the SCNTL1 register. The 53C710 SCSI reset detection logic is edge-sensitive so that multiple interrupts are not generated for a single assertion of the SCSI RST signal.
<b>0</b>	<b>PARITY ERROR</b> <i>[read-only]</i>  (PAR) This bit is set when the 53C710 detects a parity error while sending or receiving SCSI data. The Enable Parity Checking bit (bit <3> in the SCNTL0 register) must be set for this bit to become active. A parity error can occur when receiving data from the SCSI bus or when receiving data from the host bus. From the host bus, parity is checked as it is transferred from the DMA FIFO to the SODL register. A parity error can occur from the host bus only if pass-through parity is enabled (SCNTL0<3>=1, and SCNTL0<2>=0).

## 18.16 SCSI Status One (SSTAT1)

Figure 18–16 SCSI Status One (SSTAT1)



## 18.16 SCSI Status One (SSTAT1)

Table 18–21 SCSI Status One Description

Field	Description
7	<b>SIDL REGISTER FULL</b> <i>[read-only]</i> (IFL) This bit is set when the SCSI Input Data Latch register (SIDL) contains data. Data is transferred from the SCSI bus to the SCSI Input Data Latch register before being sent to the DMA FIFO and then to the host bus. The SIDL register contains SCSI data received asynchronously. Synchronous data received does not flow through this register.
6	<b>SODR REGISTER FULL</b> <i>[read-only]</i> (ORF) This bit is set when the SCSI Output Data Register (SODR), a hidden buffer register which is not directly accessible, contains data. The SODR register is used by the SCSI logic as a second storage register when sending data synchronously. It is not accessible to the user (cannot be read or written). This bit can be used to determine how many bytes reside in the chip when an error occurs.
5	<b>SODL REGISTER FULL</b> <i>[read-only]</i> (OLF) This bit is set when the SCSI Output Data Latch (SODL) contains data. The SODL register is the interface between the DMA logic and the SCSI bus. In synchronous mode, data is transferred from the host bus to the SCSI Output Data Register (SODR, a hidden buffer register which is not accessible), and then to the SODL register before being sent to the SCSI bus. In asynchronous mode, data is transferred from the host bus to the SODL register, and then to the SCSI bus. The SODR buffer register is not used for asynchronous transfers. This bit can be used to determine how many bytes reside in the chip when an error occurs.
4	<b>ARBITRATION IN PROGRESS</b> <i>[read-only]</i> (AIP) Arbitration in Progress (AIP=1) indicates that the 53C710 has detected a bus free condition, asserted BSY and asserted its SCSI ID onto the SCSI bus.
3	<b>LOST ARBITRATION</b> <i>[read-only]</i> (LOA) When set, LOA indicates that the 53C710 detected a bus free condition, arbitrated for the SCSI bus, and lost arbitration because another SCSI device was asserting the SEL signal.
2	<b>WON ARBITRATION</b> <i>[read-only]</i> (WOA) When set, WOA indicates that the 53C710 detected a bus free condition, arbitrated for the SCSI bus, and won arbitration. The arbitration mode selected in the SCNTL0 register must be full arbitration and selection for this bit to be set.
1	<b>SCSI RST SIGNAL</b> <i>[read-only]</i> (RST) This bit represents the current status of the SCSI RST signal. This signal is not latched and may be changing when read.
0	<b>SCSI SDP PARITY SIGNAL</b> <i>[read-only]</i> (SDP) This bit represents the current status of the SCSI SDP parity signal. This signal is not latched and may be changing when read.

## 18.17 SCSI Status Two (SSTAT2)

Figure 18–17 SCSI Status Two (SSTAT2)

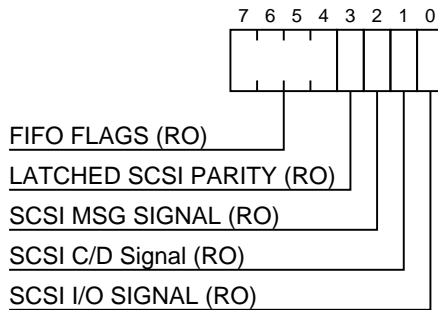


Table 18–22 SCSI Status Two Description

Field	Description																																																		
<b>7:4</b>	<b>FIFO FLAGS</b> <i>[read-only]</i> (FF3—FF0) These four bits define the number of bytes that currently reside in the 53C710's SCSI synchronous data FIFO. These bits are not latched and they change as data moves through the FIFO. Because the FIFO is only 8 bytes deep, values over 8 do not occur.																																																		
	<table border="1"> <thead> <tr> <th>FF3</th> <th>FF2</th> <th>FF1</th> <th>FF0</th> <th>Bytes in the SCSI FIFO</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td>2</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>3</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td>4</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>1</td><td>5</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td>6</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>1</td><td>7</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td>8</td></tr> </tbody> </table>	FF3	FF2	FF1	FF0	Bytes in the SCSI FIFO	0	0	0	0	0	0	0	0	1	1	0	0	1	0	2	0	0	1	1	3	0	1	0	0	4	0	1	0	1	5	0	1	1	0	6	0	1	1	1	7	1	0	0	0	8
FF3	FF2	FF1	FF0	Bytes in the SCSI FIFO																																															
0	0	0	0	0																																															
0	0	0	1	1																																															
0	0	1	0	2																																															
0	0	1	1	3																																															
0	1	0	0	4																																															
0	1	0	1	5																																															
0	1	1	0	6																																															
0	1	1	1	7																																															
1	0	0	0	8																																															
<b>3</b>	<b>LATCHED SCSI PARITY</b> <i>[read-only]</i> (SDP) This bit reflects the SCSI parity signal (SDP) corresponding to the data latched in the SCSI Input Data Latch register (SIDL). It changes when a new byte is latched into the SIDL register. This bit is active high. It is set when the parity signal is active.																																																		
<b>2</b>	<b>SCSI MSG SIGNAL</b> <i>[read-only]</i>																																																		
<b>1</b>	<b>SCSI C/D Signal</b> <i>[read-only]</i>																																																		
<b>0</b>	<b>SCSI I/O SIGNAL</b> <i>[read-only]</i> (I/O) These SCSI phase status bits are latched on the asserting edge of REQ when operating in either initiator or target mode. These bits are set when the corresponding signal is active. They are useful when operating in low level mode.																																																		

## 18.18 Data Structure Address (DSA)

This register contains the base address used for all table-indirect calculations. It is 32 bits wide and defaults to all zeros.

During any Memory Move operation, the contents of this register are overwritten. If the DSA value is needed for a subsequent SCSI script, save and later restore it. Note that it is possible to perform a Memory-to-DSA Move, but not a DSA-to-Memory Move.



## 18.18 Data Structure Address (DSA)

The Table Indirect scripts addressing mode is described in detail in the *53C710 Programmer's Guide*.

## 18.19 Chip Test Zero (CTEST0)

Figure 18–18 Chip Test Zero (CTEST0)

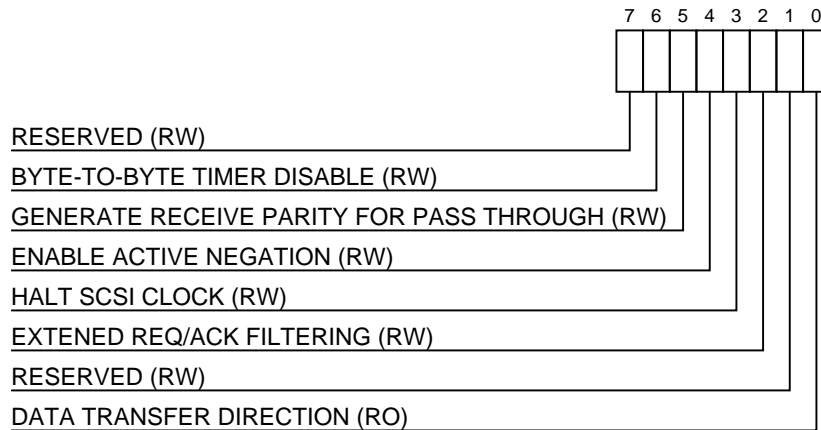


Table 18–23 Chip Test Zero Description

Field	Description
7	<b>RESERVED</b> <i>[read/write]</i> This bit must always be written zero.
6	<b>BYTE-TO-BYTE TIMER DISABLE</b> <i>[read/write]</i> (BTD) This bit, in conjunction with the Notime bit in CTEST4, provides the following selection/byte-to-byte timer options: <ul style="list-style-type: none"> <li>• At power-up or if both bits are not set, the selection and byte-to-byte time are enabled.</li> <li>• if notime is set, both functions are disabled.</li> <li>• If notime is not set and the byte-to-byte disable is set, the selection timer functions but the byte-to-byte timer is disabled.</li> </ul>
5	<b>GENERATE RECEIVE PARITY FOR PASS THROUGH</b> <i>[read/write]</i> (GRP) When this bit is set, and the 53C710 is in parity pass through mode, the parity received on the SCSI bus does not pass through to the DMA FIFO. Parity is generated as data enters the DMA FIFO, eliminating the possibility of bad SCSI parity passing through to the host bus. A SCSI parity error interrupt is generated, but a system parity problem will not be created. After reset or when the bit is cleared, while parity pass through mode is enabled, parity received on the SCSI bus passes through the 53C710 unmodified.
4	<b>ENABLE ACTIVE NEGATION</b> <i>[read/write]</i>

(continued on next page)

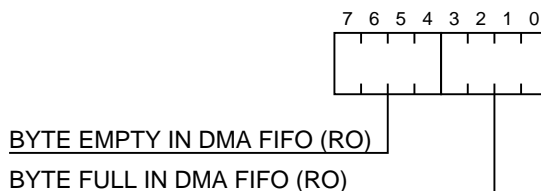
## 18.19 Chip Test Zero (CTEST0)

**Table 18–23 (Cont.) Chip Test Zero Description**

Field	Description
	(EAN) This bit provides the ability to actively deassert selected signals instead of relying on pull-ups when the 53C710 is driving. When this bit is asserted SCSI request, acknowledge, data and parity are actively deasserted. Active deassertion of these signals occurs only when the 53C710 is in an information transfer phase. When operating in a different environment or at fast SCSI trimmings, Active negation should be enabled to improve setup and hold times. Active negation is disabled after reset or when this bit is cleared.
<b>3</b>	<b>HALT SCSI CLOCK</b> <i>[read/write]</i>  (HSC) Asserting this bit causes the internal divided SCSI clock to stop in a glitchless manner. This bit may be used for test purposes or to lower $I_{dd}$ during a power-down mode. Note that SCSI registers must be reinitialized a power-up time.
<b>2</b>	<b>EXTENDED REQ/ACK FILTERING</b> <i>[read/write]</i>  (ERF) The scsi core contains a special digital filter on the REQ/ACK pins which will cause glitches on deasserting edges to be disregarded. Asserting this bit extends filter delay from 30 ns to 60 ns on the deasserting edge of the REQ and ACK signals. The 30 ns delay should be used for fast SCSI. Note that this bit must never be set during fast SCSI operation (>5 M transfers per second), because a valid assertion could be treated as a glitch. This bit does not affect transfer rates.
<b>1</b>	<b>RESERVED</b> <i>[read/write]</i>
<b>0</b>	<b>DATA TRANSFER DIRECTION</b> <i>[read-only]</i>  (DDIR) This status bit indicates the direction data is being transferred. When this bit is set, the data is transferred from the SCSI bus to the host bus. When this bit is cleared, data is transferred from the host bus to the SCSI bus. This bit cannot be written.

## 18.20 Chip Test One (CTEST1)

**Figure 18–19 Chip Test One (CTEST1)**



**Table 18–24 Chip Test One Description**

Field	Description
<b>7:4</b>	<b>BYTE EMPTY IN DMA FIFO</b> <i>[read-only]</i>

(continued on next page)

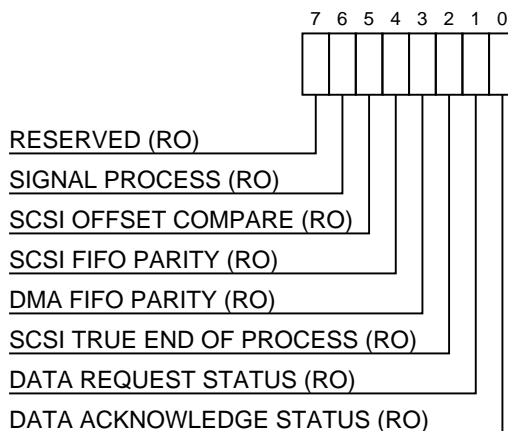
## 18.20 Chip Test One (CTEST1)

**Table 18–24 (Cont.) Chip Test One Description**

Field	Description
	(FMT3—FMT0) These bits identify the bottom bytes in the DMA FIFO that are empty. Each bit corresponds to a byte lane in the DMA FIFO. For example, if byte lane 3 is empty, then FMT3 is 1. Because the FMT flags indicate the status of bytes at the bottom of the FIFO, if all FMT bits are set, the DMA FIFO is empty.
<b>3:0</b>	<b>BYTE FULL IN DMA FIFO</b> <i>[read-only]</i> (FFL3—FFL0) These status bits identify the top bytes in the DMA FIFO that are full. Each bit corresponds to a byte lane in the DMA FIFO. For example, if byte lane 3 is full, then FFL3 is 1. Because the FFL flags indicate the status of bytes at the top of the FIFO, if all FFL bits are set, the DMA FIFO is full.

## 18.21 Chip Test Two (CTEST2)

**Figure 18–20 Chip Test Two (CTEST2)**



**Table 18–25 Chip Test Two Description**

Field	Description
<b>7</b>	<b>RESERVED</b> <i>[read-only]</i>
<b>6</b>	<b>SIGNAL PROCESS</b> <i>[read-only]</i> (SIGP) This bit is a copy of the SIGP bit in the ISTAT register (bit <5>). The SIGP bit is a flag that may be passed to or from a running script. The only scripts instruction directly offered by the SIGP bit is a Wait for Selection Re-selection. When this bit is read, the SIGP bit in the ISTAT register is cleared.
<b>5</b>	<b>SCSI OFFSET COMPARE</b> <i>[read-only]</i>

(continued on next page)

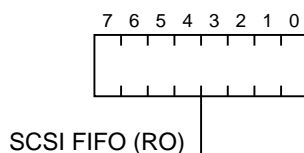
**Table 18–25 (Cont.) Chip Test Two Description**

Field	Description
	(SOFF) This bit operates differently, depending on whether the chip is an initiator or target. If the 53C710 is an initiator, this bit is set whenever the SCSI synchronous offset counter is equal to zero. If the 53C710 is a target, this bit is set whenever the SCSI synchronous offset counter is equal to the maximum synchronous offset defined in the SXFER register.
<b>4</b>	<b>SCSI FIFO PARITY</b> <i>[read-only]</i>  (SFP) This bit represents the parity bit of the SCSI synchronous FIFO corresponding to data read out of the FIFO. Reading the CTEST3 register unloads a data byte from the bottom of the SCSI synchronous FIFO. When the CTEST3 register is read, the data parity bit is latched into this bit location.
<b>3</b>	<b>DMA FIFO PARITY</b> <i>[read-only]</i>  (DFP) This bit represents the parity bit of the DMA FIFO when the CTEST6 register reads data out of the FIFO. Reading the CTEST6 register unloads one data byte from the bottom of the DMA FIFO. When the CTEST6 register is read, the parity signal is latched into this bit location and the next byte falls down to the bottom of FIFO.
<b>2</b>	<b>SCSI TRUE END OF PROCESS</b> <i>[read-only]</i>  (TOEP) This bit indicates the status of the 53C710's internal TEOP signal. The TEOP signal acknowledges the completion of a block move through the SCSI portion of the 53C710. When this bit is set, TEOP is active. When this bit is clear, TEOP is inactive.
<b>1</b>	<b>DATA REQUEST STATUS</b> <i>[read-only]</i>  (DREQ) This bit indicates the status of the 53C710's internal Data Request signal (DREQ). When this bit is set, DREQ is active. When this bit is clear, DREQ is inactive.
<b>0</b>	<b>DATA ACKNOWLEDGE STATUS</b> <i>[read-only]</i>  (DACK) This bit indicates the status of the 53C710's internal Data Acknowledge signal (DACK). When this bit is set, DACK is inactive. When this bit is clear, DACK is active.

## 18.22 Chip Test Three (CTEST3)

Reading this register unloads the bottom byte of the 8-byte SCSI synchronous FIFO. Reading this register also latches the parity bit for the FIFO into the SCSI FIFO Parity bit in the CTEST2 register. The FIFO Full bits in the SSTAT2 register can be read to determine how many bytes currently reside in the SCSI synchronous FIFO. Reading this register when the SCSI FIFO is empty causes a SCSI Gross Error (FIFO underflow).

**Figure 18–21 Chip Test Three (CTEST3)**



## 18.22 Chip Test Three (CTEST3)

Table 18–26 Chip Test Three Description

Field	Description
7:0	<b>SCSI FIFO</b> <i>[read-only]</i>

## 18.23 Chip Test Four (CTEST4)

Figure 18–22 Chip Test Four (CTEST4)

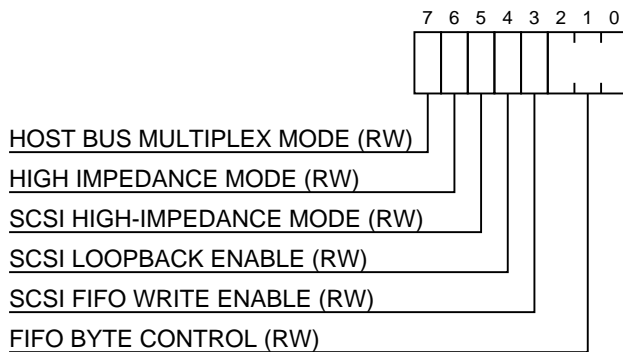


Table 18–27 Chip Test Four Description

Field	Description
7	<b>HOST BUS MULTIPLEX MODE</b> <i>[read/write]</i>  (MUX) When set, the MUX bit puts the 53C710 into host bus MUX mode. In this mode, the chip asserts a valid address for one BCLK (during which AS/TS is valid and the data bus is tristated), and then tristates the address bus and drives the data bus (if a write). This allows the address and data buses to be tied together. It should be written prior to acquiring bus mastership. The MUX mode bit allows the 53C710 to operate without external hardware on those host buses on which data and addresses share a common 32 bits.
6	<b>HIGH IMPEDANCE MODE</b> <i>[read/write]</i>  (ZMOD) Setting this bit causes the 53C710 to place all output and bidirectional pins into a high-impedance state. In order to read data out of the 53C710, this bit must be cleared. This bit is intended for board-level testing only. Setting this bit during system operation will likely result in a system crash.
5	<b>SCSI HIGH-IMPEDANCE MODE</b> <i>[read/write]</i>

(continued on next page)

Table 18–27 (Cont.) Chip Test Four Description

Field	Description
	<p>(SZM) Setting this bit causes the 53C710 to place certain SCSI outputs in a high-impedance state. The following outputs are placed in a high-impedance state:</p> <ul style="list-style-type: none"> <li>• SD 7:0      SCSI OUTPUT DATA LATCH (SODL)</li> <li>• SDP      SCSI SDP PARITY SIGNAL (SSTAT1&lt;0&gt;)</li> <li>• BSY      ASSERT SCSI BSY SIGNAL (SOCL&lt;5&gt;)</li> <li>• SEL      SELECTED OR RESELECTED (SIEN&lt;4&gt;)</li> <li>• RST      SCSI RST RECEIVED (SIEN&lt;1&gt;)</li> <li>• REQ      ASSERT SCSI REQ SIGNAL (SOCL&lt;7&gt;)</li> <li>• C/D      ASSERT SCSI C/D SIGNAL (SOCL&lt;1&gt;)</li> <li>• I/O      ASSERT SCSI I/O SIGNAL (SOCL&lt;0&gt;)</li> <li>• MSG      ASSERT SCSI MSG SIGNAL (SOCL&lt;2&gt;)</li> <li>• ACK      ASSERT SCSI ACK SIGNAL (SOCL&lt;6&gt;)</li> <li>• ATN      ASSERT SCSI ATN SIGNAL (SOCL&lt;3&gt;)</li> </ul> <p>The direction control lines (SDIR &lt;7:0&gt;, SDIRP, BSYDIR, RSTDIR, and SELDIR) are driven low and are not in a high-impedance state. In order to transfer data on the SCSI bus, this bit must be cleared.</p>
<b>4</b>	<p><b>SCSI LOOPBACK ENABLE</b> <i>[read/write]</i></p> <p>(SLBE) Setting this bit enables loopback mode. Loopback allows any SCSI signal to be asserted. 53C710 may be an initiator or a target. It also allows the 53C710 to transfer data from the SODL register back into the SIDL register. For a complete description of the tests that can be performed in loopback mode, see <i>53C710 Programmer's Guide</i>.</p>
<b>3</b>	<p><b>SCSI FIFO WRITE ENABLE</b> <i>[read/write]</i></p> <p>(SFWR) Setting this bit redirects data from the SODL to the SCSI FIFO. A write to the SODL register loads a byte into the SCSI FIFO. The parity bit loaded into the FIFO is odd or even parity depending on the status of the Assert SCSI even Parity bit in the SCNTL1 register. Clearing this bit disables this feature.</p>
<b>2:0</b>	<p><b>FIFO BYTE CONTROL</b> <i>[read/write]</i></p> <p>(FBL2—FBL0) These bits send the contents of the CTEST6 register to the appropriate byte lane of the 32-bit DMA FIFO. If the FBL2 bit is set, bits FBL1 &amp; FBL0 determine which of four byte lanes can be read or written. Each of the four bytes that make up the 32-bit DMA FIFO can be accessed by writing these bits to the proper value. For normal operation, FBL2 must equal zero (set to this value before executing SCSI scripts).</p>

## 18.24 Chip Test Five (CTEST5)

## 18.24 Chip Test Five (CTEST5)

Figure 18–23 Chip Test Five (CTEST5)

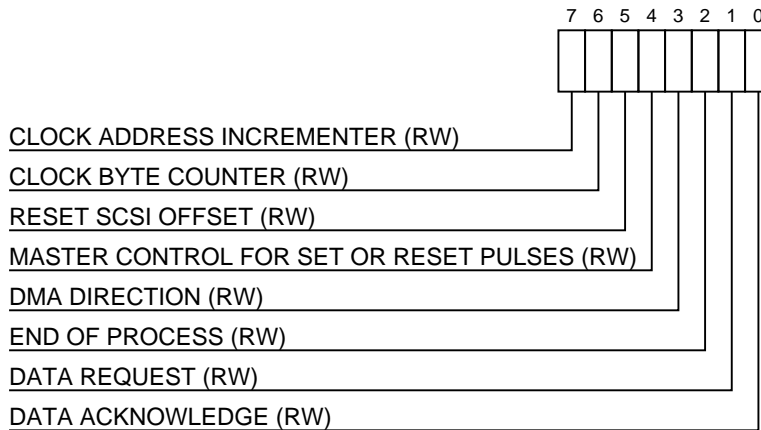


Table 18–28 Chip Test Five Description

Field	Description
7	<b>CLOCK ADDRESS INCREMENTER</b> <i>[read/write]</i> (ASCK) Setting this bit increments the address pointer contained in the DNAD register (by four bytes). The DNAD register is incremented based on the DNAD contents and the current DBC value. This bit automatically clears itself after incrementing the DNAD register.
6	<b>CLOCK BYTE COUNTER</b> <i>[read/write]</i> (BBCK) Setting this bit decrements the byte count contained in the DBC register. It is decremented based on the DBC contents and the current DNAD value. This bit automatically clears itself after decrementing the DBC register.
5	<b>RESET SCSI OFFSET</b> <i>[read/write]</i> (ROFF) Setting this bit resets the current offset pointer in the SCSI synchronous offset counter. This bit is set if a SCSI Gross Error condition occurs. The offset should be reset when a synchronous transfer does not complete successfully. This bit automatically resets itself after clearing the synchronous offset.
4	<b>MASTER CONTROL FOR SET OR RESET PULSES</b> <i>[read/write]</i> (MASR) This bit controls the operation of bits <3:0>. When this bit is set, bits <3:0> assert the corresponding signals. When this bit is reset, bits <3:0> deassert the corresponding signals. This bit and bits <3:0> should not be changed in the same write cycle.
3	<b>DMA DIRECTION</b> <i>[read/write]</i> (DDIR) Setting this bit either asserts or deasserts the internal DMA Write (DMAWR) direction signal depending on the current status of the MASR bit in this register. Asserting the DMAWR signal indicates that data is transferred from the SCSI bus to the host bus. Deasserting the DMAWR signal transfers data from the host bus to the SCSI bus.
2	<b>END OF PROCESS</b> <i>[read/write]</i>

(continued on next page)

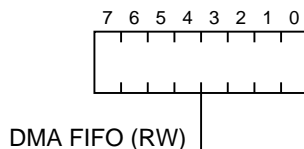
Table 18–28 (Cont.) Chip Test Five Description

Field	Description
	(EOP) Setting this bit either asserts or deasserts the internal EOP control signal depending on the current status of the MASR bit in this register. The internal EOP signal is an output from the DMA portion of the 53C710 to the SCSI portion of the 53C710. Asserting the EOP signal indicates that the last data byte has been transferred between the two portions of the chip. Deasserting the EOP signal indicates that the last data byte has not been transferred between the two portions of the chip. If the MASR bit is configured to assert this signal, this bit automatically clears itself after pulsing the EOP signal.
<b>1</b>	<b>DATA REQUEST</b> <i>[read/write]</i> (DREQ) Setting this bit either asserts or deasserts the internal DREQ (data request signal) depending on the current status of the MASR bit in this register. Asserting the DREQ signal indicates that the SCSI portion of the 53C710 request a data transfer with the DMA portion of the chip. Deasserting the DREQ signal indicates that data should not be transferred between the SCSI portion of the 53C710 and the DMA portion. If the MASR bit is configured to assert this signal, this bit automatically clears itself after asserting the DREQ signal.
<b>0</b>	<b>DATA ACKNOWLEDGE</b> <i>[read/write]</i> (DACK) Setting this bit either asserts or deasserts the internal DACK data request signal dependent on the current status of the MASR bit in this register. Asserting the DACK signal indicates that the DMA portion of the 53C710 acknowledges a data transfer with the SCSI portion of the chip. Deasserting the DACK signal indicates that data should not be transferred between the DMA portion of the 53C710 and the SCSI portion. If the MASR bit is configured to assert this signal, this bit automatically clears itself after asserting the DACK signal.

## 18.25 Chip Test Six (CTEST6)

Writing to this register writes data to the appropriate byte lane of the DMA FIFO as determined by the FBL bits in the CTEST4 register. Reading this register unloads data from the appropriate byte lane of the DMA FIFO as determined by the FBL bits in the CTEST4 register. Data written to the FIFO is loaded into the top of the FIFO. Data read out of the FIFO is taken from the bottom. When data is read from the DMA FIFO, the parity bit for that byte is latched and stored in the DMA FIFO parity bit in the CTEST2 register. To prevent DMA data from being corrupted, this register should not be accessed before starting or restarting a script.)

Figure 18–24 Chip Test Six (CTEST6)





## 18.25 Chip Test Six (CTEST6)

Table 18–29 Chip Test Six Description

Field	Description
7:0	<b>DMA FIFO</b> <i>[read/write]</i>

## 18.26 Chip Test Seven (CTEST7)

Figure 18–25 Chip Test Seven (CTEST7)

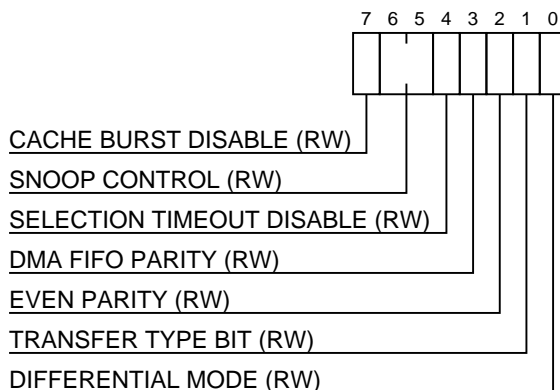


Table 18–30 Chip Test Seven Description

Field	Description
7	<p><b>CACHE BURST DISABLE</b> <i>[read/write]</i></p> <p>(CDIS) When this bit is set, the 53C710 does not request a cache-line burst. When this bit is clear, the chip attempts cache-line bursts when two conditions are met.</p> <ol style="list-style-type: none"> <li>1. The address must be lined up to a cache-line boundary (&lt;A3:A0&gt; must be zero).</li> <li>2. The transfer counter must be greater than 32.</li> </ol> <p>Cache-line burst mode eliminates the need for a full handshake between the bus master and the memory device when transferring data.</p>
6:5	<p><b>SNOOP CONTROL</b> <i>[read/write]</i></p> <p>(SC1—SC0) The SC0 and SC1 bits control the two Snoop Control pins. The SC1 bit controls the Snoop Control 1 pin all of the time. The SC0 bit controls the Snoop Control 0 pin only when the Snoop Mode bit is not set. Monitoring the SC0 bit provides advance notice of a pending 53C710 bus request. Bus snooping allows for transmission of additional information to other devices on the host bus about the current type of transfer. In Bus Mode 2, the host processor can snoop an alternate master Read/Write transfer, ensuring access to valid data. In other operating modes, these bits and pins provide additional user-defined functionality.</p>

(continued on next page)

Table 18–30 (Cont.) Chip Test Seven Description

Field	Description
<b>4</b>	<b>SELECTION TIMEOUT DISABLE</b> <i>[read/write]</i> (NOTIME) Setting this bit disables the 250 ms timer for all modes, including byte to byte.
<b>3</b>	<b>DMA FIFO PARITY</b> <i>[read/write]</i> (DFP) This bit represents the parity bit of the DMA FIFO when reading data out of the DMA FIFO via programmed I/O. In order to transfer data to or from the DMA FIFO, perform a read or a write to the CTEST6 register. When loading data into the FIFO via programmed I/O, write this bit to the FIFO as the parity bit for each byte loaded. When writing data to the DMA FIFO, set this bit with the status of the parity bit to be written to the FIFO before writing the byte to the FIFO.
<b>2</b>	<b>EVEN PARITY</b> <i>[read/write]</i> (EVP) Parity is generated for all slave mode register reads and master memory writes. This bit controls the parity sense. Setting this bit causes the 53C710 to generate even parity when driving data on the host data bus. The 53C710 inverts the parity bit received from the SCSI bus to create even parity. In addition, the even parity received from the host bus is inverted to odd parity before the 53C710 checks parity and sends the data to the SCSI bus. Clearing this bit causes the 53C710 to maintain odd parity throughout the chip.
<b>1</b>	<b>TRANSFER TYPE BIT</b> <i>[read/write]</i> (TT1) The inverted value of this bit is asserted on the TT1 pin during bus mastership (Bus Mode 2 only). This bit is not used in Bus Mode 1.
<b>0</b>	<b>DIFFERENTIAL MODE</b> <i>[read/write]</i> (DIFF) Setting this bit enables the 53C710 to interface with external differential pair receivers. The SCSI BSY, SEL, and RST are input in differential mode only. Resetting this bit enables single-ended mode. This bit should be set in the initialization routine if the differential pair interface is to be used.

## 18.27 Temporary Stack (TEMP)

This 32-bit register stores the instruction address pointer for a CALL or a RETURN instruction. The address pointer stored in this register is loaded into the DSP register. This address points to the next instruction to be executed. Do not write to TEMP while the 53C710 is executing SCSI scripts. During any Memory-to-Memory Move operation, the contents of this register are destroyed. If the TEMP value is needed for a subsequent SCSI script, save and then later restore it.

## 18.28 DMA FIFO (DFIFO)

## 18.28 DMA FIFO (DFIFO)

Figure 18–26 DMA FIFO (DFIFO)

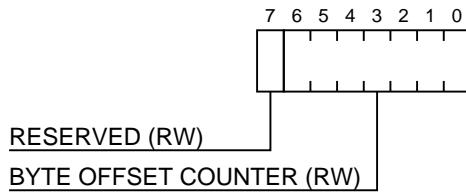


Table 18–31 DMA FIFO Description

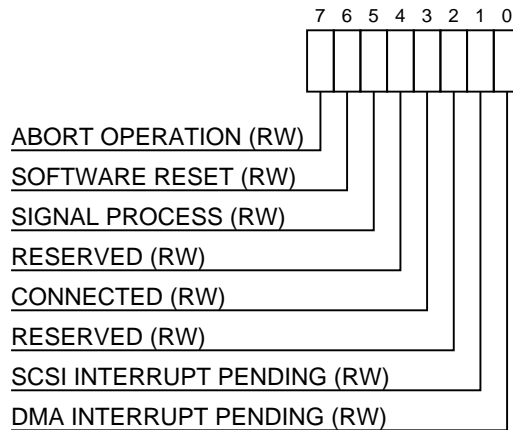
Field	Description
7	<b>RESERVED</b> <i>[read/write]</i>
6:0	<b>BYTE OFFSET COUNTER</b> <i>[read/write]</i> (BO6—BO0) These six bits indicate the amount of data transferred between the SCSI core and the DMA core. They may be used to determine the number of bytes in the DMA FIFO when a DMA error occurs. These bits are unstable while data is being transferred between the two cores. Once the chip has stopped transferring data, these bits are stable. The following steps determine how many bytes are left in the DMA FIFO when an error occurs, regardless of the direction of the transfer. <ol style="list-style-type: none"> <li>1. Subtract the seven least significant bits of the DBC register from the 7-bit value of the DFIFO register.</li> <li>2. AND the result with <math>7F_{16}</math> for a byte count between zero and 64.</li> </ol>

## 18.29 Interrupt Status (ISTAT)

This is the only register that can be accessed by the CPU while the 53C710 is executing scripts without interfering in the operation of the 53C710. It may be used to poll for interrupts if interrupts are masked. When either the SIP or DIP bit is set, the DSTAT and SSTAT0 latches close and subsequent interrupts are stacked (held in a pending register "behind" the status register). When the current interrupt is cleared by reading the appropriate status register, the stacked interrupts are transferred to the status register causing another interrupt.

When an interrupt occurs, the 53C710 halts in an orderly fashion before asserting IRQ. If, in the middle of an instruction fetch, the fetch is completed (except in the case of a Bus Fault or Watchdog Timeout), though execution does not begin. If possible, DMA write operations empty the FIFO before halting. All other DMA operations finish only the current cycle (or burst if a cache line) before halting. SCSI handshakes that have begun are completed before halting. The 53C710 attempts to clean up any outstanding synchronous offset. In the case of the Transfer Control Instructions, once instruction execution begins it continues toward completion before halting. If the instruction is a JUMP/CALL WHEN, the wait aborts and the DSP is updated to the transfer address before halting. All other instructions may halt before completing execution.

**Figure 18–27 Interrupt Status (ISTAT)**



**Table 18–32 Interrupt Status Description**

Field	Description
7	<p><b>ABORT OPERATION</b> <i>[read/write]</i></p> <p>(ABRT) Setting this bit aborts the current operation being executed by the 53C710. If this bit is set and an interrupt is received, reset this bit before reading the DSTAT register to prevent further aborted interrupts from being generated. The sequence to abort any operation is:</p> <ol style="list-style-type: none"> <li>1. Set this bit.</li> <li>2. Wait for an interrupt.</li> <li>3. Read the ISTAT register.</li> <li>4. If the SCSI Interrupt Pending bit is set, read the SSTAT0 register to determine the cause of the SCSI Interrupt and go back to step 2.</li> <li>5. If the SCSI Interrupt Pending bit is clear, and the DMA Interrupt Pending bit is set, write 00<sub>16</sub> value to this register.</li> <li>6. Read the DSTAT register to verify the aborted interrupt and to see if any other interrupting conditions have occurred.</li> </ol>
6	<p><b>SOFTWARE RESET</b> <i>[read/write]</i></p> <p>(RST) Setting this bit resets the 53C710. All registers except the DCNTL EA bit are cleared to their respective default values and all SCSI signals are deasserted. Setting this bit does not cause the SCSI RST signal to be asserted. This bit is not self-clearing; it must be cleared to remove the reset condition (a hardware reset also clears this bit). This reset does not clear the Enable Acknowledge (EA) or Function Control One (FC1) bits.</p>
5	<p><b>SIGNAL PROCESS</b> <i>[read/write]</i></p>

(continued on next page)

## 18.29 Interrupt Status (ISTAT)

**Table 18–32 (Cont.) Interrupt Status Description**

Field	Description
	SIGP is a Read/Write bit that can be written at any time, and polled & reset via CTEST2. The SIGP bit can be used in various ways to pass a flag to or from a running script. The only script instruction directly affected by the SIGP bit is Wait For Selection/Reselection. Setting this bit causes that opcode to jump to the alternate address immediately. The instructions at the alternate jump address should check the status of SIGP to determine the cause of the jump. The SIGP bit may be used at any time and is not restricted to the Wait for Selection/Reselection condition. Note that if the SIGP bit is active when a Selection/Reselection occurs, the auto-switching from/to target mode is disabled and must be manually set by either the host or a script.
<b>4</b>	<b>RESERVED</b> <i>[read/write]</i>
<b>3</b>	<b>CONNECTED</b> <i>[read/write]</i> (CON) This bit is automatically set any time the 53C710 is connected to the SCSI bus as an initiator or as a target. It is set after successfully completing arbitration or when the 53C710 has responded to a bus-initiated selection or reselection. It is also set after successfully completing arbitration when operating in low level mode. When this bit is clear, the 53C710 is not connected to the SCSI bus.
<b>2</b>	<b>RESERVED</b> <i>[read/write]</i>
<b>1</b>	<b>SCSI INTERRUPT PENDING</b> <i>[read/write]</i> (SIP) This status bit is set when an interrupt condition is detected in the SCSI portion of the 53C710. The following conditions cause a SCSI interrupt to occur: <ul style="list-style-type: none"> <li>• A phase mismatch (initiator mode) or ATN becomes active (target mode)</li> <li>• An arbitration sequence is completed</li> <li>• A selection or reselection timeout occurs</li> <li>• The 53C710 was selected or reselected</li> <li>• A SCSI gross error occurs</li> <li>• An unexpected disconnect occurs</li> <li>• A SCSI reset occurs</li> <li>• A parity error is detected</li> </ul> <p>To determine exactly which condition(s) caused the interrupt, read the SSTAT0 register.</p>
<b>0</b>	<b>DMA INTERRUPT PENDING</b> <i>[read/write]</i>

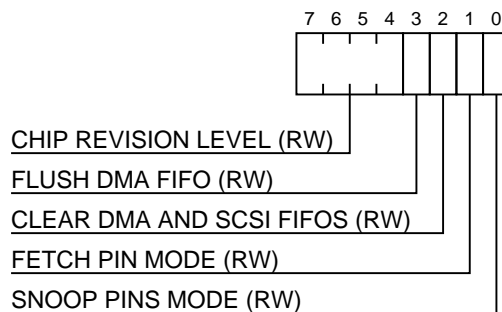
(continued on next page)

**Table 18–32 (Cont.) Interrupt Status Description**

Field	Description
	(DIP) This status bit is set when an interrupt condition is detected in the DMA portion of the 53C710. The following conditions causes a DMA interrupt to occur:
	<ul style="list-style-type: none"> <li>• A bus fault is detected</li> <li>• An abort condition is detected</li> <li>• A scripts instruction is executed in single-step mode</li> <li>• A scripts interrupt instruction is executed</li> <li>• The Watchdog Timer decrements to zero</li> <li>• An illegal instruction is detected</li> </ul>
	To determine exactly which condition(s) caused the interrupt, read the DSTAT register.

### 18.30 Chip Test Eight (CTEST8)

**Figure 18–28 Chip Test Eight (CTEST8)**



**Table 18–33 Chip Test Eight Description**

Field	Description
7:4	<b>CHIP REVISION LEVEL</b> <i>[read/write]</i> (V3—V0) These bits identify the chip revision level for software purposes. This chapter describes devices with revision level 1.
3	<b>FLUSH DMA FIFO</b> <i>[read/write]</i> (FLF) When this bit is set, data residing in the DMA FIFO is transferred to memory, starting at the address in the DNAD register. The internal DMAWR signal, controlled by the CTEST5 register, determines the direction of the transfer. This bit is not self clearing; once the 53C710 has successfully transferred the data, this bit should be reset. Note that all chip registers may be read during flush operations.

(continued on next page)

## 18.30 Chip Test Eight (CTEST8)

**Table 18–33 (Cont.) Chip Test Eight Description**

Field	Description
<b>2</b>	<p><b>CLEAR DMA AND SCSI FIFOS</b> <i>[read/write]</i></p> <p>(CLF) When this bit is set, all data pointers for the SCSI and DMA FIFOs are cleared. In addition to the SCSI and DMA FIFO pointers, the SIDL, SODL, and SODR full bits in the SSTAT1 register are cleared. Any data in either of the FIFOs is lost. This bit automatically resets after the 53C710 has successfully cleared the appropriate FIFO pointers and registers.</p>
<b>1</b>	<p><b>FETCH PIN MODE</b> <i>[read/write]</i></p> <p>(FM) When set, this bit causes the FETCH pin to deassert during indirect and table-indirect read operations. FETCH is active only during the opcode portion of an instruction fetch. This allows scripts to be stored in a PROM while data tables are stored in RAM, reducing the long delay associated with arbitrating for the host bus in order to fetch scripts instructions from system memory.</p> <p>If this bit is not set, FETCH is asserted for all bus cycles during instruction fetches.</p>
<b>0</b>	<p><b>SNOOP PINS MODE</b> <i>[read/write]</i></p> <p>(SM) When set, the two snoop pins change function and become pure outputs that are always driven, except when in ZMODE.</p> <p>When clear, the snoop pins are driven during host bus ownership with the values of the CTEST7 SC&lt;1:0&gt; bits.</p>

## 18.31 Longitudinal Parity (LCRC)

This register contains the longitudinal parity for all data crossing from the DMA FIFO to or from the SCSI core. The parity consists of an exclusive OR of all data bytes.

Writing to this register clears its contents to 00<sub>16</sub> regardless of the value written.

Like the SFBR register in the 53C710, this register is used by the SCSI core to hold the SCSI ID value during selection and reselection. The LCRC register should be used instead of the SFBR because the SFBR is used as an accumulator during many scripts operations, and may be overwritten at any time by a selection or reselection.

**Figure 18–29 Longitudinal Parity (LCRC)**



**Table 18–34 Longitudinal Parity Description**

Field	Description
<b>7:0</b>	<p><b>BITS 7-0 LONGITUDINAL PARITY</b> <i>[read/write]</i></p>

## 18.32 DMA Byte Counter (DBC)

This 24-bit register determines the number of bytes to be transferred in a Block Move instruction. While sending data to the SCSI bus, the counter is decremented as data is moved into the DMA FIFO from memory. While receiving data from the SCSI bus, the counter is decremented as data is written to memory from the 53C710. The DBC counter is decremented each time that the AS signal is pulsed by the 53C710. It is decremented by an amount equal to the number of bytes that were transferred.

The maximum number of bytes that can be transferred in any one Block Move command is 16,777,215 bytes. The maximum value that can be loaded into the DBC register is  $FFFFFF_{16}$ . If the instruction is Block Move and a value of  $000000_{16}$  is loaded into the DBC register, an illegal instruction interrupt occurs if the chip is not operating in a target mode command phase.

The DBC register is also used during table indirection I/O scripts to hold the offset value.

Figure 18–30 DMA Byte Counter (DBC)

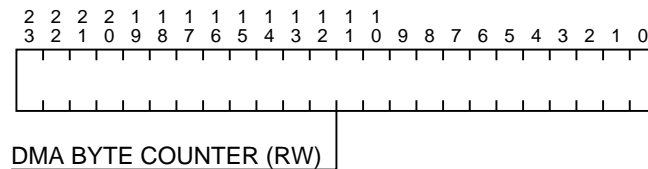


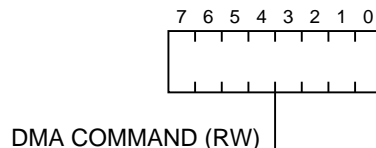
Table 18–35 DMA Byte Counter Description

Field	Description
23:0	DMA BYTE COUNTER <i>[read/write]</i>

## 18.33 DMA Command (DCMD)

This 8-bit register determines the instruction for the 53C710 to execute. This register has a different function for each instruction. For a complete description, refer to the 53C710 instruction set.

Figure 18–31 DMA Command (DCMD)





## 18.33 DMA Command (DCMD)

Table 18–36 DMA Command Description

Field	Description
7:0	<b>DMA COMMAND</b> <i>[read/write]</i>

## 18.34 DMA Next Data Address (DNAD)

This 32-bit register contains the general-purpose address pointer. At the start of some script operations, its value is copied from the DSPS register. Its value may not be valid except in certain abort conditions.

To maintain software compatibility with the 53C710, interrupt vectors should be read from the DSPS register.

Figure 18–32 DMA Next Data Address (DNAD)

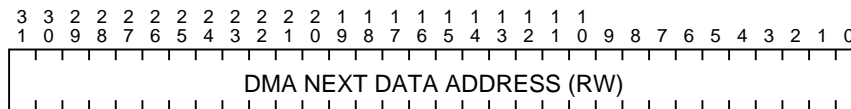


Table 18–37 DMA Next Data Address Description

Field	Description
31:0	<b>DMA NEXT DATA ADDRESS</b> <i>[read/write]</i>

## 18.35 DMA Scripts Pointer (DSP)

To execute SCSI scripts, the address of the first SCSI script must be written to this register. In normal scripts operation, once the starting address of the SCSI scripts is written to this register, the scripts are automatically fetched and executed until an interrupt condition occurs.

In single-step mode, there is a scripts step interrupt after each instruction is executed. The DSP register does not need to be written with the next address, but the Start DMA bit (DCNTL<2>), must be set each time the step interrupt occurs to fetch and execute the next SCSI script. When writing this register 8 bits at a time, writing the upper 8 bits, 2F (2C), begins execution of SCSI scripts.

Figure 18–33 DMA Scripts Pointer (DSP)

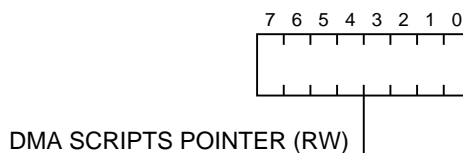


Table 18–38 DMA Scripts Pointer Description

Field	Description
7:0	<b>DMA SCRIPTS POINTER</b> <i>[read/write]</i>

### 18.36 DMA Scripts Pointer Save (DSPS)

This register contains the second longword of Read/Write or Transfer Control scripts instructions. It is overwritten each time a script instruction is executed. When a script interrupt is fetched, this register holds the interrupt vector.

Figure 18–34 DMA Scripts Pointer Save (DSPS)

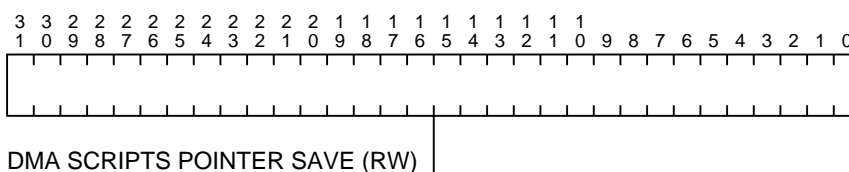


Table 18–39 DMA Scripts Pointer Save Description

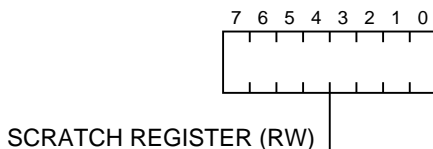
Field	Description
31:0	<b>DMA SCRIPTS POINTER SAVE</b> <i>[read/write]</i>

### 18.37 Scratch Register (SCRATCH)

This is a general-purpose, user-defined scratch pad register. Most scripts operations do not destroy the contents of this register, only register read/write and memory moves into the scratch register alter its contents.

The scratch register, combined with register-to-register move, AND, OR, and ADD operations provides the capability to write a complete SCSI interface program in scripts.

Figure 18–35 Scratch Register (SCRATCH)



## 18.37 Scratch Register (SCRATCH)

Table 18–40 Scratch Register Description

Field	Description
7:0	<b>SCRATCH REGISTER</b> <i>[read/write]</i>

## 18.38 DMA Mode (DMODE)

Figure 18–36 DMA Mode (DMODE)

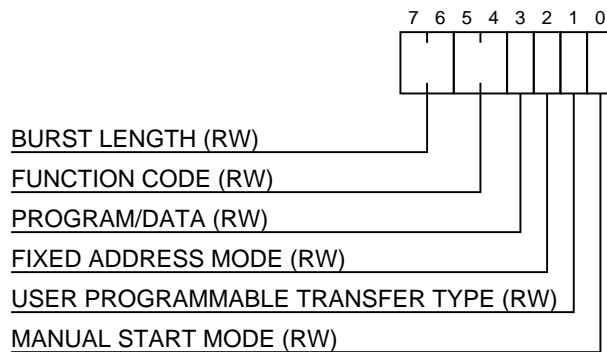


Table 18–41 DMA Mode Description

Field	Description															
7:6	<p><b>BURST LENGTH</b> <i>[read/write]</i></p> <p>(BL1—BL0) These bits control the number of bus cycles performed per bus ownership. The 53C710 asserts the Bus Request output when the DMA FIFO can accommodate a transfer of at least one burst size of data. Bus Request is also asserted during start-of-transfer/end-of-transfer cleanup &amp; alignment, even though less than a full burst of transfer may be performed.</p> <p>To perform cache line bursts, these bits must be set to 4 or 8 transfers and cache bursting must be enabled (CTEST7). Cache bursts are always four longword transfers, regardless of the setting of these bits.</p> <p>The 53C710 inserts a "fairness delay" of approximately 5 to 8 BCLKs between bus ownership. This gives the CPU and other bus master devices the opportunity to access memory between bursts.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>BL1</th> <th>BL0</th> <th>Burst length</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1-transfer</td> </tr> <tr> <td>0</td> <td>1</td> <td>2-transfer</td> </tr> <tr> <td>1</td> <td>0</td> <td>4-transfer</td> </tr> <tr> <td>1</td> <td>1</td> <td>8-transfer</td> </tr> </tbody> </table>	BL1	BL0	Burst length	0	0	1-transfer	0	1	2-transfer	1	0	4-transfer	1	1	8-transfer
BL1	BL0	Burst length														
0	0	1-transfer														
0	1	2-transfer														
1	0	4-transfer														
1	1	8-transfer														
5:4	<p><b>FUNCTION CODE</b> <i>[read/write]</i></p> <p>(FC2—FC1) These bits are user defined. Their values are asserted onto the corresponding device pins during bus mastership. These bits/pins are active in both bus modes.</p>															
3	<p><b>PROGRAM/DATA</b> <i>[read/write]</i></p>															

(continued on next page)

Table 18–41 (Cont.) DMA Mode Description

Field	Description
	<p>(PD) This bit affects the function of the FC0 pin. Setting this bit causes the 53C710 to drive the FC0 signal low when fetching instructions from memory. Clearing this bit causes the 53C710 to drive the FC0 signal high when fetching instructions from memory.</p> <p>The FC0 signal is always driven high when moving data to or from memory and can only be driven low during instruction fetch cycles. This feature can be used to allow scripts and data to be stored in separate memory banks.</p>
<b>2</b>	<p><b>FIXED ADDRESS MODE</b> <i>[read/write]</i></p> <p>(FAM) Setting this bit disables the address pointer (DNAD register) so that it does not increment after each data transfer. If this bit is clear, the pointer increments after each data transfer. This bit is used to transfer data to or from a fixed port address. The port width must equal 32-bits.</p>
<b>1</b>	<p><b>USER PROGRAMMABLE TRANSFER TYPE</b> <i>[read/write]</i></p> <p>(U0/TT0) In both bus modes, UPSO-TT0 is a general-purpose output pin. The value of this bit is asserted onto the UPSO-TT0 pin while the 53C710 is a bus master, to indicate the type of access for the current bus transfer.</p>
<b>0</b>	<p><b>MANUAL START MODE</b> <i>[read/write]</i></p> <p>(MAN) Clearing this bit causes the 53C710 to automatically fetch and execute SCSI scripts after the DSP register is written. Setting this bit disables the 53C710 from automatically fetching and executing SCSI scripts after the DSP register is written. When the Start DMA bit in the DCNTL register is cleared, it controls the start time of the operation. Once the Start DMA bit in the DCNTL register is set, the 53C710 automatically fetches and executes each instruction.</p>

## 18.39 DMA Interrupt Enable (DIEN)

This register contains the interrupt mask bits corresponding to the interrupting condition described in the DSTAT register. An interrupt is masked by clearing the appropriate mask bit. Masking an interrupt prevents IR/Q from being asserted for the corresponding interrupt, but the status bit is still set in the DSTAT register. Masking an interrupt does not prevent the ISTAT DIP from being set; all DMA interrupts are considered fatal. Setting a mask bit enables the assertion of IRQ for the corresponding interrupt.

A masked nonfatal interrupt does not prevent unmasked or fatal interrupts from getting through; interrupt stacking does not begin until either the ISTAT SIP or DIP is set.

The 53C710 IRQ output is latched; once asserted, it remains asserted until the interrupt is cleared by reading the appropriate status register. Masking an interrupt after the IRQ output is asserted does not cause IRQ to be deasserted.

## 18.39 DMA Interrupt Enable (DIEN)

Figure 18–37 DMA Interrupt Enable (DIEN)

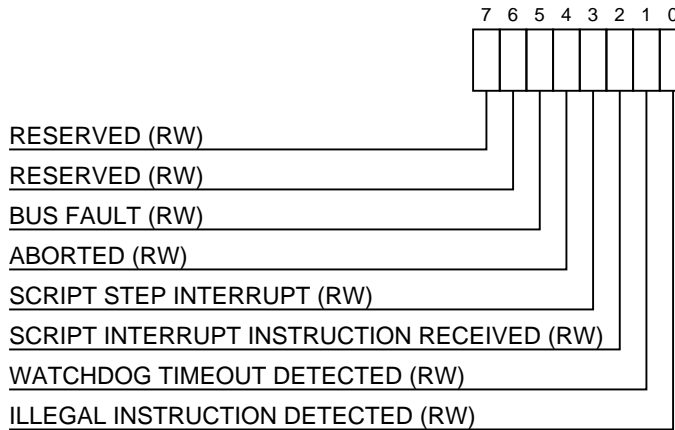


Table 18–42 DMA Interrupt Enable Description

Field	Description
7	<b>RESERVED</b> [read/write]
6	<b>RESERVED</b> [read/write]
5	<b>BUS FAULT</b> [read/write]
4	<b>ABORTED</b> [read/write]
3	<b>SCRIPT STEP INTERRUPT</b> [read/write]
2	<b>SCRIPT INTERRUPT INSTRUCTION RECEIVED</b> [read/write]
1	<b>WATCHDOG TIMEOUT DETECTED</b> [read/write]
0	<b>ILLEGAL INSTRUCTION DETECTED</b> [read/write]

## 18.40 DMA Watchdog Timer (DWT)

The DMA watchdog timer register provides a timeout mechanism during data transfers between the 53C710 and memory. This register determines the amount of time that the 53C710 waits for the assertion of the transfer acknowledge (TA) signal after starting a bus cycle. Write the timeout value to this register during initialization. Every time that the 53C710 transfers data to/from memory, the value stored in this register is loaded into the counter. Disable the timeout feature by writing 00<sub>16</sub> to this register.

The unit time base for this register is 16\* BCLK input period. For example, at 50 MHz the time base for this register is 16 x 20 ns = 320 ns. If a timeout of 50 μs is desired, this register should be loaded with a value of 9D<sub>16</sub>.

The minimum timeout value that should be loaded into this register is 02<sub>16</sub>; the value of 01<sub>16</sub> does not provide a reliable timeout period.

## 18.40 DMA Watchdog Timer (DWT)

Figure 18–38 DMA Watchdog Timer (DWT)

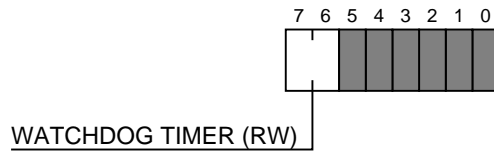


Table 18–43 DMA Watchdog Timer Description

Field	Description
7:6	<b>WATCHDOG TIMER</b> <i>[read/write]</i>

## 18.41 DMA Control Register (DCNTL)

Figure 18–39 DMA Control Register (DCNTL)

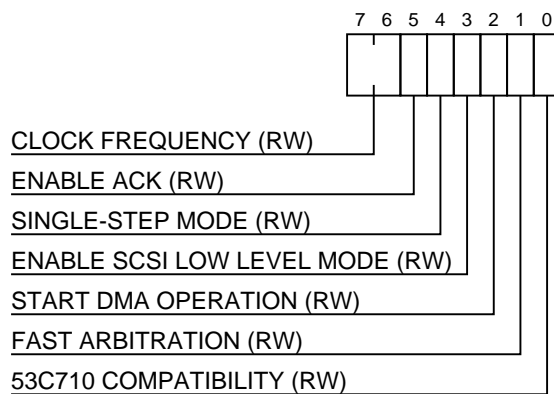


Table 18–44 DMA Control Register Description

Field	Description
7:6	<b>CLOCK FREQUENCY</b> <i>[read/write]</i>

(continued on next page)

## 18.41 DMA Control Register (DCNTL)

**Table 18–44 (Cont.) DMA Control Register Description**

Field	Description																				
	(CF1—CF0) These two bits determine the SCLK prescale factor used by the 53C710 SCSI core; the internal SCSI clock is derived from the externally applied SCLK. These bits are programmed as follows:																				
	<table border="1"> <thead> <tr> <th>CF1</th> <th>CF0</th> <th>SCSI Core Clock</th> <th>SCLK Frequency</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>1</td> <td>SCLK + 3</td> <td>50.01 - 66.67 MHz</td> </tr> <tr> <td>0</td> <td>0</td> <td>SCLK + 2</td> <td>37.51 - 50.00 MHz</td> </tr> <tr> <td>0</td> <td>1</td> <td>SCLK + 1.5</td> <td>25.01 - 37.50 MHz</td> </tr> <tr> <td>1</td> <td>0</td> <td>SCLK + 1</td> <td>16.67 - 25.00 MHz</td> </tr> </tbody> </table>	CF1	CF0	SCSI Core Clock	SCLK Frequency	1	1	SCLK + 3	50.01 - 66.67 MHz	0	0	SCLK + 2	37.51 - 50.00 MHz	0	1	SCLK + 1.5	25.01 - 37.50 MHz	1	0	SCLK + 1	16.67 - 25.00 MHz
CF1	CF0	SCSI Core Clock	SCLK Frequency																		
1	1	SCLK + 3	50.01 - 66.67 MHz																		
0	0	SCLK + 2	37.51 - 50.00 MHz																		
0	1	SCLK + 1.5	25.01 - 37.50 MHz																		
1	0	SCLK + 1	16.67 - 25.00 MHz																		
	Note that it is important that these bits be set to the proper values to guarantee that the 53C710 meets the SCSI timings as defined by the ANSI specification. These bits affect both asynchronous and synchronous timings (unless the synchronous clock is decoupled via the SBCL register).																				
<b>5</b>	<b>ENABLE ACK</b> <i>[read/write]</i>  (EA) Setting this bit causes the STERM-TA pin to become bidirectional. As a result, the 53C710 generates STERM-TA during slave accesses. When this bit is clear, the 53C710 monitors STERM-TA to determine the end of a cycle. This bit takes effect during the cycle in which it is set; setting this bit must be the first I/O performed to the 53C710 if this feature is desired.																				
<b>4</b>	<b>SINGLE-STEP MODE</b> <i>[read/write]</i>  (SSM) Setting this bit causes the 53C710 to stop after executing each scripts instruction, and generate a scripts step interrupt. When this bit is clear the 53C710 does not stop after each instruction; instead it continues fetching and executing instructions until an interrupt condition occurs. For normal SCSI scripts operation, this bit should be clear. To restart the 53C710 after it generates a script Step interrupt, the ISTAT and DSTAT registers should be read to clear the interrupt and then the START DMA bit in this register should be set.																				
<b>3</b>	<b>ENABLE SCSI LOW LEVEL MODE</b> <i>[read/write]</i>  (LLM) Setting this bit places the 53C710 in low level mode. In this mode, no DMA operations can occur, and no script instructions can be executed. Arbitration and selection may be performed by setting the Start Sequence bit as described in the SCNTL0 register. SCSI bus transfers are performed by manually asserting and polling SCSI signals. Clearing this bit allows instructions to be executed in SCSI scripts mode.																				
<b>2</b>	<b>START DMA OPERATION</b> <i>[read/write]</i>  (STD) The 53C710 fetches a SCSI scripts instruction from the address contained in the DSP register when this bit is set. This bit is required if the 53C710 is in one of the following modes:  <ol style="list-style-type: none"> <li>1. Manual start mode - Bit &lt;0&gt; in the DMODE register is set</li> <li>2. Single-step mode - Bit &lt;4&gt; in the DCNTL register is set</li> </ol> When the 53C710 is executing scripts in manual start mode, the Start DMA bit needs to be set to start instruction fetches, but does not need to be set again until an interrupt occurs. When the 53C710 is in single-step mode, the Start DMA bit needs to be set to restart execution of scripts after each single-step interrupt.																				
<b>1</b>	<b>FAST ARBITRATION</b> <i>[read/write]</i>																				

(continued on next page)

Table 18–44 (Cont.) DMA Control Register Description

Field	Description
0	<p>(FA) When this bit is set, the 53C710 becomes bus master immediately after receiving a bus grant, saving one clock cycle of arbitration time. When this bit is clear, the 53C710 follows the normal arbitration sequence.</p> <p><b>53C710 COMPATIBILITY</b> <i>[read/write]</i></p> <p>(COM) When this bit is clear, the 53C710 behaves in a manner compatible with the 53C700; selection/reselection IDs is stored in both the LCRC and SFBR registers, and auto switching is enabled.</p> <p>When this bit is set, the ID is stored only in the LCRC register, protecting the SFBR from being overwritten if a selection/reselection occurs during DMA register to register operations. The default condition of this bit (clear) causes the 53C710 to act the same as the 53C700, which does not support register to register operations. When this bit is set, auto switching is disabled.</p>

## 18.42 Adder Sum Output (ADDER)

This 32-bit register contains the output of the internal adder, and is used primarily for test purposes.

Figure 18–40 Adder Sum Output (ADDER)

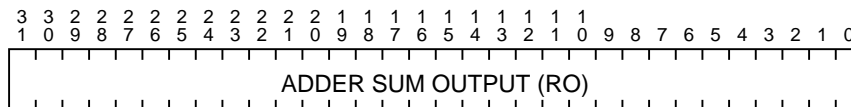


Table 18–45 Adder Sum Output Description

Field	Description
31:0	<b>ADDER SUM OUTPUT</b> <i>[read-only]</i>

### 18.42.1 SCSI Script RAM Buffer

On the I/O module, 128 kbytes of buffer memory are provided as a shared resource for the five SCSI/DSSI controllers. This memory is longword addressable by the system processor using the mailbox mechanism for secondary bus access. Bit <6> of the MAILBOX DATA STRUCTURE command field is used to select the SCSI script RAM for access by the processor. Bits <16:2> of the secondary bus address field are used to select an individual longword in the buffer for access. Bits <3:0> in the mask field of the mailbox data structure are used to selectively disable bytes within the longword. Setting a bit in the MASK field disables the corresponding byte in the longword for writing.

During DMA operations by the SCSI controllers, bits <31> and <30> of the address are used to select between main memory and SCSI script RAM. When bit <31> is a 1 and bit <30> is a 0, the SCSI script RAM is selected as the source or target of the DMA operation.



## 18.42 Adder Sum Output (ADDER)

### 18.42.2 Ethernet Station Address ROM

The Ethernet station addresses for both network interfaces are contained in 64 bytes of read only memory. This ROM is organized as byte locations on longword boundaries. The first 32 locations contain the network address for Ethernet interface 0 and the second 32 locations contain the address for Ethernet interface 1.

Bit <10> in the Lbus mailbox data structure control field selects the Ethernet Address ROM for access while bits <7:2> select the location within the ROM. Data from the ROM appears on bits <7:0> of the mailbox data structure Read Data field.

### 18.42.3 Serial Control Bus Interface

The Phillips PCD8584 provides an interrupt driven master interface to the serial control bus. The DEC 4000 AXP CPU uses this interface to communicate with the slave interfaces on the operator control panel (OCP) and power system controller (PSC), and with the 256x8 error log EEPROM devices on the memory, I/O, and CPU modules.

Bit <11> in the Lbus mailbox data structure control field selects the PCD8584 for access, while bit 2 of the Secondary Bus Address field selects one of the two registers within the device. Bits <7:0> of the mailbox data structure Write Data and Read Data fields are used for write and read operations respectively. The mask field of the mailbox data structure has no effect. Figure 18–41 shows the mapping of the mailbox data structure Secondary Bus Address field to the PCD8584 registers. When the PCD8584 device interrupts, bit <35> in the LINT register is set and the processor interrupt CIRQ\_I(0) is asserted.

Figure 18–41 Serial Control Bus Interface Register Map

31	0	address
reserved	DATA	0000 0000
reserved	CTRL	0000 0004

I2C\_CSR

### 18.42.4 FEPR0M

The I/O module contains 512 kilobytes of flash-erase programmable read only memory (FEPR0M), organized as 128 K longword locations on the Lbus. The FEPR0M is for console firmware storage and provides the ability to erase and reprogram this firmware in the field without the removal of the I/O module. The FEPR0M locations are accessible only through the mailbox mechanism. No support for direct execution of instructions from the FEPR0M is provided.

Bit <12> in the LBUS MAILBOX DATA STRUCTURE command field selects the FEPR0M for access, bits <18:2> select the longword location within the FEPR0M. FEPR0M erase and program operations are accomplished using the standard mailbox read and write operations.

# Part IV

---

## The System Bus

This part contains a detailed functional description of the DEC 4000 system bus.

---

## The DEC 4000 System Bus

The DEC 4000 system bus is a shared-memory, synchronous, multiplexed bus designed to support the Digital Alpha AXP architecture. This bus is a 143-conductor parallel bus that is routed across the system backplane. It connects up to two central processor modules, up to four memory modules, and an I/O module .

Each module on the system bus contains a unique system bus interface unit (BIU). The BIU on the I/O module interfaces the IO module to the system bus through coherent cache line buffers. The BIU on the memory modules interface the 128-bit wide system bus to the 280-bit wide error detection and correction protected memory arrays.

Each processor module contains one DECchip™ 21064 CPU chip. In addition, each processor module has a primary cache consisting of a write-through data cache, and a virtual instruction cache. It also contains a secondary direct-mapped write-back instruction and data cache (backup cache ) with a system bus interface. Arbitration logic and the system bus clock generators are located on the CPU<sub>0</sub> module.

The memory connectors provide a unique slot identification code to each memory which is used to configure the CSR registers address space. The connector for CPU<sub>1</sub> provides an identification code that disables the clock drivers and configures the CSR address space.

### 19.1 Supported Transactions

- The bus supports four transaction types:
  - Null
  - Read
  - Write
  - Exchange
- Provides low latency service to the CPU's cache miss transactions and I/O module read transactions.
- Provides sufficient bandwidth to the I/O module to allow local I/O to operate at full bandwidth and the Futurebus+ to operate at no less than 100 MB/s.
- The system bus protocol timing is capable of scaling with improvements in memory performance.

### 19.2 Address Space

Noncacheable address space is partitioned into primary and secondary sections. The primary sections are registers which are local to the system bus and respond within 6 to 10 cycles. Secondary registers are accessible via a mailbox pointer register and a system memory based mailbox data structure.

## 19.3 System Bus Transactions

### 19.3 System Bus Transactions

The system bus is a 128-bit, non-pended synchronous bus with multiplexed data and address lines and centralized arbitration.

All the components that interface with the system bus use either PHI1 to PHI1 or PHI1 L to PHI1 L single phase edge to edge clock signals supplied by the CPU<sub>0</sub> module to the backplane connectors for driving and receiving system bus signals.

#### 19.3.1 Bus Arbiter

The CPU<sub>0</sub> module is the arbiter and the default bus master. The following four transactions are supported on the system bus:

- READ, data-stream or instruction-stream read
- WRITE, Write unmasked with good or bad data
- EXCHANGE, Write unmasked with good or bad data, with read data return
- NULL, Null Transaction

#### 19.3.2 Transaction Process

Transactions begin with the arbitration controller selecting a current commander, requesting use of CPU node's backup caches, and asserting the address and command cycle enable signal in cycle 0.

The commander drives a valid address, command, and parity in cycle 1.

Every system bus node checks address and command parity and logs a fault and reports a fault via the hard error interrupt.

A commander may stall in cycle 2 before supplying write data and parity in cycles 2 and 3 along with an indication of the goodness of the write data. Read data is received in cycles 5 and 6 along with an indication of the goodness of the data.

Bystanders do not check data parity, because the responder or commander will confirm the data either through direct acknowledgment or a hard error interrupt.

The addressed responder confirms the cycle by asserting the acknowledge signal two cycles later.

The commander checks for the acknowledgment and regardless of the presence or absence, completes the number of cycles specified for the transaction.

Commanders do not abort transactions prior to the completion of the specified number of cycles for each transaction type.

The following list gives non-stalled transaction cycle times. Allow for at least one arbitration request cycle.

- READ, 7 cycles
- WRITE, 6 cycles
- EXCHANGE, 7 cycles
- NULL, 7 cycles

Cached nodes must provide probe results by cycle 3.5. A responder or bystander may stall any transaction in cycle 4 by asserting a stall signal in cycle 3, at the cycle 3.5 clock edge. The system bus will stall as long as the stall signal is asserted. Arbitration is overlapped with the last cycle of a transaction, such that tristate conflict is avoided.

## 19.4 Cache Protocol

The cache protocol is termed to be write-update. A node may choose to write invalidate by following the write transaction protocol. Memory is written on every system bus write. Two system bus signals, CSHARED L and CDIRTY L, allow a node to keep its cache coherent with all other nodes by following the policy described in this chapter.

All nodes which implement caches, monitor the system bus and probe their tag stores during all memory address space references. If the result of this probe indicates that a given node has a valid copy of the referenced cache line, the node must assert CSHARED L or, alternatively, can invalidate a line on writes that hit.

If the bus operation is a read and the target cache line is found to be dirty on a given node's cache, the CDIRTY L response must be provided. CPU nodes and/or the I/O node may provide CSHARED L responses to a system bus transaction, but the protocol ensures that only one CPU node can return a CDIRTY L response. The CDIRTY L response obligates the responding node to supply the read data to the system bus, because the memory copy is stale and the memory controller aborts the return of the read data.

A CPU that wants to access its cache considers a tag probe for read satisfied if the tag matches and the valid bit is set. A tag probe for write is satisfied if the tag matches, the valid bit is set and the shared bit is clear. A line is considered dirty if the dirty bit is set and considered clean if the dirty bit is clear.

Writes always clear the dirty bit of the referenced cache line in both the commander node and all nodes that take the update.

System bus memory transaction addresses which hit in a lock address register (which was loaded by a LDx\_L instruction) must return a probe result of SHARED, even if the line is not valid in the primary and backup cache. This forces writes to the line to be broadcast, and STxC instructions to function correctly.

A node has two options when a system bus transaction is a write and the block is found to be valid in its cache. A node should either invalidate the line or accept the line and update its copy, keeping the line valid.

If the commander node is the I/O module, the write is accepted only if the probe of the backup cache tag results in a hit (I/O update always) or if the probe of the backup cache tag and the primary cache duplicate tag hits (I/O conditional update).

Each CPU node that implements caches stores the following (for each 32 byte cache line):

- A tag consisting of some number of physical address bits
- A valid bit indicating whether or not this line can be considered
- A shared bit indicating whether or not this line may also be resident in another node's cache
- A dirty bit indicating whether or not this line has been written to by this node

A CPU running transactions to its cache and to system memory will affect the state of its cache line and the cache line on another CPU or I/O bystander node.

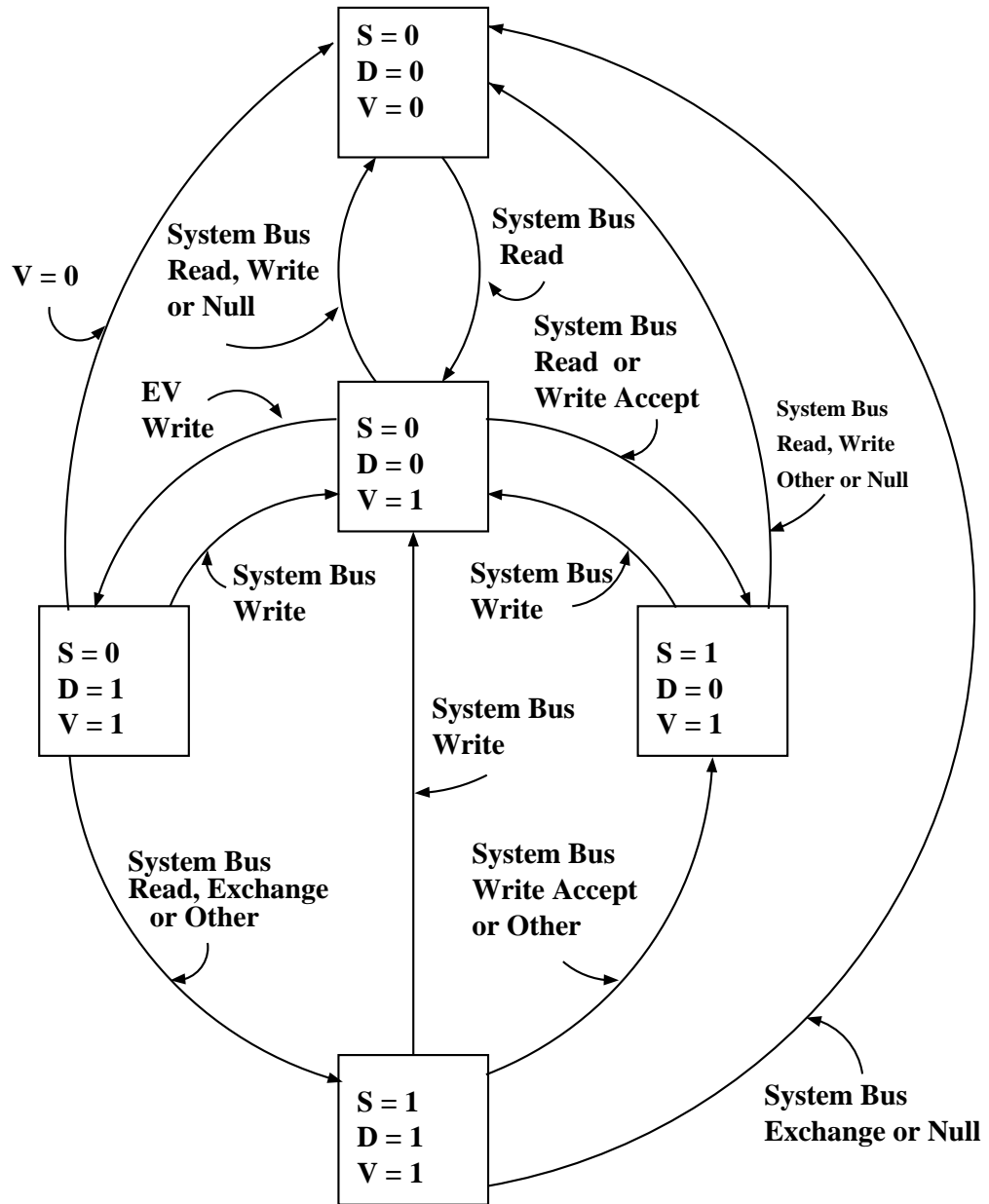
## 19.4 Cache Protocol

This snooping protocol assumes that the processor's primary cache is always maintained as a subset of the backup cache by performing invalidation on victimized lines and on I-stream reads that hit in the duplicate tag of the primary cache. Nodes are required to perform system bus write transactions if a line is valid and dirty but the tag does not match the address for the given request. In this case the line must be written back to memory.

The exchange transaction is provided for this victim write. While a victim line is pending system bus arbitration, the node must accept all read and write references. The I/O node implements an invalidate on write policy, and will signal shared status to reads which hit on a line it has buffered.

Figure 19-1 summarizes cache state transitions.

Figure 19-1 Protocol Transition Summary



## 19.5 System Bus Signals

### 19.5 System Bus Signals

A system bus interface includes a multiplexed address and data path, protocol control signals, clocks, and system control signals. All system bus control signals and parity signals are true in the low voltage state. All system bus address, and data signals are true in the high voltage state. This section begins by listing all the signals. The remainder of the section provides a description of all the pins along with connector pinouts.

The maximum system bus cycle time is 24 ns. The system bus supports up to seven nodes, data and parity signals are single stubbed with one stub per bus node. Selected control signals are double stubbed on all system bus modules.

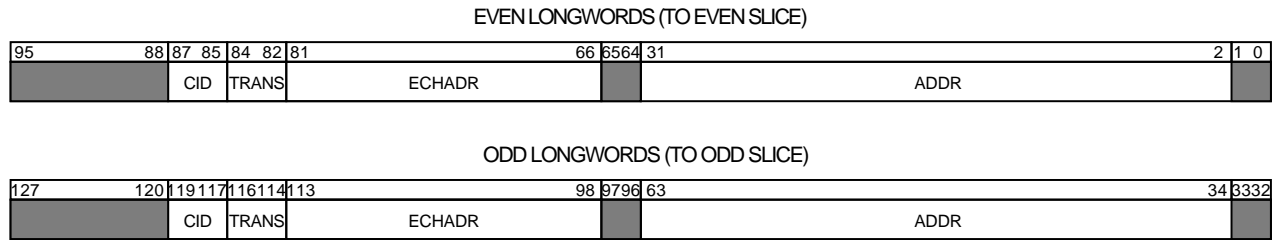
Table 19–1 shows the system bus signals and which nodes connect to each signal. The driver type is defined as pull-pull (PP), open-drain terminated (OD), or PECL (Positive ECL differential).

The connector pins are assigned to enable quadword partitioned gate arrays to be accessed by the aligned quadword on a hexword boundary.

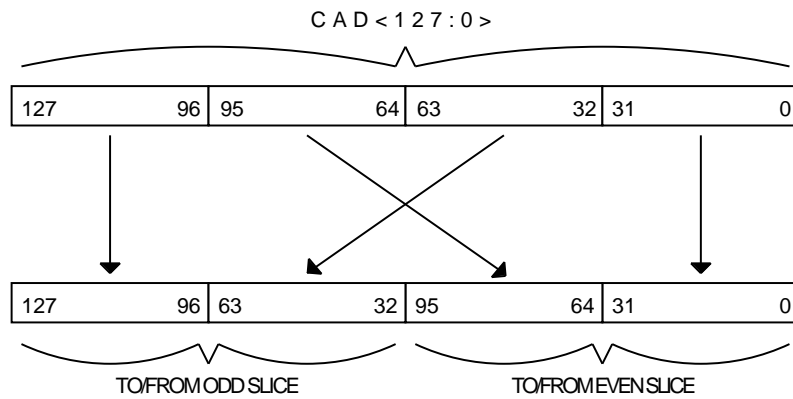


## 19.5 System Bus Signals

**Figure 19–2 Command-Address Cycle Format, Data Longword Shuffle**



- NOTES: 1. SHADED AREAS SHALL BE DRIVEN HIGH – MUST BE ONE.  
2. ODD LONGWORDS ARE DUPLICATIONS OF THE EVEN LONGWORDS.



**Table 19–1 System Bus Signals (143 Total)**

Signal Name	Total Number		Driver	Function
	P1,P2,M,IO			
CAD<127:0>	128	B, B, B, B	PP	Address and command or data
CA L	1	O, I, I, I	PP	Command cycle strobe
CSTALL0 L	1	B, B, B, B	OD	Responder or bystander stall, even slice
CSTALL1 L	1	B, B, B, B	OD	Responder or bystander stall, odd slice
CXACK L	1	B, B, B, B	PP	Responder command and address and write data acknowledge
CDIRTY L	1	O, O, I, -	OD	Memory data is stale

(continued on next page)

## 19.5 System Bus Signals

Table 19–1 (Cont.) System Bus Signals (143 Total)

Signal Name	Total		Driver	Function
	Number	P1,P2,M,IO		
CSHARED L	1	B, B, -, O	OD	Memory data is cached
CPARITY L<3:0>	4	B, B, B, B	PP	Odd longword parity for address and command or data
IOREQ L	1	I, -, -, O	PP	IO arbitration request
IOG L	1	O, -, -, I	PP	IO arbitration grant
CPUREQ L	1	I, O, -, -	PP	CPU <sub>1</sub> arbitration request
CPUG L	1	O, I, -, -	PP	CPU <sub>1</sub> arbitration grant
BCREQ L	1	O, I, -, -	PP	CPU <sub>0</sub> to CPU <sub>1</sub> backup cache arbitration request
<b>Interrupt and Error Signals (6 total)</b>				
C_ERR L	1	B, B, O, O	OD	Interrupt for hard or soft errors in the memory, CPU, or I/O subsystems
CIRQ L<1:0>	2	I, I, -, O	PP	I/O module interrupt requests
CINT_TIM	1	I, I, -, O	PP	1MS interval timer clock, distributed from I/O to both CPU connectors
CSYS_EVENT L	1	I, I, -, O	OD	Non-maskable interrupt to force the processor to return to console mode
CUCERR L	1	B, B, B, B	OD	Transaction hard error
<b>Clocking and Initialization Signals(8 total)</b>				
PHI1	1	I, I, I, I	PECL	Clock, driven from the CPU <sub>0</sub> clock connector
PHI3	1	I, I, I, I	PECL	Clock, driven from the CPU <sub>0</sub> clock connector
PHI1 L	1	I, I, I, I	PECL	Clock, driven from the CPU <sub>0</sub> clock connector
PHI3 L	1	I, I, I, I	PECL	Clock, driven from the CPU <sub>0</sub> clock connector
CRESET L	1	B, I, I, I	PP	System RESET - return to a power-up state
CPU <sub>1</sub> ID L	1	-, I, -, -	-	CPU <sub>1</sub> slot identification
MID<1:0>	2	-, -, I, -	-	Memory slot identification
<b>CPU Clock Connector (1 total)</b>				
ASYNC_RESET L	1	I, -, -, -	-	DC power unstable, supplied from the power subsystem controller to the CPU <sub>0</sub> clock connector
CHALT L	1	I, -, -, -	-	Operator control panel halt switch to the CPU <sub>0</sub> node clock connector
<b>TOTAL System Bus Connector Signal Pins</b>	157			

### 19.5.1 Command and Address Format

**Table 19–2 Command and Address Format**

Signal	Number	Function
<b>Command and Address Format</b>		
RESERVED<1:0>	2	Unassigned signals, must be driven high
ADDR<31:2>	30	Transaction aligned octaword address, corresponds to CPU address<33:4>
RESERVED<65:64>	2	Unassigned signals, must be driven high
ECHADR<81:66>	16	Exchange transaction cache line tag and index for write data address
TRANS<84:82>	3	Encodes transaction type
CID<87:85>	3	Encodes the identification of the commander
RESERVED<95:88>	8	Unassigned signals, must be driven high
RESERVED<33:32>	2	Unassigned signals, must be driven high
ADDR<63:34>	30	Transaction aligned octaword address, corresponds to CPU address<33:4>
RESERVED<97:96>	2	Unassigned signals, must be driven high
ECHADR<113:98>	16	Exchange transaction cache line tag and index for write data address
TRANS<116:114>	3	Encodes transaction type
CID<119:117>	3	Encodes the identification of the commander
RESERVED<127:120>	8	Unassigned signals, must be driven high
<b>Total Address and Command Pins:</b>	<b>128</b>	

#### Note

For each command cycle, CAD <95:64 and 31:0> and CPARITY L <2 and 0> are replicated on CAD <127:96 and 63:32> and CPARITY L <3 and 1> by the commander node. RESERVED fields carry no information during the address and command cycle, but are SUSTAINED at a high voltage level by backplane pull-up resistors and are included in the parity generation and checking. Maintaining a high level on these reserved signals ensures that they do not switch during the address and command cycle. The pulled high state of CAD<127:0> has incorrect CPARITY L<3:0> signals.

### 19.5.2 Arbitration Signals

The system bus protocol can support three arbitrated nodes, which consist of two processors on a CPU module and one I/O module. Memory modules cannot be commanders and hence do not arbitrate for bus usage. The arbiter is built into the CPU<sub>0</sub> module so that the CPU<sub>0</sub> can be a default commander of the system bus. Arbitration cycles occur in parallel with data transfer cycles.

The following sections describe the system bus arbitration signals.

## 19.5 System Bus Signals

### IOREQ L

The I/O module asserts IOREQ L on any TPHI1 clock edge, when it wants to drive the system bus. IOREQ L is a unidirectional signal from the I/O module to the CPU<sub>0</sub> module. This signal is sampled by a NOT TPHI1 edge 0.5 cycle before CPUG L is asserted on the next TPHI1 edge. Once asserted, IOREQ L is not negated until one cycle after an IOG L has deasserted. If IOREQ L is held asserted, IOG L could assert in the next arbitration cycle or the second arbitration cycle.

### IOG L

The signal IOG L asserts on a TPHI1 edge in the cycle before 0, 0.5 cycle after it was sampled, to grant the system bus to the IO module, the arbiter asserts CA L in cycle 0, and the I/O module drives the address in cycle 1. IOG L is negated on the first cycle 2, to be sampled 0.5 cycle later. IOG L is a unidirectional signal driven from CPU<sub>0</sub> to the I/O module.

### CPUREQ L

The CPU<sub>1</sub> module asserts CPUREQ L on a TPHI1 edge when it wants to drive the system bus. CPUREQ L is a unidirectional signal from the CPU<sub>1</sub> module to the CPU<sub>0</sub> module. This signal is sampled by a NOT TPHI1 edge 0.5 cycle before CPUG L is asserted on the next TPHI1 edge. Once asserted, CPUREQ L is not negated until one cycle after CPUG L has deasserted. If CPUREQ L is held asserted, CPUG L could assert in the next arbitration cycle or the second arbitration cycle.

### CPUG L

The signal CPUG L asserts on a TPHI1 edge in the cycle before 0, 0.5 cycle after it was sampled, to grant the system bus to the CPU<sub>1</sub> module, the arbiter asserts CA L in cycle 0, and the CPU<sub>1</sub> module drives the address in cycle 1. CPUG L is negated on the first cycle 2, to be sampled 0.5 cycle later. CPUG L is a unidirectional signal from the CPU<sub>0</sub> module to the CPU<sub>1</sub> module.

For CPU<sub>0</sub>, the arbitration signals are not visible to the system bus. The default state of the system bus is to be owned by the CPU<sub>0</sub> module.

### BCREQ L

This signal is sent from the CPU<sub>0</sub> arbitration controller on a TPHI1 edge one cycle prior to cycle 0 to the CPU<sub>0</sub> and CPU<sub>1</sub> nodes to cause the backup caches to be requested for use by the probe address which is available two cycles later. This signal is sampled .5 cycle after assertion. The arbiter negates this signal one cycle after cycle 1, to allow a CPU node a choice of releasing its backup cache to the processor if CSTALL L is asserted to cause multiple cycle 2(s), BCREQ L is to be sampled .5 cycle later. This signal must be negated during assertion of CRESET L.

### 19.5.3 Protocol Signals

#### CAD<127:0>

CAD <127:0> are multiplexed between address and data information. Starting at cycle 0.5 to the end of cycle 1 CAD <127:0> represent address, command, and commander identification information. For write cycles 2 and 3 and read cycles 5 and 6 they represent 128 bits of write or read data in each cycle, to complete a 256 bit data transfer.

The system bus command and address cycle is used by a commander to initiate a system bus transaction. During command and address cycles the address is driven as shown in Figure 19-2. A commander drives the RESERVED fields high (MBO - must be one) to ensure proper parity generation and to reduce switching noise.

During write data cycles the commander drives data on CAD<127:0> twice. The full 256 bits of data are written, because all memory and noncacheable address space write transactions are full hexword. During Read Data Return and Read Data Error cycles the responder drives the data on CAD 127:0.

During data cycles the data is driven as shown in Figure 19-2. A commander which does not receive acknowledgment to an initiated transaction will abort by completing the system bus protocol in the specified number of transaction cycles. The commander is responsible for reporting this fault to the processor via a hard error interrupt.

### 19.5.4 Address Field

The address space supported by the system bus is divided into memory space and noncacheable space. The 21064 CPU can generate 34-bit addresses. This system uses the 34-bit address. CPU Address bit 33 (which is the same as system bus CAD<31> and CAD<63>) distinguishes memory space from noncacheable space.

The most significant bit of this address (corresponding to lines CAD<31>) or CAD<63> selects 8 GB noncacheable space (CAD<31,63> = 1) or 8 GB memory space (CAD<31,63> = 0).

The low 2 GB of noncacheable space is called the primary I/O address space and the other 6 GB space is reserved. The primary I/O address space contains all registers which are immediately accessible to the system bus. Device registers on the I/O module, called secondary registers, are accessed via hardware assist and a system memory based mailbox data structure.

#### **ECHADR<15:0>**

The exchange address (ECHADR <15:0>) is the cache line tag address field for the victim write address. This field corresponds to CPU address bits<33:18> and system bus exchange address fields CAD<81:66> and <113:98> of the memory address for the victim write data. This tag field is decoded to select a memory CAS strobe. The concatenation of the tag and index fields form the unique victim memory address as shown in Figure 19-3.

For the 1-MB cache configuration, physical address bits<19:5> are used to select one of 32968 cache lines, requiring the tag address bits to include physical address bits<33:20>, which correspond to ECHADR<15:2> and system bus fields CAD<81:69> and <113:100>. ECHADR<1:0> are not used in this configuration, as they duplicate the two high order index field bits.

For the 4-MB cache configuration, physical address bits<21:5> are used to select one of 131872 cache lines, requiring the tag address bits to include physical address bits<33:22>, which correspond to ECHADR <15:4> and system bus fields CAD <81:70> and <113:102>. ECHADR <3:0> are not used in this configuration, as they duplicate the four high order index field bits.

## 19.5 System Bus Signals

**Figure 19–3 Exchange Address Layout**

	8	4	2	1	512	256	128	64	32	16	8	4	2	1	512	256	128	64	32	16	8	4	2	1	512	256	128	64	32	16	N/U	N/U
	GB	GB	GB	GB	MB	MB	MB	MB	MB	MB	MB	MB	MB	MB	KB	KB	KB	KB	KB	KB	KB	KB	KB	KB	KB	KB	KB	KB	KB	KB	B	B
PHYS ADDR	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
CAD ADDR	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ECHADR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																

CACHE SIZE	Cache Tag Bits																Cache Index Address				Index Address Bits (CAD Format)															
4 MB	11	10	9	8	7	6	5	4	3	2	1	0	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3							

CACHE SIZE	Cache Tag Bits																Cache Index Address		Index Address Bits (CAD Format)															
1 MB	13	12	11	10	9	8	7	6	5	4	3	2	1	0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3					

NOTE: If a tag field is smaller than 16 bits, unused ECHADR bits must be filled from the corresponding cache index address bits.

ECHAD2

### TRANS<2:0>

The TRANS<2:0> lines specify the current bus transaction type or command type during the address and command cycle 1. The RESERVED code is not acknowledged, but could be stalled and no errors shall be reported, and the transaction is dismissed. The arbiter assumes a RESERVED transaction takes a minimum of 7 cycles to complete before re-arbitrating. Table 19–3 shows how the three bits are interpreted.

**Table 19–3 TRANS<2:0> Bit Interpretations**

Levels	Abbreviation	Bus Transaction	Type	Function
000	RD	READ	READ	Request a hexword read
001	TRD	READ	READ	Request exclusive hexword read, invalidate
010	XD	EXCHANGED WRITE - READ	WRITE-READ	Write victim to memory and read a replacement cache line
011	-	Reserved		Responders and bystanders perform passive dismiss
100	WR	WRITE	WRITE	Request a hexword write
101	-	Reserved		Responders and bystanders perform passive dismiss
110	-	Reserved		Responders and bystanders perform passive dismiss
111	NUT	NULL TRANSACTION	NO ACTION	Dismiss bus arbitration

### CID<2:0>

During the command and address cycle, CID<2:0> contain the commander's ID. This ID is used to identify the source of the request during cycle 1 and to associate data errors with the commander which issued the request. The CPU module backup cache write policy distinguishes processor nodes from an I/O node to effect the write accept policy.

Each commander node on the system bus (memory modules are not commander nodes) may have one transaction outstanding. The system bus does not support more than three commander nodes.

The commander ID codes available for use by a node are shown in the following table. CID<2:0> indicate which node originated the transaction. Table 19–4 lists the commander CID assignments.

**Table 19–4 Commander CID Assignments**

Node Name	CID<2:0>
CPU <sub>0</sub> _NODE	001
CPU <sub>1</sub> _NODE	010
IO_NODE	100
RESERVED	all others

### CPARITY L< 3:0>

The CPARITY L<3:0> is computed over each longword of the 128-bit system bus. Odd parity is used, where the "exclusive OR" of all bits including the parity bit is a "0".

When the bus is idle, the system bus backplane ensures that the bus defaults to a pulled up level. The high levels cause incorrect parity to be guaranteed on an idle bus. If a device is granted the bus but chooses not to use it during a given cycle, it is responsible for driving the CAD<127:0> to valid levels and CPARITY L<3:0> with correct parity. Table 19–5 shows the CAD parity coverage.

## 19.5 System Bus Signals

Table 19–5 CAD Parity Coverage

Parity bit	protected data	parity_type
CPARITY L<3>	CAD<127:96>	odd parity
CPARITY L<2>	CAD<95:64>	odd parity
CPARITY L<1>	CAD<63:32>	odd parity
CPARITY L<0>	CAD<31:0>	odd parity

### 19.5.5 Response and Interrupt Signals

#### CA L

The CA L signal is driven only by the system bus arbiter. All nodes receive CA L to identify cycle 0 as the start of a new transaction. The address and command must be driven by the granted commander in cycle 1, following cycle 0. There are cases where the next arbitration grant, which defines the next CA L assertion will overlap the last cycle of a transaction. However tristate overlap is ensured to be avoided by the transaction timing.

#### CSTALL0 L and CSTALL1 L

System bus transactions can be stalled an integral number of cycles by writing or exchanging commanders to memory address space in cycle 2 or by responders or bystanders in cycle 4 by asserting CSTALL0 L and CSTALL1 L in cycle 1.5 or cycle 3.5.

Commander driven stalls in cycle 2, with CSTALL L driven not later than cycle 1.5, allow a cached node to read data from the cache, merge it with processor write data, and then continue the write to memory.

Stall is not asserted by a responder or bystander node later than cycle 3.5. A bystander node does not stall any noncacheable address space reference. This provides per transaction flow control. The negation in cycle 2.5 or 4.5 enables entry to cycle 3 or 5, 1.5 cycle later to continue a transaction. Arbitration request signals may assert during stalled cycles.

The bus interface chips are quadword sliced. One copy is for the even slice and the other copy is for the odd slice.

#### CXACK L

In response to each address and command cycle, a responder is required to acknowledge it had received the command and address with good parity by asserting CXACK L two cycles after cycle 1, regardless of the CSTALL L state, this could be cycle 2 or cycle 3.

A bystander must check parity for all address and command cycles, and if an error is detected, log the error and notify a CPU via the C\_ERR L interrupt. Bystanders should not check data parity.

Responders must also acknowledge the reception of write data with good parity, by asserting CXACK L two cycles after the write data is received. That is in cycle 4 and cycle 5 in no CSTALL L assertion, or in the multiple cycle 4(s) if CSTALL L is asserted.



**CXACK L** is *not* used for flow control. Commanders must check for the **CXACK L** response and complete the transaction cycle flow. If **CXACK L** assertion is not detected in the correct cycles, the commander will complete the transaction, log the failing event(s) and notify a CPU via the **C\_ERR L** interrupt. Address and command cycles to nondecoded address spaces are cycled through the transaction by the commander without acknowledgment.

### **CDIRTY L**

**CDIRTY L** is asserted by a CPU bystander from cycle 3.5 to 4.5 if it contains the referenced valid memory line that has been modified but not yet written back to memory. **CDIRTY L** does not assert during noncacheable address space transactions. **CDIRTY L** could change state in cycles 2 and 5. Drivers must not be enabled in cycle 0. During a read, exclusive read, or exchange, a cache that asserts **CDIRTY L** will supply the read data as a responder with at least one slip cycle, and the target memory will, upon seeing **CDIRTY L**, abort the read data return.

The target memory must be capable of disabling its driver in cycle 4 as the responding cache enables its driver and returns data in cycle 5. **CDIRTY L** can never assert during a write transaction. A commander which writes a shared and dirty line to memory marks the line clean because memory updates on all writes. The coherence protocol ensures that at most one cache can assert **CDIRTY L**. The I/O module is not concerned with the dirty state of a cache line, due to its invalidate on detected write policy.

### **CSHARED L**

Based on the shared state of the cache line, the commander determines whether it will do a write-through when the contents of the memory line are written. **CSHARED L** is asserted from cycle 3.5 to 4.5 by a CPU or I/O bystander that contains the referenced valid memory line during any transaction to be sampled at the end of cycle 4.5. **CDIRTY L** does not assert during noncacheable address space transactions.

**CSHARED L** could change state in cycles 2 and 5, drivers must not be enabled in cycle 0. In addition, a CPU node must respond with shared status for a system bus read of a cache line which has a valid lock address and lock flag. Both CPU nodes may assert **CSHARED L** as the I/O node performs a transaction. During a write transaction, the bystander CPU probe result may assert **CSHARED L** to signal that it will accept the write, and the commander must retain the shared state of the line. Alternatively, the bystander CPU must not assert the **CSHARED L** signal if it invalidates the cache line. Hence, CPU nodes can either accept and update writes or invalidate on writes.

The I/O node monitors transaction addresses and returns **CSHARED L** status for read transactions to a line it has buffered for merging with write data. For write transactions from another node to an I/O node buffered line, the line will be invalidated and read from the system bus again to obtain the most recent copy. This enables the I/O node to merge write data to a line which a CPU node may be polling.

A CPU cache line may be in one of five states as shown in Table 19–6.

## 19.5 System Bus Signals

**Table 19–6 CPU Cache Line States**

1.	NOT VALID		Cache line is invalid, miss.	
2.	VALID	NOT SHARED	NOT DIRTY	Valid for read or write. Cache line contains the only cached copy of the block which is identical to the memory copy.
3.	VALID	NOT SHARED	DIRTY	Valid for read or write. Cache line contains the only cached copy of the block which has been modified more recently than the memory copy.
4.	VALID	SHARED	NOT DIRTY	Valid for read or write, but write must broadcast to system bus. Cache line may be in some other cache, but the memory copy is identical.
5.	VALID	SHARED	DIRTY	Valid for read or write, but write must broadcast to system bus. Cache line may be in some other cache, but the contents have been modified more recently than the memory copy. This is a transitional state which occurs when arbitrating for the bus to broadcast a write. It is also a case when an unshared dirty line is returned to a system bus read transaction.

During all transactions other than noncacheable space transactions, all caches inspect their tag stores. Each cache returns one of three responses on the shared and dirty lines during cycle 3:

1	NOT SHARED	NOT DIRTY	Line not cached.
2	SHARED	NOT DIRTY	Line cached and clean.
3	SHARED	DIRTY	Line is cached and dirty. (Cannot occur for write transactions)

The commander interpretation of these responses is :

1	NOT SHARED	NOT DIRTY	No other cache contains the line.
2	SHARED	NOT DIRTY	One or more caches contain the line, but the memory copy is consistent.
3	SHARED	DIRTY	One or more caches contain the line, but the memory copy is stale.

### **C\_ERR L**

This synchronous signal is asserted in cycle 0 by a CPU node or cycles 0 or 1 by an I/O or memory node of a transaction, or whenever the system bus is idle, for a maximum of one cycle. The signal is sampled to cause an interrupt to both processor nodes. This interrupt is asynchronous to a system bus transaction that may have caused an error. This signal indicates a hard or soft error or a latent error to a CPU from a CPU, memory, or an I/O node. A node provides a soft error interrupt enable method in its CSRs if it is a source of soft errors. Latent errors

are defined as being reported to the CPU nodes after the associated system bus transaction has completed.

### **CIRQ L<1:0>**

These level sensitive interrupt request lines assert asynchronously from the I/O node to the CPU nodes. These are general device level interrupts. CIRQ L<1> is associated with the Futurebus+ and CIRQ L<0> is associated with I/O module resident device controllers. These interrupts are intended to have equivalent priority. System software can decide which CPU node is responsible for either or both of these signals.

### **CINT\_TIM**

This free running 1-ms interval timer clock signal is used to provide an interrupt to the CPU<sub>0</sub> and CPU<sub>1</sub> nodes, and to supply a free running clock to the processor chips for counting. It is driven from the I/O module, an edge is detected by the CPU nodes which then request an interrupt to the processor. The CPU<sub>1</sub> node inverts this signal before detecting the edge to cause an interrupt.

### **CSYS\_EVENT L**

This edge sensitive asynchronous interrupt is nonmaskable at the CPU nodes and indicates that one or more special action system events have occurred. These events could be console halt request, front panel restart, power supply power failure, etc. This signal is driven from the operator control panel, the I/O module, and the power system controller. This signal has a minimum pulse width of 200 ns and a maximum pulse width of 500 ns.

### **CUCERR L**

Per transaction unrecoverable data errors are reported via the CUCERR L signal. For read transactions CUCERR L asserts in cycle 5 or 6 with the octaword of bad data. For write transactions CUCERR L asserts in cycle 2 or 3 with the octaword of bad data. Also, CUCERR L is used for flow control of store conditional writes to the noncacheable address space mailbox pointer register and asserts in the first cycle 4 to be sampled by the commander at the end of the first cycle 4. The I/O responder may sustain the assertion of CUCERR L during multiple cycle 4 s, but negates CUCERR L no later than the end of the last cycle 4. CUCERR L is sampled by the commander to signify data errors at the end of cycle 5 or 6 for reads, and to signify STxC noncacheable address space write failures at the end of the first cycle 4. The responder samples CUCERR L to check the validity of the written data at the end of cycles 2 or 3 for writes.

### **CPU Clock Connector Initialization signal**

The power subsystem controller or the operator control panel notifies CPU<sub>0</sub> that DC system power is insufficient for normal operation or that a warm reset should occur by asserting ASYNC\_RESET L. This signal connects to CPU<sub>0</sub> through the clock connector.

### **ASYNC\_RESET L**

This asynchronous signal indicates that DC voltages are unstable for normal system operation, negation means DC voltages have been stable for a minimum of 30 ms. Assertion indicates that normal operation is not possible or that CRESET L should assert to perform a warm reset. To reset a powered system, the minimum assertion time must be greater than 10 ms to ensure all state logic returns to an initialized condition. DC voltages must be stable for 10 to 50 ms before ASYNC\_RESET L is negated, during a power-up sequence. The operator control panel delivers a TTL signal, RESET L, minimum pulse of 10 ms to the power system controller to cause a reset of the system. The power system control unit ensures

## 19.5 System Bus Signals

that `ASYNC_RESET L` is negated 10  $\mu$ s minimum before beginning to remove DC voltages.

### **CHALT L**

This asynchronous signal from the operator control panel halt switch is received by a buffer on the CPU<sub>0</sub> module and then OR tied to the non-maskable `CSYS_EVENT L` interrupt to notify CPU<sub>0</sub> and CPU<sub>1</sub>. This signal has a minimum pulse width of 200 ns and a maximum pulse width of 500 ns.

### 19.5.6 Clocking and Initialization Signals

The clocks are generated from the CPU<sub>0</sub> node and radially distributed to each system bus node.

#### 19.5.6.1 Synchronous Clock Signals

##### **PHI1 and PHI1 L**

These PECL level differential signals are one of two overlapping single phase clocks. The receiver node receives this clock into a noninverting and inverting PECL to CMOS converter to provide a clock (`TPHI1`) and not clock (`TPHI1 L`) to the system bus interface logic. This clock is used to time all system bus transactions.

##### **PHI3 and PHI3 L**

These PECL level differential signals are one of two overlapping single phase clocks. It is a 3/4 phase (late) skewed copy of the PHI1 and PHI1 L signals. Receiving this clock is optional at each node. A receiver node receives this clock into a noninverting and inverting PECL to CMOS converter to provide a clock (`TPHI3`) and not clock (`TPHI3 L`) to the system bus interface logic.

### 19.5.7 Initialization Signals

##### **CRESET L**

This signal returns the system to an initial state when asserted by asynchronously forcing each node to disable its bus drivers. During a power-up, `CRESET L` remains asserted until backplane +5 V and CPU +3.3 V power are stable, system bus oscillators are running at full harmonic content and voltage swing. These conditions are ANDed with the `ASYNC_RESET L` signal from the power system controller, synchronized to the `TPHI1` clock to negate `CRESET L` synchronous to `TPHI1`.

Nodes have 4 ns minimum setup time of the negation of `CRESET L` to their `TPHI1` clock edge. `CRESET L` may assert after the system has powered-up for a minimum of 10 ms to force all system bus nodes to an initial state. `CRESET L` asserts 100 ns maximum after the assertion of `ASYNC_RESET L`.

##### **CPU 2ID L**

This static signal is available to the CPU<sub>1</sub> slot connector and is sampled on power-up by negation of `CRESET L` to configure the CPU<sub>1</sub> node. The CPU<sub>0</sub> slot senses this signal high during reset to select itself as the primary CPU node. The I/O module ensures that it does not assert `IOREQ L` when `CRESET L` is asserted. The backplane ensures that this signal is grounded in the CPU<sub>1</sub> slot.

### MID<1:0>

These static signals supplied from the backplane are available to the memory connectors to identify which slot, base CSR position, and NVRAM serial control bus (I<sup>2</sup>C) address each memory module configures itself. The backplane supplies either GROUND or NO-CONNECTION on the MID signals. The decoding is summarized in Table 19–7:

**Table 19–7 Memory Slot ID**

MID<1:0>	Backplane Connection		
00	gnd gnd	Memory 1	SLOT 4 - closest to CPU <sub>0</sub> module slot
01	gnd NC	Memory 2	SLOT 5
10	NC gnd	Memory 3	SLOT 6
11	NC NC	Memory 4	SLOT 7 - furthest from the CPU <sub>0</sub> module slot

NC = no connection

### Serial Control Bus Signals

The two signals for the serial control bus (I<sup>2</sup>C) bus are routed on the backplane in the J1 power connector of CPU, I/O, and memory nodes. The SCL (serial clock) signal and the SDA (serial data) signal are terminated on the backplane and the operator control panel.

## 19.6 System Bus Transactions and Timing

This section describes system bus transactions and their logical timing diagrams.

The modules on the system bus can be classified as a commander, responder, or bystander. Each module type must follow specific rules. A general rule is that commanders are responsible for reporting errors to a CPU. Responders and bystanders may provide further error information for a CPU to read, but they must not send redundant hard error interrupts, except in the case of a detected parity error during a address and command cycle. A soft error interrupt for correctable memory or cache data errors could be reported by a responder. The timing of transactions ensures that the system bus can never consume a CPU node's backup cache for consecutive transactions without allowing the CPU an opportunity to have access to the backup cache.

Commanders, responders, and bystanders must always check address and command cycle parity. Bystanders should not check data parity, but commanders and responders always check data parity, and the protocol enables them to acknowledge correct transfers.

A responder or bystander which detects a RESERVED transaction type in the address and command cycle, disregards the transaction. The arbiter treats a RESERVED transaction type as a 7 cycle transaction. CSTALL0 L and CSTALL1 L could assert for a RESERVED transaction.

All transactions assume hexword data transfers, hence there is no transaction length specifier. Responders which do not return a full hexword of valid data must nevertheless drive the system bus with a full hexword of data with correct parity. Commanders which do not write a full hexword of data must nevertheless drive the system bus with a full hexword of data with correct parity.

## 19.6 System Bus Transactions and Timing

Commander reads and writes directly to primary noncacheable address space must restrict their addresses to hexword aligned quadwords or longwords. Hence all registers appear on aligned hexword address boundaries as quadword or longword data fields.

Commander reads and writes to secondary noncacheable address space are accomplished using a system memory based mailbox data structure assisted by a noncacheable address space mailbox address pointer register located on the I/O module.

The following sections describe the entire set of system bus transactions.

### 19.6.1 Read and Read Exclusive Transactions

These transactions are used to move a hexword of data from the responder to the commander. The read timing is shown in Figure 19–4. Two TRANS command codes are used. The TRANS field distinguishes the read from the exclusive read. The exclusive read differs from a read in that a CPU bystander and the I/O node invalidates the line the CPU commander is reading. The read data may be returned from a CPU or a memory node depending on the dirty response. A read exclusive to noncacheable address space should be treated identically to a read, however a node may not support this transaction type in noncacheable address space. The read data address bit <4> specifies which octaword of the aligned hexword is requested to return first. Hence data may be delivered in wraparound order.

A read transaction is initiated by a commander requesting a system bus grant from the arbiter which may be in any cycle. instruction-stream and data-stream reads are not distinguished on the system bus. The arbiter follows a priority based arbitration algorithm of I/O highest, CPU<sub>0</sub> and CPU<sub>1</sub> round robin with CPU<sub>0</sub> defaulted to current commander. The arbiter will ensure at least one idle bus cycle before issuing a grant if it observes three contiguous transactions, to avoid starving any processor chip from accessing its backup cache. The arbiter asserts a node grant and a backup cache request (BCREQ L) in the cycle prior to cycle 0, the arbiter asserts CA L during cycle 0 to signal the commander and bystanders that a new transaction is starting, a bystander may be selected as a responder. The commander drives its ID and address on the TPH1 L edge in cycle 0, the transaction type and parity can be driven with ID and address or one half cycle later on the TPH1 H edge in cycle 1, through cycle 1.

Commanders do not stall in cycle 2 during read, noncacheable address space write, or null transactions. Each commander is required to replicate the address and command on each quadword of the 128-bit bus and supply longword parity across the four longwords of the address and command cycle. The commander is not obligated to drive meaningful information in the ECHADR address field. Other nodes monitor the address and command, check parity, and decode the address to determine if they are to be a responder or a bystander.

If the responder or bystander detect bad parity, they must log this fault in a CSR and assert C\_ERR L in the next cycle 0 or for at least one cycle if the bus goes idle. The commander disables its C/A information by the end of cycle 1, and could disable the CAD drivers or leave them enabled until the end of cycle 3. A commander that drives in cycles 2 and 3 may supply valid parity with whatever information is driven on CAD signals. The commander watches for the responder to assert CXACK L in cycle 3, two cycles after CAD was driven.

## 19.6 System Bus Transactions and Timing

The arbiter deasserts grants at the end of cycle 1 and deasserts BCREQ L one cycle after cycle 1 to condition the early release of the processor's backup caches, when the read is not a DIRTY hit. The granted node may deassert its request one cycle after the arbiter negates the request. If CXACK L does not assert two cycles after CAD was driven, the commander must complete the full transaction sequence, it usually means an invalid address or a parity error fault occurred and the read data and parity must be ignored. If the address is to noncacheable address space, responders must return a hexword of read data in cycles 5 and 6 with correct parity and the commander must ignore the state of CSHARED L and CDIRTY L.

The responder may assert CSTALL0 L and CSTALL1 L to effect flow control by stalling the transaction in cycle 4.

### Stall Signals

There are two stall signals, one for each quadword sliced ASIC. Both stall signals assert and deassert at the same time. If the address is to memory space, the caches must return the results of the probe in cycle 3.5 to 4.5 to the bus on the CSHARED L and CDIRTY L signals. If the read probes dirty the CPU node ensures at least one stall cycle.

CSTALL0 L and CSTALL1 L are used to effect flow control to stall the transaction in cycle 4. CSTALL0 L and CSTALL1 L are asserted in cycle 3.5 to stall the bus in cycle 4, and negated in cycle 4.5 to allow cycle 4 to complete 1.5 cycles later and the bus to resume in cycle 5. If the probe result is NOT CDIRTY L, the memory responds by returning data in cycles 5 and 6 with correct parity, and if there is an uncorrectable error in either octaword of the data, by asserting CUCERR L with the corresponding returned data cycle. A responding node which asserts CUCERR L in cycle 5 leaves the signal asserted through cycle 6. It is possible for one or both CPU nodes to assert CSHARED L to indicate that it has that line valid. The I/O module will never assert CDIRTY L, but will assert CSHARED L if the read address matches a line it has buffered for performing an I/O write data merge.

Memories must abort the read data return by disabling their CAD drivers prior to entry of cycle 5, if the probe results in assertion of CDIRTY L. The responding cache node must not enable its CAD drivers until (NOT CSTALL0 L and NOT CSTALL1 L and cycle 4.5) or cycle 5. Only one cache node can respond with CDIRTY L, hence that node must supply the read data in cycles 5 and 6 with correct parity and if either octaword of the data is uncorrectable, signal such by asserting CUCERR L with the corresponding data cycle. It is the commanders responsibility to interrupt the CPU to report corrupted read data.

The responder may assist the commander and log the address and syndrome with the commander CID code to help with fault management reporting. The responding node must disable CAD and CPARITY L drivers at the end of cycle 6 if it is not being the next granted commander to avoid contention in cycle 0.5 of the next transaction. However, a read transaction responder that is transitioning to commander may leave its CAD and CPARITY drivers enabled from cycle 0 to 0.5 instead of trying to disable at the end of cycle 6 and enable at cycle 0.5. This decision must be made based on the state of the grant signal as the responder becomes a commander to avoid enabling during idle cycles.

CPU nodes may use this condition when transitioning from returning dirty or CSR read data to commanding the next transaction. I/O nodes may use this condition when transitioning from responding to a CSR read to commanding the next transaction. If the responder returned correctable data, to the commander, and if soft error interrupts are enabled in the responder, it should log this fault in

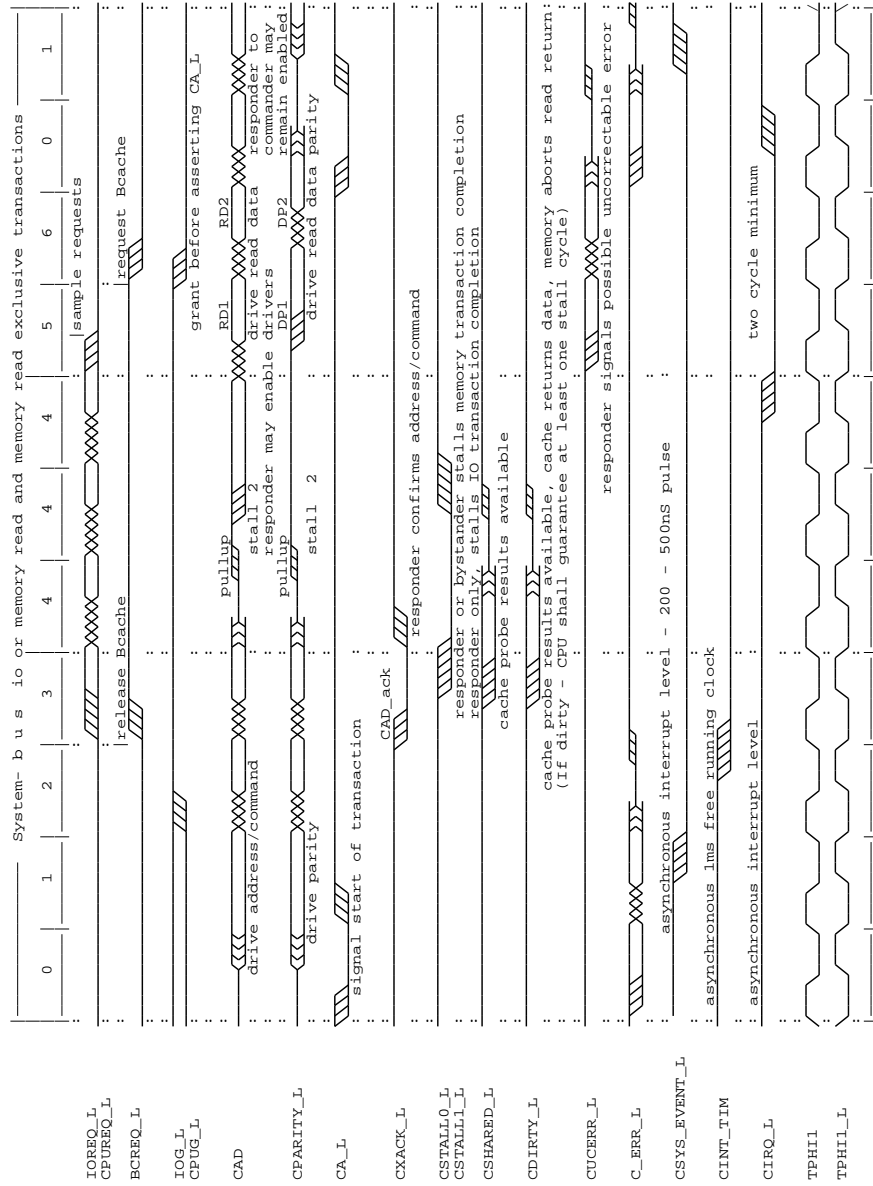
## 19.6 System Bus Transactions and Timing

a CSR and assert C\_ERR L in the next cycle 0 or for at least one cycle if the bus goes idle.

A CPU node must monitor read transactions for a target address which matches the address in its lock-address register with its lock flag set. A match on this condition forces the node to assert CSHARED L as the probe response in cycle 3.5 to 4.5. This ensures that subsequent writes remain visible to the bus even if the cache block containing the locked address should become NOT SHARED.



Figure 19-4 System Bus Read Timing



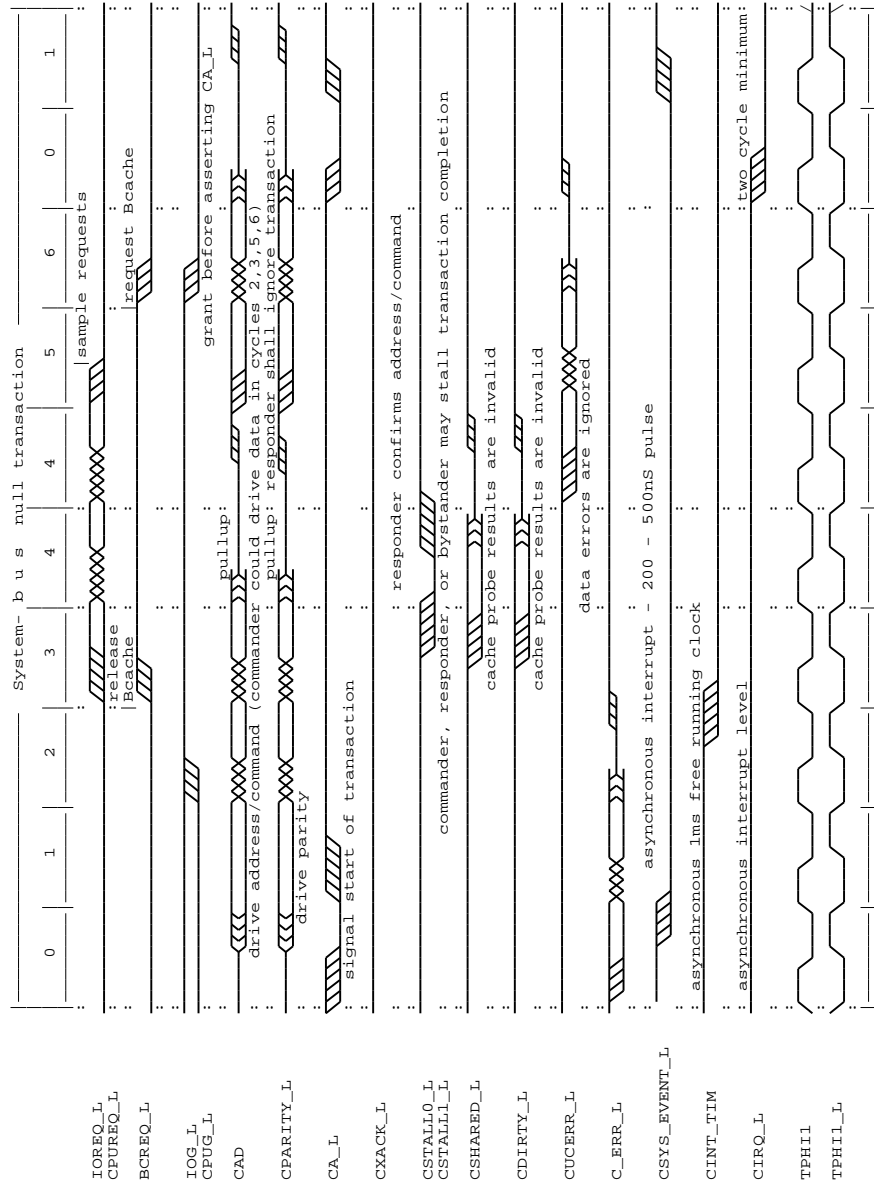
## 19.6 System Bus Transactions and Timing

### 19.6.2 Null Transactions

A commander which requests the system bus for performing a read or write transaction may need to abort the transaction, or ensure that it is granted the system bus as a form of lock on a sharable resource in order to avoid contention. See Figure 19-5 for the description of the 7 cycle NULL transaction. Because the arbitration protocol does not permit negation of a request prior to a grant, the granted commander can nullify the transaction by driving the command field to all ones and sequencing through the transaction. The addressed responder ignores the transaction. A CPU node may use this transaction, by driving the CAD lines in cycles 2,3,5,6 to update the lock register and to complete a masked write to a victimized line.

The cache probe results can be ignored. The commander should be aware that CSHARED L, CDIRTY L, and CUCERR L could assert. Also, the transaction may be stalled by a responder or a bystander in cycle 4 for memory transaction by asserting CSTALL0 L and CSTALL1 L in cycle 3.5. A bystanding node does not stall the read transaction to noncachable address space. BCREQ L negates one cycle after cycle 1 to allow a processor backup cache to be returned from bus use to processor use. The arbiter deasserts bus grant signals at the end of cycle 1 and the granted node may deassert its request one cycle later.

Figure 19-5 System Bus NULL Transaction Timing

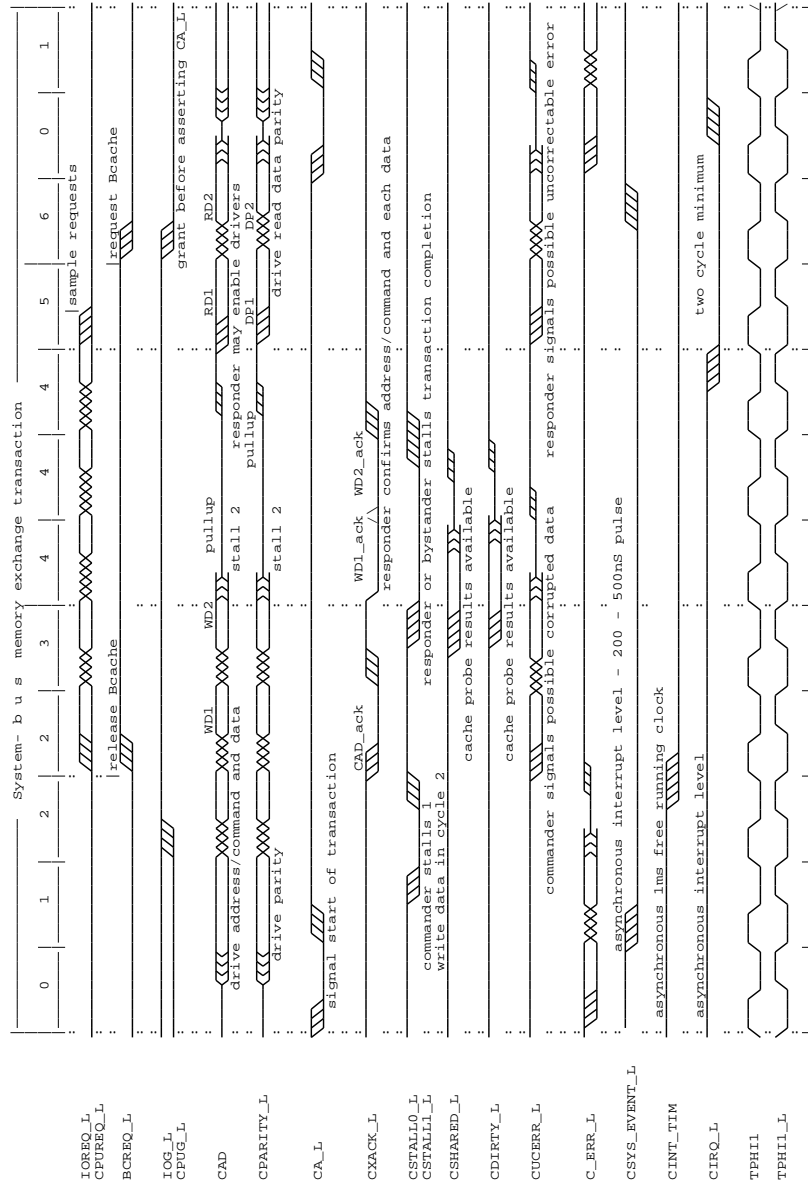


## 19.6 System Bus Transactions and Timing

### 19.6.3 Memory Exchange Transactions

These transactions are used to move a hexword of victim data from a commander's write back cache to system memory and to replace the victim by moving a hexword of data from the responder to the commander. Hence at a given index, the exchange of a dirty cache line by writing it to memory with a memory line with a different tag. This transaction is restricted to occur only to memory space. If the octaword wrap function is used, the victim write and fill read wraps or doesn't wrap with identical octaword ordering.

Figure 19-6 System Bus Memory Exchange Timing



## 19.6 System Bus Transactions and Timing

The exchange transaction begins in cycle 0 when the arbiter asserts  $CA_L$  during cycle 0. The commander drives its  $ID$  and address on the  $TPHI1_L$  edge in cycle 0. The transaction type and parity can be driven with  $ID$  and address or one half cycle later on the  $TPHI1_H$  edge in cycle 1, through cycle 1. The  $ECHADR$  fields must be driven with valid tag content to tell the memory node where to write the data. The exchanging commander may stall in cycle 2 by asserting  $CSTALL1_L$  and  $CSTALL2_L$  in cycle 1.5 to 2.5. Bystanding or responding nodes do not attempt to stall in cycle 2. The exchanging commander continues to drive the  $CAD$  signals during stalled cycle 2(s) with unspecified data and good parity until it drives write data and parity in nonstalled cycles 2 and 3 sequence. The commander drives the victim write data and parity in the nonstalled cycles 2 and 3, along with an indication of the goodness of the data on  $CUCERR_L$ .

All exchanged victim writes update memory, but are not accepted by backup caches. If the victim data has an uncorrectable error the commander asserts  $CUCERR_L$  with the data in non-stalled cycles 2 or 3, to signal memory to mark the data bad. If  $CUCERR_L$  is asserted with the first write data, it remains asserted through cycle 3. The commander releases  $CUCERR_L$  at the end of cycle 3. The commander disables its  $CAD$  and  $CPARITY_L$  drivers at the end of cycle 3. The responder confirms the address and command and each portion of the write data by asserting  $CXACK_L$  two cycles after command/address cycle 1 and two cycles after the non-stalled cycle 2 and 3 regardless of cycle 4 stalls.

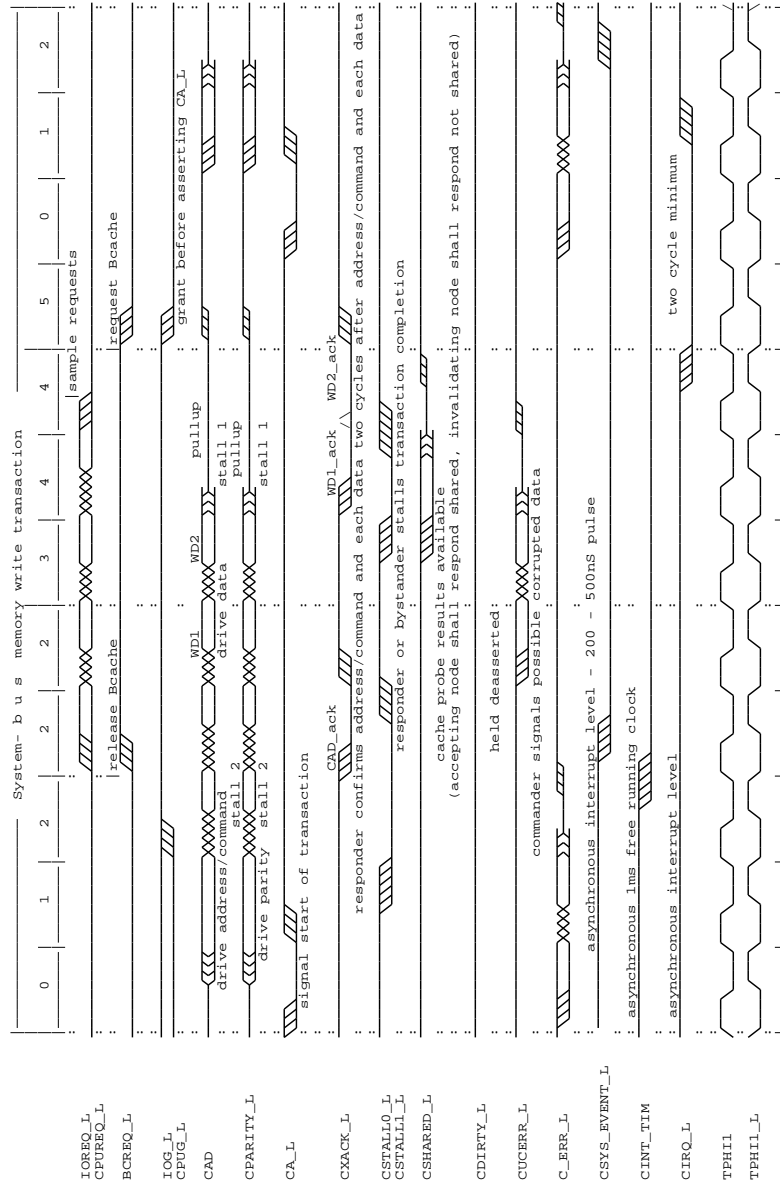
If the commander does not receive command/address  $CXACK_L$  two cycles after it was driven, it must complete the transaction sequence, ignore  $CXACK_L$  and  $CUCERR_L$  responses to the write data and ignore the read data, parity, and  $CUCERR_L$  in cycles 5 and 6. If  $CXACK_L$  is received two cycles after cycle 1, the commander continues to check confirmation of the write data two cycles after valid data was driven via  $CXACK_L$ . The responder, which could be the memory responder of the read or the write or both, or a bystander may stall the transaction in cycle 4 by asserting  $CSTALL0_L$  and  $CSTALL1_L$  in cycle 3.5. Note that  $CXACK_L$  data responses are not delayed by cycle 4 stalls.

Cache nodes return their probe results of the read address in cycle 3.5 to 4.5. Only the read address need be probed; it is not necessary to probe the victim write address. The remainder of the transaction, the read response, is identical to that described above in Figure 19-4.  $BCREQ_L$  negates one cycle after cycle 1 regardless of cycle 2 stalls to allow a processor backup cache to be returned from bus use to processor use. The arbiter deasserts bus grant signals at the end of cycle 1 and the granted node may deassert its request one cycle later.

### 19.6.4 Memory Write Transactions

These transactions are used to move a hexword of data from a commander to a memory address space responder. Figure 19-7 is a timing diagram of the system bus write cycle.

Figure 19-7 System Bus Write Timing



## 19.6 System Bus Transactions and Timing

The transaction begins in cycle 0 when the arbiter asserts  $CA_L$  during cycle 0. The commander drives its  $ID$  and address on the  $TPH1_L$  edge in cycle 0. The transaction type and parity can be driven with  $ID$  and address or one half cycle later on the  $TPH1_H$  edge in cycle 1, through cycle 1. The commander does not have to drive a meaningful address in the  $ECHADR$   $CAD$  fields. The writing commander may stall in cycle 2 by asserting  $CSTALL1_L$  and  $CSTALL2_L$  in cycle 1.5 to 2.5. Bystanding or responding nodes do not attempt to stall in cycle 2. The writing commander does not stall in cycle 2 of read, null, and noncacheable address space write transactions. The writing commander continues to drive the  $CAD$  signals during stalled cycle 2(s) with unspecified data and good parity until it drives write data and parity in non-stalled cycles 2 and 3 sequence. The commander drives the write data and parity in the non-stalled cycle 2 and 3, along with an indication of the goodness of the data on  $CUCERR_L$ . All write transactions update memory.

If the write data has an uncorrectable error, the commander asserts  $CUCERR_L$  with the data in non-stalled cycles 2 or 3, to signal memory and CPU nodes to mark the data bad. If  $CUCERR_L$  is asserted with the first write data, it remains asserted through cycle 3. The commander releases  $CUCERR_L$  at the end of cycle 3. The commander disables its  $CAD$  and  $CPARITY_L$  drivers at the end of cycle 3. The responder confirms the address and command and each portion of the write data by asserting  $CXACK_L$  two cycles after command/address cycle 1 and two cycles after the non-stalled cycle 2 and 3 regardless of cycle 4 stalls.

If the commander does not receive command/address  $CXACK_L$  two cycles after it was driven, it must complete the transaction sequence and ignore  $CXACK_L$  and  $CUCERR_L$  responses. A node may assert  $CSTALL0_L$  and  $CSTALL1_L$  in cycle 3.5 to effect flow control over the transaction by stalling it in cycle 4.  $CDIRTY_L$  never asserts during this transaction.  $CSHARED_L$  may assert in cycle 3.5 to 4.5 by a CPU node which chooses to accept the line. A CPU node does not assert  $CSHARED_L$  if it chooses to invalidate the line.

If a cache node decides to accept the write, it must mark the line  $SHARED$  and  $NOT DIRTY$ .  $BCREQ_L$  negates one cycle after cycle 1 regardless of cycle 2 stalls, to allow a processor's backup cache to be returned from bus use to processor use. The arbiter deasserts bus grant signals at the end of cycle 1 and the granted node may deassert its request one cycle later.

Latent write errors may be reported by responder nodes via the  $C\_ERR_L$  signal. If the memory nodes detect bad write data parity, they complete the write with the check field forced to identify the line with an uncorrectable error.

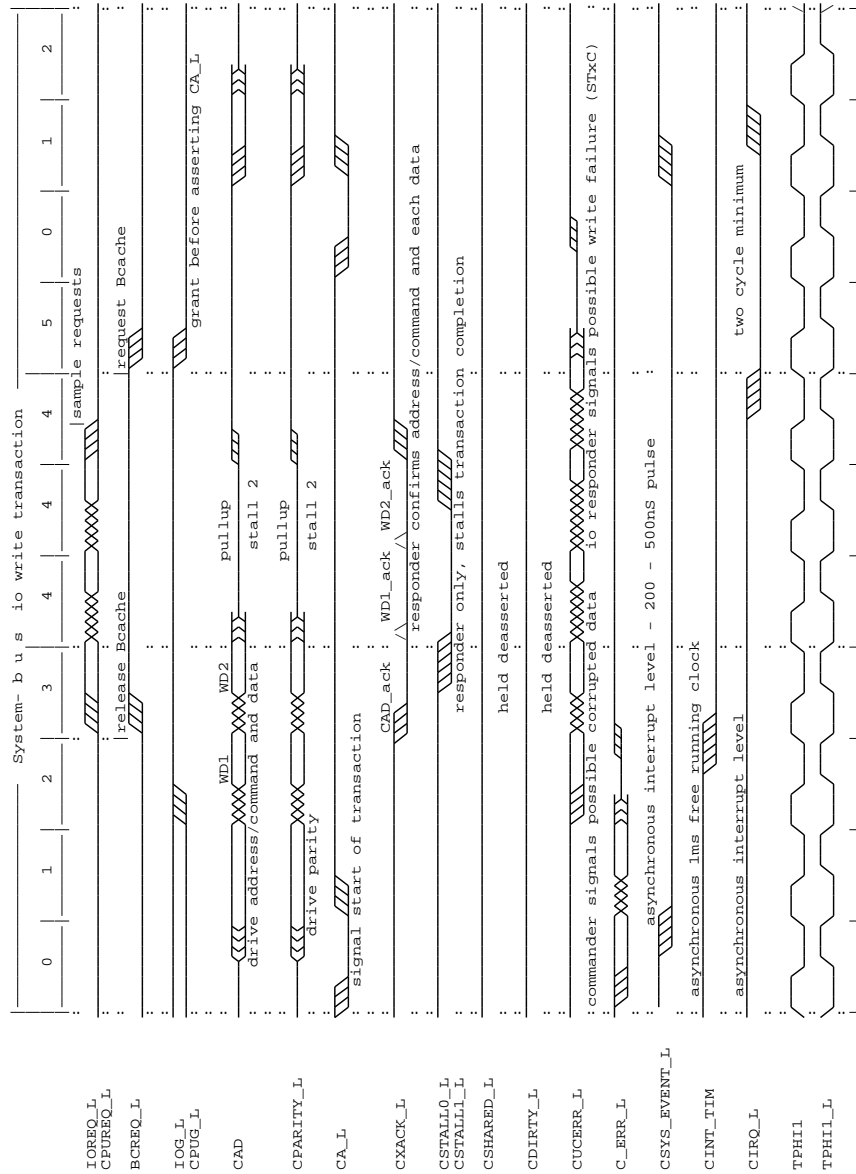
### 19.6.5 Noncacheable Address Space Write Transactions

This transaction is used to move a hexword of data from a CPU commander to a CSR in a responder. This transaction is restricted to noncacheable address space. The transaction is similar to the memory write transaction, except for the behavior of the  $CUCERR_L$  and  $CSHARED_L$  signals, and a  $CSTALL0,1_L$  assertion restriction. The responding node may inform the CPU commanding node that it can not accept this CSR write by asserting the  $CUCERR_L$  signal during the first cycle 4 and optionally all other cycle 4(s), to be sampled by the commander at the end of the first cycle 4, to fail the  $STxC$  transaction. Figure 19–8 shows the timing diagram for a system bus noncacheable address space write.



# 19.6 System Bus Transactions and Timing

Figure 19–8 System Bus Noncacheable Address Space Write Timing



## 19.6 System Bus Transactions and Timing

CSHARED L does not assert because this is a transaction in noncacheable address space and is not probed by CPU nodes. The noncacheable address space responder fails the store or store conditional by asserting CUCERR L in the first cycle 4. This store failure is used for CSR flow control from a CPU to the I/O node. Hence a write to the mailbox CSR or to remote noncacheable address space should be a STxC type instruction.

If the responder node detects a write data parity error in the first octaword of write data, it does not complete the write to the destination. If the responder detects a write data parity error in either octaword, it indicates the error to the CPU node with the C\_ERR L . A bystanding node does not stall the noncacheable address space write transaction.

# Part V

---

## The Firmware

This part contains a detailed functional description of the system firmware.

---

## System State Transitions

The DEC 4000 can be in one of six states as listed below:

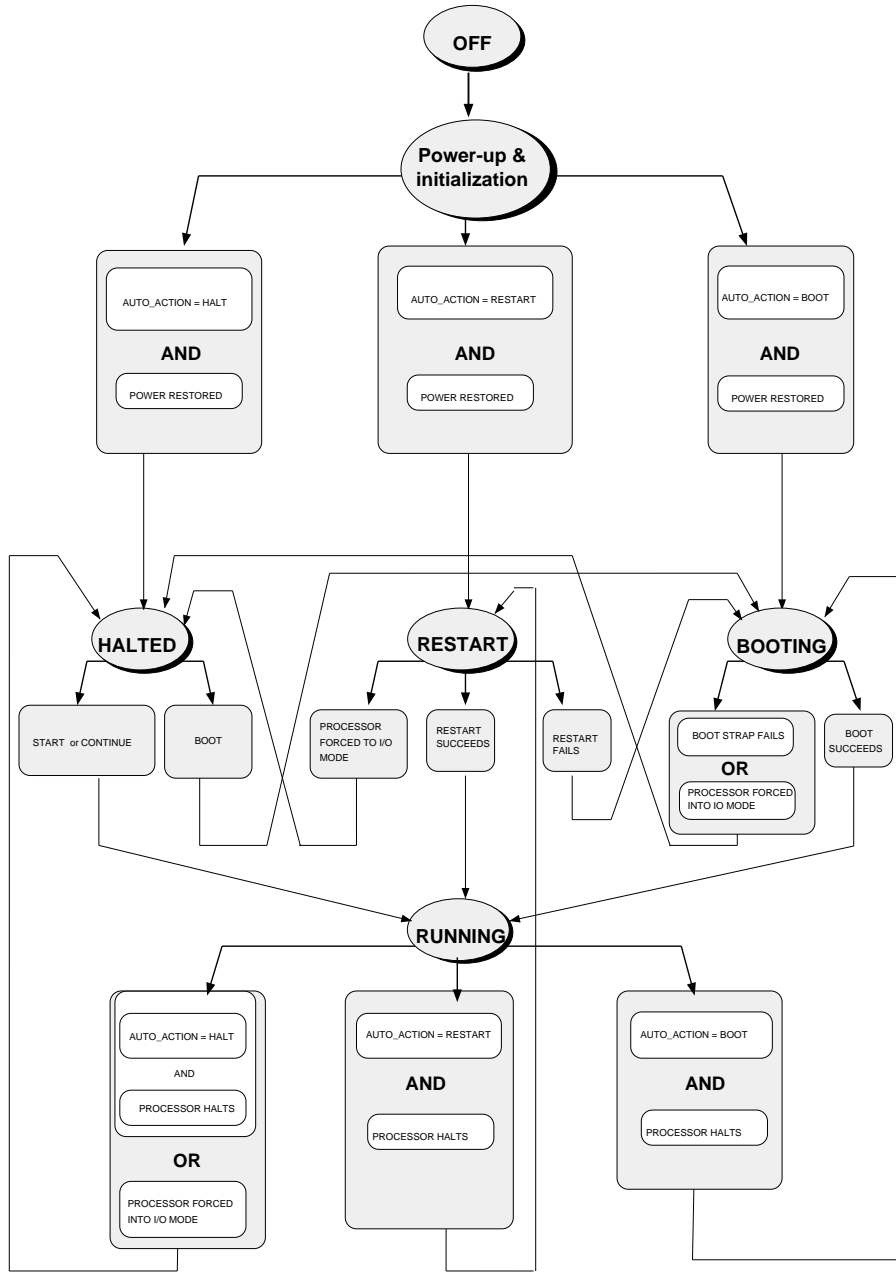
1. Powered off
2. Powering-up and Initializing
3. Bootstrapping
4. Halted
5. Restarting
6. Running

The transition between the major states are determined by the current state and a number of variables and events including the following:

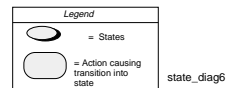
- Whether power is available to the system
- The console AUTO\_ACTION environment variable
- The bootstrap in progress (BIP) flags
- The start-capable (RC) flags
- Processor error halts
- The CAL\_PAL HALT instruction
- Console commands

Figure 20–1 shows the state transitions for the DEC 4000 system.

Figure 20–1 DEC 4000 System State Transitions



For simplicity, power fail lines have not been shown.  
Power fail on any state causes the state to transition to "OFF".



---

## The Power-Up and Initialization State

This chapter explains the behavior of the DEC 4000 system under power-up and initialization as well as reset and halt.

## 21.1 The Powered Off State

For the purposes of this discussion, the beginning state of the system is defined as powered off. No power is flowing to the system processors.

- The AC power cord is connected from the DEC 4000 system to a power main outlet.
- The AC present LED is illuminated (on the FEU).
- The AC breaker switch is in the off position (on the FEU).
- The DC On/Off switch is in the off position (on the OCP).
- The 48 volt BUS\_DIRECT line is energized and the 48 volt BUS\_SWITCHED line is deenergized.

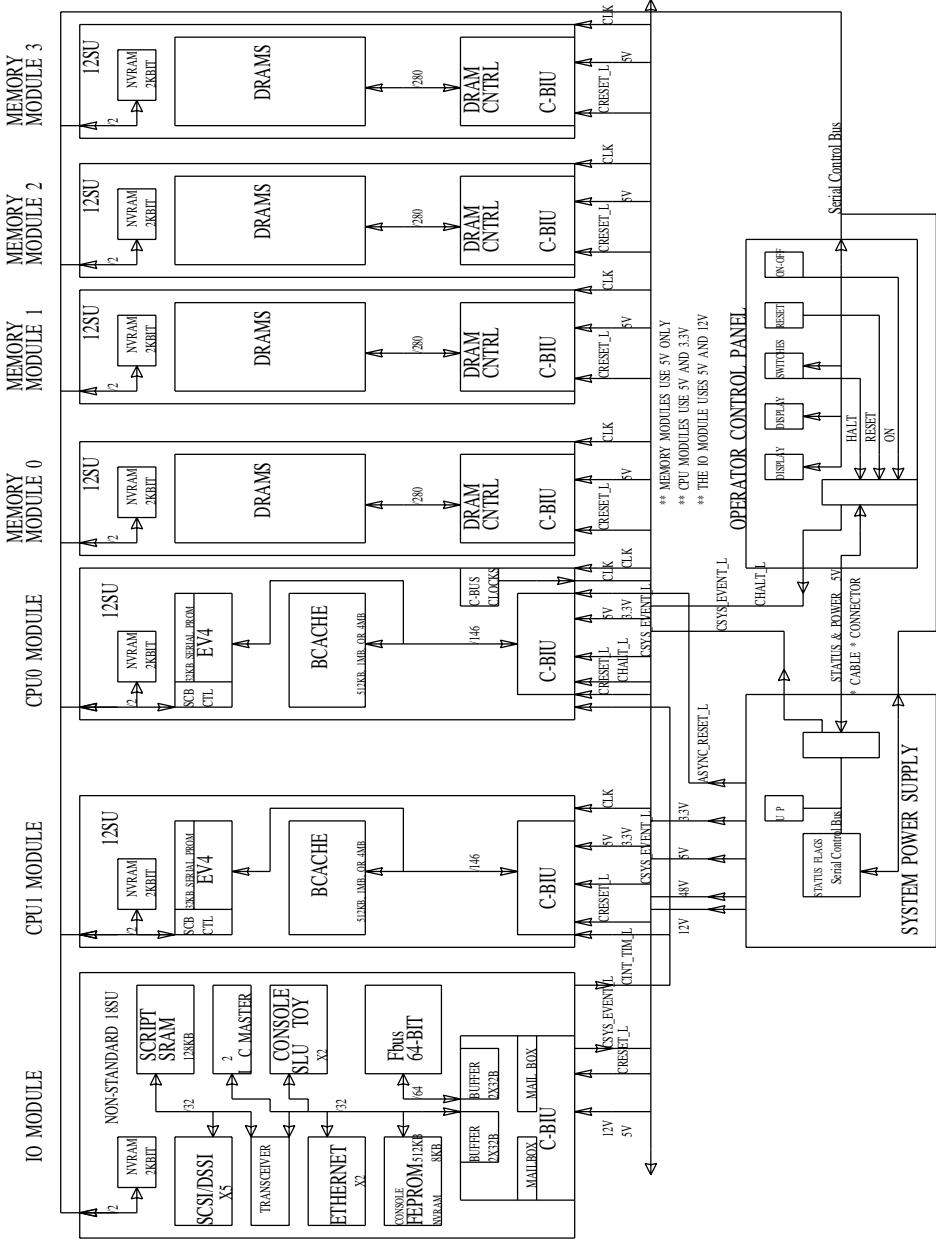
## 21.2 The Power-up Process

*Power-up* is the term given to a process that starts the flow of electrical current to a device or system. The power-up sequence requires that the AC power supplies energize and stabilize first, allowing the DC power to be requested via the OCP. On DC request, the logic circuits are energized and the DC logic initialization process takes place. There are several components that participate in the power-up and initialization process. These components and the hardware interconnects between them must be operating correctly in order for the firmware to load and execute and provide status indicator to the operator control panel. Figure 21-1 shows a diagram of the required hardware interconnects for system initialization.

- +5 volts DC voltage
- System bus clocks
- System bus reset logic
- DECchip™ 21064 CPU chip
- DECchip™ CPU's SRAMs and 87C652 D-bus microcontroller for the serial control bus
- Operator control panel



Figure 21-1 Initialization Block Diagram



## 21.2 The Power-up Process

### 21.2.1 AC Power-Up

When AC power is applied to the system, either when the AC circuit breaker in the back of the system is turned to the on position or when the electricity returns after a power outage, the following sequence is followed:

1. The front end unit (FEU) begins operation and energizes the BUS\_DIRECT output. The BUS\_SWTCHD and BUS\_SWTCHD\_RTN lines are held off. The fan power converter is also held off.
2. The bias supply on the power system controller (PSC) starts operation when energized by the BUS\_DIRECT supply. Once the PSC bias supply output voltage is valid, the microprocessor on the PSC begins its self-check.

If the self-check fails, the PSC FAILURE LED is illuminated and the system latches off. If the self check is successful, the PSC OK LED is illuminated and power-up proceeds.

3. The PSC then checks for the DC power to be commanded on. If the OCP is requesting DC power to be turned on and the system is not configured as a slave on the power bus, the DC power up proceeds. If the system is configured as a slave on the DEC 4000 power bus, the slave-in input to the PSC must also be requesting DC power to be turned on in order for the DC power-up to proceed. The OCP requests DC power to be energized via the DC\_ON\_OCP L signal. If these conditions are not met, the PSC idles, monitoring for DC to be requested.

### 21.2.2 DC Power-Up

It makes no difference in the following sequence whether the system is powering up from the application of AC power or from a DC off condition.

1. If DC has been requested, the PSC checks for an overtemperature condition. If overtemperature is detected, the PSC aborts the power-up and reports the overtemperature failure.
2. The PSC then starts the fans (at full speed) by asserting FAN\_POWER\_ENABLE H to the FEU. If the fans fail to start, the PSC aborts the power-up and responds to the fan failure.
3. With fan rotation established, ASYNC\_RESET L is asserted and POK H negated. See Section 21.2.3 for detailed information about the ASYNC\_RESET L and CRESET signals.
4. The power system control unit delivers a ASYNC\_RESET\_L signal to CPU<sub>0</sub> 30 to 50 ms after the DC voltages are stable.

As a side effect of the assertion of CRESET, the OCP will have both CPU LEDs lit.

5. ASYNC\_RESET\_L holds CRESET\_L while the system bus clocks stabilize to constant frequency and duty cycle.
6. The 48 VOLT\_SWITCHED bus is energized by asserting the BUS\_REQUEST H signal to the FEU.

If the BUS\_SWTCHD\_OK L signal is not asserted within approximately 100 ms, the power-up is aborted. The fans are turned off, the FEU\_FAILURE LED is illuminated, and the PSC enters latching-shutdown mode.

## 21.2 The Power-up Process

7. Once the 48 VOLT\_SWITCHED bus is stable, the 3.3V output is energized by asserting V3\_ENABLE H. If the 3.3V output does not come into regulation within approximately 100 ms, the power-up is aborted. The 48 VOLT\_SWITCHED bus and fans are turned off, the failure LED on the DC3 converter unit, is illuminated, and the PSC enters the latching-shutdown mode.
8. Once the 3.3V bus is in regulation, the 5V output is energized by asserting V5\_ENABLE H.  
If the 5V output does not come into regulation within approximately 100 ms, the power-up is aborted. The 3.3V and 48 VOLT\_SWITCHED buses and fans are turned off, the failure LED on the DC5 converter unit is illuminated, and the PSC enters the latching-shutdown mode.
9. Once the 5V bus is in regulation, the 2.1V output is energized by asserting V2\_ENABLE H and the DC5 OK LED illuminated.  
If the 2.1V output does not come into regulation within approximately 20 ms, the power-up is aborted. The 5V bus, 3.3V bus, 48 VOLT\_SWITCHED bus and fans are turned off, the failure LED on the DC3 converter unit is illuminated, and the PSC enters the latching-shutdown mode.
10. Once the 2.1V bus is in regulation, the 12V output is energized by asserting V12\_ENABLE H .  
If the 12V output does not come into regulation within approximately 100 ms, the power-up is aborted. The 2.1V, 5V, 3.3V, and 48 VOLT\_SWITCHED buses and fans are turned off, the failure LED on the DC3 converter unit is illuminated, and the PSC enters the latching-shutdown mode.
11. Once the 12V bus is in regulation, DC3 OK LED is illuminated.
12. At this point the entire system is energized. The PSC checks the status of the entire power system.
13. If everything is still functioning after a delay of 10 ms minimum (50 ms maximum), ASYNC\_RESET\_L is deasserted by the PSC and logically ANDed on the CPU<sub>0</sub> module with a signal that indicates that the clock has run for 20 ms. This ANDed signal is then synchronized to the PHI1 clock and TPHI1 clock to cause the synchronous deassertion of CRESET\_L.

### 21.2.3 ASYNC\_RESET\_L and CRESET\_L

The term *reset* refers to a hardware condition whereby a switch is pushed that causes the CPU to reset. The reset causes the system to invoke the initialization sequence and return to a power-up hardware state. The operator control panel has a reset button which causes the system to reset the CPU. Software initialization routines must manage devices which must be returned to initial conditions after the system was powered-up. To reset a system which has been powered-up, the operator control panel reset switch generates a 10 ms minimum pulse that is directly gated to the ASYNC\_RESET\_L signal on the power-controller module.

The reset switch asynchronously returns the system bus nodes to a power-up state via the ASYNC\_RESET\_L and CRESET\_L signals.

The following happens asynchronously on the assertion of CRESET\_L :

- CAD(127:0), CXACK\_L , and CPARITY(3:0) are tri-stated and pulled up.
- CSYS\_EVENT\_L, C\_ERR\_L, CUCERR\_L, CDIRTY\_L, CSHARED\_L, CSTALL0\_L and CSTALL1\_L are released and pulled up by their termination pullups.

## 21.2 The Power-up Process

- CALL, CPUG\_L, IOG\_L, IOREQ\_L, CPUREQ\_L, CIRQ\_L(1:0), BCREQ\_L and CINT\_TIM are driven deasserted. These signals do not have backplane pullups.
- PHI1, PHI1\_L, PHI3, and PHI3\_L free run 10 ms after stable +5v supply.

All nodes asynchronously tri-state, release, and negate all system bus signals within 100 ns of the reception of CRESET\_L.

ASYNC\_RESET\_L asynchronously causes CRESET\_L to assert 100 ns maximum later.

The negation of the operator control panel reset causes ASYNC\_RESET\_L to negate.

The CPU<sub>0</sub> module synchronizes ASYNC\_RESET\_L to the TPHI1 rising edge and negates CRESET\_L with a minimum of 4 ns setup time to TPHI1 at the receiver nodes.

CRESET\_L is asserted for not less than 10 ms.

The PHI clocks continue to run during this reset sequence to ensure that all logic returns to the initial state.

## 21.3 System Initialization

*Initialization* is the term given to the sequence of steps that prepares the system to start. A system initialization occurs following power-up and operator control panel initiated reset. During initialization, the console is initialized and all of the subsystem's logic circuits receive the message that they are getting ready to start.

Most of the initialization process is controlled by the primary processor (CPU<sub>0</sub>). Each processor module contains one 21064 CPU chip and one 87C652 D-bus microcontroller chip. For this discussion, the naming conventions will be repeated here.

- **CPU<sub>0</sub>** and **CPU<sub>1</sub>** refer to the two processor modules.  
CPU<sub>0</sub> is the primary processor.
- **P<sub>0</sub>** and **P<sub>1</sub>** refer to the 21064 CPU chips, one on CPU<sub>0</sub> and one on CPU<sub>1</sub> respectively. In some cases where the text refers to both CPU chips, the term 21064 is used.
- **Micro<sub>0</sub>** and **Micro<sub>1</sub>** refer to the two 87C652 D-bus microcontrollers on CPU<sub>0</sub> and CPU<sub>1</sub> respectively.

The startup of the 21064 CPU chip (P<sub>0</sub> or P<sub>1</sub>) is controlled by serial ROM (SROM) code that is loaded into the 21064's instruction cache. The startup of the 87C652 D-bus microcontroller (Micro<sub>0</sub> or Micro<sub>1</sub>) is controlled by firmware contained inside the D-bus microcontroller chip. On initialization, all four of these devices start in parallel after the deassertion of ASYN\_RESET\_L.

There are several synchronization points throughout the initialization process. One being the synchronization of the 21064 chip and its accompanying D-bus microcontroller, the other being the synchronization of the two processor modules.

Control is shared at first by 21064 and the D-bus microcontroller. Control is then passed to the D-bus microcontroller and later to the FEPRM firmware code. When the D-bus microcontroller is in control, testing is achieved by directing commands from the D-bus microcontroller to the 21064 CPU chip. The 21064, in general, waits in an idle loop for commands to be issued to it from the D-bus microcontroller. This is true on both modules.

## 21.3 System Initialization

By the time the testing begins, CPU<sub>0</sub> has determined that it is the primary CPU. The 21064 CPUs are in command/response mode. Each of these tests are initiated by the D-bus microcontroller by sending a command to the processor. That command initiates a test on the processor and returns either its success or an error status which then starts the next test. The initialization flow is described in Figure 21–2 through Figure 21–4.

The initialization flow is detailed in the following list. Callout numbers in the list correspond to callout numbers in Figure 21–2 through Figure 21–4 .

- ❶ The deassertion of ASYNC\_RESET\_L by the power system controller marks the start of code execution for the 21064 processor chips and the D-bus microcontrollers on both processor modules.

The D-bus microcontrollers work in conjunction with the serial control bus and the 21064 CPUs to test basic features of the processor modules. P<sub>0</sub> and P<sub>1</sub> load their respective SROM code into their instruction caches and start execution. At the same time Micro<sub>0</sub> and Micro<sub>1</sub> read their internal firmware code and start. These operations happen exactly in parallel and all four devices are shown positioned on the same starting line in Figure 21–2.

- ❷ Both P<sub>0</sub> and P<sub>1</sub>, under the direction of their respective SROM code, simultaneously initialize themselves and their backup caches. After each 21064 chip completes its initialization, it waits for a synchronization byte from its D-bus microcontroller indicating that the D-bus microcontroller has completed its initialization. The D-bus microcontrollers have a significantly greater number of steps to perform before synchronizing with the 21064 chips.

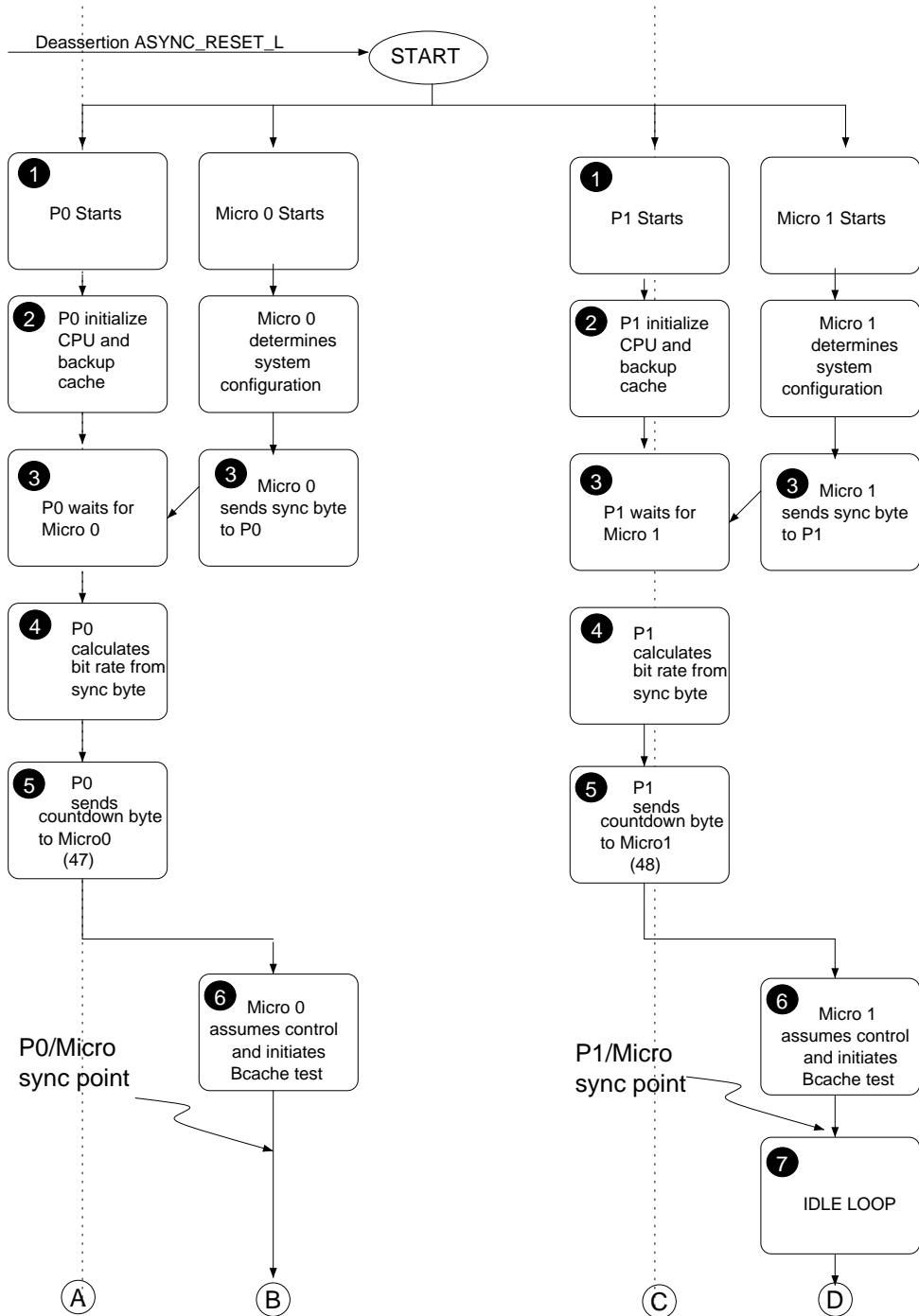
The D-bus microcontrollers determine the configuration of the system.

- The D-bus microcontrollers determine which processor they are on, via a hardware bit BCPU2\_ID L which is latched from IOREQ L from the backplane (J5 pin 161).
- They determine speed bits, and auto unload configuration by reading the EEROM on the serial control bus bus.
- They determine how many memory modules are configured in the system and the configuration byte for the memory modules by attempting to read the EEROMs on the memory boards.
- CPU<sub>0</sub> determines whether or not CPU<sub>1</sub> is present by attempting to read CPU<sub>1</sub>'s EEROM. If that READ fails it is assumed that CPU<sub>1</sub> is not present in the system and the CPU<sub>1</sub> LED on the operator control panel is turned off.

- ❸ When the D-bus microcontrollers complete the system configuration, they send a synchronization byte to their respective CPU chips and wait for the corresponding synchronization byte from the CPU in a handshake fashion. A 55<sub>16</sub> byte is sent to the CPU chip because this piece of data has the property of having alternate one and zero bits.
- ❹ Once the 21064 receives the 55, it calculates the bit rate by counting the time that it takes to get the first 9 edges, and dividing by 8. It calculates a very accurate measurement of the bit time.

## 21.3 System Initialization

Figure 21–2 Initialization Flow Diagram



INIT8

- ⑤ The 21064 sends back a countdown byte of 47 if its on CPU<sub>0</sub> and a 48 if CPU<sub>1</sub>. At this point the D-bus microcontrollers on each processor module are in control and operating in parallel.

## 21.3 System Initialization

- ⑥ The D-bus microcontroller initiates a sequence of backup cache tests and waits at the end of the testing. CPU and system bus test the backup cache's ability to access itself on the system bus and I/O module on system bus. Memory is not tested at all at this point. There is a synchronization point between the two CPUs at the end of the first block of test.
- ⑦ When CPU<sub>1</sub> finishes the test sequence, it sends a message to CPU<sub>0</sub> indicating it has completed the test and enters an idle loop. The message sent to CPU<sub>0</sub> allows CPU<sub>0</sub> continuing its testing.

CPU<sub>0</sub> waits for completion of its own test sequence, completion of CPU<sub>1</sub> test sequence, or CPU<sub>1</sub> not present and the assurance of a couple of control character bits being set (AUTO\_UNLOAD and AUTO\_TEST). This is the default case.

While any tests are being executed, LEDs on the operator control panel change to indicate which device is being tested. For example, when the backup cache is being tested, the CPU LED will be lit. When the system bus is being tested, the I/O LED is lit and the CPU LEDs are left lit.

- ⑧ CPU<sub>0</sub>'s D-bus microcontroller issues a command to the P<sub>0</sub> to perform a test on the largest memory module in the system.
- ⑨ P<sub>0</sub> tests 2 megabytes of memory. It is not a complete module test. It is a simple test for the most basic failure modes. It does not test all of the memory modules. Full testing of the system memory modules occurs in the FEPRM code and occurs later in the initialization process. When this test is complete, one memory module is enabled and 2 megabytes of it have been tested. This is done to minimize initialization time. The D-bus microcontroller determines which memory is the largest.

When CPU<sub>0</sub> gets that command, it issues a request packet back to the micro<sub>0</sub> to get the configuration bits for the memories.

CPU<sub>0</sub> configures all of the memory boards. CPU<sub>0</sub> runs a memory test on the largest memory board and enables the largest memory board that has not failed.

If it fails, the D-bus microcontroller determines if there are additional memory modules in the system. It marks the failing module as having failed by writing to its EEROM and it reissues the test. It will loop on this memory test command until it runs out of memory modules to try.

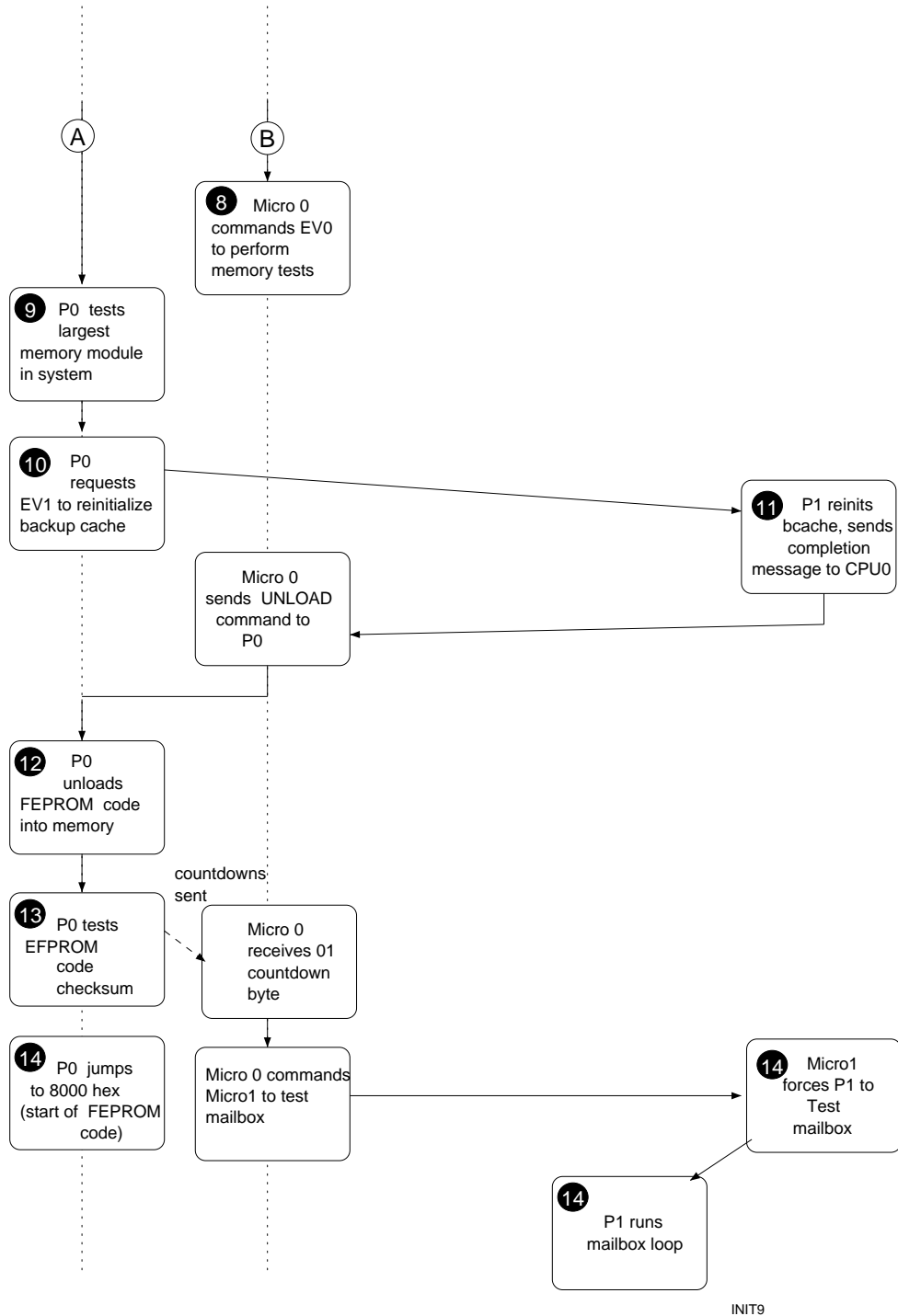
- ⑩ Once the complete memory test competes successfully, CPU<sub>0</sub> sends a packet to CPU<sub>1</sub> requesting that CPU<sub>1</sub> reinitialize its backup cache.
- ⑪ CPU<sub>1</sub> reinitializes its backup cache and sends as a packet to CPU<sub>0</sub> saying it is finished.

CPU<sub>1</sub> backup cache is now in a clean state.

Micro<sub>0</sub> issues an unload command to its 21064, which then resets the LBUS, unloads the FEPRMs, verifies the checksum, byte lanes, and byte lane consistency.

## 21.3 System Initialization

Figure 21–3 Initialization Flow Diagram



### 21.3.1 FEPROM Unloading

- 12 P<sub>0</sub> unloads the FEPROM code. The FEPROM code for P<sub>0</sub> starts at location 8000<sub>16</sub>.



## 21.3 System Initialization

- ⑬ P<sub>0</sub> checks the checksum of the FEPR0M code to make sure that code in the FEPR0MS is correct. It checks a byte across all 4 byte lanes to see that it is all the same code. It also checks an additional byte which has the byte lane number in it to see that the FEPR0MS are in the correct socket locations. Because the checksums are byte-length only, they are not the full width. So it is possible to have all 4 parts mismatched and not detect it. During this process, a set of countdowns are being sent back to the D-bus microcontroller, on CPU<sub>0</sub>.

- ⑭ CPU<sub>0</sub> jumps to the start of the FEPR0M code.

When the micro<sub>0</sub> receives the last (01) countdown byte, it sends a message to the micro<sub>1</sub> asking the D-bus microcontroller to start a special test that forces the P<sub>1</sub> to spin on its processor mailbox. P<sub>1</sub> monitors the mailbox waiting for a non-zero value, then jumps to the location contained in the mailbox.

What this achieves is it commands the P<sub>1</sub> to wait for a command from P<sub>0</sub> via the P<sub>1</sub> processor mailbox. This is how Symmetric MutiProcessing (SMP) capability is established.

P<sub>1</sub> is actually running an instruction cache based loop. Because the instruction cache is not invalidated, the location can be overwritten. If the instruction cache is running for example, at location 1000, and location 1000 is written to, it has no effect on the instruction cache. So it is possible to read and write freely through all of memory running a loop in the instruction cache and have absolutely no effect on the instructions.

- ⑮ FEPR0M code assumes control.

The FEPR0M code decompress itself because the bulk of the FEPR0M code is stored in a condensed, unexecutable format. Only the very beginning of FEPR0M code is directly executable.

To decompress, FEPR0M code copies itself out of its current location and into "higher memory". The intent of this process is that the FEPR0M code must get itself out of the place where it will be decompressing to. The entire image is decompressed down to 8000<sub>16</sub>. The first 8000 is the Palcode. The fact that the FEPR0M code is compressed is irrelevant to the transfer operation performed by the SR0M code.) Note that up to this point only 1 memory module is enabled and 8 megabytes of it have been briefly tested. The complex memory testing is done by the firmware console which has been unloaded from the FEPR0MS.

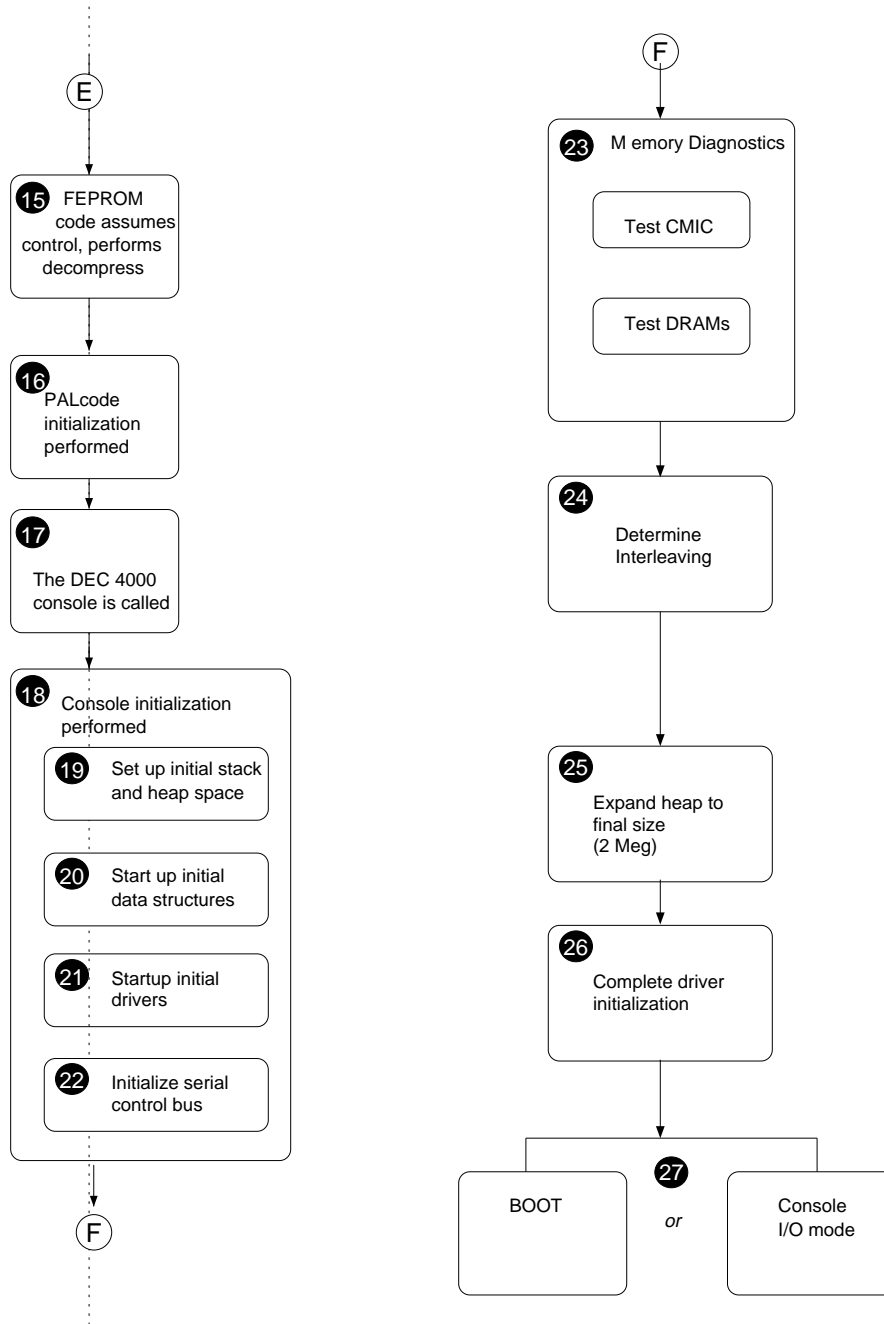
- ⑯ After the decompress is complete, control is transferred to the P<sub>0</sub> and it jumps to location 8000<sub>16</sub>. At some point further on, PALcode determines whether or not the operating system software is OSF/1, VMS, or Microsoft NT<sup>TM</sup>.

At part of the PAL init, the arbitration mask for P<sub>1</sub> is disabled so P<sub>1</sub> is at this point stalled waiting to get access to the system bus. This is done to minimize the performance impact of P<sub>1</sub>. Part of the PAL init, disables P<sub>1</sub> arbitration ability so the next time P<sub>1</sub> attempts to get. attempts to go read its mailbox which requires a system bus access it will stall waiting for the system bus grant.

- ⑰ Once PAL is initialized and its data structure is in state it calls the DEC 4000 console.

## 21.3 System Initialization

Figure 21–4 Initialization Flow Diagram



INIT10

### 21.3.2 Console Initialization

- 18 The console starts its initialization process in order to enable it to acquire control of the processor.

During console initialization, the console code builds the HWRPB and establishes a mere minimal environment.

- ⑲ It sets up its initial stack space, and initial heap, which is simply a block of memory used by the console routines. Stack space for the console is set up there.
- ⑳ It sets up the initial data structures for file names, and process context. Some of this happens before memory, some happen after memory testing and configuration. Several data structures are established at the console base in some portion of memory. This procedure is done prior to memory testing and is executed out of 1 Meg, essentially out of the backup cache. The data structures in this heap are file headers, file descriptors data structures, and device driver data structures.
- ㉑ It then starts the the initial drivers. The driver initialization sequence is as follows:
  - Phase 0 - TTY and serial control bus drivers
  - Phase 1 - Software Drivers
  - Phase 2 - Device Drivers

The driver startup RAM disk driver sequence is defined below:

- The Lbus driver is started and eventually communication to the console is available.
- The serial control bus is initialized so that the memory EEPROM may be access. This also uses the Lbus driver.
- The serial control bus driver causes the Lbus driver to test its mailboxes.
- ㉒ The serial control bus driver must be initialized for the memory test. To initialize the serial control bus driver, several other drivers must be functioning. The intent is to test the memory and eventually invoke the console. By initializing the serial control bus, the memory error log may be accessed. The serial control bus initialization sequence is as follows:
  - Driver initialization
  - Float 0 and 1 across three CSRs in the 8584 chip
  - Driver detected errors

### 21.3.3 Memory Testing

- ㉓ Full memory diagnostics are performed. The memory diagnostics, in general, first test the CMIC operation and then perform full tests on the DRAMS making sure that there is enough memory to operate at a very basic level.

Up to this point instructions have been executed out of 1 Meg of backup cache. There exists a 100 Kb used in heap space. The first 800 Kbytes of code are the console. This is the deviceless state, the "software-only" state is set up initially. Memory cell testing is performed by banking all four memory modules together and testing in parallel.

During any and all memory testing, errors that are detected are logged in the memory module's EEPROM that sits on the serial control bus. All errors are considered hard. When a certain threshold is reached, the board is removed from the configuration. The event logger uses a FIFO algorithm to capture

## 21.3 System Initialization

128 lines of text. Because it uses a FIFO algorithm, the user must transfer the error data into a file if it is to be used more than once.

The initial memory module test sequence follows:

- CSR tests
- Stream buffer tests
- Address line tests
- Bank uniqueness tests
- EDC correctable errors
- EDC uncorrectable errors
- EDC odd slice/even slice errors

The full DRAM testing includes

- Three pass gray code
- EDC used for error detection
- Modules tested in parallel mode

First, the error logs from the EPROMS on each memory module are extracted and used to precede the memory failure tables.

Unlike past consoles, this console will detect and mark out any pages that were previously marked in the EEPROM error logs on the subsequent reboot from some prior operation. Any failures that these tests detect will be written in to those EPROMS. The concept is any or less detected failures will also be written into those EEPROMS so if the system has been up and running and the operating system detects a bad memory location it will be recorded in the EPROMS and at the same point the next time the system is booted, it will be marked out of the bit map.

This also provides a way of getting around those soft correctable errors in places like VMS kernal where they can not get remapped or the next time around they get remapped because they are taken out before VMS gets there so that precedes the table.

- 24 Once the testing is complete, a combination of environment variables and module size are checked to determine interboard interleaving. In the absence of a user specified interleave, the default interleave for the fastest operation possible is used. To accomplish the interleaving requires that the console fit in one megabyte of memory executed out of backup cache. If the user has specified a set of interleave constraints, they will be validated to assure that they are achievable within the current module set and if they are achievable, then that is the interleaving that will be done.
- 25 Once memory testing is complete, the heap is expanded to its final size (which is about 2 Meg) and the rest of the drivers are started. Here, a memory heap structure be used by the diagnostics for the rest of memory is established.

### 21.3.3.1 Memory Testing and the Bitmap

The console must test enough memory to allocate memory for its own image and data structures, the HWRPB, PALcode, scratch areas, page tables, memory descriptors, and system software.

A hard error is defined as one which is uncorrectable by memory hardware, like a double-bit error. A correctable error (single bit error) is also considered a hard error in the context of the console powerup. Pages containing any errors are marked as unusable. The bitmap will also reflect errors logged in the EEPROM error log by the operating system. The pages will be mark unusable during the next console initialization.

### 21.3.4 Driver Initialization

② Then the rest of the drivers are started. Their hardware is tested as each driver is initialized. For example, when the driver for the NCR port is activated, the NCR chip is tested, The DSSI/SCSI initialization test sequence is as follows:

- Script RAM data/address lines, memory cells
- Data lines on the Lbus sides of each NCR chip
- Driver startup detects subsystem failures
- Diagnostic script testing of remaining components

When the TGEC's are brought up, the bus termination is checked. All the function testing that is done by the console is done as the drivers are initialized by the driver initialization routines.

Once all the drivers initialize, the power-on script is invoked which checks some other functions on the I/O module. It checks to make sure that the TOY clock batteries have never gone dead.

It checks jumpers in the SCSI only port to make sure the continuity cards are in place.

Access to the power subsystem is initialized. A series of basic network loopback tests is run and then a series of memory exercisers is run.

The Futurebus+ drivers are initialized.

- Size the system—The initial configuration of Profile B I/O devices on Digital systems is split between the console code and the device drivers. Following power-on the console configures the various registers on the system bus bridge and then probes the Futurebus+ to determine which devices are present. The probing of each node is begun when all nodes on the bus have released the reset line. The IEEE Futurebus+ standard allows up to two nodes on each module (sides 0 and 1). Modules that have only one node are required to place the node on side 0. An empty slot will not acknowledge either node address of the module.
- Set Configuration—The configuration routine performs the following operations on each module after power-up, system reset, and bus initialization.
  - Check TEST\_STATUS CSR for successful completion of initialization and initialization test.

## 21.3 System Initialization

- Read MODULE\_VENDOR\_ID and MODULE\_SW\_VERSION to determine the device driver associated with the module.
- Concatenate MODULE\_VENDOR\_ID and MODULE\_SW\_VERSION to form a unique number that relates a device to a driver, The IEEE 1212 CSR standard supports other ROM entries for this purpose if the module has more than one type of node or unit.
- Write the BUS\_PROPAGATION\_DELAY CSR in each node to properly set the glitch filters.
- Read the MODULE\_LOGICAL\_CAPABILITY ROM CSR of each node to determine the module capabilities for address and data width.
- Read the NODE\_CAPABILITIES\_EXT\_ROM CSR of each node to determine the node address and data width capabilities.
- Read the NODE\_CAPABILITIES\_EXT\_ROM CSR of each node to verify that the node is capable of extended test.
- Write the NODE\_IDS SCR to update the bus number, if applicable (the default value is 1023).
- Write the SPLIT\_TIMEOUT, TRANSITION\_TIMEOUT, BUSY\_RETRY\_DELAY, BUSY\_RETRY\_COUNTER, ERROR\_RETRY\_DELAY, and ERROR\_RETRY\_COUNTER CSRs to the correct value for the system.
- Write the LOGICAL\_MODULE\_CONTROL CSR to set the correct values for the 64\_BIT\_ADDRESS bit, DATA\_WIDTH field, MASTER\_ENABLE bit, and COMPELLED\_DATA\_LENGTH field (PARITY\_REPORT\_ENABLE bit clear). The values written to these fields are determined by the module's capabilities CSRs. There is no need to write the LOGICAL\_COMMON\_CONTROL CSR on Profile B modules.
- If the node has implemented extended tests (as determined earlier by the read of the NODE\_CAPABILITIES ROM), then the ARGUMENT CSRs will be written with the starting address of the 4-Kbyte block of memory for use by extended diagnostics.
- Write the TEST\_START CSR to initiate the default extended test suite.
- Check the TEST\_STATUS for successful completion of extended tests.
- Enable parity error reporting on the node by rewriting the LOGICAL\_MODULE\_CONTROL CSR with the correct values for PARITY\_REPORT\_ENABLE, 64\_BIT\_ADDRESS bit, DATA\_WIDTH field, MASTER\_ENABLE bit, and COMPELLED\_DATA\_LENGTH field.

The device driver is responsible for configuring and MEMORY\_BASE, MEMORY\_BOUNDS, UNIT\_BASE, UNIT\_BOUNDS and any initial unit space CSRs.

- 27 Once console initialization is complete, auto action flags, auto boot flags, are read. (Auto action flags are all SRM defined.) Here console initialization is complete and the next action is either to boot, restart, or remain at the console level, depending on the state of autoboot.

## 21.3 System Initialization

### 21.3.4.1 I/O Adapter Configuration

The console performs only minimal initialization of the I/O bus adapters following Self test.

At this point the 21064 SROM code and the D-bus microcontroller are synchronized. The console display shown system status.





*Halt* is the term given to the action of stopping the processor from normal processing.

Halt can occur in one of several ways:

- The HALT switch on the OCP
- A system error
- A HALT command entered at the console on a multiprocessor system.

## 22.1 The HALT command

Internal system error or halt button operation on the operator control panel. A HALT command entered at the console terminal, has no effect on single processor systems because it is assumed that the CPU is already halted if the firmware code has been invoked. However on a dual processor system, the second processor can be halted. If a processor is halted via an error detection, processor control is typically passed to the firmware code.

## 22.2 The HALT Switch on the OCP

The operator control panel halt switch causes `CSYS_EVENT_L` to assert and the state of the switch can be read by a CPU via the serial control bus. The `CHALT_L` pulse generated by the operator control panel is 200 to 500 ns. The switch is latched, so that software can use the switch state to decide to reboot. The switch may be in either state during power-up, reset, or power-down sequences, however the `CHALT_L` signal is normally negated.

## 22.3 System Error

A system error will cause a system halt and force the processor into console IO mode.

---

## The Bootstrap State

Bootstrapping is the process of locating and loading primary program image and transferring control to it. The system firmware uses a bootstrap procedure defined by the Alpha AXP Architecture. On a DEC 4000 AXP system, bootstrap can be attempted *only* by the primary processor, commonly referred to as the "boot processor". The firmware uses device and optional filename information specified either on the command line or in appropriate environment variables. There are only three conditions where the boot processor attempts to bootstrap the operating system:

1. The BOOT command is typed on the console terminal.
2. The system is reset and AUTO\_ACTION = BOOT.
3. An operating system restart is attempted and fails.

The function of the firmware in the bootstrap state is to load a program into memory and begin its execution. This program may be a primary bootstrap program such as Alpha Primary Boot (APB), Ultrixboot, or any other applicable program specified by the user or residing in the boot block.

## 23.1 DEC 4000 Bootstrap Algorithm

The console maintains a "bootstrap in progress" flag (BIP). This cold start flag is used to prevent repeated attempts to automatically bootstrap a failed system. The following algorithm is used to perform the system bootstrap:

1. Build a valid restart parameter block.
2. If this boot attempt is a result of a console BOOT command (an expected entry), skip to step 3 and clear the expected entry flag.
3. If the BIP flag is set, the boot fails.
4. Set the BIP flag.
5. Search the device database for a callback that matches the specified boot device. If none is found, the boot fails.
6. Load the boot parameters into the HWRPB.
7. Load the boot image from the boot device.
8. Transfer control to the boot device.

If the bootstrap fails, the console will display a message on the console terminal and return to the console prompt. If bootstrap succeeds, the operating system is responsible for clearing the BIP flag.

## 23.2 Environment Variables

An Alpha AXP system console uses environment variables. Environment variables provide a mechanism for defining console state and controlling console operation. Environment variables may be changed with system software, may change as a result of console state changes, and may be modified at the console. They may be read, written, or saved depending on the variable attribute. Each variable consists of an identifier (ID) and a value maintained by the console. There are three classes of environment variable.

1. Common to all Alpha AXP implementations
2. Specific to a given console

### 3. Specific to system software

The DEC 4000 system uses the Alpha defined variables as well as variables specific to its console (class 2). Appendix E lists all the environment variables, their attributes, their function, and identifies the environment variables that are specific to the DEC 4000 system.

## 23.3 Hardware Restart Parameter Block

Following successful completion of the power-up script, the firmware initializes a data structure called the Hardware Restart Parameter Block or simply HWRPB. The HWRPB is a page-aligned data structure that serves as a communications area between the console and the operating system, as well as between CPUs in a multiprocessor system. The HWRPB is a critical resource during powerfail recoveries and other machine restart situations.

The HWRPB is created by the primary processor following a system reset. The HWRPB is not affected by a node reset of any device. Further information and status may be written into the HWRPB by software during and after bootstrapping. To speed the search for the HWRPB during CPU restarts and to ensure that only the real HWRPB is found, the HWRPB must be located in the lowest physical addresses of good memory. The console ensures that the HWRPB is both physically and virtually contiguous.

The console stores the physical address of the HWRPB in the non-volatile HWRPB\_ADDR environment variable during console initialization. When attempting a restart operation in a system where the contents of memory are backed up by battery, the console examines the memory pointed to by the environment variable for a valid HWRPB. System software must never move the HWRPB to a different memory location; operation of the system is UNDEFINED if such an attempt is made.

During the boot process, the console virtually maps the entire HWRPB. The total length of the HWRPB is dependent on the number of per-CPU slots supported. DEC 4000 allocates space for the maximum of 2 CPU slots, regardless of whether the slots are actually filled.

## 23.4 Console Callback Routines

The firmware uses a set of minimal device handler routines, called "call backs", to read the bootblock and, subsequently the primary bootstrap program into memory from the boot device. This technique is called "bootblock" booting. A pre-defined interface to these routines allows them to also be used by the primary bootstrap to load a secondary bootstrap or system image. The callback routines are defined by the Alpha AXP Architecture.

To begin a bootstrap, the firmware writes boot device information to the HWRPB and appropriate environment variables to be used by the boot callbacks and secondary bootstrap programs. The console locates a boot callback for the specified device. If a suitable callback is found, control is transferred to it.

If the target device is a disk, the callback loads logical block zero (the "bootblock") into memory. Unlike the VAX bootblock, the Alpha bootblock contains no code for loading the primary bootstrap. Rather, it contains descriptors that point to the logical block numbers (LBN) where the primary bootstrap program can be found. Initially, the LBN descriptor will point to the first of multiple contiguous blocks containing the primary bootstrap program. However, the architecture allows for

## 23.4 Console Callback Routines

the primary bootstrap program to be stored on discontinuous blocks, which would all be loaded into memory by the boot callback to form a single program.

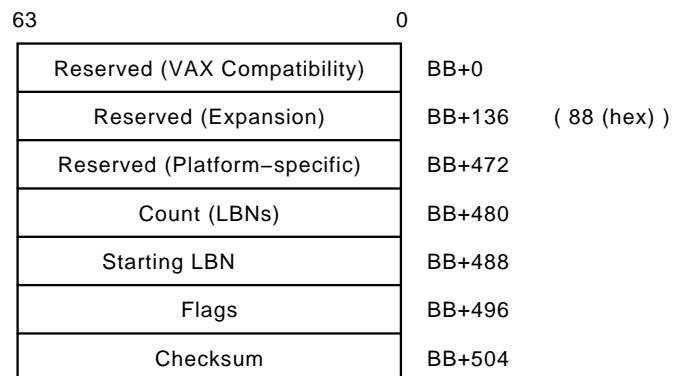
The structure of the Alpha bootblock is given in Figure 23–1. The Alpha LBN descriptor information is located at the end of the block. The VAX bootblock structure remains in the beginning of the block. This permits a VAX and Alpha to share a single, combined bootblock.

## 23.5 Boot Block Calling Interface

The CPU loads the primary bootstrap program by invoking a boot callback located in ROM. The boot callback initializes the boot device and reads the first logical block from the device into the first page of good memory. This "boot block" shown in Figure 23–1 contains information about the location of the primary bootstrap on the device. The boot callback uses this information to then load the primary bootstrap program.

The callbacks may use one or more pages at the top of the first 256Kb block of good memory to hold data structures required by the I/O adapters. Data read by the boot callback should not overwrite these structures.

**Figure 23–1 Alpha AXP Boot Block**



BOOTBLOCKA

Table 23–1 Alpha AXP Boot Block Field Definitions

Field Name	Description
Block Count	if Flag<0> is 0, this is the size, in blocks, of the primary bootstrap program. If Flag<0> is 1, then this is the number of additional boot blocks.
Start LBN	If Flag<0> is 0, then this is the starting LBN of the primary bootstrap program. If Flag<0> is 1, then this is the starting LBN of the additional boot blocks.
Flags	If bit 0 of the flag field is a 0, then the primary bootstrap program is contiguous and is completely described by the Block Count and Start LBN. If bit <0> is a 1, then the primary bootstrap program is discontinuous, and Block Count and Start LBN describe the additional boot blocks. Bits <63:1> are reserved.
Checksum	Checksum of bytes 0-1F7. Algorithm is a 64 bit 2's complement sum, ignoring overflows.

If the target device is not a disk, the boot callback must know how to locate the bootblock or the primary bootstrap program. Once the primary bootstrap program has been loaded, the console passes control to it.

## 23.6 Boot Devices

The boot devices supported are determined by the boot callbacks stored in ROM, and by the devices supported by the primary bootstrap program.

Table 23–2 Supported Boot Devices

Adapter	Bus	Device	Name
IO module	Ethernet	TGEC	EZAn
IO module	DSSI/SCSI	Disk	DIan
IO module	DSSI/SCSI	Tape	MIan
Futurebus+	FDDI	tbd	tbd

## 23.7 Boot Parameters

The console is responsible for loading various parameters into the HWRPB before transferring control to the primary bootstrap. These parameters are used to pass information to the operating system. Other information is passed to the operating system via environment variables. The boot parameters passed from the console to the operating system are described in Table 23–3.

Table 23–3 Boot Parameters

Parameter	Passed By
Boot device specification	BOOTED_DEV environment variable

(continued on next page)

## 23.7 Boot Parameters

**Table 23–3 (Cont.) Boot Parameters**

Parameter	Passed By
Boot control flags	BOOTED_OSFLAGS environment variable
Boot filename	BOOTED_FILE environment variable
Physical address of HWRPB	HWRPB_ADDR environment variable
Halt PC	Per-CPU Slot of HWRPB
Halt PS	Per-CPU Slot of HWRPB
Halt code	Per-CPU Slot of HWRPB
Address of good memory	Memory Data Descriptor Block

## 23.8 Disk and Tape Booting

Each tape boot callback rewinds the tape before it performs the first read, and before transferring control to the loaded image. The boot callback for a tape device 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 the console is assumed to be the first file on the tape. In this case, the boot callback skips to the first tapemark, and reads blocks into memory, storing them beginning at the address passed in 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.

## 23.9 Ethernet Booting

The local Ethernet device is the TGEC which resides on the I/O module. The DEC 4000 supports up to two local Ethernet ports, referenced by the firmware as devices, "eza0" and "ezb0".

Whenever a network bootstrap is selected on a DEC 4000 system, the bootstrap routine makes continuous attempts to boot from the network. The network bootstrap continues, until either a successful boot occurs, a fatal controller error occurs, or the boot is terminated with a Control-C.

Two Ethernet protocols are supported for network bootstraps, DECNET MOP and TCP/IP BOOTP. The environment variables, "eza0\_protocols" and "ezb0\_protocols", are used to define the default protocols to be used for booting from each port. These variables can be set to either "mop" or "bootp". By default, these variables are set to "mop bootp", enabling both boot protocols. Alternately, MOP and BOOTP attempts are made until the boot succeeds.

When the bootstrap succeeds control is passed to the loaded image.

### 23.9.1 MOP Booting

Whenever the environment variables, "eza0\_protocols" and "ezb0\_protocols", contain the string "mop"; the DEC 4000 bootstrap uses the DECNET MOP program load sequence for bootstrapping the system and the MOP "dump/load" protocol type for load related message exchanges.

The bootstrap routine, the requester, starts by sending a REQ\_PROGRAM message in the appropriate envelope to the MOP "dump/load" multicast address. In order to support both DECNET Phase VI and Phase V servers, the messages are sent in both formats, first four MOP V3 messages, then four MOP V4 messages at one second intervals. It then waits for a response in the form of a VOLUNTEER message from another node on the network, the MOP load server. If a response is received, then the destination address is changed from the multicast address to the node address of the server. The same REQ\_PROGRAM message is retransmitted to the server as an acknowledge which initiates the load.

Next, the console begins sending REQ\_MEM\_LOAD messages in response to either:

- A MEM\_LOAD message, while there is still more to load,
- A MEM\_LOAD\_w\_XFER, if it is the end of the image, or
- A PARAM\_LOAD\_w\_XFER, if it is the end of the image and operating system parameters are required.

The "load number" field in the load messages is used to synchronize the load sequence. At the beginning of the exchange, both the requester and server initialize the load number. The requester only increments the load number if a load packet has been successfully received and loaded. This forms the acknowledge to each exchange. The server will resend a packet with a specific load number, until it sees a request with the load number incremented. The final acknowledge is sent by the requester and has a load number equivalent to the load number of the appropriate PARAM\_LOAD\_w\_XFER message + 1.

Because the request for load assistance is a MOP "must transact" operation, the network bootstrap continues indefinitely until a volunteer is found. The REQ\_PROGRAM message is sent out in bursts of eight at four second intervals, the first four in MOP Version 4 IEEE 802.3 format and the last four in MOP Version 3 Ethernet format. The backoff period between bursts doubles each cycle from an initial value of four seconds, to eight seconds,... up to a maximum of five minutes. However, to reduce the likelihood of many nodes posting requests in lock-step, a random "jitter" is applied to the backoff period. The actual backoff time is computed as  $(.75 + (.5 * \text{RND}(x))) * \text{BACKOFF}$ , where  $0 \leq x < 1$ .

#### 23.9.1.1 MOP Network "Listening"

Whenever the DEC 4000 console is running, it "listens" on each of its ports for other maintenance messages directed to the node and periodically identifies itself at the end of each 8 to 12 minute interval, prior to a bootstrap retry. In particular, this "listener" supplements the MOP functions of the console load requester typically found in bootstrap firmware and supports:

- A remote console server that generates COUNTERS messages in response to REQ\_COUNTERS messages, unsolicited SYSTEM\_ID messages every 8 to 12 minutes and solicited SYSTEM\_ID messages in response to REQUEST\_ID messages, as well as, recognition of BOOT messages.
- A loopback server that responds to Ethernet LOOPBACK messages by echoing the message to the requester.



## 23.9 Ethernet Booting

- An IEEE 802.2 responder which replies to both XID and TEST messages.

### 23.9.2 BOOTP booting

Whenever the environment variables, "eza0\_protocols" and "ezb0\_protocols", are set to "bootp"; the DEC 4000 attempts a Internet boot. The console implements TFTP and BOOTP client protocols for network bootstrapping in an Internet environment. Supporting these TFTP and BOOTP requires pieces of UDP, IP and ARP.

Note that behavior of this firmware depends in part upon behavior of the software running on the server host, which varies from server to server. For example, the exact format of the file path name specification used with TFTP depends on the server: the Ultrix TFTP server requires a partial path name, and the OSF server requires a complete path name. Unix systems frequently name the TFTP server "tftpd" and the BOOTP server "bootpd". (See the appropriate system documentation for server details. )

#### 23.9.2.1 Setting up the Console for BOOTP Boots

For BOOTP and TFTP to operate reliably, several network parameters, contained in environment variables, must be properly configured. The Internet protocols are robust and thus may work intermittently if the parameters are misconfigured, which can make debugging a misconfiguration difficult. The next few paragraphs define what you need to know to get the BOOTP software working.

---

#### Note

---

Each network port has a set of environment variables prefixed with the port device name, either "eza0\_" or "ezb0\_". To examine the current network port environment variables use the console "show" command, for instance:

```
>>>show eza0*
>>>show ezb0*
```

In the following discussion only the port "eza0" is configured, however the same procedure would apply to "ezb0" for BOOTP configuration.

---

First, the environment variable "eza0\_protocols" should include the string "bootp" to enable BOOTP, TFTP, etc. (The variable may also include "mop" to enable MOP booting.) In particular, if not enabled the BOOTP software will not be invoked for booting. Also, the network driver may not enable reception of broadcast packets, which breaks ARP.

Each interface has a small database of information required to operate the BOOTP software on that interface. Internally the database is kept in a 300 byte structure having the same format as a BOOTP packet. This database can be directly read and written in binary form through the BOOTP protocol driver; more on that later. The four most important fields of the database can be accessed in a friendlier fashion through the environment variables "eza0\_inetaddr", "eza0\_sinetaddr", "eza0\_ginetaddr", "eza0\_inetfile". The first three are the Internet addresses for the interface (eza0), the remote server host, and remote gateway host, respectively. These variables use Internet standard dotted decimal notation; e.g., "16.123.16.53". "Eza0\_inetfile" contains a file to be booted and is formatted simply as a string.

The most important of these four is the local address, "eza0\_inetaddr". TFTP and ARP will not operate properly without the correct address. "Eza0\_ginetaddr" is the address of an Internet gateway on the local network. TFTP cannot communicate beyond the local network if this gateway address is not correct. "Eza0\_sinetaddr" is the address of a server, which may or may not be on the local network. Ordinarily this is the server from which to boot. This is the default remote host contacted by TFTP. "Eza0\_inetfile" is ordinarily the file to be booted. This should be a fully qualified file name, according to whatever rules are specified by the TFTP server on the remote host. This is the default file name requested by TFTP.

The interface database must be initialized somehow before TFTP can be used. The database can be initialized by manually setting the four database variables, by explicitly invoking BOOTP, or automatically on the first invocation of TFTP. Whether initialization occurs on the first TFTP depends on whether the database has been marked as initialized. The database will be marked as initialized on the first occurrence of any of three events: the invocation of TFTP, the invocation of BOOTP, or the setting of any of the four database environment variables.

The most common case is the invocation of TFTP. When TFTP is invoked and the database has not been marked initialized then the database will be automatically initialized by one of two methods, as specified by the environment variable "eza0\_inet\_init". If "eza0\_inet\_init" is set to "bootp" (the default) the BOOTP protocol driver will be invoked to initialize the database by broadcasting a BOOTP request and storing the response in the database. If "eza0\_inet\_init" is set to "NVRAM" then the database will be initialized by copying the contents of four nonvolatile default variables into the four database variables. The four nonvolatile default variables are "eza0\_def\_inetaddr", "eza0\_def\_sinetaddr", "eza0\_def\_ginetaddr" and "eza0\_def\_inetfile". These variables obviously must be set in advance, for example:

```
>>>set eza0_def_inetaddr 16.123.16.53
>>>set eza0_def_sinetaddr 16.123.16.242
>>>set eza0_def_ginetaddr 16.123.16.242
>>>set eza0_def_inetfile bootfiles/vmunix
>>>set eza0_inet_init NVRAM
```

When BOOTP is invoked (either explicitly or via the automatic initialization discussed above) the database is marked as initialized. In the usual case where BOOTP is successfully invoked without the "nobroadcast" parameter (ie, as "bootp/eza0" or "bootp:broadcast/eza0") the received reply packet is copied into the database, thus initializing it. If the "nobroadcast" parameter is specified (ie, "bootp:nobroadcast/eza0") then no request is broadcast and thus no reply is received to copy into the database. However, the database is still marked initialized, so a following TFTP will not automatically initialize the database.

When one of the four database environment variables is set the database is marked as initialized. Thus a following TFTP will not automatically initialize the database, regardless of whether the environment variables were set to sensible values.

TFTP, BOOTP, and ARP all use retransmission to improve robustness. If an initial transmission is not answered appropriately, the protocol software will retransmit. Each protocol has an environment variable which controls the number of retries before giving up. The variables are named "eza0\_arp\_tries", "eza0\_bootp\_tries", and "eza0\_tftp\_tries". The default value of these is 3, which translates to an average of 12 seconds before failing (see the discussion of retransmission timing below). If the value of one of these variables is less than

## 23.9 Ethernet Booting

1, the protocol will fail immediately. Machines located on very busy networks or associated with heavily loaded servers may need these variables set higher.

The retransmission algorithms use a randomized exponential backoff delay. If the first try fails a second try will occur about 4 seconds later. A third try would come after another 8 seconds, a fourth after 16 seconds, and so forth up to 64 seconds. These times are actually averages, however, since random jitter of about +/- 50% is added to each delay. This implies that with "eza0\_arp\_tries" set to 3 ARP will fail if it doesn't get a response within about 12 seconds on average, but the actual timeout will be somewhere between 6 and 18 seconds.

### 23.9.2.2 Using BOOTP bootstrap

The normal use of BOOTP and TFTP is for bootstrapping across a network. However, they may be explicitly invoked as protocol drivers. The BOOTP and TFTP protocols must be followed by a network in the protocol tower.

Reading from the BOOTP protocol driver reads the 300 byte binary database associated with the network. If the parameter "broadcast" is specified, or if no parameter is specified, the protocol will first broadcast a BOOTP request and update the data base with the reply. If the parameter "nobroadcast" is specified then the broadcast and update are omitted, and the existing content of the database is read. Writing to the protocol will overwrite the database. (When manipulating the database by hand it is usually easier to use the database environment variables.) Some examples follow. Note that the "hd" (hex dump) command is used to display the binary data.

```
>>># Examine the current contents of eza0's database:
>>>hd bootp:nobroadcast/eza0
>>># Update the database from a BOOTP server and examine the result:
>>>hd bootp/eza0
>>># Update the database from a BOOTP server but don't look at the result
>>>cat bootp/eza0 >nl
>>># Copy the contents of (binary) file foo into eza0's database:
>>>cat foo >bootp/eza0
```

When a BOOTP request is broadcast, the environment variable "eza0\_bootp\_server" is copied into the "sname" field of the request packet and the variable "eza0\_bootp\_file" is copied into the "file" field of the request packet. The exact interpretation of these fields depends on the BOOTP server. The "sname" field should be the name of a specific host which the local machine wants to boot from. If it doesn't matter which server answers, then the variable "eza0\_bootp\_server" should be left empty. The server should use the "file" field in the request to decide which boot file to specify in the response. For example, the client could supply a generic name like "unix" or "lat", and the server would respond with the fully qualified file path to be used with TFTP. If a machine will always be booting the same file then "eza0\_bootp\_file" can be left empty.

The TFTP protocol driver is used to read files across the network. Writes are currently unimplimented. TFTP accepts one parameter, the host address concatenated to the file name of the remote file to be read. The host address is specified in dotted decimal notation and is separated from the file name by ':'. If the file name includes '/' they must be doubled to '//'. The following example displays the file '/usr/foo/bar' from the host whose address is 16.123.16.242:

```
>>>cat tftp:16.123.16.242://usr//foo//bar/eza0
```

## 23.9 Ethernet Booting

For convenience the address could be saved in an environment variable:

```
>>>set ktrose 16.123.16.242
>>>cat tftp:$ktrose://usr//foo//bar/eza0
```

If no parameter is specified TFTP uses the file name and server address from the interface database (ie, "eza0\_sinetaddr" and "eza0\_inetfile").

Note that when booting with TFTP, the boot command passes the contents of the environment variable "boot\_file" as the parameter to tftp. If "boot\_file" does not have the correct format TFTP will fail. The most common use is probably to leave "boot\_file" empty in which case TFTP will default to using "eza0\_sinetaddr" and "eza0\_inetfile", as above.



On DEC 4000 system, a processor will attempt to restart the operating system only when it has halted after encountering an error halt condition. DEC 4000 does not implement "warm start" or power-fail restart, because the battery backup supports the entire system cabinet.

If the AUTO\_ACTION environment variable is clear, then any CPU that is in need of a restart will remain halted. Specifically, on an error halt on any one CPU, that CPU remains halted. Conversely, if AUTO\_ACTION is enabled, then that state is true for all CPUs. For any one CPU that incurs an error halt, that CPU will be restarted whether or not it is a primary CPU. In other words, the environment variable dictates a system wide behavior.

Restart of the operating system is controlled by information contained in the hardware restart parameter block. The HWRPB\$Q\_RESTART field of the HWRPB contains the virtual address of the processor restart routine. This field is filled in by the operating system. The console also uses flags in the Per-CPU Slot of the HWRPB to indicate that a restart is in progress (RIP). The console code checks this flag to avoid repeatedly attempting to restart a failing system. The operating system is responsible for clearing the flag following the successful restart of a processor.

## 24.1 Restart Failure

If the restart of a primary processor fails, a message is displayed on the console terminal, and a bootstrap is attempted. A failed restart is considered a sufficiently serious condition to warrant abandoning whatever might still be running on the other processors.

A restart failure on a secondary processor is handled differently. The console will examine the BIP flag. If the bit corresponding to the failing processor is clear, the console will force a bootstrap. If the corresponding bit is set, the console will not force a bootstrap and the failing processor will enter console I/O mode.

The BIP flag for secondary processors is set by the operating system when it attempts to start a secondary processor. The operating system clears these bits when it is satisfied that the secondary processor has successfully started executing.

The purpose of this extra state is to avoid the following scenario peculiar to multiprocessors where:

1. A secondary processor encounters an error halt and then fails to restart.
2. The console forces a bootstrap.
3. The primary processor boots and begins running the operating system.
4. The primary processor starts the defective secondary processor (this sequence assumes that the secondary passes selftest).
5. The secondary repeats its error halt and failed restart.
6. The console again forces a bootstrap, causing the cycle to repeat.

---

### Note

---

A secondary processor cannot directly perform a reboot, because it cannot notify the other secondary processors that an expected entry is planned. Should the location of the primary change during the system reset, the fact that a boot was in progress could be lost. To avoid this problem, a secondary must force the primary into console mode (via NHALT)

and then signal that a boot is needed. The primary then performs the bootstrap.

---

## 24.2 Restart Parameters

The console will transfer control to the restart address specified in the HWRPB. The Per-CPU Slot in the HWRPB also contains all of the halt parameters needed to restart by the restart routine. These parameters include:

SLOT\$T_HWPCB	SLOT\$Q_HALT_PCBB	SLOT\$Q_HALT_SP
SLOT\$Q_HALT_PC	SLOT\$Q_HALT_PS	SLOT\$Q_HALTCODE

To maintain a consistent user interface across Alpha AXP platforms, the DEC 4000 console command syntax was designed to be common across Alpha platforms. The DEC 4000 console supports a set of common commands useful for system configuration and operating system bootstrap. Additionally, the DEC 4000 console provides services commonly found in VAX consoles and also implements many "U\*x-style" commands. This optional set of commands has been provided for development, manufacturing, repair, and support personnel; and are not necessary for normal system operation.

## 25.1 Console Prompt

DEC 4000 is by definition "halted" or in "console mode", whenever the firmware is executing. When halted, the firmware communicates with an operator through the device that is designated as the system console. The firmware delivers the following prompt on the console terminal indicating that it is awaiting command input.

```
>>>
```

In certain cases, the following line continuation prompt may appear.

```
_>
```

In general, the continuation prompt indicates that the operator has either explicitly requested line continuation using the "\", backslash or has implicitly invoked line continuation by not completing a console command. The console will return to the standard prompt when the command has been properly terminated.

## 25.2 Command Conventions

The DEC 4000 console accepts commands of lengths up to 255 characters, excluding the terminating carriage-return or deleted characters. Longer commands cause an error message to be issued. A command line with no characters is considered a valid, null command. No action is taken, but the console redisplay the prompt.

White space is any contiguous sequence of spaces or tabs and is treated as a single space. White space is required to separate command line elements. However, leading and trailing white space is ignored.

Command keywords may be abbreviated to the smallest number of characters to unambiguously identify the keyword. Certain frequently used commands have single character abbreviations, which uniquely identify the command. For instance, "E" is always interpreted as "EXAMINE".



## 25.2 Command Conventions

Command "qualifiers" or "options" are prefixed with a dash "-", which in turn must be preceded by at least one space. Options may be placed anywhere after the command keyword. However, any option arguments must follow the option designator and be preceded by at least one space.

---

### Note

---

The terms "qualifiers" and "options" in this document are synonymous.

---

All numbers are in hexadecimal unless otherwise specified. Note that the register names (R0, R1, etc.) are not considered numbers, and use decimal notation. The console commands are not case-sensitive, and may be invoked by entering characters in either case. Characters are echoed in the case that they are input.

The console supports command line recall and editing.

## 25.3 Console Special Characters

Control characters are entered by holding down the Control key and then hitting the desired KEY. The following is a summary of the control characters and other special characters which the DEC 4000 console supports.

**Table 25-1 Console Special Characters**

<code>Return</code>	Terminates command line input. No action is taken on a command until after it is terminated by a carriage return. A null line terminated by a carriage return is treated as a valid, null command. No action is taken, and the console re-prompts for input. Carriage return is echoed as carriage return, line feed.
<code>&lt;x</code>	When the operator hits DELETE key, the console deletes the character that the operator previously typed. The console does not delete characters past the beginning of a command line. If the operator types more DELETES than there are characters on the line, the extra DELETES are ignored. If a DELETE is typed on a blank line, it is ignored.
<code>Ctrl A</code>	Toggle insertion/overstrike mode for command line editing. By default, the console powers up to overstrike mode.
<code>Ctrl B</code> or <code>up-arrow</code> or <code>down-arrow</code>	Recall previous command(s). The last sixteen commands are stored in the recall buffer.
<code>Ctrl C</code>	Echo "^C" and to terminate foreground command process. The Control-C function will terminate any command running in the foreground and return control to the command shell which invoked it. On the command line, Control-C behaves the same as Control-U. Control-C has no effect as part of a binary data stream. Control-C clears control-S, and reenables output stopped by control-O.
<code>Ctrl D</code> or <code>left-arrow</code>	Moves the cursor left one position.
<code>Ctrl E</code>	Moves the cursor to the end of the line.

(continued on next page)

Table 25–1 (Cont.) Console Special Characters

<code>Ctrl F</code> or <code>right-arrow</code>	Moves the cursor right one position.
<code>Ctrl H</code>	Moves the cursor to the beginning of the line.
<code>Ctrl O</code>	Suppress output to the console terminal until the next Control-O is entered. Control-O is echoed as "^O" when it disables output, but is not echoed when it reenables output. Output is reenabled if the console prints an error message, or if it prompts for a command from the terminal. Displaying a REPEAT command does not reenables output. When output is reenabled for reading a command, the console prompt is displayed. Output is also enabled Control-S.
<code>Ctrl Q</code>	Resumes output to the console terminal. Additional Control-Q strokes are ignored. Control-Q is not echoed.
<code>Ctrl S</code>	Stops output to the console terminal, until Control-Q is typed. Control-S is not echoed.
<code>Ctrl U</code>	Echoes "^U" and deletes the entire line. If Control-U is typed on an empty line, it is echoed, and the console prompts for another command.
<code>Ctrl R</code>	Echoes Return and Linefeed followed by the current command line.
<code>Ctrl P</code>	When in console mode, <code>Control-P</code> is ignored. When in operating system mode and "tta*_halts" is set to 2, <code>Control-P</code> is enabled and causes the processor to halt and enter console mode. When in operating system mode and "tta*_halts" is set to 0, <code>Control-P</code> is disabled.
<code>BREAK</code>	When in console mode, <code>BREAK</code> is ignored. When in operating system mode and "tta*_halts" is set to 4, <code>BREAK</code> is enabled and causes the processor to halt and enter console mode. When in operating system mode and "tta*_halts" is set to 0, <code>BREAK</code> is disabled.

## 25.4 Environment Variables

An environment variable is a name and value association maintained by the console program. These variables control various console features, and are used to pass console information to the operating system. The value associated with an environment variable is either an ASCII string or an integer.

Environment variables may be modified, displayed, and deleted (if possible) using the SET, SHOW, and CLEAR commands. The SET command also allows the user to create new environment variables. Certain environment variables are typically modified by a user to tailor the recovery behavior of the system on power-ups and other fatal system failures. Additional environment variables are used to control execution of diagnostics.

The environment variables which are defined by the console program are summarized in Appendix E. A default value is associated with any variables which are stored in NVRAM. This default is used if the variable is not set, or if NVRAM is unreadable.

## 25.5 Command Overview

The DEC 4000 console is a hybrid of a VAX console and U\*X shell. Although the command line parser expects U\*X style commands, many commands are very similar to VAX console commands. Some commands are unique to this console. Table 25–2 lists frequently used commands from each of these groups. Appendix D is an alphabetized command reference section containing complete descriptions of the each of the console commands.

## 25.5 Command Overview

By cloning certain U\*X functions and carrying along some VAX console functions, we have taken advantage of existing paradigms, rather than reinventing or renaming similar functions. Due to popular demand, however, the prompt has remained the same as VAX consoles, the triple angle prompt, ">>>".

Instead of VAX-like **/qualifiers**, the DEC 4000 console uses U\*X-like **-options**. For example, a VAX console command, such as **e/b 0**, must be typed **e -b 0**, because you must use a space to separate the option from the command. If you type **e-b 0**, the console issues an error message.

Once you are familiar with console operation, use the command summary in Table 25-4 and shell operators in Table 25-3 as a quick reference guide. A complete description of all the commands is provided the repository in Appendix D. Below are a few of the frequently used console commands.

Table 25-2 Frequently Used Commands

VAX-like Commands	U*X-like Commands	Unique Commands
boot	cat	cbcc
examine	echo	cdp
deposit	eval	edit
help	grep	exer
set	hd	memexer
show	ls	memtest
test	man	nettest
	ps	sa
	sleep	

## 25.6 Getting Information About the System

The following commands may be used to provide information about resident software and hardware in the system.

## 25.6 Getting Information About the System

```
>>>show version
version                               V2.5-5744 Oct 28 1992 09:08:24
>>>show pal
pal                                   VMS PALcode X5.25A, OSF PALcode X1.14A

>>>show device
dka200.2.0.0.0                        DKA200                                RZ57
dkd0.0.0.3.0                          DKD0                                  RZ35
dkd100.1.0.3.0                        DKD100                               RZ35
dkd200.2.0.3.0                        DKD200                               RZ35
dkd300.3.0.3.0                        DKD300                               RZ35
duc1.1.0.2.0                          $1$DIA1 (RF0201)                     RF72
mke100.1.0.4.0                        MKE100                                TLZ04
eza0.0.0.6.0                          EZA0                                  08-00-2B-1D-02-8F
ezb0.0.0.7.0                          EZB0                                  08-00-2B-1D-02-90
fbc0.0.0.6.1                          FBC0                                  Fbus+ Profile_B Exercis
p_b0.7.0.1.0                            Bus ID 7
pka0.7.0.0.0                          PKA0                                  SCSI Bus ID 7
pkd0.7.0.3.0                          PKD0                                  SCSI Bus ID 7
pke0.7.0.4.0                          PKE0                                  SCSI Bus ID 7
puc0.7.0.2.0                          PIC0                                  DSSI Bus ID 7

>>>show config
Console V2.5-5744                      VMS PALcode X5.25A, OSF PALcode X1.14A
CPU 0      P      B2001-AA DECchip (tm) 21064-2
CPU 1      -
Memory 0   P      B2002-BA 32 MB
Memory 1   -
Memory 2   -
Memory 3   P      B2002-CA 64 MB
Ethernet 0 P      08-00-2B-1D-02-8F
Ethernet 1 P      08-00-2B-1D-02-90

          ID 0   ID 1   ID 2   ID 3   ID 4   ID 5   ID 6   ID 7
A      SCSI   P          RZ57
B      P
C      DSSI   P      RF72
D      SCSI   P      RZ35  RZ35  RZ35
E      SCSI   P      TLZ04
Futurebus+ P          FBC0

System Status Pass          Type b to boot

>>>
```

## 25.7 Online HELP

The DEC 4000 console also provides online **help**. In ROM-based images of the console, *brief* help is supplied for each command. The *brief* help displays a one-line description and the syntax for command. The syntax line lists all possible options and arguments for the command. For instance, in the following example, the command requests help on **help**:

## 25.7 Online HELP

```
>>>help help
NAME
    help or man
FUNCTION
    Display information about console commands.
SYNOPSIS
    help or man [<command>...]
    Command synopsis conventions:
    <item> Implies a placeholder for user specified item.
    <item>... Implies an item or list of items.
    [] Implies optional keyword or item.
    {a,b,c} Implies any one of a, b, c.
    {a|b|c} Implies any combination of a, b, c. )

>>>
```

The console also supports the U\*x alias **man** for on-line help. If you do not specify a help topic when invoking the **help** or **man** command, a complete list of commands is displayed in addition to the help **help** brief text.

```
>>>man
NAME
    help or man
FUNCTION
    Display information about console commands.
SYNOPSIS
    help or man [<command>...]
    Command synopsis conventions:
    <item> Implies a placeholder for user specified item.
    <item>... Implies an item or list of items.
    [] Implies optional keyword or item.
    {a,b,c} Implies any one of a, b, c.
    {a|b|c} Implies any combination of a, b, c. )
```

The following help topics are available:

```
alloc      bin          boot          build         cat
cbcc       cdp          check         chmod         chown
clear      cmp          continue     crc           date
deposit    dynamic     echo         edit          eval
examine    exer        exer_read    exer_write    exit
fbus_diag find_field  free         grep          hd
help/man   Init        io_test     kill          kill_diags
line       ls          memexer     memexer_mp    memtest
net        netexer     nettest     ntlpex        ps
psc        rm          sa           semaphore     set
set host   show        show cluster show config   show device
show error show fru    show hwrpb  show map      show memory
show_status sleep      sort         sp            start
stop       sw          test         tr            uniq
update     wc

>>>
```

You can specify multiple topics with the **help** command, as shown below. Type a space between topics to keep them separate from each other:

```

>>>help examine deposit
NAME
    examine
FUNCTION
    Display data at a specified address.
SYNOPSIS
    examine [-{b,w,l,q,o,h,d}] [-{physical,virtual,gpr,fpr,ipr}]
            [-n <count>] [-s <step>]
            [<device>:]<address>

NAME
    deposit
FUNCTION
    Write data to a specified address.
SYNOPSIS
    deposit [-{b,w,l,q,o,h}] [-{physical,virtual,gpr,fpr,ipr}]
            [-n <count>] [-s <step>]
            [<device>:]<address> <data>

>>>

```

The **help** command also supports a type of wildcarding. In the following example, **help on** any command beginning with the letters **da** will be displayed.

```

>>>help da
NAME
    date
FUNCTION
    Set or display the current time and date.
SYNOPSIS
    date [<yymmddhhmm.ss>]

>>>

```

To display help for all commands, type **help \***.

```
>>>help *
```

## 25.8 Examining and Depositing

In the DEC 4000 console (as is true in U\*x) many commands act on byte streams. A byte stream is similar in concept to a VAX console address space and may represent an extent of memory, a set of registers, a device or a file. The console manipulates these byte streams by performing typical device operations—open, read, write, close. Therefore, throughout this document the term *device* will be used to refer to any such byte stream or address space regardless of its actual physical implementation. With this understood a traditional VAX address space, /P, can be accessed as a *device*, PMEM.

Hence, the **examine** and **deposit** commands manipulate *devices* when accessing data within the system. The default *device* is physical memory, which sticks (all subsequent implicit references access that *device*) until explicitly changed. When another *device* is specified, that *device* becomes the default.

Internally, the console uses *drivers* as the access mechanism for referencing different *devices*. Specifically, the console provides *drivers* for the following Alpha *devices*.

```

pmem: - physical memory
vmem: - virtual memory
gpr: - general purpose registers
fpr: - floating point registers

```

## 25.8 Examining and Depositing

**i**pr: - internal processor registers

On DEC 4000 the following platform specific *devices* are also available.

**f**erom: - Intel 28F010 firmware FEPROM

**e**erom: - Environment variable NVRAM

**n**isa: - Ethernet station address ROM

**i**ic: - PCD8584 registers I2Cbus controller

**n**cr0(1,2,3,4): - NCR53710 registers DSSI/SCSI (ports 0-4)

**s**cram: - Script RAM

**t**gec0(1): - TGEC registers Ethernet (ports 0,1)

**t**oy: - DS1287A registers clock chip and NVRAM

**u**art: - Z8530 registers Console serial port

In this paradigm the address argument of a VAX console command becomes a byte *offset* within a *device* in a DEC 4000 console command. For example, **pmem:0** explicitly refers to the location in physical memory at offset zero, i.e. physical address 0. If no *device* name is supplied, the *offset* implicitly applies to the last *device* referenced (**pmem** by default). In the remaining discussions, however, the terms *address* and *offset* will be used synonymously.

In the console there is also the notion of a last referenced address. An **examine** or **deposit** command without an explicit address will always reference the next address (computed as the last referenced address plus the current data size). The characters +, \*, and - are symbolic addresses for the next, current, and previous addresses, respectively.

Data width options are analogous to the corresponding VAX qualifiers. That is, **-b**, **-w**, **-l**, **-q**, **-o**, **-h** options correspond to the size of the accessed data—byte, word, longword, quadword, octaword, and hexaword.

### 25.8.1 Accessing Memory

Before randomly experimenting with memory, it is important to find a "safe" area in memory to alter. Since the console itself and other critical data structures reside in memory, care should be taken not to alter them. The **alloc** command may be used to allocate a 1000 byte block of memory, as shown in the following example.

```
>>>alloc 1000
03FFF000
>>>
```

The **alloc** command returns the address of the allocated block, in this case, 03FFF000. Hence, in the following examples this block will be used for experimentation.

The next example shows the **examine** and **deposit** commands to the **pmem:** device (physical memory) at the allocated address.

```
>>>deposit pmem:3fff000 1 # Deposit a 1 at address 3fff000.
>>>examine pmem:3fff000 # Examine the location.
pmem: 3FFF000 00000001
>>>
```

The next example shows the use of the abbreviated form of the commands, **e** and **d**. Abbreviations for commands are permitted and typically also, as in this case, the device specifier is absent. Assuming the state left by the previous example, the current device is still physical memory (or **pmem:** ).

## 25.8 Examining and Depositing

```
>>>d 3fff000 deadbeef          # Deposit new data there.
>>>e 3fff000                  # Check it out.
pmem:          3FFF000 DEADBEEF
```

Below is an example using a console command option.

---

### Note

---

Remember, the console uses U\*X-like **-options**, not VAX-like **/qualifiers**. Also notice the inconspicuous use of white space.

---

In this example the **-n** option is used to specify a repeat count. Each command is executed over "n+1" successive addresses.

```
>>>d 3fff000 aaaa5555 -n 3      # Write to 4 locations, yes 4!
>>>e 3fff000 -n 3              # Notice that -n 3 yields n+1 or 4!
pmem:          3FFF000 AAAA5555
pmem:          3FFF004 AAAA5555
pmem:          3FFF008 AAAA5555
pmem:          3FFF00C AAAA5555
>>>
```

The console provides a *hex dump* command, **hd**, as an alternate method for dumping memory (or other devices or files). Here the **-l** option specifies the number of bytes to display. (Why **-l** for **hd** and **-n** for **examine**? Because **examine** is a VMS-like command and **hd** is a U\*X cloned command.)

```
>>>hd pmem:3fff000 -l 10      # Dump the allocated memory.
00000000 55 55 aa aa 55 55 aa aa 55 55 aa aa 55 55 aa aa UU^a^UU^a^UU^a^UU^a^
>>>hd -l 20 show_status      # Dump part of SHOW_STATUS script.
00000000 65 63 68 6f 20 27 64 2f 53 27 20 3e 24 24 73 73 echo 'd/S' >$$ss
00000010 0a 65 63 68 6f 20 27 2d 2d 2d 27 20 3e 3e 24 24 .echo '---' >>$$
>>>
```

### 25.8.2 Examining registers

The following examples show the **examine** and **deposit** commands used to reference registers. Registers may be addressed:

- *Symbolically*, for instance **r0** or **ksp**
- *Explicitly*, as offsets within device address space; for instance, **gpr:0** or **ipr:0**
- *Implicitly*, as offsets within the current device address space; for instance **0**

Also notice the usage of the symbolic relative addresses **+**, **\***, **-**, and the implied address increment (no address specified):



## 25.8 Examining and Depositing

```
>>>e r0                                # Examine R0 symbolically,...
gpr:                                     0 (   R0) 0000000000000002
>>>e gpr:0                               # explicitly as device offset,...
gpr:                                     0 (   R0) 0000000000000002
>>>e 0                                    # or implicitly as device offset.
gpr:                                     0 (   R0) 0000000000000002
>>>e 8                                    #                               R1,
gpr:                                     8 (   R1) 000000000000C408
>>>e                                     # and the next R2.
gpr:                                    10 (   R2) 0000000000000000
>>>e ipr:0                                # Try an IPR...
ipr:                                     0 (  ASN) 0000000000000000
>>>e                                     # and the next...
ipr:                                     1 ( ASTEN) 0000000000000000
>>>e +                                    # and the next...
ipr:                                     2 ( ASTSR) 0000000000000000
>>>e *                                    # and the current...
ipr:                                     2 ( ASTSR) 0000000000000000
>>>e -                                    # and the previous.
ipr:                                     1 ( ASTEN) 0000000000000000
>>>e ksp                                  # One by name...
ipr:                                    12 (  KSP) 000000000000F30
>>>e                                     # and the next.
ipr:                                    13 (  ESP) 0000000000000000
>>>
```

The **examine** and **deposit** commands support symbolic representation of certain processor registers. In the following example, **pc**, **sp**, **ps**, are abbreviations for program counter, stack pointer, and process status longword.

```
>>>e pc                                  # Program Counter
PC psr:                                 0 (   PC) 000000000000D30
>>>e ps                                  # Process Status
ipr:                                    17 (   PS) 000000000001F00
>>>e sp                                  # Stack Pointer
gpr:                                    F0 (  R30) 000000000000F30
>>>
```

## 25.9 Using Pipes (|) and grep to Filter Output

The **grep** command is a convenient means of searching for information by filtering an input according to the expression argument supplied. A *pipe* (|) enables the output of one command to be the input to the next command, without creating an intermediate file. Because the **grep** command requires input, a pipe is used to channel the output of the **examine** command into the **grep** command.

In the following example, **grep** is used to search for a pattern in memory. In this case **grep** parses all the output lines from the **examine** command, but only permits lines which contain *deadbeef* to reach the display. The **grep** command can also be used to search for patterns that don't match the model provided; that is, search for every line that does not contain the input pattern.

## 25.9 Using Pipes (|) and grep to Filter Output

```
>>>d pmem:3fff000 0 -n 8 # Clear some memory.
>>>d 3fff020 deadbeef # Drop in a target.
>>>e 3fff000 -n 8 # Display memory.
pmem: 3FFF000 0000000000000000
pmem: 3FFF008 0000000000000000
pmem: 3FFF010 0000000000000000
pmem: 3FFF018 0000000000000000
pmem: 3FFF020 00000000DEADBEEF
pmem: 3FFF028 0000000000000000
pmem: 3FFF030 0000000000000000
pmem: 3FFF038 0000000000000000
pmem: 3FFF040 0000000000000000
>>>e 3fff000 -n 8 | grep DEADBEEF # Display only lines with DEADBEEF.
pmem: 3FFF020 00000000DEADBEEF
>>>
```

## 25.10 Using I/O Redirection (>)

Although default output goes to the console, you can redirect output to other devices or files by using the redirection operator `>`. In the following example, the output of an **examine** command is redirected to a file *foo*, which is created dynamically out of the console's memory heap. The console **cat** command, similar to the VMS **copy** command, is used in this example to display the contents of the created file *foo*. The **rm** command, similar to the VMS **delete** command, is used to delete *foo*.

```
>>>ls foo # Check to see if foo exists.
foo no such file
>>>e 3fff000 -n 1 > foo # Redirect examine output to file foo.
>>>ls foo # Does foo exist now?
foo
>>>cat foo # Yes! List foo.
pmem: 3FFF000 0000000000000000
pmem: 3FFF008 0000000000000000
>>>rm foo # Remove (delete) file foo.
>>>ls foo # Does foo exist now?
foo no such file
>>> # No.
```

## 25.11 Running Commands in the Background "&"

In a design verification testing (DVT) environment, the ability to run tasks in the background is an especially helpful feature. You can execute any command in the background by placing the background operator **&** at the end of the command. This capability alone makes it extremely easy to generate random activity in a system.

In the following example, three processes are started in the background. The first process, invoked with the console **exer** command, performs reads to block 0 of disk *dub2*, which you determine by using the console **show device** command). Next, two instantiations of the console memory test are created, using the **memtest** command. In all three cases, the console immediately returns with the console prompt and awaits further commands.

## 25.11 Running Commands in the Background "&"

```
>>>show device                                # See what devices are available.
dub2.2.0.1.0                                RF0102$DIA2          RF72
eza0.0.0.0.0                                EZA0                 08-00-2B-1D-02-91
ezb0.0.0.1.0                                EZB0                 08-00-2B-1D-02-92
pka0.7.0.0.0                                PKA0                 SCSI Bus ID 7
pkc0.7.0.2.0                                PKC0                 SCSI Bus ID 7
pkd0.7.0.3.0                                PKD0                 SCSI Bus ID 7
pke0.7.0.4.0                                PKE0                 SCSI Bus ID 7
pub0.7.0.1.0                                PIB0                 DSSI Bus ID 7
>>>exer dub2 -sb 0 -p 0 &                   # Read block 0 forever.
>>>memtest -p 0 &                             # Start up the memory test forever.
>>>memtest -p 0 &                             # Startup another memory test task.
>>>
```

## 25.12 Monitoring Status

The console monitors process status in several ways. The **ps** command is similar to the VMS **show system** command, as shown below. And **grep** can be used here to avoid unnecessary output.

```
>>>ps                                          # Display complete process status.
  ID      PCB      Pri CPU Time Affinity CPU Program      State
-----
0000006c 001423a0 3      2 00000001 0      ps running
0000005c 00144b40 2      19253 00000001 0      memtest ready
0000005b 00147a60 2      9 00000001 0      sh_bg waiting on 00144B40
00000059 0014c060 2      21750 00000001 0      memtest ready
00000058 0014edc0 2      5 00000001 0      sh_bg waiting on 0014C060
00000056 00152860 2      3 00000001 0      exer_kid waiting on mscp_rsp
00000055 00153ae0 2      2 00000001 0      exer waiting on exer_tqe
00000054 00181580 2      6 00000001 0      sh_bg waiting on 00153AE0
0000004f 00154d60 5      38 ffffffff 0      pke0_poll waiting on tqe
.
.
.
>>>ps | grep exer                             # Check exer.
00000056 00152860 2      6 00000001 0      exer_kid waiting on mscp_rsp
00000055 00153ae0 2      2 00000001 0      exer waiting on exer_tqe
>>>
```

Another method for monitoring diagnostic status is the **show\_status** command that lists the status of diagnostics or exercisers, such as **exer** or **memtest**. Note that the **show\_status** command is actually implemented as a built-in script, which you can execute by typing its name. You can easily list the contents of this script, other scripts or any other file by using the **cat** command.

## 25.12 Monitoring Status

```
>>>show_status
      ID      Program      Device      Pass Hard/Soft Bytes Written      Bytes Read
-----
00000001      idle      system      0      0      0      0      0
00000056      exer_kid dub2.2.0.1.0      0      0      0      0      3807232
00000059      memtest      mem      47451      0      0      388710400      388710400
0000005c      memtest      mem      46370      0      0      379854848      379854848
>>>
>>>cat show_status
# Sample built-in script.
echo 'd/S' >$$include
echo '---' >>$$include
echo system >>$$include
echo cpu >>$$include
echo memtest >>$$include
echo exer_k >>$$include
echo nettest >>$$include
echo cbcc >>$$include
echo io_test >>$$include
echo fbus_diag >>$$include

echo bg_N >$$exclude
echo ' nl ' >>$$exclude
echo ' tt ' >>$$exclude
echo tta0 >>$$exclude
echo ntlpex >>$$exclude
show_iobq|grep -f $$include|grep -v -f $$exclude
rm $$include
rm $$exclude
>>>
```

## 25.13 Killing a Process

To stop a process just use the process id from a **ps** command as the argument of the **kill** command.

```
>>>ps | grep memtest
0000005c 00144b40 2      135733 00000001 0      memtest ready
00000059 0014c060 2      138258 00000001 0      memtest ready
>>>kill 59
>>>ps | grep memtest
0000005c 00144b40 2      135733 00000001 0      memtest ready
>>>
```

## 25.14 Creating Scripts

A *script* is a file which contains console commands. The console contains many built-in scripts which you may execute by simply typing the name of the script file. The **show\_status** command describe earlier is an example of such a built-in script.

The console also provides a crude means of creating scripts. In the following example, the **echo** command is used to write characters to file *foo* using the output creation operator, **>**. The script *foo* is then displayed and executed.

```
>>>echo e pmem:3fff000 > foo
>>>cat foo
e pmem:3fff000
>>>foo
pmem:          3FFF000 0000000000000000
>>>
```

## 25.14 Creating Scripts

In the next example, addition characters are appended to *foo*. Note the usage of the single quote ' grouping character which encloses the desired text. The use of single quotes in the command line prevents the command-separator character ; from prematurely terminating the **echo** command. Also notice that the output append operator >> is used to extend *foo*.

```
>>>echo 'd 3fff000 5 ; e 3fff000' >> foo # Append "d 0 5 ; e 0" to foo.
>>>cat foo # List foo.
e pmem:3fff000
d 3fff000 5 ; e 3fff000
>>>foo # Execute foo.
pmem: 3FFF000 0000000000000000
pmem: 3FFF000 0000000000000005
>>>
```

You may enter a much longer script by reordering the command. Open a string with the single quote ', then enter the script (on several lines), then close the string with a second single quote '. For example,

```
>>>echo > foo 'ex 3fff000
_>d 3fff000 7
_>e 3fff000
_>d 3fff000 5
_>e 3fff000'
>>>cat foo
ex 3fff000
d 3fff000 7
e 3fff000
d 3fff000 5
e 3fff000
>>>foo
pmem: 3FFF000 0000000000000000
pmem: 3FFF000 0000000000000007
pmem: 3FFF000 0000000000000005
>>>
```

## 25.15 Using Flow Control

The console supports a limited number of flow control structures at the shell command level. The syntax for these constructs is as follows:

- **while** command\_sequence **done**
- **while** command\_sequence **do** command\_sequence **done**
- **until** command\_sequence **done**
- **until** command\_sequence **do** command\_sequence **done**
- **for** name **do** command\_sequence **done**
- **for** name **in** list **do** command\_sequence **done**
- **case** word **in** case\_part\_list  
pattern ) command\_sequence ;;  
[ pattern ) command\_sequence ;; ]  
**esac**

- **if** command\_sequence  
**then** command\_sequence  
[ **elif** command\_sequence **then** command\_sequence ]  
[ **else** command\_sequence ]  
**fi**

### Conditional Branching

Conditional branching in **if**, **while**, **until** loops is determined by the exit status of the command sequence following the control structure. In general, an exit status of zero indicates *success* and results in the execution of the *true* path.

In the following example, the **eval** command is used to extract an exit status from variable **junk**. The variable is initialized with the console **set** command.

```
>>>set junk 0
>>>show junk
junk                0
>>>eval junk
0
>>>if (eval junk) then (echo true) else (echo false) fi
0
true
>>>set junk 1
>>>if (eval junk) then (echo true) else (echo false) fi
1
false
>>>set junk 2
>>>if (eval junk)
_>then (echo true)
_>else (echo false) fi
2
false
>>>
```

### Loop Constructs

Byte swapping is a useful function when dealing with foreign bus architectures, such as Futurebus+. Byte swapping is the transposition of bytes in a bus address. A simple script called **bswap** is created here which will transform a longword value by swapping most significant bytes to least significant bytes.

Here the **for..do..done** construct is used in conjunction with some arbitrary environment variables **aa** and **bb** pass arguments to the script. The **for** variable **aa** takes on the values of all the arguments on the command line at the invocation of **bswap**. The **set** command is used to create **bb**, which is simply **aa** with **0x** prepended to it. (This permits the user to enter hexadecimal numbers without having to specify the radix prefix **0x**.)

In following example, the **eval** command is used to perform the transformation. The **eval** command uses:

- **&** operator for logical "and" function
- **|** operator for logical "or" function
- **>>** operator for logical shift right function
- **<<** operator for logical shift left function.

## 25.15 Using Flow Control

A close inspection of the postfix expression should reveal how it works.

```
>>>echo > bswap 'for aa; do
  _>set bb 0x$aa
  _>eval -x "$bb 0xff & 24 << $bb 0xff00 & 8 << $bb 0xff0000 & 8 >> $bb 0xff000000 & 24 >> | | |
  _>done'
>>>bswap 12345678
78563412
>>>bswap 12 1234 123456 12345678
12000000
34120000
56341200
78563412
>>>
```

In the following example, a simple **for** loop is used to create a more generic process status command:

```
>>>echo > stat 'for i
  _>do ps | grep $i
  _>done'
>>>cat stat
for i
do ps | grep $i
done
>>>stat memtest
00000131 00114e80 2          0 00000001 0      memtest ready
>>>stat memtest ps
00000131 00114e80 2          0 00000001 0      memtest ready
00000167 00108ea0 3          0 00000001 0          ps running
>>>
```

## 25.16 Console Shell

The DEC 4000 console is similar to a U\*X *shell*. A *shell* is a command line interpreter, an interface between the operator and the firmware. The lexical analyzer and parser for the DEC 4000 console implement a subset of the Bourne Shell with some minor modifications.

---

### Note

---

The article, *An Introduction to the UNIX Shell*, by S.R. Bourne describes the use of the *shell*. This article can be found in Part 4 of *ULTRIX-32 Supplementary Documentation Volume 1 User*.

---

An integral part of the console is its set of *shell* operators. These operators qualify the operation of commands, permit redirection of I/O, and allow for sequencing of operations. These operators are described in Table 25-3.

The DEC 4000 console command set consists of many U\*X-like commands, several VAX-like commands, and a unique set of commands specifically developed for diagnostics and design verification environments. Table 25-4 provides a quick lookup facility for the supported console commands.

---

### Note

---

For a complete description of all console commands refer to Appendix D.

---

Table 25–3 Console Shell Operators

Operator	Name	Form	Description
>	Output creation	>destination	Write output to destination.
>>	Output append	>>destination	Append output to destination.
<	Input redirection	<source	Read input from source.
<<	Here document	<<string...	Read input from standard input, until string is seen at the beginning of a line.
	Pipe	cmd1   cmd2	Pipe output of first command to input of second command.
;	Sequence	cmd1 ; cmd2	Run first command to completion before running second command.
\	Line continuation	cmd1 \ _> cmd2	Continue command on the next line. The command line prompt changes to "_>", until the command is completed.
#	Line comment	# text	The text following the number sign is ignored. This is useful for imbedding comments in command scripts or logs.
&	Background	cmd &	Run command in background, don't wait for command to complete.
&a	Affinity	&a m	Sets the processor affinity mask to allow this process to run on the CPU's defined by mask m. Multiple processors may also be specified as a list or range.
(,){	Grouping		Used to override precedence of pipe, sequence, and background operators.
*,?,[...]	Pattern specifiers		Characters used to form a regular expression for pattern matching. Where "*" matches on any character or characters or none, "?" matches on any single character, and "[...]" matches on any of the enclosed characters.
\$string	Environment variable substitution		The string is treated as a legal environment variable and translated.
'xxx'	String with no substitution		The string is passed untouched.
"String"	String with substitution		The string is passed after wildcards and environment variables are expanded.
'cmd'	Command substitution		Treat the string as a command string, execute it, and substitute in the resulting output.

In addition to shell operators, the console uses the following reserved words:  
**if then else elif fi case in esac for while until do done**



## 25.16 Console Shell

Table 25–4 DEC 4000 Console Command Summary

Command	Options	Parameters
<b>VAX-like Console Commands</b>		
boot	[-file filename] [-flags root,bitmap] [-halt]	[ boot_device ]
continue		
deposit	[-{b,w,l,q,o,h}] [-n val] [-s val]	[device:]address data
examine	[-{b,w,l,q,o,h,d}] [-n val] [-s val]	[device:]address
help		[command]
man		[command]
initialize	[-c] [-d device_path]	[slot-id]
set host	[-dup] [-task t]	node
show		{ <i>envar</i> , config, device, error, fru, hwrpb, memory}
start		address
<b>U*X-like Console Commands</b>		
cat		file...
check	[-{f   r   w}]	inode
chmod	[-   +   =]{r   w   x   b   z}	file...
clear		<i>envar</i>
cmp	[-s val] [-e val] [-l val] [-b val]	file1 file2
crc	[-s val] [-e val] [-l val]	file
dynamic	[-h] [-v] [-c] [-z ha]	
echo	[-n]	args...
eval		postfix_expression
exit		exit_value
grep	[-{v   c   n   y   x}] [-f filename]	expression [file...]
hd	[-s val] [-e val] [-l val]	file...
kill		pid...
ls	[-l]	[file...]
more	[-n pagesize]	file...
ps		
rm		file...
set		<i>envar</i> value

In this summary, the following conventions are used:

[*item*] - indicates the *item* is optional.

{*a,b,c*} - indicates any one of *a*, *b*, or *c*.

{*a / b / c*} - indicates any combination of *a*, *b*, or *c*.

*device*: specifies the name of the driver for a "device" address space and is one of:

**pmem:**, **vmem:**, **gpr:**, **fpr:**, **ipr:**, **pio:**

**eerom:**, **enet:**, **ferom:**, **iic:**, **ncr0(1,2,3,4):**, **scram:**, **tgec0(1):** **toy:**, **uart:**

*envar*: specifies the name of an "environment variable", for example, **boot\_device** .

(continued on next page)

Table 25–4 (Cont.) DEC 4000 Console Command Summary

Command	Options	Parameters
<b>U*X-like Console Commands</b>		
sleep		time
sort		file...
tr	[-c   d   s]	string1 [string2]
uniq		file...
wc	[-l   w   c]	file...
<b>Unique Console Commands</b>		
alloc	[-z heap_address]	size [modulus] [remainder]
cdp	[-{i,n,a,u,o}] [-sn] [-sa val] [-su val]	[dssi_device]
exer	[-sb startblock] [-eb endblock] [-p passcount] [-l blocks] [-bs blocksize] [-bc block_per_io] [-d1 buf1_string] [-d2 buf2_string] [-a action_string] [-sec seconds] [-m] [-v]	[device]...
find_field		field_no
free		address...
memtest	[-sa address] [-ea address] [-l length] [-bs block size] [-i inc] [-p n] [-f] [-m] [-z] [-h] [-rs n] [-rb] [-mb]	
net	[-sa] [-s] [-i] [-ri] [-ic] [-se] [-re] [-rc] [-l1] [-l2] [-els] [-kls] [-l file_name] [-id node_address] [-lc number] [-l0 node_address] [-bd burst_interval] [-cm mode_string] [-sv mop_version]	[port]
nettest	[-f filename] [-mode string] [-p n] [-sv mop_version] [-to loop_time] [-w number]	[port]
sa		process_id affinity_ mask
semaphore		
show_status		
sp		process_id new_ priority
stop		device_path

## 26.1 Diagnostic Overview

The strategy used for the DEC 4000 system diagnostics is to detect and isolate failures contained within the CPU for the engineering debug, design verification test (DVT), manufacturing, and customer services environments. The specific nature of these environments and the diagnostics supplied for each is different. This document identifies the diagnostics for the DEC 4000 system and notes the environment each test is targeted toward.

### 26.1.1 Diagnostic Classes

The diagnostic support for the DEC 4000 system is provided by two classes of diagnostic: tests and exercisers.

**Diagnostic Tests** are modules that determine the functionality of the system. The tests perform by generating a stimulus and comparing the measured response to an expected value. Any deviation from expected functional behavior is reported as an error. Each DEC 4000 test module is targeted at a single functional or physical component of the system. These modules are not intended for concurrent execution. The diagnostic tests for the DEC 4000 system consist of two groups of tests as follows:

- The first group is the initial power-up tests which verify that sufficient functionality exists within the system to load and execute the FEPR0M-based console program kernel.
- The second group runs under the FEPR0M-based console program kernel and tests all of the Kernel system including all boot path devices. These tests are intended to detect and isolate structural faults within the system and do not provide system or device exerciser functions.

**Diagnostic exercisers** are modules that are intended to run concurrently to force the manifestation of nonlogical defects within the system. Each module is designed to provide a controllable degree of physical activity within the target subsystem. These modules are supplied to address the following failure model which is difficult to support with diagnostic tests.

- An interconnect that fails intermittently, which can be aggravated by mechanical or thermal means
- Design bugs whereby devices can not obtain enough bandwidth or whose latency requirements are not met
- Hard failures that require too much context for a test directed approach

## 26.1 Diagnostic Overview

The design of the exerciser modules allows simultaneous execution of independent modules targeted at different devices within the system. These modules use the existing console device drivers to support access to the target devices. Testing is focused on providing the maximum bus interaction possible.

### 26.1.2 Diagnostic User Interface

#### 26.1.2.1 Starting Diagnostic Tests and Exercisers

Diagnostic modules may be run as either foreground or background processes. To invoke a diagnostic test module, the user may do one of two things:

1. Enter the TEST command with or without a test target argument.

Entering the TEST command without a test target argument concurrently tests (exercises) most devices in the system.

2. Enter the actual name of the desired diagnostic module.

The diagnostic modules that may be executed are enumerated below. Some of these modules run a single test on one device and other modules are exercisers which run concurrent tests on multiple devices. Some of these modules are scripts which execute other diagnostic modules.

- memtest
- memexer
- memexer\_mp
- nettest
- netexer
- ntlpex
- ntlpex1
- ntlpex2
- fbus\_diag
- exer\_read
- exer\_write
- exer

Refer to Appendix D for specific information about the syntax of these commands.

#### 26.1.2.2 Diagnostic Control Flags

The DEC 4000 console uses environment variables and command qualifiers to control the execution of individual diagnostic tests and exercisers.

Section 26.1.2.2.1 describes the global environment variables and command qualifiers that are used to control all diagnostic test and exerciser modules.

Each individual diagnostic module description describes any module specific environment variables or command qualifiers used to control that module. Refer to the SET and SHOW commands in Appendix D for information on setting or displaying the current state of an environment variable.

**26.1.2.2.1 Global Diagnostic Environment Variables** The following global environment variables are available to the user for control of diagnostics. These environment variables are global to all diagnostic programs that are executing, and thus affect all diagnostic programs.

Refer to Table E-1 for a complete description of these and all other environment variables.

```
>>>show d_*
d_bell                off
d_cleanup             on
d_complete           off
d_eop                off
d_group              field
d_harderr            halt
d_oper               off
d_passes             1
d_report             summary
d_softerr            continue
d_startup            off
d_trace              off
>>>
```

**26.1.2.2.2 Local Diagnostic Command Qualifiers for most diagnostics** The following command qualifiers affect the operation of an invocation a diagnostic test module by overriding the corresponding global diagnostic environment variable for the duration of that test. These command qualifiers may be used with most diagnostics.

Refer to Appendix D for a description of the diagnostic commands and their respective command qualifiers.

Qualifier	Definition
-p n	Value of n overrides "diag_passes" as the number of passcounts to run
-g "group_name"	The specified group name overrides d_group.

**26.1.2.2.3 Local Diagnostic Command Qualifiers for specific diagnostics**

In addition to the global command qualifiers that control all diagnostics each diagnostic module has its own set of command qualifiers to control the execution of that diagnostic module.

Refer to Appendix D for a description of the diagnostic commands and their respective command qualifiers.

### 26.1.2.3 Monitoring the Exerciser Status

Status monitoring of one or more diagnostic tests is provided by a separate process called show\_status. This process uses the status block within the PCB to display the current runtime status for each diagnostic process currently executing within the console. show\_status must be invoked as a shell command so it can only be executed when the diagnostic modules are running as background processes. The format of the status display is shown below:

## 26.1 Diagnostic Overview

**Figure 26–1 Diagnostic Status Display**

```
>>>show_status
-----
      ID      Program      Device      Pass Hard/Soft Bytes Written      Bytes Read
-----
00000001      idle ❶ system          0   0   0              0              0
00000056      exer_kid dub2.2.0.1.0      0   0   0              0      3807232
00000059      memtest      mem 47451   0   0      388710400      388710400
0000005c ❷ memtest ❸      mem ❹ 46370 ❺ 0 ❻ 0 ❼ 379854848 ❽ 379854848
```

In Figure 26–1, the fields are defined as follows.

- ❶ idle process - Errors not attributable to a specific device or process are listed on this line of the display so this line is always displayed.
- ❷ ID - Unique process ID within the life of the system for each process found
- ❸ Program - Program name for this diagnostic process, normally the module name
- ❹ Device - Device being tested by this process
- ❺ Pass - Number of passes completed by this process
- ❻ Hard Errors - Number of hard errors detected by this process
- ❼ Soft Errors - Number of soft errors detected by this process
- ❽ Bytes Written - Number of bytes written by this process
- ❾ Bytes Read - Number of bytes read by this process

### 26.1.2.4 Terminating Diagnostic Tests and Exercisers

Diagnostic modules normally terminate after completing all device tests. If it is desirable to terminate a diagnostic before it has completed it is often possible to do so. If the diagnostic is running as a foreground process then a control-C, ^C, will usually terminate the diagnostic module. If one or more diagnostics are running as background processes then the kill\_diags command will terminate all diagnostics.

## 26.1.3 Diagnostic Output Displays

### 26.1.3.1 Operator Control Panel LED Displays

The DEC 4000 system operator control panel (OCP) contains eight yellow fault LED's to indicate that a system component is being tested or has failed. The eight LEDs are shown in Figure 1–4 in <REFERENCE>(OCP\_SECTION) and listed below.

- FBUS - One of the Futurebus+ modules in the system
- MEM3 - Memory Module 3
- MEM2 - Memory Module 2
- MEM1 - Memory Module 1
- MEM0 - Memory Module 0
- CPU0 - CPU Module 0
- CPU1 - CPU Module 1
- I/O - I/O Module

On system power-up, the two CPU fault/testing LEDs illuminate and the remaining memory, I/O, and Futurebus+ LEDs stay extinguished. Successful completion of the power-up testing extinguishes all of the LEDs.

### 26.1.3.2 Console Terminal Error Messages

Error messages are broken into 3 levels. These levels are referred to as the device driver level, the SUMMARY level and the FULL level. The device driver level is quite often absent. It consists of one or more lines of error messages describing the reason for a low level device driver operation failure. When a device driver operation succeeds but the diagnostic code detects a failure symptom then the device driver doesn't issue any error messages. When errors are issued by a device driver during a diagnostic test failure the device driver message(s), if any, will appear in the following example at ❶ .

The SUMMARY level consists of information at the top header portion of the screen ( fields ❷ through ❹ in the following example, and the FULL level consists of information in field in the bottom half of the message (❺. Selection of the SUMMARY and FULL levels displayed is controlled with the D\_REPORT environment variable.

- If D\_REPORT is set to SUMMARY, then the FULL level information is not displayed.
- If D\_REPORT is set to FULL, then the FULL level information is displayed as well as the SUMMARY level information.
- D\_REPORT can also be set to OFF to disable display of both SUMMARY and FULL error information. By default, D\_REPORT is set to SUMMARY.

Figure 26–2 gives an example of an error message.

**Figure 26–2 Example Error Message**

```

❶
*** Hard ❷ Error - Error #15 ❸ - External loopback error, no packet received ❹ ***
Diagnostic Name      ID          Device Pass Test Hard/Soft 01-JAN-1990
nettest ❺          23 ❻          eza0.0.0.6.0 ❼ 5 ❽ 2 ❾ 1 ❿ 9 ⓫ 12:00:01 ⓬
Buffer contents are as follows: ⓭
      Expected      Received
      -----      -
20150000: AAAAAAAA      00000000
20150004: 55555555      00000000
20150008: CCCCCCCC      00000000
2015000C: 33333333      00000000
20150010: 88888888      00000000
20150014: 77777777      00000000
20150018: FFFFFFFF      00000000
*** End of Error *** ⓬

```

In Figure 26–2, the fields are explained as follows:

- ❶ Zero or more lines of device driver error messages may appear here
- ❷ Error type. Possible values are:
  - HARD - Hard error

## 26.1 Diagnostic Overview

- SOFT - Soft error
  - FATAL - Fatal error
  - NONE - No error type will be displayed.
- ③ Error number
  - ④ Summary Level Error message
  - ⑤ Diagnostic/module name
  - ⑥ ID—Unique process ID of the diagnostic test program
  - ⑦ Target device
  - ⑧ Current pass count
  - ⑨ Current test number
  - ⑩ Current hard error count
  - ⑪ Current soft error count
  - ⑫ Time stamp
  - ⑬ Full-level error message information may appear here
  - ⑭ Error message delimiter

### 26.1.4 Startup Message

The display of the diagnostic startup message is controlled by the D\_STARTUP environment variable. If D\_STARTUP is set to ON, the startup message is displayed. By default, D\_STARTUP is set to OFF.

A startup message consists of two lines of information used to indicate that the diagnostic has started execution. An example is as follows.

**Figure 26–3 Startup Message**

```
Diagnostic Name      ID           Device Pass Test Hard/Soft 01-JAN-1990 ①
nettest ②          23 ③       eza0.0.0.6.0 ④ 0 ⑤      0 ⑥ 0 ⑦ 12:00:01 ⑧
```

In Figure 26–3, the fields are explained as follows.

- ① Startup message header
- ② Diagnostic name
- ③ Target device
- ④ Pass count of 0
- ⑤ Hard error count of 0
- ⑥ Soft error count of 0
- ⑦ Time stamp



### 26.1.5 Completion Message

The display of the diagnostic completion message is controlled by the D\_COMPLETE environment variable. If D\_COMPLETE is set to ON, then the completion message is displayed when the diagnostic process is terminated. By default, D\_COMPLETE is set to OFF.

A completion message consists of summary information on the run of the diagnostic. An example follows.

**Figure 26–4 Completion Message**

```
Testing Complete ①
Diagnostic Name   ID           Device Pass Test Hard/Soft 01-JAN-1990
nettest ②       23 ③       eza0.0.0.6.0 ④ 1 ⑤       1 ⑥ 9 ⑦ 12:00:01 ⑧
* End of Run ⑨ - Failed ⑩ *
```

In Figure 26–4, the fields are explained as follows.

- ① Completion message indicator
- ② Diagnostic name
- ③ ID—Unique process ID of the diagnostic test program
- ④ Target device
- ⑤ Final pass count
- ⑥ Final hard error count
- ⑦ Final soft error count
- ⑧ Time stamp
- ⑨ Completion message delimiter
- ⑩ Pass/fail indicator. One of:
  - Passed - No hard/fatal errors were reported (if a soft error occurs the diagnostic will still 'pass')
  - Failed - One or more hard/fatal errors were reported

### 26.1.6 End-of-pass Message

The display of the diagnostic end-of-pass message is controlled by the D\_EOP environment variable. If D\_EOP is set to ON, the end-of-pass message is displayed. By default, D\_EOP is set to OFF.

An end-of-pass message consists of one line of information used to indicate that the diagnostic has completed one pass. Current error totals are also given. An example of the standard end-of-pass message follows.

**Figure 26–5 End of Pass Message**

```
Diagnostic Name   ID           Device Pass Test Hard/Soft 01-JAN-1990 ①
nettest ②       88 ③       eza0.0.0.6.0 ④ 1 ⑤       1 ⑥ 9 ⑦ 12:30:10 ⑧
```

## 26.1 Diagnostic Overview

In Figure 26–5, the fields are defined as follows.

- ❶ Header will exist from previous error callout (or startup message if no error)
- ❷ Diagnostic name
- ❸ ID—Unique process ID of the diagnostic test program
- ❹ Target device
- ❺ Pass count
- ❻ Hard error count
- ❼ Soft error count
- ❽ Time stamp

### 26.1.7 Test Trace Message

The display of the diagnostic test trace message is controlled by the D\_TRACE environment variable. If D\_TRACE is set to ON, then the test trace message is displayed. By default, D\_TRACE is set to OFF.

A test trace message consists of one line of information used to indicate that the diagnostic is starting a test. Some diagnostic modules have multiple tests, each with a different test number. An example of the test trace message follows.

**Figure 26–6 Test Trace Message**

```
Diagnostic Name      ID           Device  Pass  Test  Hard/Soft  01-JAN-1990 ❶  
nettest ❷          88 ❸      eza0.0.0.6.0 ❹ 1 ❺ 3 ❻ 1 ❼ 9 ❽ 12:00:01 ❾
```

In Figure 26–6, the fields are defined as follows.

- ❶ Header exists from previous error callout (or startup message if no error)
- ❷ Diagnostic name
- ❸ ID—Unique process ID of the diagnostic test program
- ❹ Target device
- ❺ Current pass count
- ❻ Current test number
- ❼ Current hard error count
- ❽ Current soft error count
- ❾ Time stamp

### 26.1.8 Diagnostic Power-up Flow

When the DEC 4000 system is powered-up, a series of diagnostic tests are executed to ensure the proper operation of the system. These tests consist of initial power-up tests and FEPR0M-based tests. In addition, the power-up sequence runs exercisers on all configured system devices.

## 26.2 CPU/Cache Subsystem

The diagnostics in this section test all of the major functional blocks found on the CPU module. These include the backup cache RAMs, the system bus gate arrays and the paths from these two areas to the 21064 (EV4).

### 26.2.1 Diagnostic Strategy

This section of tests verifies several aspects of the CPU module. Several pieces of logic must be working properly before the contents of the FEPROMs (located on the I/O module) can be copied into the backup cache RAMs. The 21064 microprocessor must be able to communicate with the backup cache and the system bus gate arrays. The functionality of the backup cache must also be verified. The diagnostics that verify the hardware, prior to the loading of the FEPROMs, are located in the SROMs (labeled below as 'serial-ROM group'). More extensive testing of the CPU subsystem is performed by diagnostics stored in the FEPROMs (labeled below as 'FEPROM group'). These tests are executed after code stored in the the FEPROMs is successfully loaded into the backup cache RAMs.

### 26.2.2 Power-up Strategy

On assertion of C RESERT\_L, the first "I-cache size" worth of code will be loaded from the SROM into the I-cache. Once the D-cache is disabled, SROM tests insure the basic functionality of the CPU subsystem. The address and data lines from the 21064 microprocessor to the backup cache are tested first, followed by the data RAMs, the TAG RAMs and the ECC logic. Some reads of CSRs are then performed in the ASICs to ensure access to the system bus.

At this point CPU<sub>1</sub> determines if CPU<sub>0</sub> failed. If CPU<sub>0</sub> does not fail, CPU<sub>1</sub> will stall so that only one CPU attempts to configure memory. Next, the address and data lines on the system bus are tested, using the I/O module. The first memory module is found and its configuration data is extracted from the serial control bus. The memory module is then configured and its first 8 MB is tested.

The backup cache is disabled and the code from the FEPROMs is loaded into main memory. The backup cache testing continues with tag, tag-parity and control flag tests. Finally the FEPROM code is moved from the ROMs to memory and control is passed to location 8000<sub>16</sub>.

### 26.2.3 Fault Architecture

The current fault strategy is to place as much error information as possible into the EEPROMs on the CPU board a failure occurs during the power-up sequence. This is accomplished with the 87C652 microprocessor. The micro is programmed in such a way that it allows communication between the 21064 microprocessor to the serial control bus (and thus the EEPROMs).

### 26.2.4 Interpreting Error Printouts

This section describes how to interpret the serial ROM diagnostic printouts.

### 26.2.5 Error Message Display

The following is an example of an error report. Any errors that are detected by the serial ROM diagnostics are visible only when the SHOW ERROR command is used from the firmware console.

## 26.2 CPU/Cache Subsystem

### Examples

```
Error Number: 01 Subtest Number: 01
Address: 12345678
Expected Data: 12345678,12345678
Received Data: 12345678,12345678
```

In the following paragraphs, each section that describes a serial ROM diagnostic that detects errors, is followed by a sample error printout.

### 26.2.6 Test name: B-Cache init

#### Test number: 01

**Overview:** This is not a test. Running test 1 reinitializes the backup cache

#### Description:

1. Initialize every block in the cache with a tag of 0 and Valid, Shared and Dirty values of 0-0-0.
2. Clears all error registers in the CPU ASICs.
3. Enables the B-Cache for normal operation, that is, B-Cache probing is enabled from the processor side and allocation and all checking is turned on from the gate array side.

#### Errors:

The backup cache initialization routine does not detect errors.

#### Operator Control Panel LEDs:

does not affect LEDs

### 26.2.7 Test name: SROM Address Test

#### Test number: 02

**Overview:** This test verifies the B-cache data RAMs on the CPU module.

#### Description:

1. Every block of memory in the area to be tested is written with a quadword (QW) of a graycode pattern and a QW of an inverse graycode pattern.
2. Every quadword is verified and if correct written back with its inverse.
3. The entire test area is flooded with zeros.

---

#### Note

---

The first memory module tested is the largest module with the lowest slot number. If the module fails the next ranking module is tested. This is done until the test finds 8 Meg of good memory.

---

**Errors:** This test uses the subtest field in the error printout. It indicates the number of the failing quadword within the cache block under test.

- 00 Failure on verifying data from QW0
- 01 Failure on verifying data from QW1
- 02 Failure on verifying data from QW2

03 Failure on verifying data from QW3

**Figure 26–7 Test 02 sample printout**

```
Test Number: 02 Subtest Number: 02
Address: 00000000,00000010
Expected: ffffffff,fffffff
Received: fffffeff,fffffff
```

**Operator Control Panel LEDs:**

```
0 0001 00 0 for memory module #0
0 0010 00 0 for memory module #1
0 0100 00 0 for memory module #2
0 1000 00 0 for memory module #3
```

### 26.2.8 Test name: SROM Tag Store Test

**Test number: 03**

**Overview:** This test verifies the tag store and tag control RAMs on the CPU module.

**Description:**

1. The gate arrays on the CPU module are put into B-cache init mode allowing each B-cache block to be written with various tag and tag control bit patterns. Each cache block is written with an incrementing pattern in its tag field.
2. The 21064 microprocessor (Ev) is caused to probe each block and latch the corresponding tag/tag control values in one of its internal registers. The values are then compared to the initialization values for that particular cache block.
3. The cache is then re-written with a decrementing pattern in the tags and a new pattern in the tag control field.
4. Again, the 21064 probes the entire cache and verifies the tag and tag control fields of every block in the cache.

**Errors:**

For this test the following subtests are possible.

- 01 Failure while verifying the tag\_ADDR bits during the routine which increments the tags from 0 (first pass)
- 02 Failure while verifying the tag\_ADDR bits during the routine which decrements the tags to 0 (second pass)
- 03 Failure while verifying the tag\_CTRL bits during the routine which increments the tags from 0
- 04 Failure while verifying the tag\_CTRL bits during the routine which decrements the tags to 0 (second pass)

The failing address which is printed out is the address that was used to trigger the last B-cache probe (the one that had an incorrect tag or control field bit(s)).

The expected data reflects only the control bits or tag bits that were being checked for.

The received data has the entire contents of 21064's BC\_TAG IPR.

## 26.2 CPU/Cache Subsystem

When comparing expected to received, you should look only at the field in the received quadword that corresponds to the particular subtest.

### Sample printouts:

1.

```
Test Number: 03 Subtest Number: 03
Address: 00000000,00000020
Expected: 00000000,00000004
Received: 00000000,00000100
```

Subtest 03 means that the tag control bits were being verified when the failure occurred. This means that only bits <2:4> of the received field need to be looked at (disregard the tag field). In this example the dirty bit was expected to be set but was not.

2.

```
Test Number: 03 Subtest Number: 01
Address: 00000000,00000020
Expected: 00000000,00000200
Received: 00000000,00000304
```

Subtest 01 means that the tag field was being verified when the failure occurred. This means that only bits <5:18> of the received field need to be looked at (disregard the control bit field). In this example only one tag bit was expected to be set.

3.

```
Test Number: 03 Subtest Number: 03 or 04
Address: 00000000,00000000
Expected: 00000000,00000003
Received: 00000000,00000004
```

In the case of subtest 03 or 04, there was a control bit error. The expected data ignores the hit bit in the `bc_tag` IPR so the value that should have been in `bc_tag` is actually the expected data, shifted left once. In this case, a 3 means that it was expected that both the parity bit and the dirty bit be set. Because the 4 is the actual value returned from `bc_tag`, it indicates that only the dirty bit was set. Therefore the first bit that should be suspected is the parity bit.

### Operator Control Panel LEDs:

```
0 0000 10 0 for CPU0
0 0000 01 0 for CPU1
```

### 26.2.9 Test name: SRAM B-Cache Data Line Test

**Test number: 04**

**Overview:** This test verifies the data bus between 21064 and the B-cache.

#### Description:

1. Set a bit in a cache block.
2. Verify that only that bit is a one.
3. Shift to the next bit and verify again.

This is done for all 128 bits on the 21064 data bus.

### Errors:

This test does not use the subtest field in the error printout (it will always be 00). The point at which the test failed will be obvious just by looking at the expected data pattern.

### Sample printout:

```
Test Number: 04 Subtest Number: 00
Address: 00000000,00000000
Expected: 00010000,00000000
Received: 00000000,00000000
```

### Operator Control Panel LEDs:

```
0 0000 10 0 for CPU0
0 0000 01 0 for CPU1
```

### 26.2.10 Test name: SROM ECC test

#### Test number: 05

**Overview:** This test verifies ECC generation by the 21064 (Ev).

#### Description:

1. The entire cache is written with 10010029. This pattern's ECC is 2A (there are only 7 check bits, not 8).
2. Every longword is read out of the cache and BIU\_STAT is checked for an ECC error.
3. Now each longword is written with 1000012d, which has an ECC of 55.
4. Every longword is read out of the cache and BIU\_STAT is checked for an ECC error.

### Errors:

The Subtest number is not used (always 00). If the expected pattern is 10010029, the check bits were expected to be all A's. If the expected data is 1000012d, the check bits were expected to be all 5's. The received data may match the expected data because an ECC error may not necessarily be associated with a data error. Also, the ECC bits generated by 21064 cannot be read directly, so they must be checked using an analyzer.

### Sample printout:

```
Test Number: 05 Subtest Number: 00
Address: 00000000,00010080
Expected: 00000000,1001002d
Received: 00000000,1001002d
```

### Operator Control Panel LEDs:

```
0 0000 10 0 for CPU0
0 0000 01 0 for CPU1
```

## 26.2 CPU/Cache Subsystem

### 26.2.11 Test name: CPU Spin on Processor Mailbox

**Test number: 06**

**Overview:**

This is not a test. Running test 6 causes a CPU to read its PMBX in a loop. This test is useful only in an SMP scenario.

**Description:**

1. The CPU from which the test command is issued begins reading its processor mailbox in anticipation of an address. It reads its PMBX approximately once every millisecond.
2. The firmware console can then be loaded from the other CPU. At some point the firmware console writes an address to the secondary CPU's mailbox. The CPU that is spinning on its mailbox reads that address and jumps to it.

**Errors:**

The CPU spin on mailbox is used only for multiprocessor initialization. It does not detect any errors.

**Operator Control Panel LEDs:**

does not affect LEDs

### 26.2.12 Test name: Cbus (System Bus) test

**Test number: 07**

**Overview:** This test does a quick verification of the system bus.

**Description:**

1. A 1 and a 0 are floated across the processor mailbox and four different I/O module CSRs. The following registers are tested by CPU0/1:

CPU0	CPU1
DLCA0	DLCB0
DLCA1	DLCB1
DLCA2	DLCB2
DLCA3	DLCB3

**Errors:**

An error is detected if there is a stuck bit on the system bus. This includes detecting bad backplane connections. The subtest field is not used in this test. The address field will identify the failing CSR.

**Sample printout:**

```
Test Number: 07 Subtest Number: 00
Address: 00000002,00000140
Expected: 00000000,80000000
Received: 00000000,00000000
```

**Operator Control Panel LEDs:**

0 0000 00 1 for I/O module



**26.2.13 Test name: Disable failing CPU****Test number: 08****Overview:**

This is not a test. Running test 8 causes the other CPU to be disabled. This test is used on power-on to disable a CPU that has failed its power up.

**Description:**

1. First, the other CPU's arbitration mask is turned off. This prohibits that CPU from accessing the system bus.
2. That CPU's lock register (CSR13) is then cleared.
3. Finally, the CPU to be turned off has its BCC written with x000.1000. This disables parity checking in the duplicate tag, disables ECC checking and correction in the backup cache, forces the tag control bits to valid = 0, and disables backup cache allocation. This is so any system bus references that coincidentally hit in this "disabled" CPU will update the backup cache to invalidate the block so that it will not hit again.

**Errors:**

This test is used only in multiprocessor initialization. It does not detect any errors.

**Operator Control Panel LEDs:**

does not affect LEDs

**26.2.14 Test name: Memory test****Test number: 99**

**Overview:** This test finds and tests 2 MB of good memory.

**Description:**

1. The configuration information from the 652 is parsed to figure out which is the largest memory module in the system.
2. All present modules are written with the correct size bits but only the largest module is enabled. This module is also written with a refresh value of ^xE7 and its refresh is enabled.
3. EDC reporting is then disabled and the graycode test (test 2) is called to test the first 8 Meg on this module. Because the state of the EDC bits is undefined at power-up, EDC reporting is disabled to eliminate seeing any errors. These errors would cause machine checks which cannot be handled while executing SROM code.
4. If the test passes, the EDC RAMs on the memory module being tested are put into swap mode (they are used for data instead of EDC) and the graycode test is called a second time. Swap mode causes the EDC RAMs to be used as data RAMs, corresponding to data bits <0:11>.

**Errors:**

The definitions of the subtest numbers are the same as those for test number 2, with two exceptions.

- If the memory test fails during the first pass, the test number will be 99.

## 26.2 CPU/Cache Subsystem

- If the memory test fails during the second pass (while testing the EDC RAMs), the test number will be 98.

### Sample printout:

```
Test Number: 99  Subtest Number: 00
Address: 00000000,00000040
Expected: 00000000,00000003
Received: 00000000,80000003
```

In this case data bit 30 seems to be stuck high. If the Cbus test (test 7) has been run successfully the problem can be either in the cache or the memory module. If tests 2 and 4 also pass, then the memory should be looked at.

```
Test Number: 98  Subtest Number: 00
Address: 00000000,00000040
Expected: 00000000,00000003
Received: 00000000,80000003
```

In this example, the same error occurred but during the second pass of the memory test, while testing the EDC RAMs. A good way to find this problem is to put the EDC RAMs in and out of swap mode and try to examine/deposit some locations manually.

### Operator Control Panel LEDs:

```
0 0001 00 0 for memory module #0
0 0010 00 0 for memory module #1
0 0100 00 0 for memory module #2
0 1000 00 0 for memory module #3
```

### 26.2.15 Test name: Unload

#### Test number: 00

This is the command that unloads the firmware console from the I/O FEPRoMs.

#### Description:

Although this command does not have a test number, it is mentioned here because it can print out several different errors during its execution. The basic algorithm for unloading the flash ROMs is the following.

1. Reset the L-bus on the I/O module.
2. Write the data structure base address to the L-bus Mailbox Pointer Register (LMBPR).
3. Wait for the I/O module to respond with the data from the FEPRoMs then write it to memory.
4. Once the entire contents of the FEPRoMs have been written to memory re-read all of the data from memory and calculate the checksum.
5. Compare this to the correct checksum stored in the FEPRoMs and if correct pass control to the Palcode.
6. Verify that all four FEPRoMs have the same value in the firmware console revision field.
7. Verify that the FEPRoMs are in the correct locations.

#### Errors:

There are four ways in which the unload may fail:

- If the 21064 CPU (Ev) times out waiting for the I/O module to set the done bit in the data structure

## 26.2 CPU/Cache Subsystem

- If the checksum calculation returns the incorrect result
- If the FEPROM revision fields are inconsistent
- If one or more of the FEPROMS are in the wrong location on the I/O module.

### **Sample printouts;**

Several different subtests are possible.

## 26.2 CPU/Cache Subsystem

Subtest Number	Meaning	Possible cause
02	Time out waiting for done bit from I/O module	The I/O module may be unplugged or poorly seated. There may be a problem accessing the L-bus. Try reading the FEPRoms manually. See the instructions in the I/O spec.
03	Bad Checksum in byte 0	For subtests 03 through 06, starting with the most likely cause:
04	Bad Checksum in byte 1	The memory module may be missing.
05	Bad Checksum in byte 2	The memory module may have an incorrectly programmed EEROM.
06	Bad Checksum in byte 3	Was test 99 run first ? It has to be. There is a problem with one or more of the FEROMs. There may be a problem with the memory module.
07	Revision Error	One or more of the FEPRoms is of the wrong revision.
08	Lane Error	One or more the FEPRoms is in the wrong location.

Note: Only the first FEPRom that is bad will be flagged.

```
Test Number: 00 Subtest Number: 02
Address: 00000000,00001000
Expected: 00038080,00038080
Received: 00000a00,00000a00
```

Both the expected and received values in this case are meaningless. The address corresponds to the base address of the data structure in memory (always 1000).

```
Test Number: 00 Subtest Number: 03/04/05/06
Address: 00000000,00000000
Expected: 00000000,fa34bc16
Received: 00000000,fcfcfcfc
```

A failure occurred while verifying the checksum for byte lane 0, but it appears as though all of the bytes came up in error. The first component that can be suspected is memory.

```
Test Number: 00 Subtest Number: 03
Address: 00000000,00000000
Expected: 00000000,fa34bc16
Received: 00000000,fa34bc00
```

In this example, the combination of the subtest number and the received data tell us to look at the FEPRom for byte 0.

```
Test Number: 00 Subtest Number: 07
Address: 00000000,00087fec
Expected: 00000000,43434343
Received: 00000000,43432943
```

In this example, the FEPROM 1 is of the wrong revision.

```
Test Number: 00 Subtest Number: 03
Address: 00000000,00087fe8
Expected: 00000000,03020100
Received: 00000000,03000102
```

In this example, FEPROMS 0 and 2 are in the wrong location.

### Operator Control Panel LEDs:

```
0 0000 00 1 for I/O module
```

## 26.3 Memory Subsystem

The DEC 4000 memory module provides between 16 MB and 128 MB of high-performance storage, depending on the technology of the DRAMs used and the number of banks installed, with error detection and correction capabilities. A pair of application-specific integrated circuits (CMOS gate arrays) provides the DEC 4000 system interconnect data interface and transaction control, DRAM interface and control, EDC generate, error detection/correction and error syndrome generation, and diagnostic/self test functions.

### 26.3.1 Initialization

#### 26.3.1.1 Initialization Tests

This test sizes the memory subsystem both for presence of modules and board size. The DEC 4000 memory module configuration CSR is setup. For each memory module, one at a time, basic data, address and bank uniqueness is tested. Major functionality of the CMOS gate arrays is also verified, including Error Detection and Correction(EDC) logic and stream buffer storage. The DRAMs are verified in parallel utilizing diagnostic mode. The modules are configured using interleave functionality by default or in a user specified configuration. Failing modules are not configured into the system. All errors are reported and logged into the modules EEROM.

#### Description

Verify the Memory integrated circuit (CMIC):

CMIC register storage - Float a one and a zero across all bits in one CMIC register.

CMIC stream buffer - Write/read/verify all stream buffer storage cells with a one and zero.

Bank uniqueness (across banks) - Write/read/verify locations on all banks using a graycode/inverse graycode algorithm.

Chip/bank address uniqueness(within bank) - Float a one and a zero across all address lines for each bank.

## 26.3 Memory Subsystem

CMIC EDC logic - Force correctable and uncorrectable errors and verify correct reorting and CSR logging.

Verify all DRAMS on available Modules:

Write every cell of the DRAMs with a 1 and a 0 using a graycode/ inverse graycode algorithm.

All EDC bits are tested because a complement of a data pattern forces the complement of the EDC bits.

Use of dignostic mode function during testing allows access to multiple memory modules in the same address range.

Use of EDC logic to detect and report all errors decreases test time by eliminating explicit verification of data.

### 26.3.1.2 Memory Configuration

#### Description

Memory is configured, by default, using full interleave capability. All non-failing modules are configured largest to smallest. Two like modules are configured using 2-way interleaving and four like modules are configured using 4-way interleave. Multiple smaller modules may also be logically combined to be interleaved with a larger module of like size. For example, two 64MB modules can be combined to form a logical 128MB module and be interleaved with a 128MB module.

The interleave environment variable may be used to control the memory configuration and force a custom interleave. The interleave environment variable may be set to "none" which will configure the memory largest to smallest in a non-interleaved setup.

Special interleave syntax can be used to define specific configurations.

#### Note

The interleave environment variable is nonvolatile, it will be preserved across a power cycle. The new, specified, memory configuration will not take affect until the "init" command is typed or the system is reset.

**Table 26–1 Interleave environment variable syntax**

Syntax	Function	Description
0,1,2,3	Board number	used to identify memory board
,	Interleave set	used to separate groups of boards that will be interleaved
:	Combine board	used to designate multiple memory modules to appear as one logical interleave unit
+	Interleave Unit	used to identify modules that are to be interleaved

For example, in a system with memory modules of 128MB, 32MB, 32MB, 64MB in slots 0 through 3. Interleave could be set to:

Table 26–2 Interleave environment variable examples

Command Syntax	Memory Configuration
>>>set interleave 0+3:1:2	Interleave the 128MB with a logical 128MB made up of the 64MB, and the two 32MB modules. Same as "default" interleave.
>>>set interleave 0,3+1:2	Configure the 128MB alone, then interleave the 64MB with a logical 64MB made up of two 32MB modules.
>>>set interleave 0,3,1+2	Configure the 128MB and 64MB alone, then interleave the two 32MB modules.
>>>set interleave 0,3,1,2	Configure all modules with no interleaving. Same as "none" interleave.
>>>set interleave 0,3,1	Configure the 128MB, 64MB and one 32MB alone. Do not configure the 32MB module in slot 2.

### 26.3.1.3 Failure Reporting and Logging

#### Description

Failure information is reported and logged after the initialization tests are completed. All errors are logged into the IIC EEROM error structure. Refer to chapter on IIC EEPROM error structure for complete details. All new, unique errors are also reported to the console terminal and the event logger(el) during powerup.

### 26.3.2 Exerciser Tests

#### Description

The exerciser uses a system wide memory exerciser service available from the shell. This allows the memory subsystem to be run in a resource competing environment. Multiple exercisers may be started to test different sections of memory.

Other exercisers may also be started to produce more activity. A system-wide exerciser status reporting program is used to report the state of all exercisers currently executing

For a complete description of the memory exercisers refer to the command Appendix sections for memtest, memexer, and test.

## 26.4 SCSI/DSSI Subsystem

The SCSI/DSSI (mass storage) subsystem is part of the I/O module in a DEC 4000 system and consists of the following:

- Five NCR 53C710 controller chips
- 128 Kbytes of buffer memory shared by the NCR chips (Script RAM)
- Associated logic to communicate with four SCSI/DSSI and one SCSI-only port on the I/O module backplane
- The disk and tape units
- The CPU and host memory

## 26.4 SCSI/DSSI Subsystem

The four NCR chips can serve as either DSSI or SCSI adapters. Each has a PAL and a transceiver chip on the SCSI side of the NCR chip. The fifth NCR chip has its SCSI side routed directly to the SCSI bus connector.

### 26.4.1 Expected Failures

The four major areas in the mass storage subsystem for failure are:

- Mechanical and electrical connections
- Chip failures (both NCR and associated logic)
- Disk controllers or tape units
- System interaction including latency and bandwidth issues

### 26.4.2 Diagnostic Strategy

The diagnostic strategy falls into three basic categories: power-up, functional, and exerciser.

Power-up diagnostics test most of the NCR pin connections, the script RAM, and some of the NCR chip internals. The emphasis is on testing as many of the NCR chip pin connections as possible. Lesser emphasis is given to testing all of the internals of the NCR chips themselves. These diagnostics run before the SCSI/DSSI drivers complete their initialization code and cannot be repeated without doing a hardware reset. Disk and tape units perform self-tests and those units that fail their self tests do not appear in the system device list.

Functional tests perform reads and writes of data to/from the disk and tape units. They also test some of the pin connections not previously tested during the driver initialization. Functional tests are executed after the driver processes have successfully completing driver initialization.

Exerciser tests are performed while running in a resource competing environment. The goal of the exerciser tests is to detect interaction faults.

### 26.4.3 Power-up Tests

These tests are subdivided into three categories:

- I/O module self tests
- Driver startup/sizing
- Disk/tape unit self tests

Those components of the I/O module used by the mass storage subsystem are tested in parallel with the disk and tape unit self tests. After the I/O module completes self test, the drivers complete their initialization and size the system to determine what units are present.

These tests are invoked when the console starts running, before the console prompt is issued. These tests cannot be invoked from the shell because other applications, in addition to diagnostics, may be using the driver concurrently. The driver initialization is run only once. Hence, these initialization tests are run only once.



### 26.4.3.1 IO Module Self Tests

All of these tests execute even if the five SCSI/DSSI interfaces do not have any disk or tape units attached. These tests cover the I/O module components in the following order: script RAM, NCR 53C710 chip host lines and the SCSI/DSSI lines.

### 26.4.3.2 Driver Startup/Sizing

After the I/O module completes self test, the class and port drivers startup and report any errors detected. Once the drivers have started, they size the system, that is, they build a device list of the units that respond. SCSI/DSSI ports with no disk or tape units attached get very little test coverage from this driver startup code.

#### Expected Failures

Failure models not previously covered are: faulty cables and cable connections; failed disk or tape units (but no media); more of the NCR 53C710 chip internals than previously covered.

#### Test Description

The drivers startup and report any errors detected. The drivers then size the system by sending command packets to units that may be there. Units that do not respond are assumed to be either not present or have failed their self tests. Units that respond are entered into the system device list. SCSI/DSSI ports with no disk or tape units attached get very little test coverage from this driver startup code.

If one or more units per SCSI/DSSI port shows up in the system device table, there exists a high probability that the port's cables and cable connectors as well as the disk/tape unit are functional. If no units show up for any particular port, there is not sufficient information about the integrity of the port's cables and connectors. Units that are physically attached to a port but do not show in a device table are faulty or the cable/connectors are faulty.

### 26.4.3.3 Disk/Tape Unit Self Tests

These tests reside in the firmware of the disk/tape units themselves and are run in response to a hardware reset or device power-up. These tests run independently and in parallel with the DEC 4000 system self tests (which includes the I/O module self tests described above).

Because the method of retrieval of this information varies depending on the unit type and the nature of an error the test results are not retrieved by the console firmware.

#### Expected Failures

Defective disk or tape unit.

#### Test Description

These tests vary depending on the unit type. The units that fail these tests do not responded to normal commands and do not show up in the system device list.

## 26.4.4 Functional Tests

These tests can be invoked from the shell if the operator chooses to run them. These tests are not run at power-up time. They read and write to one or more disk units concurrently. Tape units can be read but not written by the console firmware.

## 26.4 SCSI/DSSI Subsystem

### Expected Failures

Faulty cables and cable connections; failed disk or tape units; bad media, and faulty disk subsystem components on the I/O module.

### Test Description

The two functional tests for mass storage devices are `exer_read` and `exer_write`. These two commands are described in the Console Command Repository Appendix.

### 26.4.5 Exerciser Tests

The third section of testing is the exerciser. The main goal is to test the disk subsystem running in a resource competing environment (both hardware and CPU compute time). These tests detect interaction faults and provide some functional testing as well.

#### Test Description

In general, these tests read, write and verify data to/from the devices. They are started as processes and run concurrently with other exercisers under the DEC 4000 Shell. A status program, `show_status`, can be run to report the progress of the exercisers.

The console command, `test`, concurrently exercises all disks as well as other system hardware. Only one test command at a time should be run. The test command is described in the Console Command Repository Appendix.

The console command, `exer`, provides virtually unlimited ways of specifying device and test specifications to facilitate triggering failures that aren't easily reproduceable. In almost all cases this same functionality is provided with the `exer_read`, `exer_write`, and `test` commands. The `exer` command is intended only for extremely knowledgeable service personnel and then in only rare circumstances. The `exer` command is described in the Console Command Repository Appendix.

## 26.5 Network Subsystem

The network subsystem is part of the I/O module in a DEC 4000 system. It consists of two TGEC chips and associated logic to communicate with the Thinwire and Thickwire ports on the I/O module handle, the CPU and host memory.

### 26.5.1 Initialization

#### 26.5.1.1 Driver Initialization Tests

The initialization tests are completed before the console driver is started and are only invoked on a: power-cycle; hardware reset; or console init command. For each TGEC chip installed the chip itself is verified, and internal and external lookbacks are performed. Errors detected in initialization will be flagged by the LEDs on the OCP and may prohibit the driver from being fully functional.

### Description

Verify the TGEC chip:

Run the TGEC chip-level Power-up Self Test

Verify the port station address ROM

TGEC CSR Storage - Float a one and zero across all bits in one TGEC register.

Initialize the chip - Set up the transmit and receive descriptors, build and send the setup frame.

Verify logic from the TGEC chip to the I/O module handle:

Transmit loopback packets - Send internal and external loopback messages of varying size and alignment.

---

### Note

---

The loopback tests require a correctly terminated network port.

---

## 26.5.2 Exerciser Tests

### Description

The exerciser uses a system wide network exerciser available from the shell. This allows the NI subsystem to run in a resource-competing environment (both hardware and CPU compute time). Both internal and external loopback testing may be performed as well as live-network testing. The more extensive coverage comes from the external or live-network testing.

These tests rely heavily on the EZ port driver and MOP driver for detection and reporting of error information. Because the diagnostic may not be the only application using the driver, it cannot own the driver nor can it force the hardware to do things that might interfere other applications.

The exerciser tests complete the verification of the subsystem that is not covered by the initialization code. The tests are simple in principle but, through customization of environment variables and user created scripts, are able to detect a wide range of problems.

For a complete description of the network exercisers refer to the command Appendix sections for `nettest`, `netexer`, `ntlpex`, `ntlpex1`, `ntlpex2`, and `net`.

## 26.6 Futurebus+ Subsystem

### 26.6.0.1 Hardware Overview

The Futurebus+ Adapter on the I/O module provides an interface between the DEC 4000 system and devices attached to the Futurebus+. This interface makes the physical memory available for access by the Futurebus+ devices as well and provides access to Futurebus+ device CSR's and routes Futurebus+ device interrupts to the system processor.

## 26.7 Futurebus+ Devices

## 26.7 Futurebus+ Devices

### 26.7.1 Hardware Overview

The DEC 4000 system supports up to 6 Futurebus+ devices via a 6 slot Futurebus+ backplane in the system chassis. These devices may be a mixture of both Digital and Non-Digital produced devices.

The DEC 4000 system console supports the testing of only those devices that comply with the Digital Futurebus+ Handbook suggestions for a Standard Test interface.

#### 26.7.1.1 Diagnostic Strategy

The diagnostic strategy for the Futurebus+ devices uses the built-in tests on these devices. Three test sets of diagnostics are used during the automated testing of these devices provided that the device under test implements all three test sets. Should a given device not implement one or more of these test sets, that test set is reported as passed. The three tests used are defined as follows:

1. Power-on Self Tests -

The completion status of the power-on self test is verified during the Futurebus+ sizing. Failing devices are reported. Power-on tests are defined by the Digital Futurebus+ Handbook as being tests that shall not require any external resources. Every test that can be run without support or cooperation with other nodes or resources in the system shall be included in this category.

2. Extended Self Tests -

Once the Futurebus+ has been sized and configured, the extended self-test is executed on each device and the completion status checked. Extended tests are defined by the Digital Futurebus+ Handbook as being tests that shall use a buffer external to the unit under test as scratch space. The handle or pointer to the scratch space is passed to tests in the extended category in the ARGUMENT CSRs. To be consistent with the CSR standard, the buffer shall be 4 Kbytes in size.

3. System Tests -

As part of the DEC 4000 system testing, the system test set for each Futurebus+ device is started and the completion status monitored. System tests are defined by the Digital Futurebus+ Handbook as tests that may require additional system resources beyond those required by a test in the extended category. The exact use of this category is defined by the vendor.

### 26.7.2 Futurebus+ Sizing

#### Overview

The initial configuration of Profile B I/O devices on Digital systems is split between the console code and the device drivers. Following power-on the console configures the various registers on the system bus bridge and then probes the Futurebus+ to determine which devices are present.

Note that the IEEE Futurebus+ standard allows up to two nodes on each module (sides 0 and 1). Modules that have only one node are required to place the node on side 0.

The configuration routine performs the operations outlined in the System Configuration Software section of the Digital Futurebus+ Handbook.

## 26.8 SLU Subsystem

### 26.8.1 Overview

#### 26.8.1.1 Hardware Overview

The I/O module contains two serial line units: The console serial line and the auxiliary serial line. Both serial lines operate in asynchronous mode only. These serial lines are implemented using a single Zilog 85C30 device. Access to the 85C30 is via two byte wide registers for each of the available serial lines which are accessible on L-bus addresses.

### 26.8.2 85C30 Register Test

The 85C30 register test verifies that each bit in the internal registers can be independently set and cleared. Completion of this test provides a high confidence level that the chip is functional and the driver initialization will complete successfully. This test is executed prior to initialization of the console serial line driver and can not be invoked from the console CLI.

---

#### Note

---

Due to the function of some the bits within the registers, not all bits of the registers are tested.

---

#### Failure Model

The focus of this test is to verify the pins of the 85C30 serial line controller chip. The pins to be tested are listed below:

- A!/B
- !CE
- D0-D7
- D!/C
- !RD
- !WR

#### Environment Variables Used

None.

#### Parameters/Options Used

None.

#### Test Description

This test execute a floating 0's and 1's pattern test for the following registers.

- TBD

## 26.8 SLU Subsystem

### 26.8.3 85C30 Internal Loop-back Test

The 85C30 internal loop-back test verify its functionality by looping the transmit signal to the receive signal internal to the chip.

#### Failure Model

Internal transmit and receive logic of the 85c30 chip

#### Environment Variables Used

None

#### Parameters/Options Used

None

#### Test Description

This test sets the 85C30 chip into internal loop-back and transmits and receives a series of characters, a each of several Baudrates. A check is made to ensure that the character transmitted is the same as the character received.

### 26.8.4 85C30 External Loop-back Test

The 85C30 external loop-back test uses an external loop-back connector to loop the transmit signal to the receive signal external to the chip in order verify the functionality of this component.

#### Test Description

This tests utilizes the EXER command along with the external loopback connector to transmit and recieve a series of characters. One each block of characters has been transfered, EXER will compare the transmitted and recieved characters to ensure that the same number of characters were recieved as were transmitted and that the recieved characters match the transmitted characters. The EXER command line to implement this test is as follows:

```
>>>exer tta'n' -bs 1 -a "W-rc" -l 1000
```

This test requires the use of an external loop-back connector and therefore may not be used to test a serial line currently be used as the console input/output channel. In order to execute this test on the console serial line, the console input /output must be redirected to either the auxiliary serial line or the MOP based remote console.

## 26.9 Multi-Processor

The DEC 4000 may be configured as a multi-processor system. While it is possible to run diagnostics locked to a single processor, most tests will be scheduled to run on the next available processor. This allows devices to be tested from both processors.

It may also be desirable to stress the dependency of the processor caches in a multi-processor system. This may be done using the memexer\_mp script. This script locks a memory exerciser to each processor. Each exerciser will test every other longword. This interaction causes a high incidence of shared and dirty cache blocks.

The cache status may be examined using the cbcc command. Cbcc verifies the coherhency of the caches on each processor. It can also list the value of the tag and the tag control bits at a specific point in time for the desired cache lines.

For a complete descriptions of the commands related to multi-processing refer to the command Appendix sections for memexer\_mp, memtest and cbcc.

### 26.10 Serial Control Bus Subsystem

#### 26.10.1 Controller Register Test (iic\_reg\_test)

The serial control bus controller register test verifies that each bit the internal registers can be independently set and cleared. Completion of this test provides a high-confidence level that the chip is functional and that the driver initialization will complete successfully. This test is executed prior to initialization of the serial control bus driver, and can not be invoked from the console CLI.

---

**Note**

---

Because of the function of some of the bits within the Control register, not all bits of this register are tested.

---

#### **Test Description**

Execute a floating 0's and 1's pattern test for the following registers.

- Data shift (S0) - all bits
- Own address (S0') - all bits
- Control/status (S1) - bits <5:0>
- Clock register (S2) - all bits
- Interrupt vector (S) - all bits

#### 26.10.2 Bus Access Test (iic\_acc\_test)

The serial control bus access test verifies the ability to access devices attached to the serial control bus. This test makes uses the LED driver contained on the OCP as the test target. This test is executed by the serial control bus driver initialization routine, and can not be invoked from the console CLI.

#### **Test Description**

This test reads the current value of the OCP LED driver and verifies that it contains the power-up default value of 00h.

## IPR State on Power-up and RESET

The table below lists the state of all the IPRs immediately following reset. The table also specifies which IPRs need to be initialized by power-up PALcode.

**Table A–1 IPR Reset State**

IPR	Reset State	Comments
ITB_TAG	undefined	
ITB_PTE	undefined	
ICCSR	cleared	Floating point disabled, single issue mode, VAX mode enabled, ASN = 0, jsr predictions disabled, branch predictions disabled, branch history table disabled, performance counters reset to zero, Perf Cnt0(16b) : Total Issues/2, Perf Cnt1(12b) : Dcache Misses
ITB_PTE_TEMP	undefined	
EXC_ADDR	undefined	
SL_RCV	undefined	
ITBZAP	n/a	PALcode must do a itbzap on reset.
ITBASM	n/a	
ITBIS	n/a	
PS	undefined	PALcode must set processor status.
EXC_SUM	undefined	Palcode must clear exception summary and exception register write mask by doing 64 reads.
PAL_BASE	cleared	Cleared on reset.
HIRR	n/a	
SIRR	undefined	PALcode must initialize.
ASTRR	undefined	PALcode must initialize.
HIER	undefined	PALcode must initialize.
SIER	undefined	PALcode must initialize.
ASTER	undefined	PALcode must initialize.
SL_XMIT	undefined	PALcode must initialize. Appears on external pin.
DTB_CTL	undefined	Palcode must select between SP/LP dtb prior to any TB fill.
DTB_PTE	undefined	
DTB_PTE_TEMP	undefined	
MMCSR	undefined	Unlocked on reset.

(continued on next page)



**Table A–1 (Cont.) IPR Reset State**

IPR	Reset State	Comments
VA	undefined	Unlocked on reset.
DTBZAP	n/a	PALcode must do a dtbzap on reset.
DTBASM	n/a	
DTBIS	n/a	
BIU_ADDR	undefined	Potentially locked.
BIU_STAT	undefined	Potentially locked.
SL_CLR	undefined	PALcode must initialize.
DC_ADDR	undefined	Potentially locked.
DC_STAT	undefined	Potentially locked.
FILL_ADDR	undefined	Potentially locked.
ABOX_CTL	see comments	[11..0] <- ^x0100 Write buffer enabled, machine checks disabled, correctable read interrupts disabled, Icache stream buffer disabled, Dcache disabled, forced hit mode off.
ALT_MODE	undefined	
CC	undefined	Cycle counter is disabled on reset.
CC_CTL	undefined	
BIU_CTL	see comments	Bcache disabled, parity mode undefined, chip enable asserts during RAM write cycles, Bcache forced-hit mode disabled. BC_PA_DIS field cleared. BAD_TCP cleared. BAD_DP undefined.  Note: The Bcache parameters BC RAM read speed, BC RAM write speed, BC write enable control, and BC size are all undetermined on reset and must be initialized before enabling the Bcache.
FILL_SYNDROME	undefined	Potentially locked.
BC_TAG	undefined	Potentially locked.
PAL_TEMP[31..0]	undefined	

PALcode should execute four jsr call instructions to initialize the jsr stack. This is necessary to insure deterministic behavior for testers. The following code will initialize the stack once the ICCSR [JSE] bit is set.

```

                BSR    r1,stk_1    ; push RET PC
stk_1:
                BSR    r2,stk_2    ; push RET PC
stk_2:
                BSR    r3,stk_3    ; push RET PC
stk_3:
                BSR    r4,stk_4    ; push RET PC
stk_4:

```

## A.1 TB Miss Flows

This section describes hardware specific details to aid the PALcode programmer in writing ITB and DTB fill routines. These flows were included to highlight trade-offs and restrictions between PAL and hardware. The PALcode source that is released with EVx should be consulted before any new flows are written. A working knowledge of the ALPHA memory management architecture is assumed.

### A.1.1 ITB Miss

When the Ibox encounters an ITB miss it latches the VPC of the target instruction-stream reference in the EXC\_ADDR IPR, flushes the pipeline of any instructions following the instruction which caused the ITB miss, waits for any other instructions which may be in progress to complete, enters PALmode, and jumps to the ITB miss PAL entry point. The recommended PALcode sequence for translating the address and filling the ITB is described below.

1. Create some scratch area in the integer register file by writing the contents of a few integer registers to the PAL\_TEMP register file.
2. Read the target virtual address from the EXC\_ADDR IPR.
3. Fetch the PTE (this may take multiple reads) using a physical-mode HW\_LD instruction. If this PTE's valid bit is clear report TNV or ACV as appropriate.
4. Since the ALPHA SRM states that translation buffers may not contain invalid PTEs, the PTE's valid bit must be explicitly checked by PALcode. Further, since the ITB's PTE RAM does not hold the FOE bit, the PALcode must also explicitly check this condition. If the PTE's valid bit is set and FOE bit is clear, PALcode may fill an ITB entry.
5. Write the original virtual address to the TB\_TAG register using HW\_MTPR. This writes the TAG into a temp register and not the actual tag field in the ITB.
6. Write the PTE to the ITB\_PTE register using HW\_MTPR. This HW\_MTPR causes both the TAG and PTE fields in the ITB to be written. Note it is not necessary to delay issuing the HW\_MTPR to the ITB\_PTE after the MTPR to the ITB\_TAG is issued.
7. Restore the contents of any modified integer registers to their original state using the HW\_MFPR instruction.
8. Restart the instruction stream using the HW\_REI instruction.

### A.1.2 DTB Miss

When the Abox encounters a DTB miss it latches the referenced virtual address in the VA IPR and other information about the reference in the MMCSR IPR, and locks these registers against further modifications. The Ibox latches the PC of the instruction which generated the reference in the EXC\_ADDR register, drains the machine as described above for ITB misses, and jumps to the DTB miss PALcode entry point. Unlike ITB misses, DTB misses may occur while the CPU is executing in PALmode. The recommended PALcode sequence for translating the address and filling the DTB is described below.

1. Create some scratch area in the integer register file by writing the contents of a few integer registers to the PAL\_TEMP register file.
2. Read the requested virtual address from the VA IPR. Although the act of reading this register unlocks the VA and MMCSR registers, the MMCSR register only updates when D-stream memory management errors occur. It therefore will retain information about the instruction which generated this DTB miss. This may be useful later.
3. Fetch the PTE (may require multiple reads). If the valid bit of the PTE is clear, a TNV or ACV must be reported unless the instruction which caused the DTB miss was FETCH or FETCH/M. This can be checked via the opcode field of the MMCSR register. If the value in this field is 18 (hex), then a

## A.1 TB Miss Flows

FETCH or FETCH/M instruction caused this DTB miss, and as mandated by the ALPHA SRM, the subsequent TNV or ACV should NOT be reported. Therefore PALcode should read the value in EXC\_ADDR, increment it by four, write this value back to EXC\_ADDR, and do a HW\_REI.

4. Write the register which holds the contents of the PTE to the DTB\_CTL IPR. This has the effect of selecting either the small or large page DTB for subsequent DTB fill operations, based on the value contained in the granularity hint field of the PTE.
5. Write the original virtual address to the TB\_TAG register. This writes the TAG into a temp register and not the actual tag field in the DTB
6. Write the PTE to the DTB\_PTE register. This HW\_MTPR causes both the TAG and PTE fields in the DTB to be written. Note it is not necessary to delay issuing the HW\_MTPR to the DTB\_PTE after the MTPR to the DTB\_TAG is issued.
7. Restore the contents of any modified integer registers.
8. Restart the instruction stream using the HW\_REI instruction.

The following sections give a summary of the hardware flows for various error conditions for the 21064 (EV4) CPU.

### B.1 I-stream ECC error

- data put into Icache unchanged, block gets validated
- machine check
- BIU\_STAT: FILL\_ECC, FILL\_IRD set, FILL\_SEO set if multiple errors occurred
- FILL\_ADDR[33..5] & BIU\_STAT[FILL\_QW] give bad QW's address
- FILL\_SYNDROME contains syndrome bits associated with failing quadword
- BIU\_ADDR, BIU\_STAT[6..0] locked - contents are UNPREDICTABLE
- DC\_STAT locked - contents are UNPREDICTABLE
- BC\_TAG holds results of external cache tag probe if external cache was enabled for this transaction

### B.2 D-stream ECC error

- data put into Dcache unchanged, block gets validated
- machine check
- BIU\_STAT: FILL\_ECC set, FILL\_IRD clear, FILL\_SEO set if multiple errors occurred
- FILL\_ADDR[33..5] & BIU\_STAT[FILL\_QW] give bad QW's address
- FILL\_ADDR[4..2] contain PA bits [4..2] of location which the failing load instruction attempted to read
- FILL\_SYNDROME contains syndrome bits associated with failing quadword
- BIU\_ADDR, BIU\_STAT[6..0] locked - contents are UNPREDICTABLE
- DC\_STAT: RA identifies register which holds the bad data. LW,LOCK,INT,VAX\_FP identify type of load instruction
- BC\_TAG holds results of external cache tag probe if external cache was enabled for this transaction

### **B.3 BIU: tag address parity error**

### **B.3 BIU: tag address parity error**

- recognized at end of tag probe sequence
- lookup uses predicted parity so transaction misses the external cache
- BC\_TAG holds results of external cache tag probe
- machine check
- BIU\_STAT: BC\_TPERR set
- BIU\_ADDR holds address

### **B.4 BIU: tag control parity error**

- recognized at end of tag probe sequence
- transaction forced to miss external cache
- BC\_TAG holds results of external cache tag probe
- machine check
- BIU\_STAT: BC\_TCPERR set
- BIU\_ADDR holds address

### **B.5 BIU: system external transaction terminated with CACK\_SERR**

- CRD interrupt.
- BIU\_STAT: BIU\_SERR set, BIU\_CMD holds cReq\_h[2..0].
- BIU\_ADDR holds address.

### **B.6 BIU: system transaction terminated with CACK\_HERR**

- machine check
- BIU\_STAT: BIU\_HERR set, BIU\_CMD holds cReq\_h[2..0]
- BIU\_ADDR holds address

### **B.7 BIU: I-stream parity error (parity mode only)**

- data put into Icache unchanged, block gets validated
- machine check
- BIU\_STAT: FILL\_DPERR set, FILL\_IRD set
- FILL\_ADDR[33..5] & BIU\_STAT[FILL\_QW] give bad QW's address
- FILL\_SYNDROME identifies failing longword(s)
- BIU\_ADDR, BIU\_STAT[6..0] locked - contents are UNPREDICTABLE
- DC\_STAT locked - contents are UNPREDICTABLE
- BC\_TAG holds results of external cache tag probe if external cache was enabled for this transaction

## B.8 BIU: D-stream parity error (parity mode only)

### B.8 BIU: D-stream parity error (parity mode only)

- data put into Dcache unchanged, block gets validated
- machine check
- BIU\_STAT: FILL\_DPERR set, FILL\_IRD clear
- FILL\_ADDR[33..5] & BIU\_STAT[FILL\_QW] give bad QW's address
- FILL\_ADDR[4..2] contain PA bits [4..2] of location which the failing load instruction attempted to read
- FILL\_SYNDROME identifies failing longword(s)
- BIU\_ADDR, BIU\_STAT[6..0] locked - contents are UNPREDICTABLE
- DC\_STAT: RA identifies register which holds the bad data.  
LW,LOCK,INT,VAX\_FP identify type of load instruction
- BC\_TAG holds results of external cache tag probe if external cache was enabled for this transaction

---

## AC and DC Characteristics

**Table C–1 System bus AC and DC Characteristics**

Parameter	Description	Min	Max	Units	Notes
<b>DC Characteristics</b>					
Voh	Output high voltage @Ioh = 8ma	2.4	-	V	
Vol	Output low level @Iol= -8ma	-	.4	V	
Vohck	Output high voltage @Ioh = -8ma	4.5	-	V	
Volck	Output low voltage @Iol = 8ma	-	.5	V	
Volod	Output low voltage @Iol = 48ma	-	.4	V	
Ioh	Output high current @Voh = min	8	-	ma	
Iol	Output low current @Vol = max	8	-	ma	
Iohck	Output high current @Voh= max	8	-	ma	
Iolck	Output low current @Vol= max	8	-	ma	
Iolod	Output low current @Vol= max	48	-	ma	
Ioz	Tri-state leakage current Vpin= 0 to 4.74v	-100	+100	ua	
Vih	Input high voltage	2	-	V	
Vil	Input low voltage	-	.8	V	
Vihck	Input high clock voltage	70%-Vdd	-	V	
Vilck	Input low clock voltage	-	30%-Vdd	V	

# D

---

## Console Command Repository



## alloc

---

### alloc — Allocate a block of memory from the heap.

Exports the 'malloc' routine out to the shell, so that users may allocate a block of memory from the heap. The resulting block may then be used simultaneously by several test routines (there can be several readers but only one writer).

#### Syntax

```
alloc size [modulus] [remainder] [-flood] [-z heap_address]
```

#### Arguments

***size***

Specifies the size (hex) in bytes of the requested block.

***modulus***

Specifies the modulus (hex) for the beginning address of the requested block.

***remainder***

Specifies the remainder (hex) used in conjunction with the modulus for computing the beginning address of the requested block.

#### Options

**-flood**

Flood memory with 0s. By default, alloc does not flood.

**-z *heap\_address***

Allocate from the memory zone starting at address *heap\_address*. This address is usually obtained from the output of a 'dynamic' command).

#### Examples

```
>>>alloc 200
00FFFE00
>>>free fffe00
>>>set base `alloc 400`
>>>show base
base                00FFFC00
>>>mementest $base
>>>free $base
>>>clear base
```

#### Command References

dynamic, free

---

## bin — Convert an ASCII representation of hex digits into binary values.

Converts a ASCII representation of hexadecimal digits (0..9, a..f) into their binary values. All other characters are ignored.

### Syntax

```
bin hex_digit..
```

### Arguments

*hex\_digit*..

Specifies the hexadecimal digit(s) to be converted to binary.

### Options

None.

### Examples

```
>>>bin 12 > foo
>>>hd foo
00000000 12 .
>>>bin 1 2 3 4 5 6 7 8 > foo
>>>hd foo
00000000 21 43 65 87 !Ce.
>>>bin 12345678 abcdef > foo
>>>hd foo
00000000 78 56 34 12 ef cd ab xV4.íí«
>>>bin 01 23 45 67 89 ab cd ef > foo
>>>hd foo
00000000 01 23 45 67 89 ab cd ef .#Eg.«íí
>>>bin g h i j k l m n o > foo
>>>hd foo
```

### Command References

None

## boot

---

### boot — Bootstrap the system.

Initializes the processor, loads a program image from the specified boot device, and transfers control to that image. If you do not specify a boot device in the command line, the default boot device is used. The default boot device is determined by the value of the 'bootdef\_dev' environment variable.

If you specify a list of devices, a bootstrap is attempted from each device in order. Then control passes to the first successfully booted image. In a list, always enter network devices last, since network bootstraps only terminate if a fatal error occurs or an image is successfully loaded.

The -flags option can pass additional information to the operating system about the boot that you are requesting. On a VMS system, the -flags option specifies the system root number and boot flags. If you do not specify a boot flag qualifier, then the default boot flags value specified by the 'boot\_osflags' environment variable is used.

The -protocol option allows selection of either the DECNET MOP or the TCP/IP BOOTP network bootstraps. The keywords 'mop' or 'bootp' are valid arguments for this option. Note, it is possible to set the default protocol for a port by setting the environment variable, 'eza0\_protocols' or 'ezb0\_protocols' to the appropriate protocol.

Explicitly stating the boot flags or the boot device overrides the current default value for the current boot request, but does not change the corresponding environment variable.

### Syntax

```
boot [-file filename] [-flags longword[,longword]]  
      [-protocols enet_protocol] [-halt]  
      [boot_device]
```

### Arguments

#### **boot\_device**

A device path or list of devices from which the firmware attempts to boot, or a saved boot specification in the form of an environment variable. Use the set bootdef\_dev command to define the default boot device.

### Options

#### **-file filename**

Specifies the name of a file to load into the system. For booting from Ethernet, this name is limited by the MOP V3 load protocol to 15 characters. Use the set boot\_file command to specify a default boot file.

#### **-flags longword[,longword]**

Specifies additional information to the operating system. In the case of VMS, specifies system root number and boot flags. These values are passed to the operating system for interpretation. Preset default boot flag values are 0,0. Use the set boot\_osflags command to change the default boot flag values.

**-protocols *enet\_protocol***

Specifies the Ethernet protocol(s) to be used for the network boot. Either the keyword 'mop' or 'bootp' may be specified. If both are specified, each protocol is attempted to solicit a boot server.

**-halt**

Forces the bootstrap operation to halt and invoke the console program once the image is loaded and page tables and other data structures are set up. Console device drivers are not shut down when this qualifier is present. Transfer control to the image by entering the continue command.

**Examples**

In the following example, the system boots from the default boot device. The console program returns an error message if a default boot device has not been set.

```
>>>boot
```

In the next example, the system boots from the Ethernet port, eza0.

```
>>>boot eza0
```

In the next example, the system boots the file named 'dec\_4000.sys' from Ethernet port mke0.

```
>>>boot -file dec_4000.sys mke0
```

In the next example, the system attempts a TCP/IP BOOTP network boot from Ethernet port eza0.

```
>>>boot -protocol bootp eza0
```

In the next example, the system boots from the default boot device using boot flag settings 0,1.

```
>>>boot -flags 0,1
```

In the next example, the system loads the operating system from the SCSI disk, dka0, but remains in console mode. Subsequently, you can enter the continue command to transfer control to the operating system.

```
>>>boot -halt dka0
```

**Command References**

set, show

## cat

---

### cat — Copy files to standard output.

Concatenates files that you specify to the standard output. If you do not specify files on the command line, cat copies standard input to standard output.

You can also copy or append one file to another by specifying I/O redirection.

#### Syntax

```
cat [-l num] file...
```

#### Arguments

**file...**

Specifies the name of the input file or files to be copied.

#### Options

**-l num**

Specifies the number of bytes of the file to copy.

#### Examples

```
>>>echo > foo 'this is a test.'  
>>>cat foo  
this is a test.  
>>>cat -l 6 foo  
this i  
>>>
```

#### Command References

echo, ls, rm

---

## cbcc — Check backup cache coherency.

The cbcc command checks backup cache coherency. It can be used to test cache coherency in both single and dual processor systems.

The cbcc command has three distinct pieces, the coherency master process, the coherency slave process, and the display flag. In a multiprocessor environment each processor will run either a master or a slave process. The master process will be run on only one processor and the other active processors will run slave processes. Display of the memory and cache information is done with the -s qualifier.

### Syntax

```
cbcc [-cm] [-cs] [-dc] [-s] [-sa] [-sum]
     [-si start_index] [-l index_count]
     [-p pass_count] [-w wait_time]
```

### Arguments

None.

### Options

**-cm**

Create the master cache coherency checker.

**-cs**

Create the slave cache coherency checker.

**-dc**

Enable D-cache.

**-s**

Show tag, cache, and memory.

**-sum**

Show summary of the cache entries checked.

**-si *start\_index***

Specifies the starting cache index to be checked.

**-l *index\_count***

Specifies the number of indices to be checked.

**-p *pass\_count***

Specifies the number of passes of the cache test. If 0, then run forever or until CTRL-C. The default is 1.

**-w *wait\_time***

Specifies the number of seconds to wait between checks of the cache. The default is 10 seconds.

## cbcc

### Examples

```
# These examples are for a multi-processor system.

>>>cbcc -cs &pl& # start the coherency checker running on the
# slave. This process runs forever.
# display a summary of 32 cache lines starting at 0x100

>>>cbcc -cm -si 100 -l 20 -sum &p0
Master B-Cache summary Entries: 32 V: 32 S: 9 D: 21 SD: 2
Slave B-Cache summary Entries: 32 V: 3 S: 3 D: 0 SD: 0
>>>
# show the full status of cache line 0x1000.

>>>cbcc -cm -si 1000 -s &p0
Master:
TAG: 009 TP: 0 I: 1000 HIT: 0 S: 0 D: 1 V: 1 CP: 0
efefefef efefefef efefefef efefefef : 1000 C
efefefef efefefef efefefef efefefef
efefefef efefefef efefefef efefefef : 00920000 M
efefefef efefefef efefefef efefefef : 00920010 M
Slave:
TAG: 001 TP: 1 I: 1000 HIT: 0 S: 0 D: 0 V: 0 CP: 0
efefefef efefefef efefefef efefefef : 1000 C
efefefef efefefef efefefef efefefef
efefefef efefefef efefefef efefefef : 00120000 M
efefefef efefefef efefefef efefefef : 00120010 M
>>>
# display a summary of all 32K cache lines

>>>cbcc -cm -si 0 -l 8000 -sum &p0
Master B-Cache summary Entries: 32768 V: 32434 S: 4809 D: 24827 SD: 660
Slave B-Cache summary Entries: 32768 V: 4333 S: 3764 D: 331 SD: 174
>>>

# These examples are on a single processor system

# show the full status of cache line 0x200.

>>>cbcc -cm -si 200 -s
Master:
TAG: 035 TP: 0 I: 0200 HIT: 0 S: 0 D: 1 V: 1 CP: 0
efefefef efefefef efefefef efefefef : 0200 C
efefefef efefefef efefefef efefefef
efefefef efefefef efefefef efefefef : 03504000 M
efefefef efefefef efefefef efefefef : 03504010 M
>>>
# display a summary of 0x200 cache lines starting at line 0.

>>>cbcc -cm -l 200 -sum
Master B-Cache summary Entries: 512 V: 512 S: 0 D: 490 SD: 0
>>>
# display a summary of all 32K cache lines

>>>cbcc -cm -l 8000 -sum
Master B-Cache summary Entries: 32768 V: 32768 S: 332 D: 26266 SD: 0
>>>
```

### Command References

None

---

## cdp — Configure DSSI parameters.

The `cdp` command permits the modification of DSSI device parameters from the console without explicit connection to a node's DUP server. The parameters which are modified are the DUP task PARAMS: NODENAME, ALLCLASS, and UNITNUM.

Entering `cdp` without an option or target device will list these parameters for all DSSI devices in the system.

### Syntax

```
cdp  [-(i,n,a,u,o)] [-sn]
      [-sa allclass] [-su unitnum] [dssi_device]
```

### Arguments

#### *dssi\_device*

Name of the DSSI device or DSSI controller name. Only the parameters for this device or the devices on this controller will be modified.

### Options

#### **-i**

Selective interactive mode, set all parameters.

#### **-n**

Set device node name, NODENAME (upto 16 characters).

#### **-a**

Set device allocation class, ALLCLASS.

#### **-u**

Set device unit number, UNITNUM.

#### **-o**

Override warning messages.

#### **-sn**

Set the node name (NODENAME) for all DSSI devices in the system to either RFhscn or TFhscn, where "h" is the device hose number (0), "s" is the device slot number (0), "c" is the device channel number (0..3), and "n" is the device node ID number (0..6).

#### **-sa *allclass***

Set the allocation class (ALLCLASS) for all DSSI devices in the system to the value specified.

#### **-su *unitnum***

Set the starting unit number (UNITNUM) for the first DSSI device in the system to the value specified. The unit number for subsequent DSSI devices will be incremented from this base.



## cdp

### Examples

```
>>>show device
dua5.5.0.0.0      BASHFL$DIA5      RF71
dub44.4.1.0.0    $1$DIA44 (BLANK4) RF71
>>>cdp -i

pua.5.0.0.0:
Node Name [BASHFL]?
Allocation Class [0]?
Unit Number [5]?

pub.4.1.0.0:
Node Name [BLANK4]?
Allocation Class [1]?
Unit Number [44]?

>>>cdp -n dua5

pua.5.0.0.0:
Node Name [BASHFL]?
>>>cdp -a

pua.5.0.0.0:
Allocation Class [0]?

pub.4.1.0.0:
Allocation Class [1]?
>>>
```

### Command References

set host -dup

---

## check — Evaluate a string or the attributes of a inode.

Check evaluates a predicate and returns the result as status. By default, check returns 'true', if the argument 's' is not a null string.

### Syntax

```
check [-{f,r,w,x,b}] [!] [string]
```

### Arguments

#### *string*

Specifies a string or an inode name to be checked.

### Options

#### **-f**

Return true, if the inode exists.

#### **-r**

Return true, if the inode is readable.

#### **-w**

Return true, if the inode is writeable.

#### **-x**

Return true, if the inode is executable.

#### **-b**

Return true, if the inode is binary.

#### **!**

Return the negation of the evaluated check status.

### Examples

```
>>> echo > foo 'Hello World.'
>>>cat foo
Hello World.
>>>ls -l foo
-rwx-  rd          13/2048          0  foo
>>>if check -f foo; then cat foo; fi
Hello World.
>>>if check -w foo; then cat foo; fi
Hello World.
>>>if check -b foo; then cat foo; fi

>>>rm foo
>>>if check -f foo; then cat foo; fi

>>>if check -f ! foo; then cat foo; fi
Hello World.
>>>
```

### Command References

None

## chmod

---

### chmod — Change the mode of a file.

Changes the specified attributes of a file. The chmod command is a subset of the equivalent U\*x command.

You alter a file's attributes by entering an operation on the command line: A minus sign (-) indicates to remove the attribute, a plus sign (+) indicates to add the attribute, and an equals sign (=) indicates to set the attribute absolutely (clear all other attributes not in the list). You can only enter one operation per command line.

To specify an option, you must place the option as the first argument in a command line.

#### Syntax

```
chmod  [{-|+|=}{r,w,x,b,z}] [file...]
```

#### Arguments

**file...**

Specifies the file(s) or inode(s) to be modified.

#### Options

**r**

Set or clear the read attribute.

**w**

Set or clear the write attribute.

**x**

Set or clear the execute attribute.

**b**

Set or clear the binary attribute.

**z**

Set or clear the expand attribute.

#### Examples

```
>>> chmod +x script ! makes file script executable
>>> chmod =r errlog ! sets error log to read only
>>> chmod -w dk* ! makes all scsi disks non writeable
```

#### Command References

chown, ls -l

---

## chown — Change the process ownership of a block of memory.

Changes the ownership of a memory block to the specified process.

### Syntax

```
chown pid address..
```

### Arguments

***pid***

Specifies a process id (hex). PIDs may be displayed using the ps command.

***address..***

Specifies an address (hex) or list of addresses of allocated block(s) for which ownership is to be changed.

### Options

**None.**

### Examples

```
>>>chown `ps|grep idle|find 0` `alloc 200`
```

### Command References

alloc, dynamic, ps

## clear

---

### clear — Delete an environment variable.

Deletes an environment variable from the name space.

Note that some environment variables, such as `bootdef_dev`, are permanent and cannot be deleted.

#### Syntax

```
clear envvar
```

#### Arguments

*envvar*

Specifies the name of the environment variable to be cleared.

#### Options

None.

#### Examples

In the following example, an environment variable, 'foo', is deleted.

```
>>>clear foo
>>>
```

#### Command References

set, show

---

## cmp — Compare two files.

Compares the contents of two files that you specify and reports offsets where the files are different.

The `cmp` command reads the files into internal buffers in blocks of 1024 bytes (the block size can be changed with the `-b` qualifier) and compares the files on a byte by byte basis. If the files are equal, `cmp` does not issue a report.

One of the files may be '-', in which case `stdin` is used.

You can compare portions of files by using the `-n` qualifier and the skipcounts for each file. Also using the `-n` qualifier, you can specify how many characters, in decimal, are to be compared.

If you specify skip parameters, then the appropriate number of characters are ignored from the input stream. If one of the files is a pipe, then the pipe is drained for skipcount characters by reading and discarding the data. Otherwise, the characters are skipped by an `fseek` call. You specify skipcounts in decimal.

If you specify skipcounts, offsets are reported relative to the skipcount.

### Syntax

```
cmp [-n bytes] [-b size] file1 file2 [skip1] [skip2]
```

### Arguments

**file1**

Specifies the first file to be compared.

**file2**

Specifies the second file to be compared.

**skip1**

Specifies the number of characters to skip in file1.

**skip2**

Specifies the number of characters to skip in file2.

### Options

**-n bytes**

Specifies the number of bytes to compare.

**-b size**

Specifies the user defined buffer size.

### Examples

```
>>>echo abc > foo
>>>echo abcd > bar
>>>echo zzabc > foobar
>>>cmp foo bar
>>>cmp foo foobar 0 2
>>>cmp foo foo
>>>cat foo | cmp foo
>>>
```

**cmp**

## **Command References**

cat, echo

---

## continue — Resume program execution on the specified processor.

Continues execution on the specified processor, or the primary processor if a processor is not specified. The processor begins executing instructions at the address currently contained in the program counter. The processor is not initialized.

The continue command is only valid if an operator has halted the system by one of two methods: either by pressing the Halt button on the control panel or by entering Ctrl/P on the console terminal.

Note that some console commands, for example, test and boot, may alter the machine state so that program mode cannot be successfully resumed.

### Syntax

```
continue
```

### Arguments

None.

### Options

None.

### Examples

In the following example, the primary processor resumes operating system mode.

```
>>>continue
```

In the next example, a system's second processor is commanded to resume operating system mode.

```
>>>continue &p 2
```

### Command References

start, stop



## crc

---

### crc — Calculate a CRC on a file.

Calculates a CRC on a file. If you do not specify any options, then computes the CRC on the entire file.

#### Syntax

```
crc [-s val] [-e val] [-l val] file
```

#### Arguments

**file**  
file on which to calculate the CRC

#### Options

**-s**  
starting offset within the file

**-e**  
ending offset within the file

**-l**  
number of bytes to compute the crc on.

#### Examples

```
>>>cat foo
hello world
>>>crc foo
0x00000466
>>>hd foo
00000000 68 65 6c 6c 6f 20 77 6f 72 6c 64 0a hello world.
>>>eval -x \
_>0x68 0x65 0x6c 0x6c 0x6f 0x20 0x77 0x6f 0x72 0x6c 0x64 0x0a \
_> + + + + + + + + + + +
00000466
>>>d -b -n 8 pmem:0 0
>>>hd -l 8 pmem
00000000 00 00 00 00 00 00 00 00 .....
>>>crc -l 8 pmem
0x00000000
>>>d -b -n 8 pmem:0 1
>>>hd -l 8 pmem
00000000 01 01 01 01 01 01 01 01 .....
>>>crc -l 8 pmem
0x00000008
>>>crc -s 4 -l 4 pmem
0x00000004
>>>
```

---

## date — Set or display the current time and date.

Displays or modifies the current time and date. If you do not specify any arguments, date displays the current date and time. If you specify arguments, date modifies the arguments that you specify in the TOY clock.

If you want to modify the date or time, you must specify at least four digits, those that represent the hour and minute. Omitted fields are inherited. When setting the date, the day of the week is automatically generated.

### Syntax

```
date [yyyymmddhhmm.ss]
```

### Arguments

#### *yyyymmddhhmm.ss*

Specifies the date and time string consisting of decimal pairs, where:

- 'yyyy' (0000-9999) is the year,
- 'mm' (01-12) is the two digit month,
- 'dd' (01-31) is the two digit day,
- 'hh' (00-23) is the two digit hour,
- 'mm' (00-59) is the two digit minute, and
- 'ss' (00-59) is the two digit second.

-bias What the year is biased by. By default, this is 1858

### Examples

```
>>>date 199208031029.00
>>>date
10:29:04 August 3, 1992
>>>
```

### Command References

None

## deposit

---

### deposit — Write data to a specified address.

Writes data to an address that you specify: a memory location, a register, a device, or a file.

After initialization, if you have not specified a data address or size, the default address space is physical memory, the default data size is a quadword, and the default address is zero.

You specify an address or "device" by concatenating the device name with the address, for example, PMEM:0 and by specifying the size of the space to be written to.

If you do not specify an address, the data is written to the current address, in the current data size (the last previously specified address and data size).

If you specify a conflicting device, address, or data size, the console ignores the command and issues an error response.

The display line consists of the device name, the hexadecimal address (or offset within the device), and the examined data also in hexadecimal.

The EXAMINE command supports most of the same options. Additionally, EXAMINE supports instruction decoding, the -d option, which disassembles instructions beginning at the current address.

### Syntax

```
deposit  [-(b,w,l,q,o,h)] [-(physical,virtual,gpr,fpr,ipr)]  
        [-n count] [-s step]  
        [device:]address data
```

### Arguments

#### [*device*:]

The optional device name (or address space) selects the device to access. The following platform independent devices supported:

- pmem: Physical memory.
- vmem: Virtual memory. All access and protection checking occur. If the access would not be allowed to a program running with the current PS, the console issues an error message. If memory mapping is not enabled, virtual addresses are equal to physical addresses.

The following platform dependent devices are supported:

- eerom: Environment variable and error log EEROM
- enet: Ethernet station address ROM
- fbus: FutureBus+
- ferom: Intel 28F010 firmware FEPR0M
- iic: PCD8584 registers I2Cbus controller
- ncr0(1,2,3,4): NCR53710 registers DSSI/SCSI (port
- scam: Script RAM
- tgec0(1): TGEC registers Ethernet (ports 0,1)

- toy: DS1287A registers clock chip and NVRAM
- uart: Z8530 registers Console serial port

**address**

An address that specifies the offset within a device into which data is deposited. The address may be any valid hexadecimal offset in the device's address space. Note: Memory addresses of the form: Fdd, where dd is a 1-2 digit decimal number, must be preceded by a 0 to prevent recognition as a Floating-Point Register. Ex: 0f0 is a valid memory address; f0 is not. The address may also be any legal symbolic address. However, a symbolic name cannot be used if the address space was explicitly specified. The following forms are valid symbolic addresses:

- gpr-name - A symbol representing a General Purpose Register.
- ipr-name - A symbol representing the Internal Processor Register.
- PC - Program Counter (execution address). The last address, size, and type are unchanged.
- "+" - The location immediately following the last location referenced in an examine or deposit. For references to physical or virtual memory, the location is the last address plus the size of the last reference. For other address spaces, the address is the last address referenced plus one.
- "-" - The location immediately preceding the last location referenced in an examine or deposit. For references to physical or virtual memory, the location is the last address minus the size of the last reference. For other address spaces, the address is the last address referenced minus one.
- "\*" - The location last referenced by an examine or deposit.
- "@" - The location addressed by the last location referenced in an examine or deposit.

**data**

The data to be deposited. If the specified data is larger than the deposit data size, the console ignores the command and issues an error response. If the specified data is smaller than the deposit data size, it is extended on the left with zeros.

**Options****-b**

The data type is byte.

**-w**

The data type is word.

**-l**

The data type is longword.

**-q**

The data type is quadword.

**-o**

The data type is octaword.

**-h**

The data type is hexaword.

## deposit

### **-physical**

The address space is physical memory.

### **-virtual**

The address space is virtual memory.

### **-gpr**

The address space is general purpose registers.

### **-fpr**

The address space is floating point registers.

### **-ipr**

The address space is internal processor registers.

### **-n *count***

Specifies the number of consecutive locations, *val* (hex), to modify. The console deposits to the first address, then to the specified number of succeeding addresses.

### **-s *step***

Specifies the address increment size (hex). Normally this defaults to the data size, but is overridden by the presence of this qualifier. This qualifier is not inherited.

## Examples

```
>>>d -b -n 1FF pmem:0 0      ! Clear first 512 bytes of physical memory.
>>>d -l -n 3 vmem:1234 5     ! Deposit 5 into four longwords starting at virtual memory address 1234.
>>>d -n 8 R0 FFFFFFFF        ! Loads GPRs R0 through R8 with -1.
>>>d -l -n 10 -s 200 pmem:0 8 ! Deposit 8 in the first longword of the first 17 pages in physical memory.
>>>
```

## Command References

examine

---

## dynamic — Show the state of dynamic memory.

Show the state of dynamic memory. Dynamic memory is split into two main heaps, the console's private heap and the remaining memory heap.

The `-h`, `-v`, and `-c` options only work on the default memory zone. To perform these actions on other zones you must specify the other zone using the `-z` option.

### Syntax

```
dynamic [-h] [-v] [-c] [-r] [-p]  
        [-extend byte_count] [-z heap_address]
```

### Arguments

None.

### Options

**-c**

Perform a consistency check on the heap.

**-h**

Display the headers of the blocks in the heap.

**-p**

Display dynamic memory statistics on a per process basis.

**-r**

Repair a broken heap by flooding free blocks with `DYN$K_FLOOD_FREE` if and only if they have been corrupted. Repairing broken heaps is dangerous at best, as it is masking underlying errors. This flag takes effect only if a consistency check is being done.

**-v**

Perform a validation test on the heap.

**-extend *byte\_count***

Extend the default memory zone by the byte count at the expense of the main memory zone. This command assumes that these two zones are physically adjacent.

**-z *heap\_address***

Perform the dynamic command on the heap specified by `heap_address`.

## dynamic

### Examples

```
>>> dynamic
zone      zone      used   used   free   free   utili-  high
address   size      blocks bytes  blocks bytes  zation  water
-----
00097740 1048576   389    358944 17     689664 34 %    371872
001D2B80 14805504 1       32      1     14805504 0 %     0

>>>dynamic -cv -z 97740
zone      zone      used   used   free   free   utili-  high
address   size      blocks bytes  blocks bytes  zation  water
-----
00097740 1048576   398    359520 17     689088 34 %    371872

>>>dynamic -h
zone      zone      used   used   free   free   utili-  high
address   size      blocks bytes  blocks bytes  zation  water
-----
00097740 1048576   392    359136 17     689472 34 %    389280
a 00097740 000E1600_001E0600 000E1608_001BF628 00000000 00097740 32
f 000E1600 0017E600_00097740 00189E68_00097748 FFFFFFFF 000E1600 643072
a 0017E600 001823C0_000E1600 001BF448_001B0D6C 00000023 0017E600 15808
.
.
.
>>>
```

### Command References

alloc, free

---

## echo — Echo the command line.

Echoes the text entered on the command line. The echo command separates arguments in the line to be echoed by blanks. The end of the echoed line is signified by a new line on the standard output.

Whenever specifying pipes or I/O redirection, be explicit by enclosing the text within single quotes.

### Syntax

```
echo [-n] args..
```

### Arguments

*args...*

Specifies any arbitrary set(s) of character strings.

### Options

**-n**

Suppress newlines from output.

### Examples

```
>>>echo this is a test.
this is a test.
>>>echo -n this is a test.
this is a test.cs>
>>>echo 'this is a test' > foo
>>>cat foo
this is a test
>>>echo > foo 'this is the simplest way
_to create a long file. All characters will be echoed
_file foo until the closing single quote.'
>>>cat foo
this is the simplest way
to create a long file. All characters will be echoed
file foo until the closing single quote.
>>>
```

### Command References

cat



## edit

---

### edit — Invoke the console BASIC-like line editor on a file.

'edit' is a console editor which behaves much like a BASIC line editor. With 'edit' lines of a file may be added, inserted, or deleted. The editor has the following set of subcommands: HELP, LIST, RENUMBER, EXIT, and QUIT.

To invoke the editor simply 'edit filename', where filename is an existing file.

The editor may be used to modify the user powerup script, 'nvram', or any other user created file, such as 'foo' in the examples below. Note 'nvram' is a special script which is always invoked during the powerup sequence. Hence, any actions which the user wishes to execute on any powerup, can be saved in the non-volatile memory using the 'nvram' script file.

### Syntax

```
edit  file
      [Subcommands: HELP, LIST, RENUMBER, EXIT or CTL/Z, QUIT]
      [nn : Delete line number nn.]
      [nn text : Add or overwrite line nn with text.]
```

### Arguments

#### *file*

Specifies the name of the file to be edited. Note the file must already exist.

### Options

#### **HELP**

Display the brief help file.

#### **LIST**

List the current file prefixed with line numbers.

#### **RENUMBER**

Renumber the lines of the file in increments of 10.

#### **EXIT**

Leave the editor and close the file saving all changes.

#### **QUIT**

Leave the editor and close the file without saving changes.

#### **CTL/Z**

Acts the same as EXIT, leave and save all changes.

#### *nn*

Delete line number *nn*.

#### *nn text*

Add or overwrite line *nn* with *text*.

## Examples

The following is an example showing how to create and modify a script 'foo' using edit.

```
>>>echo > foo 'this is a test'      # Create a sample file.
>>>cat foo
this is a test
>>>edit foo                        # Edit the newly created file.
editing 'foo'
15 bytes read in
*help
Think "BASIC line editor", and see if that'll do the trick
*list
 10 this is a test
*20 of the console BASIC-like line editor
*30 This editor supports HELP, LIST, RENUMBER, EXIT, and QUIT.
*list
 10 this is a test
 20 of the console BASIC-like line editor
 30 This editor supports HELP, LIST, RENUMBER, EXIT, and QUIT.
*10 This is a test of the console BASIC-like line editor.
*20
*list
 10 This is a test of the console BASIC-like line editor.
 30 This editor supports HELP, LIST, RENUMBER, EXIT, and QUIT.
*15 It may be used to create scripts files at the console.
*list
 10 This is a test of the console BASIC-like line editor.
 15 It may be used to create scripts files at the console.
 30 This editor supports HELP, LIST, RENUMBER, EXIT, and QUIT.
*renumber
*list
 10 This is a test of the console BASIC-like line editor.
 20 It may be used to create scripts files at the console.
 30 This editor supports HELP, LIST, RENUMBER, EXIT, and QUIT.
*exit
168 bytes written out
>>>cat foo                        # Note EXIT saves changes.
This is a test of the console BASIC-like line editor.
It may be used to create scripts files at the console.
This editor supports HELP, LIST, RENUMBER, EXIT, and QUIT.
>>>edit foo
editing 'foo'
168 bytes read in
*20
*list
 10 This is a test of the console BASIC-like line editor.
 30 This editor supports HELP, LIST, RENUMBER, EXIT, and QUIT.
*quit
>>>cat foo                        # Note QUIT does not save changes.
This is a test of the console BASIC-like line editor.
It may be used to create scripts files at the console.
This editor supports HELP, LIST, RENUMBER, EXIT, and QUIT.
>>>
```

The next example shows how do modify the non-volatile user-defined power-up script, 'nvram'.

## edit

```
>>>edit nvram                                # Modify user powerup script, nvram.
editing 'nvram'
0 bytes read in
*10 set boot_dev eza0
*20 set boot_osflags 0,0
37 bytes written out
>>>nvram                                       # Execute the silent script, nvram.
>>>edit nvram
editing 'nvram'
37 bytes read in
*15 show boot_dev
*25 show boot_osflags
*list
  10 set boot_dev eza0
  15 show boot_dev
  20 set boot_osflags 0,0
  25 show boot_osflags
*exit
69 bytes written out
>>>cat nvram                                  # List the modified file.
set boot_dev eza0
show boot_dev
set boot_osflags 0,0
show boot_osflags
>>>nvram                                       # Execute nvram, note the SHOWs.
boot_dev          eza0
boot_osflags      0,0
>>>
#
# Reset system, note nvram execution.
#
Cobra powerup script start
boot_dev          eza0
boot_osflags      0,0
Cobra powerup script end

Cobra/Laser (COBRA) console X1.3-1505, built on Feb 13 1992 at 01:28:52
```

## Command References

cat, echo

---

## eval — Evaluate a postfix expression.

Evaluates a specified arithmetic expression and displays the results. You can specify standard operations, such as addition, subtraction, and bitwise operators. Eval can also substitute environment variables. Internally, the evaluator keeps a 16 entry value stack which bounds the complexity of the expression.

All arithmetic operations are done with unsigned integers. Boolean operators (less than, greater than, etc.) always return 0 or 1. Spaces must be used to separate tokens in an expression.

You must place certain operators in quotes to protect them from the shell.

The following operators are supported:

```
+ addition
- subtraction
* multiplication
/ unsigned division
% modulus
& bitwise and
| bitwise inclusive or
^ bitwise exclusive or
< unsigned less than
<= unsigned less than or equal
== equal
!= not equal
>= unsigned greater than or equal
> unsigned greater than
~ one's complement
if conditional operator
ent replicate top entry on value stack
del pop off top entry on value stack
'=' assign a value to an environment value
srand random number with a seed
rnd random number
<< shift logical left
>> shift logical right
~ ones complement
```

### Syntax

```
eval [-{ib,io,id,ix}] [-{b|d|o|x}] postfix_expression
```

### Arguments

#### *postfix\_expression*

Specifies the postfix expression to be evaluated.

### Options

#### **-ib**

Set default input radix to binary.

#### **-id**

Set default input radix to decimal, the default.

#### **-io**

Set default input radix to octal.

#### **-ix**

Set default input radix to hexadecimal.

#### **-b**

Print output in binary.

## eval

**-d**  
Print output in decimal, the default.

**-o**  
Print output in octal.

**-x**  
Print output in hexadecimal.

### Examples

```
>>>eval 5 6 '*'
30
>>>eval -bodx '3 4 *'
1100 14 12 C
>>>eval -x '0xaa 0x55 |'
FF
>>>cat bswap
for aa; do
set bb 0x$aa
eval -x '$bb 0xff & 24 << $bb 0xff00 & 8 << $bb 0xff0000 & 8 >> $bb 0xff000000'
done
>>>bswap 01234567 89abcdef
67452301
EFCDAB89
>>>
```

### Command References

None

---

## examine — Display data at a specified address.

The EXAMINE command displays the contents of an address that you specify: either a memory location, a register, a device, or a file.

After initialization, if you have not specified a data address or size, the default address space is physical memory, the default data size is a quadword, and the default address is zero.

You specify an address or "device" by concatenating the device name with the address, for example, PMEM:0, and by specifying the size of the data to be displayed.

If you do not specify an address, the system displays the data at an address that is based on the current address and data size (the last previously specified address and data size).

If you specify a conflicting device, address, or data size, the console ignores the command and issues an error response.

The display line consists of the device name, the hexadecimal address (or offset within the device), and the examined data, also in hexadecimal.

EXAMINE uses the same options as DEPOSIT. Additionally, the EXAMINE command supports instruction decoding, the -d option, which disassembles instructions beginning at the current address.

### Syntax

```
examine  [-(b,w,l,q,o,h,d)] [-(physical,virtual,gpr,fpr,ipr)]
         [-n count] [-s step]
         [device:]address
```

### Arguments

#### [*device:*]

The optional "device" name (or address space) selects the device to access (see DEPOSIT).

#### *address*

The address specifies the first location to examine within the current device. The address can be any legal address specifier (see DEPOSIT).

### Options

#### -b

The data size is byte.

#### -w

The data size is word.

#### -l

The data size is longword.

#### -q

The data size is quadword.

## examine

**-o**

The data size is octaword.

**-h**

The data size is hexaword.

**-d**

The data displayed is the decoded macro instruction. Alpha instruction decode (-d) does not recognize machine-specific PAL instructions.

**-physical**

The address space is physical memory.

**-virtual**

The address space is virtual memory.

**-gpr**

The address space is general purpose registers.

**-fpr**

The address space is floating point registers.

**-ipr**

The address space is internal processor registers.

**-n *count***

Specifies the number of consecutive locations to examine.

**-s *step***

Specifies the address increment size, val (hex). Normally this defaults to the data size, but is overridden by the presence of this option. This option is not "sticky".

## Examples

```

>>>e r0                                ! Examine R0 by symbolic address.
gpr: 0 ( R0) 0000000000000002
>>>e -g 0                                ! Examine R0 by address space.
gpr: 0 ( R0) 0000000000000002
>>>e gpr:0                               ! Examine R0 by device name.
gpr: 0 ( R0) 0000000000000002
>>>examine pc                             ! Examine the PC.
gpr: 0000000F ( PC) FFFFFFFC
>>>examine sp                             ! Examine the SP.
gpr: 0000000E ( SP) 00000200
>>>examine ps1                            ! Examine the PSL.
  CM TP FPD IS CURMOD PRVMOD IPL DV FU IV T N Z V C
PSL 00000000 0 0 0 0 KERNEL KERNEL 00 0 0 0 0 0 0 0
>>>examine -n 5 R7                        ! Examine R7 through R12.
gpr: 00000007 ( R7) 00000000
gpr: 00000008 ( R8) 00000000
gpr: 00000009 ( R9) 801D9000
gpr: 0000000A ( R10) 00000000
gpr: 0000000B ( R11) 00000000
gpr: 0000000C ( AP) 00000000
>>>examine ipr:11                         ! Examine the SCBB, IPR 17.
ipr: 00000011 ( SCBB) 2004A000
>>>examine scbb                          ! Examine the SCBB using symbolic name.
ipr: 00000011 ( SCBB) 2004A000
>>>examine pmem:0                         ! Examine physical address 0.
pmem: 00000000 00000000
>>>examine -d 40000                       ! Examine with instruction decode.
pmem: 00040000 11 BRB 20040019
>>>examine -d -n 5 40019                  ! Disassemble from branch.
pmem: 00040019 D0 MOVL I^#20140000,@#20140000
pmem: 00040024 D2 MCOML @#20140030,@#20140502
pmem: 0004002F D2 MCOML S^#0E,@#20140030
pmem: 00040036 7D MOVQ R0,@#201404B2
pmem: 0004003D D0 MOVL I^#201404B2,R1
pmem: 00040044 DB MFPR S^#2A,B^44(R1)
>>>examine                                ! Look at next instruction.
pmem: 20040048 DB MFPR S^#2B,B^48(R1)
>>>

```

## Command References

deposit, hd



---

## exer — Exercise one or more devices.

Exercise one or more devices by performing specified read, write, and compare operations. Optionally, report performance statistics.

A 'read' operation reads from a device that you specify into a buffer. A 'write' operation writes from a buffer to a device that you specify. A 'compare' operation writes from two devices that you specify into two buffers and compares the contents of the buffers.

The `exer` command uses two buffers, 'buffer1' and 'buffer2', to carry out the operations. A read or write operation can be performed using either buffer. A compare operation uses both buffers.

You can tailor the behavior of `exer` by using options to specify the following:

1. an address range to test within the test device(s),
2. the packet size, also known as the IO size, which is the number of bytes read or written in one IO operation,
3. the number of passes to run,
4. how many seconds to run for,
5. a sequence of individual operations performed on the test device(s) The qualifier used to specify this is called the action string qualifier.

### Syntax

```
exer [-sb start_block] [-eb end_block] [-p pass_count]
      [-l blocks] [-bs block_size] [-bc block_per_io]
      [-d1 buf1_string] [-d2 buf2_string] [-a action_string]
      [-sec seconds] [-m] [-v]
      device_name...
```

### Arguments

***device\_name...***

Specifies the name(s) of the device(s) or filestream(s) to be exercised.

### Options

***-sb start\_block***

Specifies the starting block number (hex) within filestream. The default is 0.

***-eb end\_block***

Specifies the ending block number (hex) within filestream. The default is 0.

***-p pass\_count***

Specifies the number of passes to run the exerciser. If 0, then run forever or until Ctrl-C. The default is 1.

***-l blocks***

Specifies the number of blocks (hex) to exercise. `l` has precedence over `eb`. If only reading, then specifying neither `l` nor `eb` defaults to read till eof. If writing, and neither `l` nor `eb` are specified then `exer` will write for the size of device. The default is 1.

**-bs *block\_size***

Specifies the block size (hex) in bytes. The default is 200 (hex).

**-bc *block\_per\_io***

Specifies the number of blocks (hex) per I/O. On devices without length (tape) use the specified packet size or default to 2048. The maximum block size allowed with variable length block reads is 2048 bytes. The default is 1.

**-d1 *buf1\_string***

string arg for eval to gen buffer1 data pattern from. buffer1 is initialized only once and that is before any IO occurs. default = all bytes set to hex 5A's

**-d2 *buf2\_string***

string arg for eval to gen buffer2 data pattern from. buffer2 is initialized only once and that is before any IO occurs. default = all bytes set to hex 5A's

**-a *action\_string***

Specifies an exerciser 'action string', which determines the sequence of reads, writes, and compares to various buffers. The default action string is '?r'. The action string characters are:

- r - Read into buffer1.
- w - Write from buffer1.
- R - Read into buffer2.
- W - Write from buffer2.
- n - Write without lock from buffer1.
- N - Write without lock from buffer2.
- c - Compare buffer1 with buffer2.
- - - Seek to file offset prior to last read or write.
- ? - Seek to a random block offset within the specified range of blocks. exer calls the program, random, to 'deal' each of a set of numbers once. exer chooses a set which is a power of two and is greater than or equal to the block range. Each call to random results in a number which is then mapped to the set of numbers that are in the block range and exer seeks to that location in the filestream. Since exer starts with the same random number seed, the set of random numbers generated will always be over the same set of block range numbers.

**-sec *seconds***

Specifies to terminate the exercise after the number of seconds have elapsed. By default the exerciser continues until the specified number of blocks or passcount are processed.

**-m**

Specifies metrics mode. At the end of the exerciser a total throughput line is displayed.

**-v**

Specifies verbose mode, data read is also written to stdout. This is not applicable on writes or compares. The default is verbose mode off.

## exer

### Examples

```
>>>exer du*.* dk*.* -p 0 -secs 36000
```

Read all DSSI and SCSI type disks for the entire length of each disk. Repeat this until 36000 seconds, 10 hours, have elapsed. All disks will be read concurrently. Each block read will occur at a random block number on each disk.

```
>>>exer -l 2 duc0
```

Read block numbers 0 and 1 from device duc0.

```
>>>exer -sb 1 -eb 3 -bc 4 -a 'w' -d1 '0x5a' duc0
```

Write hex 5a's to every byte of blocks 1, 2, and 3. The packet size is bc \* bs, 4 \* 512, 2048 for all writes.

```
>>>ls -l du*.* dk*.*
d**.* no such file
r--- dk 0/0 0 dka0.0.0.0.0
>>>exer dk*.* -bc 10 -sec 20 -m -a 'r'
dka0.0.0.0.0 exer completed

packet size I/Os bytes read bytes written I/Os /sec bytes/sec elapsed idle
      8192 3325 27238400 0 166 1360288 20 19
>>>exer -eb 64 -bc 4 -a '?w-Rc' duc0
```

A destructive write test over block numbers 0 thru 100 on disk duc0. The packet size is 2048 bytes. The action string specifies the following sequence of operations:

1. Set the current block address to a random block number on the disk between 0 and 97. A four block packet starting at block numbers 98, 99, or 100 would access blocks beyond the end of the length to be processed so 97 is the largest possible starting block address of a packet.
2. Write a packet of hex 5a's from buffer1 to the current block address.
3. Set the current block address to what it was just prior to the previous write operation.
4. From the current block address read a packet into buffer2.
5. Compare buffer1 with buffer2 and report any discrepancies.
6. Repeat steps 1 thru 5 until enough packets have been written to satisfy the length requirement of 101 blocks.

```
>>>exer -a '?r-w-Rc' duc0
```

A non-destructive write test with packet sizes of 512 bytes. The action string specifies the following sequence of operations:

1. Set the current block address to a random block number on the disk.
2. From the current block address on the disk, read a packet into buffer1.
3. Set the current block address to the device address where it was just before the previous read operation occurred.
4. Write a packet of hex 5a's from buffer1 to the current block address.
5. Set the current block address to what it was just prior to the previous write operation.
6. From the current block address on the disk, read a packet into buffer2.
7. Compare buffer1 with buffer2 and report any discrepancies.

8. Repeat the above steps until each block on the disk has been written once and read twice.

```
>>>set myd 0
>>>exer -bs 1 -bc a -l a -a 'w' -d1 'myd myd ~ =' foo
>>>clear myd
>>>hd foo -l a
00000000 ff 00 ff 00 ff 00 ff 00 ff 00 .....
```

Use an environment variable, myd, as a counter. Write 10 bytes of the pattern ff 00 ff 00... to RAM disk file foo. A packet size of 10 bytes is used. Since the length specified is also 10 bytes then only one write occurs. Delete the environment variable, myd. The hd, hex dump of foo shows the contents of foo after exer is run.

```
>>>set myd 0
>>>exer -bs 1 -bc a -l a -a 'w' -d1 'myd myd 1 + =' foo
>>>hd foo -l a
00000000 01 02 03 04 05 06 07 08 09 0a .....
```

Write a pattern of 01 02 03 ... 0a to file foo.

```
>>>set myd 0
>>>exer -bs 1 -bc 4 -l a -a 'w' -d1 'myd myd 1 + =' foo -m
foo exer completed

packet      IOs      bytes read  bytes written  IOs      elapsed idle
size       /sec    /sec      /sec          /sec    seconds  secs
   4         3         0           10    3001    10001     0    0
>>>hd foo
00000000 01 02 03 04 01 02 03 04 01 02 .....
```

```
>>>show myd
myd 4

>>>echo '0123456789abcdefghijklmnopqrstAB' -n >foo3
>>>exer -bs 1 -v -m foo3
b2lkfmp8jatsnAlgri54B69o3qdc7eh0foo3 exer completed

packet      IOs      bytes read  bytes written  IOs      elapsed idle
size       /sec    /sec      /sec          /sec    seconds  secs
   1         32         32           0    5333    5333     0    0
```

## Command References

exer\_read, exer\_write, memexer, netexer

## Description

Exercise one or more devices. As described in the preceding overview section, exer uses two buffers, buffer1 and buffer2. Buffer1 and buffer2 are in main memory in the 'memzone' heap.

Both buffer1 and buffer2 are initialized to a data pattern before any IO operations occur. These buffers never reinitialized, even after completing one or more passes. The data patterns that the buffers are initialized with are either a hex 5A in every byte of each buffer or it is specified via the string arguments to the optional data pattern qualifiers, -d1, -d2

The d1, d2 qualifiers use a postfix string argument to initialize a buffer's contents as follows. For each byte in the specified buffer, starting with the first byte, this postfix string is passed to the eval command which returns a byte value which is then written to the specified buffer.

Several exer qualifiers are used to specify the amount of device data to be processed. The qualifiers -sb, -eb, -l, -bs, and -bc specify, respectively: starting block, ending block, number of blocks, block size in bytes, and the number of blocks in a packet, where a packet is the amount of data transferred in one IO operation.

Reading, writing, comparing buffers, and other operations can be specified to occur in various combinations and sequences. These operations are specified by a string of one-character command codes known as the 'action string'. The action string is specified as an argument to the action string qualifier, -a.

Each command code character in the action string is processed in a sequence from left to right. Each time that exer completes all of the operations specified by the action string, exer will reduce the remaining amount of device data to be processed by the size of the last packet processed by the action string. The action string is repeatedly processed until the specified amount of device data has been processed.

Lower-case action string characters, 'rwn', specify operations that involve buffer1. Upper-case action string characters, RWN, specify operations that involve buffer2. The action string character, 'c', involves both buffer1 and buffers. The action string characters, '-?', do not involve either buffer1 or buffer2.

A random number generator can be used to seek to varying device locations before performing either a read or write operation. Randomization is achieved by calling the function, 'random', which uses a linear congruential generator (LCG) to generate the numbers. This algorithm isn't truly random, but it comes closest to meeting the needs of exer. Each time that 'random' is called, it returns a number from a specified range. If the range of numbers is a power of two, then each subsequent call to 'random', is guaranteed to return a different number from the range until all possible numbers within the range have been chosen. If the range of numbers isn't a power of two, then 'random', is used with an upper bounds that is greater than the actual range size but is a power of two. Then a modulus operation with the range size is done to the number that 'random' returns, thereby ensuring that a random number is generated within the random range size.

The total number of bytes read or written on each pass of the exerciser is specified by the length in blocks or the starting/ending block address option arguments. If neither the ending address nor the length options are specified, then on each pass the number of bytes processed could vary depending on whether or not the filestream is being written to or just being read. If the filestream is not being written to by exer, then exer will read until EOF is reached. If exer will be writing to the file (as specified in the action string), then the number of bytes processed per pass is equal to the allocation size of the file which is usually larger than the length of the file for RAM disk files, but equal to the length for disk devices.

Note that disk device I/O will fail, if the blocksize is not equal to 1 or a multiple of 512. Partial block read/writes are not supported so a length which is not a multiple of the blocksize will result in no errors, but the last partial block I/O of data won't occur.

Any combination of writing, reading, or comparing the buffer1 and buffer2 can be executed in the sequence as specified in the action string. Depending on the option arguments, one or two of these three operations (read/write/compare) may be omitted without affecting the execution of the other operations.

The exer command will return an error code immediately after a read, write, or compare error, if the 'd\_harderr' environment variable is set to 'halt'. When an error occurs and 'continue' or 'loop on error' is specified, then subsequent operations specified by the action string qualifier will occur except for compares. For instance, if a read error occurs, a subsequent compare operations will be skipped since a read failure preceding a compare operation guarantees that the

compare will fail. If subsequent block I/O's succeed, then compares of those blocks will occur. When `exer` terminates because of completing all passes or by operator termination, then the status returned will be that of the last failed write, read or compare operation, regardless of subsequent successful IO's.

**Source File**`exer.c`

## exer\_read

---

### exer\_read — Use exer to read data from 1 or more devices

Seek to a random block number on the device then read a packet of 2048 bytes. Repeat this until one of the following conditions occur:

1) All blocks on the device(s) have been read for passcount of d\_passes. 2) The exer process has been killed via ^C or by the kill command. 3) The specified time has elapsed.

Nothing is displayed unless an error occurs.

#### Syntax

```
exer_read [-sec seconds] [device_name device_name...]
```

#### Arguments

##### **device\_name**

One or more device names. Defaults to du\*.\* dk\*.\*, all DSSI and SCSI disks that are online.

#### Options

##### **-sec seconds**

Number of seconds to run. Defaults to length of device times number of number of passcounts specified by the d\_passes environment variable.

#### Examples

```
>>>exer_read #Run on all, if any, disks that are online.
```

#### Source File

```
exer_read
```

---

## exer\_write — Use exer to test the writing of data to one or more devices

Use `exer` to run a non-destructive write test on one or more devices. The data is read and written in packets of 2048 bytes.

Seek to a random block on the device then read a 2048 byte packet of data then write that same data back to the same location on the device. Then read the written data back and compare it to the data originally read and display any discrepancies.

This is repeated until one of the following conditions occurs:

1) All blocks on the device(s) have been read for the number of passes specified by the `d_passes` environment variable. 2) The `exer` process has been killed via `^C` or by the `kill` command. 3) The specified time has elapsed.

Nothing is displayed unless an error occurs.

### Syntax

```
exer_write [-sec seconds] [device_name device_name...]
```

### Arguments

**device\_name**

one or more device names. Defaults to `du*.* dk*.*`, all DSSI and SCSI disks that are online.

### Options

**-sec seconds**

Number of seconds to run. Defaults to length of device times number of number of passcounts specified by the `d_passes` environment variable.

### Examples

```
>>>exer_write #Run on all, if any, disks that are online.
```

### Source File

```
exer_write
```



## exit

---

### exit — Exit the current shell.

Exit the current shell with a status or return the status of the last command.

#### Syntax

```
exit exit_value
```

#### Arguments

***exit\_value***

Specifies the status code to be returned by the shell.

#### Options

**None.**

#### Examples

```
>>>exit # exits with status of previous command
>>>exit 0 # exits with success
>>>test || exit # runs test and exits if an error
```

#### Command References

None

---

## fbus\_diag — Start a diagnostic on a Futurebus+ node.

The `fbus_diag` command is used to start execution of a diagnostic test script on a specific Futurebus+ node. Process options and command line arguments are used to specify the specific Test or Test Script to be executed as well as the target Futurebus+ node for this command.

This command will utilize the Futurebus+ standard Test CSR interface to initiate commands on specific Futurebus+ nodes within the Cobra/Laser system. Once the command has been sent to the Futurebus+ node the routine will wait for test completion and report the results to the console. If an error is reported by the node the diagnostic will issue a `dump_buffer` command to gain any available extended information which will also be reported to the console.

Those test categories which require a buffer pointer in the argument CSR will have a default buffer provided by this diagnostic if the user does not specify a buffer address.

### Syntax

```
fbus_diag  [-rb] [-p pass_count]
           [-st test_number] [-cat test_group] [-opt test_option]
           node_name [test_arg]
```

### Options

#### **-rb**

Randomly allocate from memzone on each pass with a block size of 4096.

#### **-p *pass\_count***

Specifies the number of times to run the test. If 0, the test runs forever. This overrides the value of environment variable, `pass_count`. In the absence of this option, `pass_count` is used. The default for `pass_count` is 1.

#### **-st *test\_number***

Specifies the test number to be run. The default is 0 which runs the default tests in the category.

#### **-cat *test\_group***

Specifies the test category to be executed. The default for this qualifier is the init category. The possible categories are as follows:

- Init : Initialization tests
- Extended : Extended tests
- System : System tests
- Manual : Manual tests
- *x* : Bit mask of the desired test categories

#### **-opt *test\_option***

Specify the Test Start CSR Option field bits to be set. The possible option bits are as follows:

- Loop\_error : Loop on Test if an error is detected
- Loop\_test : Loop on this test

## **fbus\_diag**

- Cont\_error : Continue if an error is detected
- x : Bit mask of the desired option bits

The default value for this qualifier is based on the current values in the global environment variables as follows:

- Loop\_test : 1 if D\_PASSES == 0 ; 0 otherwise
- Loop\_error : 1 if D\_HARDERR == "Loop" ; 0 otherwise
- Cont\_error : 1 if D\_HARDERR == "Continue" ; 0 otherwise

### **Examples**

```
>>>fbus_diag -p 100 -cat init -st ff fb1 00
>>>
>>>fbus_diag -p 100 -cat "init,system" -st 1 -opt "loop_error" fbb0 100
>>>
```

### **Command References**

fbus\_exer

---

**fbus\_sizer** — routine used for sizing the Fbus+

**Syntax**

COMMAND ARGUMENTS:

**Examples**

## find\_field

---

### find\_field — Extract a field from each input line and write it.

This command may be replaced by a non-standard grep option sometime in the future. Look for the new grep option if this cmd vaporizes.

Extract one specified field from each string in stdin. Output these fields as lines (with linefeeds) to stdout.

Maximum string length supported is 133 characters. Strings larger than the max supported size will be broken up into multiple strings before field extraction.

### Syntax

```
find_field field_number
```

### Arguments

#### *field\_number*

Specifies a number from 0 to the number of fields on a line minus 1. Larger numbers will not produce a field match.

### Options

None.

### Examples

```
>>>echo > foo 'this is a sample file
_to show how find_field works.
_>It should pick off the first word
_on each line.'
>>>cat foo
this is a sample file
to show how find_field works.
It should pick off the first word
on each line.
>>>find_field 0 <foo
this
to
It
on
>>>find_field 1 < foo
is
show
should
each
>>>
```

### Command References

grep

---

## free — Return an allocated block of memory to the heap.

Frees a block of memory that has been allocated from a heap. The block is returned to the appropriate heap.

### Syntax

```
free  address...
```

### Arguments

***address...***

Specifies an address (hex) or list of addresses of allocated block(s) to be returned to the heap.

### Options

**None.**

### Examples

```
>>>alloc 200
00FFFE00
>>>free fffe00
>>>free 'alloc 10' 'alloc 20' 'alloc 30'
>>>
```

### Command References

alloc, dynamic

## grep

---

### grep — Globally search for regular expressions and print matches.

Globally search for regular expressions and print any lines containing occurrences of the regular expression. Since `grep` is line oriented, it only works on ASCII files.

A regular expression is a shorthand way of specifying a 'wildcard' type of string comparison. Regular expressions may need to be enclosed by quotes to prevent metacharacters from being interpreted by the shell. `Grep` supports the following metacharacters:

```
^ matches the beginning of line.
$ matches end of line.
. matches any single character.
[ ] set of characters; [ABC] matches either 'A' or 'B' or 'C'; a
dash (other than first or last of the set) denotes a range
of characters: [A-Z] matches any upper case letter; if the
first character of the set is '^', then the sense of match
is reversed: [^0-9] matches any non-digit; several
characters need to be quoted with backslash (\) if they
occur in a set: '\', ']', '-', and '^'
* repeated matching; when placed after a pattern, indicates that
the pattern should match any number of times. For example,
'[a-z][0-9]*' matches a lower case letter followed by zero or
more digits.
+ repeated matching; when placed after a pattern, indicates that
the pattern should match one or more times '[0-9]+' matches
any non-empty sequence of digits.
? optional matching; indicates that the pattern can match zero or
one times. '[a-z][0-9]?' matches lower case letter alone or
followed by a single digit.
\ quote character; prevent the character which follows from
having special meaning.
```

### Syntax

```
grep [-c|i|n|v] [-f file] [expression] [file...]
```

### Arguments

#### **expression**

Specifies the target regular expression. If any regular expression metacharacters are present, the expression should be enclosed with quotes to avoid interpretation by the shell.

#### **file...**

Specifies the file(s) to be searched. If none are present, then stdin is searched.

### Options

#### **-c**

Print only the number of lines matched.

#### **-i**

Ignore case in the search. By default `grep` is case sensitive.

#### **-n**

Print the line numbers of the matching lines.

#### **-v**

Print all lines that don't contain the expression.

#### **-f file**

Specifies to take regular expressions from a file, instead of command.

## Examples

In the following example, the output of the ps command (stdin) is searched for lines containing 'eza0'.

```
>>>ps | grep eza0
0000001f 0019e220 3          2 ffffffff 0  mopcn_eza0 waiting on mop_eza0_cnw
00000019 0018e220 2          1 ffffffff 0  mopid_eza0 waiting on tqe
00000018 0018f900 3          3 ffffffff 0  mopdl_eza0 waiting on mop_eza0_dlw
00000015 0019c320 5          0 ffffffff 0   tx_eza0 waiting on eza0_isr_tx
00000013 001a2ce0 5          2 ffffffff 0   rx_eza0 waiting on eza0_isr_rx
```

In the next example, grep is used to search for all quadwords in a range of memory which are non-zero.

```
>>>alloc 20
00FFFE0
>>>deposit -q pmem:fffff0 0
>>>e -n 3 fffffe0
pmem:          FFFFE0 EFFFFFFEFFFFFFEFFF
pmem:          FFFF88 EFFFFFFEFFFFFFEFFF
pmem:          FFFFF0 0000000000000000
pmem:          FFFF88 EFFFFFFEFFFFFFEFFF
>>>e -n 3 fffffe0 | grep -v 0000000000000000
pmem:          FFFFE0 EFFFFFFEFFFFFFEFFF
pmem:          FFFF88 EFFFFFFEFFFFFFEFFF
pmem:          FFFF88 EFFFFFFEFFFFFFEFFF
>>>free fffffe0
>>>
```

## Command References

None



## hd

---

### hd — Dump the contents of a file in hexadecimal and ASCII.

Dump the contents of a file in hexadecimal and ASCII.

#### Syntax

```
hd [-s|e|l] file...
```

#### Arguments

**file...**

Specifies the file or files to be displayed.

#### Options

**-s**

Specifies the starting offset within the file.

**-e**

Specifies the ending offset within the file.

**-l**

Specifies the number of bytes to dump.

#### Examples

```
>>>cat fred
a script called fred.
>>>hd fred
00000000 61 20 73 63 72 69 70 74 20 63 61 6C 6C 65 64 20 a script called
00000010 66 72 65 64 0A fred.
>>>hd -l 16 foo
00000000 72 2d 2d 2d 20 20 20 6e 6c 20 30 30 30 30 30 30 r--- nl 000000
>>>hd -s 512 -e 522 foo
00000200 20 20 72 64 20 30 30 30 31 37 rd 00017
>>>hd -s 512 -l 10 foo
00000200 20 20 72 64 20 30 30 30 31 37 rd 00017
>>>
```

#### Command References

cat

---

## help or man — Display information about console commands.

Defines and shows the syntax for each command that you specify on the command line. If you do not specify a command, displays information about the help command and lists the commands for which additional information is available.

For each argument (or command) on the command line, help tries to find all topics that match that argument. For example, if there are topics on 'exit', 'examine' and 'entry', the command 'help ex' would generate the help text for both 'exit' and 'examine'.

Wildcards are supported, so that 'help \*' generates the expected behavior. Topics are treated as regular expressions that have the same rules as regular expressions for the shell. Help topics are case sensitive.

In describing command syntax, the following conventions are used.

- <ITEM> Angle brackets enclose a placeholder, for which the user must specify a value.
- [<ITEM>] Square brackets enclose optional parameters, qualifiers, or values.
- {a,b,c} Braces enclosing items separated by commas, imply mutually exclusive items. Choose any one of a, b, c.
- {a | b | c} Braces enclosing items separated by vertical bars, imply combinatorial items. Choose any combination of a, b, c.

The commands help and man can be used interchangeably.

### Syntax

```
help or man  [command...]
```

Command synopsis conventions:

- item* Implies a placeholder for user specified item.
- item...* Implies an item or list of items.
- [] Implies optional keyword or item.
- {a,b,c} Implies any one of a, b, c.
- {a|b|c} Implies any combination of a, b, c.

### Arguments

**command...**  
Specifies the command(s) or topic(s) for which help is requested.

### Options

None.

### Examples

In the following example, a list of topics for which help is available is requested.

```
>>>help          # List all topics.
```

In the next example, help is requested on all topics.

```
>>>help *        # List all topics and associated text.
```

In the next example, help is requested on all commands that begin with 'ex'.

## help or man

```
>>>help ex
```

In the next example, help is requested on the boot command.

```
>>>help boot
```

## Command References

None

---

**initialize** — Initializes the console, a device, or a processor.

Initializes the console, a device, or the specified processor. If a processor is not specified, the primary processor is initialized.

**Syntax**

```
initialize [-c] [-d device] [slot-id]
```

**Arguments****slot-id**

Specifies the processor "n" to be initialized.

**Options****-c**

Specifies the console device.

**-d device**

Specifies the device to be initialized.

**Examples**

```
cs>init 2
cs>initialize -d eza0
cs>
```

## io\_diag

---

### io\_diag — Test miscellaneous devices within the system

io\_diag uses information contained in various devices within the system to verify these devices. See the subtest descriptions for more details. Multiple io\_diags may be started and run concurrently. io\_diag may be used in conjunction with other exercisers to test the complete system.

#### Syntax

```
io_diag [-p n] [-t n]
```

#### Arguments

None.

#### Options

**-p**  
passcount; if = 0 then run forever or until ^C; default = 1

**-t**  
list of desired tests to execute; default is all tests

#### Examples

```
cs> io_test                ;execute all subtest once
cs> io_test -p 0 &        ;execute all subtest until stoped in the background
cs> io_test -t 1          ;execute only the scsi test for one pass
```

#### Description

#### Source File

io\_diag.c

---

## kill — Stop and delete a process.

Kill the processes listed on the command line. Processes are killed by making a kernel call with the process id (PID) as the argument.

### Syntax

```
kill [pid...]
```

### Arguments

***pid...***

Specifies the PID(s) as shown by the 'ps' command of the process(es) to be killed.

### Options

**None.**

### Examples

```
>>> memtest -p 0 &
>>> ps | grep memtest
000000f1 00217920 2      9357 ffffffff 0      memtest ready
>>> kill f1
>>> ps | grep memtest
```

### Command References

ps

## kill\_diags

---

### kill\_diags — Kill all currently executing diagnostic processes.

Kill the processes running specific diagnostics. These are: memtest, exer, nettest, and cbcc.

#### Syntax

```
kill_diags
```

#### Arguments

None.

#### Options

None.

#### Examples

```
>>>memexer 2 & # Start up two memtest processes.
>>>show_status
  ID      Program      Device  Pass Hard/Soft Bytes Written  Bytes Read
-----
00000001  idle      system    0   0   0           0           0
00000351  memtest   memory    0   0   0      37748736    37748736
00000352  memtest   memory    0   0   0      37748736    37748736
>>>kill_diags
>>>show_status
  ID      Program      Device  Pass Hard/Soft Bytes Written  Bytes Read
-----
00000001  idle      system    0   0   0           0           0
>>>
```

#### Source File

```
kill_diags
```

---

## line — Read a line from standard input and write it to standard output.

Copy one line (up to a new-line) from the standard input channel of the current process to the standard output channel of the current process. Always outputs at least a new-line.

It is often used in scripts to read from the user's terminal, or to read lines from a pipeline while in a for/while/until loop.

### Syntax

line

### Arguments

None.

### Options

None.

### Examples

Use interactively as follows:

```
>>>line
type a line of input followed by carriage return
type a line of input followed by carriage return
>>>line >nl
type a line of input followed by carriage return
>>>
```

Use w/in a script as follows:

```
>>>echo -n 'continue [Y, (N)]? '
>>>line <tt >tee:foo/nl
>>>if grep <foo '[yY]' >nl; then echo yes; else echo no; fi
>>>
```

### Command References

None



## ls

---

### ls — List files or inodes in file system.

List files or inodes in the system. Inodes are RAM disk files, open channels and some drivers. RAM disk files include script files, diagnostics and executable shell commands.

If no strings are present on the command line, then list all files or inodes in the system.

#### Syntax

```
ls [-l] [file...]
```

#### Arguments

**file...**

Specifies the file(s) or inode(s) to be listed.

#### Options

**-l**

Specifies to list in line format. Each file or inode is listed on a line with additional information. By default just file names are listed.

#### Examples

```
>>>ls examine
examine
>>>ls d*
d          date          debug1      debug2      decode      deposit
dg_pidlist dka0.0.0.0.0
dub0.0.0.1.0
dynamic
>>>
```

#### Command References

None

---

## memexer — Invoke Gray code memory exerciser.

Start the requested number of memory tests running in the background. Randomly allocate and test 2MB blocks of memory using all available memory. The pass count is 0 to run the started tests forever.

Nothing is displayed unless an error occurs.

### Syntax

```
memexer [number]
```

### Arguments

**number**  
the number of memory exercisers to start. Default 1.

### Options

None.

### Examples

```
>>>memexer 2 # Start two memtests running in the background. Test
>>>          # in blocks of 2MB across all availavble memory
>>>show_status
  ID      Program      Device      Pass  Hard/Soft Bytes Written  Bytes Read
-----
00000001      idle system          0    0    0           0           0
00000107      memtest memory        10    0    0      541065216      541065216
00000108      memtest memory        10    0    0      541065216      541065216
>>>
```

### Source File

memexer

## memexer\_mp

---

### memexer\_mp — Invoke Gray code memory exerciser on multiprocessor system.

Start the requested number of memory exerciser sets running in the background. A set is comprised of a memory test running on each processor in an incrementing longword count. For example, with two processors, one test will run on the even lws on the first processor and the other will run on second processor on the odd lws. This will work for any number of processors. 2MB blocks of memory are allocated for the test set to run on. The blocks do no change. The pass count is 0 to run the started tests forever.

Nothing is displayed unless an error occurs. Do not call memexer\_mp multiple times as a background process!!

#### Syntax

```
memexer_mp [number]
```

#### Arguments

**number**

The number of memory exerciser sets to start. Default 1.

#### Options

**None.**

#### Examples

```
>>>memexer_mp 2 # Start two memtest sets running in the background on
>>> # a block of 2MB.
>>>show_status
  ID      Program      Device      Pass  Hard/Soft Bytes Written  Bytes Read
-----
00000001      idle system          0    0    0           0           0
00000122      memtest memory         65    0    0      134217728     134217728
0000012c      memtest memory         72    0    0      148897792     148897792
0000014d      memtest memory         55    0    0      113246208     113246208
00000157      memtest memory         62    0    0      127926272     127926272
>>># ID      PCB      Pri CPU Time Affinity CPU Program      State
>>>#-----
>>>ps | grep memtest
00000157 00135680 2      56292 00000002 1      memtest running
0000014d 0013d160 2      55297 00000001 0      memtest ready
0000012c 00145b00 2      58450 00000002 1      memtest ready
00000122 0014d6e0 2      57318 00000001 0      memtest ready
>>>
```

#### Source File

```
memexer_mp
```

---

## memtest — Exercise a section of memory.

Memtest uses a gray codes to exercise a specified section of memory. The gray code algorithm used is:  $data = (x \gg 1)^x$ , where  $x$  is an incrementing value. Multiple memtests may be started and run concurrently. Memtest may be used in conjunction with other exercisers to test the complete system. Three passes are made on the memory under test.

The first pass writes alternating graycode inverse graycode to each octaword. This will cause one data bit to toggle between each longword write, and almost all bits will toggle between octawords. For example  $graycode(0)=0x00000000$  while  $inverse\ graycode(1)=0xFFFFFFFF$ .

The second pass reads each location verifies the data and writes the inverse of the data. The read-verify-write is done one longword at a time. This will cause: all data bits to be written as a one and zero; one data bit toggle between longword writes; many bits to toggle between octaword write; and will identify address shorts.

The third pass reads and verifies each location.

The -f "fast" option can be specified so the verify sections of the second and third loops is not performed. This will not catch address shorts but will stress memory with a higher throughput. The ECC/EDC logic will be used to detect failures.

### Syntax

```
memtest [-sa start_address] [-ea end_address] [-l length]
        [-bs block_size] [-i address_inc] [-p pass_count]
        [-d data_pattern] [-rs random_seed] [-rb]
        [-f] [-m] [-z] [-h] [-mb]
```

### Arguments

None.

### Options

**-sa start\_address**

Specifies the starting address for the test. The default is the first free space in memzone.

**-ea end\_address**

Specifies the ending address for the test. The default is the start address plus the length.

**-l length**

Specifies the length of section to test in bytes. The default is the block\_size, except with the -rb option which uses the zone size. -l has precedence over -ea.

**-bs block\_size**

Specifies the block (or packet) size (hex) in bytes The default is 8192 bytes. This is only used for the random block test. For all other tests the block size equals the length.

## memtest

### **-i *address\_inc***

Specifies the address increment value in longwords. This value will be used to increment the address through the memory to be tested. The default is 1 (longword). This is only implemented for the graycode test. An address increment of 2 tests every other longword. This option is useful for multiple CPUs testing the same physical memory.

### **-d *data\_pattern***

Specifies to use this data pattern for testing memory. This is only used for the march test. The default pattern is all 5's.

### **-p *pass\_count***

Specifies the number of times to execute the test. If 0, then run forever or until CTRL-C. The default is 1.

### **-rs *random\_seed***

Specifies the random seed (only used with -rb). The default is 0.

### **-rb**

Specifies to randomly allocate and test all of the specified memory address range. Allocations are done of `block_size`.

### **-f**

Specifies fast mode. If -f is specified, the data compare is omitted. Only ECC/EDC errors are detected.

### **-m**

Specifies to time the memory test. At the end of the test the elapsed time is displayed. By default the timer is off.

### **-z**

Specifies the test will use the specified memory address without an allocation. This bypasses all checking but will allow testing in addresses outside of the main memory heap. It will also allow unaligned testing. WARNING: This flag permits testing and corrupting ANY memory!

### **-h**

Specifies to allocate test memory from the firmware heap.

### **-mb**

Specifies to use memory barriers after each memory access. This flag is only used in -f graycode test. When set, an Alpha mb instruction will be done after every memory access. This will guarantee serial access to memory. Only performed on ALPHA based machines.

## Examples

The first example writes gray codes starting at 0x200000 (-sa) for 0x1000 bytes (-l).

```
>>>memtest -sa 200000 -l 1000
```

In the next example gray codes are written from 0x200000 for 0x1000 bytes, but data is not verified (-f).

```
>>>memtest -sa 200000 -l 1000 -f
```

In the next example a default block size of 8192 bytes is written from 0x300000 for 10 passes (-p).

```
>>>memtest -sa 300000 -p 10
```

In the next example gray codes are written to arbitrary 8192 byte blocks in memory without verification. After every read and write to memory an MB, memory barrier instruction, is executed (-mb).

```
>>>memtest -f -mb
```

In the next example gray codes are written from 0x200000 to 0x3ffff. Every block within this range is randomly allocated (-rb). With -rb memtest will not error if a block within the range can't be allocated.

```
>>>memtest -sa 200000 -ea 400000 -rb
```

In the next example the console heap (-h) is tested by randomly mallocing 0x100 byte blocks (-bs).

```
>>>memtest -h -rb -bs 100
```

In this example gray codes are written across all of memzone (all memory excluding the HWRPB, the PAL area, the console, and the console heap. It is run in the foreground until CTRL-C.

```
>>>memtest -rb -p 0
```

In the next example two memtest processes are started to run forever concurrently. The first memtest writes gray codes to every other longword (-i 2) from 0x300000 for 0x200000 bytes. The second memtest tests the same range, but writes to the alternating longwords. The memory allocation is done by the first memtest. These are run on separate processors to test cache coherency. The first memtest is run on processor 0 (&p0) and the second memtest is run on processor 1 (&p1). Both memtests are run in the background.

```
>>>memtest -sa 300000 -l 200000 -i 2 -p 0 &p0&
>>>memtest -sa 300004 -l 200000 -i 2 -p 0 -z &p1&
```

## Command References

memexer, memexer\_mp, test

---

## net — Access a network port and execute MOP maintenance functions.

Using the specified port, perform some maintenance operations. If no port is specified, the default port is used.

The net command performs basic MOP operations, such as, loopback, request IDs, and remote file loads. The net command also provides the means to observe the status of a network port. Specifically, the 'net -s' will display the current status of a port including the contents of the MOP counters. This is useful for monitoring port activities and trying to isolate network failures.

To get a port's Ethernet station address, use 'net -sa'.

### Syntax

```
net [-s] [-sa] [-ri] [-ic] [-id] [-l0] [-l1] [-rb] [-csr]
    [-els] [-kls] [-cm mode_string] [-da node_address]
    [-l file_name] [-lw wait_in_secs] [-sv mop_version]
    port_name
```

### Arguments

***port\_name***

Specifies the Ethernet port on which to operate.

### Options

**-s**

Display port status information including MOP counters.

**-sa**

Display the port's Ethernet station address.

**-ri**

Reinitialize the port drivers.

**-ic**

Initialize the MOP counters.

**-id**

Send a MOP Request ID to the specified node. Uses -da to specify the destination address.

**-l0**

Send an Ethernet loopback to the specified node. Uses -da to specify the destination address.

**-l1**

Do a MOP loopback requester.

**-rb**

Send a MOP V4 boot message to remote boot a node. Uses -da to specify the destination address.

**-csr**

Displays the values of the Ethernet port CSRs.

**-els**

Enable the extended DVT loop service.

**-kls**

Kill the extended DVT loop service.

**-cm *mode\_string***

Change the mode of the port device. The mode string may be any one of the following abbreviations.

- nm = Normal mode
- in = Internal Loopback
- ex = External Loopback
- nf = Normal Filter
- pr = Promiscious
- mc = Multicast
- ip = Internal Loopback and Promiscious
- fc = Force collisions
- nofc = Don't Force collisions
- df = Default

**-da *node\_address***

Specifies the destination address of a node to be used with -l0, -id, or -rb, options.

**-l *file\_name***

Attempt a MOP load of the file name.

**-lw *wait\_in\_secs***

Wait the number of seconds specified for the loop messages from the -l1 option to return. If the messages don't return in the time period, an error message is generated.

**-sv *mop\_version***

Set the preferred MOP version number for operations. Legitimate values are 3 or 4.

## Examples

Display the local Ethernet ports' station address.

```
>>>net -sa
-eza0: 08-00-2b-1d-02-91
>>>net -sa ezbo
-ezb0: 08-00-2b-1d-02-92
```

Display the eza0 port status, including the MOP counters.



## net

```
>>>net -s
DEVICE SPECIFIC:
  TI: 203 RI: 42237 RU: 4 ME: 0 TW: 0 RW: 0 BO: 0
  HF: 0 UF: 0 TN: 0 LE: 0 TO: 0 RWT: 39967 RHF: 39969 TC: 54
PORT INFO:
tx full: 0 tx index in: 10 tx index out: 10
rx index in: 11
MOP BLOCK:
  Network list size: 0
MOP COUNTERS:
Time since zeroed (Secs): 2815
TX:
  Bytes: 116588 Frames: 204
  Deferred: 2 One collision: 52 Multi collisions: 14
TX Failures:
  Excessive collisions: 0 Carrier check: 0 Short circuit: 0
  Open circuit: 0 Long frame: 0 Remote defer: 0
  Collision detect: 0
RX:
  Bytes: 116564 Frames: 194
  Multicast bytes: 13850637 Multicast frames: 42343
RX Failures:
  Block check: 0 Framing error: 0 Long frame: 0
  Unknown destination: 42343 Data overrun: 0 No system buffer: 22
  No user buffers: 0
>>>
```

## Command References

netexer, nettest

---

## nettest — Test the network ports using MOP loopback.

This is a network test. It can test the ez ports in internal loopback, external loopback or live network loopback mode.

Nettest contains the basic options to allow the user to run MOP loopback tests. It is assumed that nettest will be included in a script for most applications. Many environment variables can be set to customize nettest. These may be set from the console before nettest is started. Listed below are the environment variables, a brief description, and their default values.

Note: Each variable name is preceded by eza0\_ or ezb0\_ to specify the desired port.

Experienced users may also desire to change other network driver characteristics by modifying the port mode. Refer to the `-mode` option.

### Syntax

```
nettest [-f file] [-mode port_mode] [-p pass_count]
        [-sv mop_version] [-to loop_time] [-w wait_time]
        [port]
```

### Arguments

#### *port*

Specifies the Ethernet port on which to run the test.

### Options

#### **-f file**

Specifies the file containing the list of network station addresses to loop messages to. The default file name is `lp_nodes_eza0` for port `eza0`. The default file name is `lp_nodes_ezb0` for port `ezb0`. The files by default have their own station address.

#### **-mode port\_mode**

Specifies the mode to set the port adapter (TGEC). The default is 'ex' (external loopback). Allowed values are:

- `df` : default, use environment variable values
- `ex` : external loopback
- `in` : internal loopback
- `nm` : normal mode
- `nf` : normal filter
- `pr` : promiscious
- `mc` : multicast
- `ip` : internal loopback and promiscious
- `fc` : force collisions
- `nofc` : do not force collisions
- `nc` : do not change mode

## nettest

### **-p *pass\_count***

Specifies the number of times to run the test. If 0, then run forever. The default is 1. Note: This is the number of passes for the diagnostic. Each pass will send the number of loop messages as set by the environment variable, `eza*_loop_count`.

### **-sv *mop\_version***

Specifies which MOP version protocol to use. If 3, then MOP V3 (DECNET Phase IV) packet format is used. If 4, then MOP V4 (DECNET Phase V IEEE 802.3) format is used.

### **-to *loop\_time***

Specifies the time in seconds allowed for the loop messages to be returned. The default is 2 seconds.

### **-w *wait\_time***

Specifies the time in seconds to wait between passes of the test. The default is 0 (no delay). The network device can be very CPU intensive. This option will allow other processes to run. ENVIRONMENT VARIABLE(S):

### **`eza*_loop_count`**

Specifies the number (hex) of loop requests to send. The default is 0x3E8 loop packets.

### **`eza*_loop_inc`**

Specifies the number (hex) of bytes the message size is increased on successive messages. The default is 0xA bytes.

### **`eza*_loop_patt`**

Specifies the data pattern (hex) for the loop messages. The following are legitimate values.

- 0 : all zeroes
- 1 : all ones
- 2 : all fives
- 3 : all 0xAs
- 4 : incrementing data
- 5 : decrementing data
- ffffffff : all patterns

### **`loop_size`**

Specifies the size (hex) of the loop message. The default packet size is 0x2E.

## Examples

```
>>>nettest      - internal loopback test on port eza0
>>>nettest ez*  - internal loopback test on ports eza0/ezb0
>>>nettest -mode ex      - external loopback test on port eza0
>>>nettest -mode ex -w 10 - external loopback test on port eza0 wait 10
seconds between tests
>>>nettest -f foo -mode nm - normal mode loopback test on port eza0
using the list of nodes contained in the
file foo
```

## **Command References**

net, netexer

ps

---

## ps — Print process status and statistics.

The `ps` command displays the system state in the form of process status and statistics.

### Syntax

`ps`

### Arguments

None.

### Options

None.

### Examples

```
>>>ps
  ID      PCB      Pri CPU Time Affinity CPU  Program      State
-----
0000008f 0010e8a0 3          0 00000001 0      ps running
00000020 00110160 1          0 ffffffff 0      puc_poll waiting on tqe
0000001f 0013cb60 6          0 ffffffff 0      puc_receive waiting on puu_receive
0000001c 0013ed00 1          0 ffffffff 0      pub_poll waiting on tqe
0000001b 0014fc00 6          0 ffffffff 0      pub_receive waiting on puu_receive
0000001a 00111a20 3          0 00000001 0      sh ready
00000015 001176a0 2          0 ffffffff 0      mopcn_eza0 waiting on mop_eza0_cnw
00000014 00119140 2          0 ffffffff 0      mopid_eza0 waiting on tqe
00000013 0011ac20 2          0 ffffffff 0      mopdl_eza0 waiting on mop_eza0_dlw
00000012 0011f6a0 6          0 ffffffff 0      tx_eza0 waiting on eza0_isr_tx
00000011 00121140 6          0 ffffffff 0      rx_eza0 waiting on eza0_isr_rx
00000010 00122ac0 1          0 ffffffff 0      pua_poll waiting on tqe
0000000f 001244e0 6          0 ffffffff 0      pua_receive waiting on pua_receive
00000009 00147460 5          0 ffffffff 0      lad_poll waiting on tqe
00000008 00148f00 5          0 ffffffff 0      dup_poll waiting on tqe
00000007 0014a9a0 5          0 ffffffff 0      mscp_poll waiting on tqe
00000006 0014e1a0 5          0 00000001 0      entry_00 waiting on entry_00
00000004 001516e0 2          0 ffffffff 0      dead_eater waiting on dead_pcb
00000003 00153140 7          0 ffffffff 0      timer waiting on timer
00000002 00158740 6          0 ffffffff 0      tt_control waiting on tt_control
00000001 0005cfd8 0          0 00000001 0      idle ready
>>>
```

### Command References

`show_status`, `sa`, `sp`

---

## psc — Communicate with the power system controller via the IIC bus.

Provides a user interface to the Cobra power system controller. The console passes packets to the power system controller via the IIC bus.

### Syntax

```
psc [clear param] [set param] [show] [reset]
```

### Arguments

**clear *param***

Clear a parameter in the power system controller.

**set *param***

Set a parameter in the power system controller.

**show**

Show the internal status of the power system controller.

**reset**

Request the power system controller to reset the system.

### Options

None.

### Examples

```
>>>psc show
>>>
```

### Command References

None

rm

---

## rm — Remove files from the file system.

Remove the specified file or files from the file system. Any allocated memory is returned to the heap.

### Syntax

```
rm file...
```

### Arguments

**file...**

Specifies the file or files to be deleted.

### Options

None.

### Examples

```
>>>ls foo
foo
>>>rm foo
>>>ls foo
foo no such file
>>>
```

### Command References

cat, ls

---

## sa — Set process affinity.

Set affinity, 'sa', is used to change the affinity mask of a process. The process may then execute on any processors specified by the mask.

### Syntax

```
sa process_id affinity_mask
```

### Arguments

***process\_id***

Specifies the PID of the process to be modified.

***affinity\_mask***

Specifies the new affinity mask which indicates which processors the process may run on. Bits 0 and 1 of the mask correspond to processors 0 and 1, respectively.

### Options

None.

### Examples

```
>>>memtest -p 0 &
>>>ps | grep memtest
00000025 001a9700 2      23691 00000001 0      memtest ready
>>>sa 25 2
>>>ps | grep memtest
00000025 001a9700 2      125955 00000002 1      memtest running
>>>
```

### Command References

ps, sp



## semaphore

---

### semaphore — Show system semaphores.

Show all the semaphores known to the system by traversing the semaphore queue.

#### Syntax

semaphore

#### Arguments

None.

#### Options

None.

#### Examples

```
>>>semaphore
-----
Name                Value  Address  First Waiter
-----
      dyn_sync      00000001 00050378
      dyn_release  00000001 000503A0
      shell_iolock  00000001 0015D684
      exit_iolock   00000001 0015D770
      grep_iolock   00000001 0015DB20
      eval_iolock   00000001 0015DC0C
      chmod_iolock  00000001 0015DCF8
^c
>>>
```

#### Command References

None

---

## set — Set or modify the value of an environment variable.

Sets or modifies the value of an environment variable. Environment variables are used to pass configuration information between the console and the operating system.

### Syntax

```
set  envar value
     [-default] [-integer] [-string]
     where
     envar={auto_action,bootdef_dev,boot_file,boot_osflags,...}
```

### Arguments

***envar***

The environment variable to be assigned a new value. Refer to the list of commonly used environment variables below.

***value***

The value that is assigned to the environment variable. Either a numeric value or an ASCII string.

### Options

**-default**

Restores an environment variable to its default value.

**-integer**

Creates an environment variable as an integer.

**-string**

Creates an environment variable as a string. ENVIRONMENT VARIABLE(S):

**auto\_action**

Sets the console action following an error, halt, or power-up, to halt, boot, or restart. The default setting is halt.

**bootdef\_dev**

Sets the default device or device list from which the system attempts to boot. For systems which ship with factory installed software (FIS), the default device is preset at the factory to the device that contains FIS. For systems which do not ship with FIS, the default setting is null.

**boot\_file**

Sets the file name to be used when a bootstrap requires a file name. The default setting is null.

**boot\_osflags**

Sets additional parameters to be passed to system software. When using VMS software, these parameters are the system root number and boot flags. The default setting is 0,0.

## set

### **tta\*\_baud**

Sets the baud rate for the auxilliary serial port. Possible settings are 600, 1200, 2400, 4800, 9600, or 19,200. The default setting is 1200.

### **tta\*\_halts**

Specifies halt characters recognized on the console serial ports, tta0 and tta1. The value is an integer bitmap, where:

- bit 0 : Enables(1) or disables(0) CTRL-P to init from the console.
- bit 1 : Enables(1) or disables(0) CTRL-P halts from the operating system.
- bit 2 : Enables(1) or disables(0) BREAK halts from the operating system. Note, since tta1 is intended for modem support, this bit is ignored on tta1 (i.e. BREAKs are not permitted on the auxiliary port).

The default for tta0 is a 2, enabling CTRL-P halts from the operating system. The default for tta1 is a 0, disabling halts from the operating system.

## Examples

In the following example, the default device from which the system attempts to boot is set to eza0.

```
>>>set bootdef_dev eza0
```

In the next example, the system's default console action following error, halt, or power-up is set to boot.

```
>>>set auto_action boot
```

In the next example, the file name to be used when the system's boot requires a file name is set to vax\_4000.sys.

```
>>>set boot_file vax_4000.sys
```

In the next example, the system's default boot flags are set to 0,1.

```
>>>set boot_osflags 0,1
```

In the next example, the Baud rate of the auxillary console serial port, tta1, is set to 1200.

```
>>>set tta1_baud 1200
```

In the next example, the Baud rate of the auxillary console serial port, tta1, is set back to its default value, 9600.

```
>>>set -default tta1_baud
```

In the next example, an environment variable called foo is created and given a value of 5.

```
>>>set foo 5
```

## Command References

clear, set host, show

---

## set host — Connect the console to another server.

Connects the console to another server. Using the `-dup` option invokes the MSCP DUP server on the selected 'target' node. You can use the DUP protocol to examine and modify parameters of a DSSI device.

The `-task` option permits connection to the desired MSCP DUP service utility.

### Syntax

```
set host [-dup] [-task task_name] target
```

### Arguments

#### *target*

Specifies the target processor number or the target device for the service connection.

### Options

#### `-dup`

Specifies connection to an MSCP DUP server. The DUP service may be used to examine and modify parameters of a DSSI device.

#### `-task task_name`

Specifies which DUP service utility to invoke. Refer to example below for a list of utilities. Note that in the absence of this qualifier, a directory of utilities is displayed.

### Examples

The following example shows how to find DSSI disks in the system and then connect to the MSCP DUP server on a disk.

```
>>>show device du
dud0.0.0.3.0          R2YQYA$DIA0          RF72
>>>set host -dup dud0
starting DIRECT on pud0.0.0.3.0 (R2YQYA)

Copyright (C) 1990 Digital Equipment Corporation
PRFMON V1.0 D 2-NOV-1990 10:30:58
DKCOPY V1.0 D 2-NOV-1990 10:30:58
DRVEXR V2.0 D 2-NOV-1990 10:30:58
DRVTST V2.0 D 2-NOV-1990 10:30:58
HISTORY V1.1 D 2-NOV-1990 10:30:58
DIRECT V1.0 D 2-NOV-1990 10:30:58
ERASE V2.0 D 2-NOV-1990 10:30:58
VERIFY V1.0 D 2-NOV-1990 10:30:58
DKUTIL V1.0 D 2-NOV-1990 10:30:58
PARAMS V2.0 D 2-NOV-1990 10:30:58
Total of 10 programs.
Task?
>>>set host -dup -task params dud0
starting PARAMS on pud0.0.0.3.0 (R2YQYA)

Copyright (C) 1990 Digital Equipment Corporation

PARAMS> show allclass

Parameter      Current      Default      Type      Radix
-----
ALLCLASS              0              0      Byte      Dec      B

PARAMS> exit
Exiting...
>>>
```

**set host**

## **Command References**

cdp

---

## sh — Create a new shell process.

This routine implements most of the functionality of the Bourne shell. It is not the Bourne shell itself, but a reverse engineered programs.

### Syntax

```
sh  [-{x | v | d}] [arg...]
```

### Arguments

**arg**  
any text string terminated with whitespace.

### Options

**-v**  
print lines as they are read in

**-x**  
show commands just before they are executed

**-d**  
delete stdin when shell is done

**-l**  
trace lexical analyzer (show tokens as they are recognized)

**-r**  
trace parser (show rules as they fire)

**-p**  
trace execution engine (show routines called)

### Examples

```
cs> sh          # start a new shell
cs>            # the new shell's prompt

cs> sh -v <foo # execute command file "foo" and show lines as read in
cs> sh -x <foo # print out commands as they are executed and after
               # all substitutions have been performed.
```

## show

---

### show — Display an environment variable value or other information.

Displays the current value (or setting) for an environment variable that you specify.

Also displays other information about the system, according to the arguments that you enter on the command line. For example, you can display the system configuration by entering 'show config'.

#### Syntax

```
show    [{config,device,hwrpb,memory,pal,version,...}]
        [envvar]
        where:
        envvar={auto_action,bootdef_dev,boot_file,boot_osflags,...}
```

#### Arguments

##### **cluster**

Displays open virtual circuits.

##### **config**

Displays the current system configuration.

##### **device**

Displays devices and controllers in the system.

##### **error**

Displays error log information.

##### **fbus**

Displays Futurebus+ devices.

##### **fru**

Displays system FRU information.

##### **hwrpb**

Displays the Alpha HWRPB.

##### **map**

Displays system virtual memory map.

##### **memory**

Displays the memory module configuration.

##### **pal**

Displays the versions of VMS and OSF PALcode.

##### **version**

Displays the version of the console firmware.

##### **memory**

Displays the memory module configuration.

**envvar**

Displays the value of the environment variable specified. Refer to the list of commonly used environment variables below.

**Options**

**None.**

ENVIRONMENT VARIABLE(S):

**auto\_action**

Displays the console action following an error halt or power-up: either halt, boot, or restart.

**baud\_modem**

Displays the baud rate for the auxilliary console port.

**bootdef\_dev**

Displays the device or device list from which bootstrapping is attempted.

**boot\_file**

Displays the file name to be used when a bootstrap requires a file name.

**boot\_osflags**

Displays the additional parameters to be passed to system software.

**eza0\_rm\_boot, ezb0\_rm\_boot**

Displays the state of MOP remote boot enable. A 0 indicates enabled, a 1 indicates disabled.

**language**

Displays the language in which system software and layered products are displayed.

**password**

Displays whether a password is required to gain access to privileged console commands.

**Examples**

In the following example, the version of firmware on a system is displayed. The firmware version is X1.4-2129.

```
>>>show version
version                V3.0-1 Sep 20 1992 00:28:54
>>>
```

In the next example, the default system powerup action is displayed.

```
>>>show auto_action
boot
>>>
```

In the next example, a system's default boot device is displayed. The default boot device in the example is eza0.

```
>>>show bootdef_dev
eza0
>>>
```

In the next example, the baud rate for a system's auxillary console port is displayed. The baud rate in the example is 1200.



## **show**

```
>>>show baud_modem  
1200  
>>>
```

## **Command References**

clear, set, show cluster, show config, show device, show error show fru, show fbus,  
show hwrpb, show map, show memory

---

## show cluster — Display open virtual circuits.

Shows any open virtual circuits in the system on DSSI.

### Syntax

```
show cluster
```

### Arguments

None.

### Options

None.

### Examples

In the following example, a system's cluster is displayed.

```
>>>show cluster
      pud0.0.0.3.0          R2YQYA          RF72/RFXX T251
>>>
```

### Command References

show device

## show config

---

### show config — Display the current system configuration.

Displays the buses found on the system and the devices found on these buses.

You can use the information from show config to identify target devices for commands like 'boot' and 'exer', as well as to verify that the system sees all devices that are installed.

Currently executes the two powerup screens.

#### Syntax

```
show config
```

#### Arguments

None.

#### Options

None.

#### Examples

In the following example, a system's configuration is displayed.

```
>>>show config
      .
powerup screens...
      .
>>>
```

#### Command References

show cluster, show device, show memory



## show device

```
>>>show device e          # Show Ethernet devices.
eza0.0.0.6.0             EZA0          08-00-2B-1D-27-AA
ezb0.0.0.7.0             EZB0          08-00-2B-1D-27-AB
>>>show device eza0      # Show Ethernet port 0.
eza0.0.0.6.0             EZA0          08-00-2B-1D-27-AA
>>>show device du        # Show DSSI disks.
dud0.0.0.3.0             R2YQYA$DIA0   RF72
>>>show device *k*       # Show SCSI devices.
dkc0.0.0.2.0             DKC0          RZ57
mke0.0.0.4.0             MKE0          TLZ04
>>>show device dk        # Show SCSI disks.
dkc0.0.0.2.0             DKC0          RZ57
>>>show device mk        # Show SCSI tape drives.
mke0.0.0.4.0             MKE0          TLZ04
>>>
```

## Command References

show cluster, show config

---

## show error — Display error information from serial control bus EEROM.

Display's the serial control bus EEROM error log events stored in each module in the system.

### Syntax

```
show error [module...]
```

### Arguments

#### *module...*

Specifies the module for which the FRU information is displayed. This is an optional argument, if not supplied the FRU information for all modules in the system will be displayed. Module names are cpu0, cpu1, io, mem0, mem1, mem2, and mem3.

### Options

None.

### Examples

```
>>> show error          # Show error information for all system modules.
IO Module EEROM Event Log
Undefined
CPU1 Module EEROM Event Log
Module Not Present
CPU0 Module EEROM Event Log
Undefined
MEM0 Module EEROM Event Log
Module Not Present
MEM1 Module EEROM Event Log
Module Not Present
MEM2 Module EEROM Event Log
Module Not Present
MEM3 Module EEROM Event Log
Entry Offest RAM # Bit Mask Multi-Chip Event Type
  0 4200 0 1      0      2
  1 8200 0 1      0      2
  2 c200 0 1      0      2
>>>
>>> show error mem3    # Show error information for a memory module.
MEM3 Module EEROM Event Log
Entry Offest RAM # Bit Mask Multi-Chip Event Type
  0 4200 0 1      0      2
  1 8200 0 1      0      2
  2 c200 0 1      0      2
>>>
```

**show error**

## **Command References**

build, clear\_error, show error

---

## show fru — Display FRU information based on the serial control bus EEROM.

Displays the part number, serial number, hardware and software revision levels along with the SDD and TDD event counts for selected system modules.

### Syntax

```
show fru [module...]
```

### Arguments

#### *module...*

Specifies the module for which the FRU information is displayed. This is an optional argument, if not supplied the FRU information for all modules in the system will be displayed. Module names are `cpu0`, `cpu1`, `io`, `mem0`, `mem1`, `mem2`, and `mem3`.

### Options

None.

### Examples

```
>>> show fru          # Show FRU information for all system Modules
      Rev  Events logged
Slot Option Part#  Hw Sw Serial#  SDD TDD
 1 IO B2201-AA D5 1 AY42100001 00 00
 2
 3 CPU0 B2001-AA D5 1 AY24200012 00 00
 4
 5
 6
 7 MEM3 B2002-BA C1 0 GA32400321 03 00
>>>
>>> show fru cpu1    # Show FRU information for the CPU Module
      Rev  Events logged
Slot Option Part#  Hw Sw Serial#  SDD TDD
 3 CPU0 B2001-AA D5 1 AY24200012 00 00
>>>
```

### Command References

`build`, `clear_error`, `show error`



## show hwrpb

---

### show hwrpb — Display the address of the HWRPB.

Display the address of the Alpha HWRPB.

#### Syntax

```
show hwrpb
```

#### Arguments

None.

#### Options

None.

#### Examples

```
>>>show hwrpb  
HWRPB is at 2000  
>>>
```

---

## show map — Display the system virtual memory map.

Display the current system virtual memory map. This routine will not show pte that are hardwired into the chip (as in the case of EV4 pass 2).

### Syntax

```
show map
```

### Arguments

None.

### Options

None.

### Examples

```
>>>show map
pte 00001020 v FFFFC0902408000 p 00000000 V KR SR FR FW
pte 00001028 v FFFFC090240A000 p 00000000 V KR SR FR FW
pte 00001020 v FFFFC0902C08000 p 00000000 V KR SR FR FW
pte 00001028 v FFFFC0902C0A000 p 00000000 V KR SR FR FW
pte 00001020 v FFFFC0B02408000 p 00000000 V KR SR FR FW
pte 00001028 v FFFFC0B0240A000 p 00000000 V KR SR FR FW
pte 00001020 v FFFFC0B02C08000 p 00000000 V KR SR FR FW
pte 00001028 v FFFFC0B02C0A000 p 00000000 V KR SR FR FW
>>>
```

## show memory

---

### show memory — Show memory configuration.

Shows the main memory configuration on a board by board basis. Also reports the addresses of bad pages defined by the bitmap.

Additionally, shows the normally inaccessible areas of memory, such as, the PFN bitmap pages, the console memory pages, PALcode, and shared data structures such as the HWRPB.

#### Syntax

```
show memory
```

#### Arguments

None.

#### Options

None.

#### Examples

In the following example, a system's memory is displayed.

```
>>>show memory
  Module  Size  Base Addr  Intlv Mode  Intlv Unit
  -----  ---  -
  0      64Mb  00000000   1-Way      0
  1      Not Installed
  2      Not Installed
  3      Not Installed
Total Bad Pages 0
>>>
```

#### Command References

```
show config
```

---

## show\_status — Show the status of any currently executing diagnostics.

Reports one line of information per currently running diagnostic. The info includes error counts, passes completed and bytes read/written to the device being tested. The source of each line of info is an IOB, IO block.

### Syntax

```
show_status
```

### Arguments

None.

### Options

None.

### Examples

```
>>>memexer 2 &          # Start up two memtest processes.
>>>show_status
-----
ID      Program      Device  Pass Hard/Soft Bytes Written  Bytes Read
-----
00000001  idle      system    0    0    0           0           0
00000351  memtest   memory    0    0    0       37748736    37748736
00000352  memtest   memory    0    0    0       37748736    37748736
>>>
```

### Source File

```
show_status
```

## sleep

---

### sleep — Suspend execution for a time.

The sleep command suspends execution of the process for a specified number of seconds. It temporarily wakes up every second to check for and kill pending bits.

#### Syntax

```
sleep [-v] time_in_secs
```

*time\_in\_secs* Unmatched angle brackets! The default is 1 second.

#### Options

**-v**  
Specifies that the value supplied is in milliseconds. By default this is 1000 (1 second).

#### Examples

```
>>>(sleep 10; echo hi there)&
>>>
(10 seconds expire...)
hi there
>>> sleep -v 20 # sleep for 20 milliseconds
```

#### Command References

None

---

## sort — sort a file and write the sorted data into another file

Sort a files in lexicographic order and write the results onto stdout. Sort has limitations on the size of files that it can sort, due to the finite amount of memory.

### Syntax

```
sort file
```

### Arguments

**file**  
Name of the unsorted input file to be sorted.

### Options

**None.**

### Examples

```
>>>echo > foo 'banana
_>pear
_>apple
_>orange'
>>>sort foo
apple
banana
orange
pear
>>>
```

sp

---

## sp — Set process priority.

Set priority, 'sp', is used to modify the priority of a process. Changing the priority of process will impact the behaviour of the process and the rest system.

### Syntax

```
sp process_id new_priority
```

### Arguments

***process\_id***

Specifies the PID of the process to be modified.

***new\_priority***

Specifies the new priority for the process. Priority values range from 0 lowest to 7 highest.

### Options

None.

### Examples

```
>>>memtest -p 0 &
>>>ps | grep memtest
00000025 001a9700 2      23691 00000001 0      memtest ready
>>>sp 25 3
>>>ps | grep memtest
00000025 001a9700 3      125955 00000001 0      memtest ready
>>>
```

### Command References

ps, sa

---

## start — Start a processor at the specified address or drivers.

Starts program execution on a processor at the specified address or start drivers.

### Syntax

```
start [-drivers [device_prefix]] [address]
```

### Arguments

***address***

Specifies the PC address at which to start execution.

### Options

***-drivers* [*device\_prefix*]**

Specifies the name of the device or device class to stop. If no device prefix is specified, then all drivers are stopped.

### Examples

```
>>>start 400          # Start the primary at address 400.  
>>>start 100000 &p 2  # Start processor 2 at address 100000.  
>>>start -drivers     # Start the drivers in the system.
```

### Command References

continue, init, stop



## stop

---

### stop — Stop the specified processor or device.

Stops the specified processor of device.

#### Syntax

```
stop [-drivers device_prefix] [processor_num]
```

#### Arguments

***processor\_num***

Specifies the number of the processor to stop.

#### Options

**-drivers [*device\_prefix*]**

Specifies the name of the device or device class to stop. If no device prefix is specified, then all drivers are stopped.

#### Examples

```
>>>stop 2  
>>>
```

#### Command References

continue, init, start

---

## sw — swap little/big endian conversion

Swaps the bytes within a longword to convert from one endian format to the other

### Syntax

```
sw lwd1 lwd2 lwd3 ... lwdn hex
```

### Arguments

None.

### Options

None.

### Examples

```
>>>sw 12345678
      87654321
>>>sw 12345678 deadbeef
      87654321 efbeadde
>>>
```

## test

---

### test — Test the system, a subsystem, or a specific device.

Test the entire system, a subsystem, or a specific device, depending on the device list argument. A list of the subsystems and devices that can be tested can be obtained from the SHOW CONFIG and SHOW DEVICE commands.

If no subsystems or devices are specified, the TEST command tests all subsystems. Tapes cannot be tested with the TEST command.

Test a subsystem by entering TEST and then the name of the subsystem to be tested. Note that TEST performs read and write tests on memory.

Test a device by entering TEST and then the name of the device to be tested. TEST will then parse the device specification to find the appropriate test or script for that device. Note that only read tests are performed on disks.

All tests run concurrently for a minimum of 30 seconds. Tests complete when all component tests have completed at least one pass. Test passes are repeated on any component that completes its test before other components.

The run time of a test is proportional to the amount of memory to be tested and the number of disk drives to be tested.

Only one instance of test can be run at a time. test can be run as either a background or foreground process.

Use the SET command to establish parameters, such as whether to halt, loop, or continue on error. The passcount environment variable, d\_passes is ignored by test.

### Syntax

```
test [disk] [dssi] [scsi] [memory] [ethernet] [fbus] [device_list]
```

### Arguments

#### disk

Do read-only test of all disk drives. One pass consists of seeking to a random block on the disk(s) and reading a packet of 2048 bytes and repeating this until 512 packets are read.

#### dssi

Do read-only test of all dssi disks.

#### scsi

Do read-only test of all scsi disks.

#### memory

Test memory subsystem.

#### ethernet

Test network subsystem.

#### fbus

Run extended test number 0 on all (if any) Fbus+ devices.

**device\_list**

Use to specify an Fbus+ device or a subset of the disk subsystem. Use `fbus_diag` to run extended test 0 on the device if it is an Fbus+ device. For all other devices use `exer` to perform a read-only test on one or more devices. Legal device names are disk device names.

**Options**

None.

**Examples**

```
>>>test
>>>test &
>>>show_status
-----
ID      Program      Device  Pass Hard/Soft Bytes Written  Bytes Read
-----
00000001      idle      system    0   0   0           0           0
0000009f      memtest      memory    0   0   0    100663296    100663296
000000a5      fbus_diag  fbc0.0.0.6.1    1   0   0           12            92
000000b4      exer_kid   dub0.0.0.1.0    0   0   0           0          444416
000000b5      exer_kid   duc0.6.0.2.0   122   0   0           0          1249280
000000b6      exer_kid   dud0.7.0.3.0   122   0   0           0          1249280
000000b7      exer_kid   dka0.0.0.0.0    0   0   0           0          256000
000000be      nettest   eza0.0.0.6.0   13   0   0          20888         20888
000000be      nettest   ezb0.0.0.7.0   13   0   0          17904         17904
>>>
tests done
>>>show_status
-----
ID      Program      Device  Pass Hard/Soft Bytes Written  Bytes Read
-----
00000001      idle      system    0   0   0           0           0
>>>
```

**Source File**

test

tr

---

## tr — translate characters from stdin to stdout

The `tr` command copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in `string1` are mapped into the corresponding characters of `string2`. When `string2` is short it is padded to the length of `string1` by duplicating its last character. Any combination of the options `-cds` may be used: `-c` complements the set of characters in `string1` with respect to the universe of characters whose ASCII codes are 0 through 0377 octal; `-d` deletes all input characters in `string1`; `-s` squeezes all strings of repeated output characters that are in `string2` to single characters.

In either string the notation `a-b` means a range of characters from `a` to `b` in increasing ASCII order. The backslash character (`\`) followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A `\` followed by any other character stands for that character.

TR is binary transparent; its operation is not restricted to ASCII strings.

### Syntax

```
tr [-cds] [string1 [string2]]
```

### Arguments

**string1**  
input characters

**string2**  
translated characters

### Options

**c**  
complement characters in `string1`

**d**  
delete all characters in `string1` from output

**s**  
squeeze all repeated characters in `string2` to one character

### Examples

```
tr a-z A-Z <foo # translate from lower case to upper case
tr -s [\ \011\012] \012 # break out words one per line
tr -s [\ \011] [\ \011] # collapse multiple blanks and tabs
tr -sd [ ] # remove blanks
tr -sd '\000' # remove nulls from stdin
```

---

## uniq — cull out duplicate lines in a file

UNIQ culls out duplicate lines in a file. U\*x versions of uniq also have a qualifier that matches on fields. Similar functionality can be obtained by passing the input stream through a filter that removes whitespace (see the command `tr`).

### Syntax

```
uniq [filenames]
```

### Arguments

**filename**  
name of file(s) to search.

### Options

**-c**  
skip n characters

### Examples

```
>>>hd dua0 | uniq -1 | more
```

---

## update — Update flash roms on the system

Update FEPROMS with new firmware.

FEPROMS may be updated using the 'update' shell command. Normally, no arguments or qualifiers are needed if just the console roms are being updated.

Update kits normally contain the required images for updating. The update program looks for these images (in the ram disk). If the images are not present, or some other image is to be used, then one of the following command constructs must be used:

1) specify a device, protocol and filename, eg: `update -file cobra_ev4p2 -protocol mopdl -device eza0`

2) specify the fully qualified pathname, eg: `update -path mopdl:cobra_ev4p2.sys /eza0` The process works as follows: 1) The image is loaded into memory 2) consistency checks are applied to the image. Specifically, checksums must match, signatures must match, sizes must match etc. Consistency checks also insure that a VAX program doesn't get burned on an ALPHA. The program complains if any of the consistency checks fail. The program will then exit unless the -ignore qualifier is present. 3) The program asks the user to confirm that an update is desired. 4) The program actually burns the FEPROMS. If all of these requirements are satisfied, the user is given a console prompt of 'Do you really want to continue [Y/N]?' If 'n' (or 'N') is typed, the program will be aborted with a return to the console prompt. If 'y' (or 'Y') is typed, the blast will commence. At this point it is imperative that the program is NOT broken, halted, or interrupted for any reason. If interrupted, the module will most likely be left in an inoperable state and the FEPROMS will need to be replaced.

There are three steps to the blasting process:

1. All longwords are programmed to '00000000'. This is accomplished via function `uniform_program()` internal to the update command.
2. All longwords are erased. The erased state is 'ffffff'. This is accomplished via function `erase()` internal to the update command.
3. All longwords are reprogrammed to the values in the image booted into memory. This is accomplished via the function `program()` internal to the update command.

Progress messages to the user are printed out for each of the three steps.

Each longword of the FEPROM is verified in each of the three steps. Each step provides for a certain number of chances to perform the operation successfully on a particular longword of the FEPROM. These algorithms are taken directly from the 28f010 spec. If a failure occurs in any of the steps, an error message is printed to the console.

If the blasting operation is successful, a success message is printed to the console.

Note that it is necessary to reset or cycle power on the system to unload the new image of the FEPROMS into memory. Until a power down, the old image is still being executed.

## Syntax

```
update [-file filename]
        [-protocol transport] [-device source_device]
        [-target target_name]
        [-path full_pathname]
```

## Arguments

None.

## Options

### **-file filename**

Specifies the name of the new FEPRM update image. The default is the name of the platform that the update is running on. For example, if the update is running on a COBRA using a pass 4 chip, the default name is COBRA\_EV4P2.

### **-protocol transport**

Specifies the source transport protocol. The default is MOPDL.

### **-device**

Specifies the device from which to load the new FEPRM update image file. The default is EZA0.

### **-target device**

Specifies the device which contains the FEPRMs to be upgraded. The default is IO (the FEPRMs on I/O module). If the target is not recognized, then it is assumed to be the name of device driver this is of type flash rom.

### **-path full\_pathname**

This qualifier is used if the file, protocol and device qualifiers are not sufficient to fully specify the source filename. This qualifier over rides the file, protocol and device qualifiers.

### **-ignore**

Ignore the signature and consistency checks. If specified, the burn occurs regardless of the built in consistency checks. This qualifier should be used with caution!

## Examples

```
cs>update -file cobra_feb01_e42
                FEPRM UPDATE UTILITY
                -----> CAUTION <-----
EXECUTING THIS PROGRAM WILL CHANGE YOUR CURRENT ROM!
Do you really want to continue [Y/N] ? : y
                DO NOT ATTEMPT TO INTERRUPT PROGRAM EXECUTION!
                DOING SO MAY RESULT IN LOSS OF OPERABLE STATE.
The program will take at most several minutes.
Starting uniform programming to 0's of block...
00000 10000 20000 30000 40000 50000 60000 70000
Starting erase of block...
00000 10000 20000 30000 40000 50000 60000 70000
Starting reprogramming of block...
00000 10000 20000 30000 40000 50000 60000 70000
update successful!
cs>
```



---

## wc — Count bytes, words and lines and report totals.

WC counts bytes, words and lines in files and reports the totals on the standard output.

### Syntax

```
wc [-l|w|c] [file...]
```

### Arguments

**file...**

Specifies the file(s) on which to perform the counts.

### Options

**-l**

Specifies to count lines and display the number of lines.

**-w**

Specifies to count lines and display the number of words.

**-c**

Specifies to count lines and display the number of characters.

### Examples

```
>>>wc fred
  1      4     21 fred
  1      1      1 total
>>>wc -l fred
  1 fred
  1 total
>>>wc -w fred
  4 fred
  1 total
>>>wc -c fred
 21 fred
  1 total
>>>
```

### Command References

cat, ls

## Environment Variables

All supported environment variables are listed below in Table E-1.

**Table E-1 Environment Variables**

#	Variable	Attributes	Function
<b>Alpha SRM Defined Environment Variables</b>			
00			Reserved
01	auto_action	NV,W	The action the console should take following an error halt or powerfail. Defined values are:  "BOOT" (544F4F42) - Attempt bootstrap. "HALT" (544C4148) - Halt, enter console I/O mode. "RESTART" (54524154534552) - Attempt restart. If restart fails, then try boot.  Any other values cause a halt, and console mode is entered.
02	boot_dev	W	The default device or device list from which booting is attempted when no boot path is specified by the BOOT command. This variable may be a boot search list. The console derives the value from bootcmd_dev at console initialization; the value is preserved across warm bootstraps. The format of value is independent of the console presentation layer.
03	bootdef_dev	NV	The device or device list from which booting is to be attempted, when no path is specified on the command line.
04	booted_dev	RO	The device from which booting actually occurred.
05	boot_file	NV,W	The default file name used for the primary bootstrap when no file name is specified by the BOOT command. The default value when the system is shipped is NULL.

Key to variable attributes:

NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.

W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.

RO - Read-only. The variable cannot be modified by system system software or console commands.

(continued on next page)

**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>Alpha SRM Defined Environment Variables</b>			
06	booted_file	RO	The file name used for the primary bootstrap during the last boot. The value is NULL if boot_file is NULL and no bootstrap file name was specified by the BOOT command.
07	boot_osflags	NV,W	<p>Default additional parameters to be passed to system software during booting if none are specified by the boot command.</p> <p>On the OpenVMS AXP operating system, these additional parameters are the root number and boot flags. The default value when the system is shipped is NULL. The following parameters are used with the DEC OSF/1 operating system:</p> <ul style="list-style-type: none"> <li>a     Autoboot. Boots /vmunix from bootdef_dev, goes to multiuser mode. Use this for a system that should come up automatically after a power failure.</li> <li>s     Stop in single-user mode. Boots /vmunix to single-user mode and stops at the # (root) prompt.</li> <li>i     Interactive boot. Request the name of the image to boot from the specified boot device. Other flags, such as -kdebug (to enable the kernel debugger), may be entered using this option.</li> <li>D     Full dump, implies "s" as well. By default, if DEC OSF/1 V2.1 crashes, it completes a partial memory dump. Specifying "D" forces a full dump at system crash.</li> </ul> <p>Common settings are a, autoboot; and Da, autoboot; but create full dumps if the system crashes.</p>
08	booted_osflags	RO	Additional parameters, if any, specified by the last BOOT command that are to be interpreted by system software. The default value when the system is shipped is NULL.

**Key to variable attributes:**

NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.  
W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.  
RO - Read-only. The variable cannot be modified by system software or console commands.

(continued on next page)

**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>Alpha SRM Defined Environment Variables</b>			
09	boot_reset	NV,W	<p>Indicates whether a full system reset is performed in response to an error halt or BOOT command. Defined values and the action taken are:</p> <p>"OFF" (46464F) - warm boot, no full reset is performed.                      "ON" (4E4F) - cold boot, a full reset is performed.</p> <p>The default value when the system is shipped is "OFF".</p>
0A	dump_dev	NV,W	<p>The complete device specification of the device to which operating system dumps should be written. The default value when the system is shipped indicates a valid implementation-dependent device.</p>
0B	enable_audit	NV,W	<p>Indicates whether audit trail messages are to be generated during bootstrap.</p> <p>"OFF" (46464F) - Suppress audit trail messages.                      "ON" (4E4F) - Generate audit trail messages.</p> <p>The system is shipped with this set to "ON".</p>
0D	char_set	NV,W	<p>Indicates the character-set encoding currently selected to be used for the console terminal.</p> <p>0 - ISO-LATIN-1 character encoding                      other - TBD</p> <p>The default value when the system is shipped is 0.</p>

Key to variable attributes:

NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.  
 W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.  
 RO - Read-only. The variable cannot be modified by system system software or console commands.

(continued on next page)

**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>Alpha SRM Defined Environment Variables</b>			
0E	language	NV,W	The default language to display critical system messages.  00 none (cryptic) 30 Dansk 32 Deutsch 34 Deutsch (Schweiz) 36 English (American) 38 English (British/Irish) 3A Espanol 3C Francais 3E Francais (Canadian) 40 Francais (Suisse Romande) 42 Italiano 44 Nederlands 46 Norsk 48 Portugues 4A Suomi 4C Svenska 4E Vlaams
0F	tty_dev	NV,W,RO	Specifies the current console terminal unit. Indicates which entry of the CTB Table corresponds to the actual console terminal. The value is preserved across warm bootstraps. The default value is "0" 30 (hex).
10-3F			Reserved for Digital.
40-7F			Reserved for console use.
80-FF			Reserved for operating system use.
<b>System Dependent Environment Variables</b>			
	cpu_enabled	NV	A bitmask indicating which processors are enabled to run (leave console mode). If this variable is not defined, all available processors are considered enabled.
	d_bell		Specifies whether or not to bell on error if error is detected.  OFF (Default) ON

Key to variable attributes:

NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.  
W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.  
RO - Read-only. The variable cannot be modified by system system software or console commands.

(continued on next page)

**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>System Dependent Environment Variables</b>			
	d_cleanup		Specifies whether or not cleanup code is executed at the end of a diagnostic.  ON (Default) OFF
	d_complete		Specifies whether or not to display the diagnostic completion message.  OFF (Default) ON
	d_eop		Specifies whether or not to display end-of-pass messages.  OFF (Default) - Disable end-of-pass messages. ON - Enable end-of-pass messages.
	d_group		Specifies the diagnostic group to be executed.  FIELD (Default) MFG other diagnostic group string (up to 32 characters)
	d_harderr		Specifies the action taken following hard error detection.  CONTINUE HALT (Default) LOOP
	d_oper		Specifies whether or not an operator is present.  ON - Indicates operator present. OFF (Default) - Indicates no operator present.
	d_passes		Specifies the number of passes to run a diagnostic module.  1 (Default) 0 - Indicates to run indefinitely. an arbitrary value

Key to variable attributes:

- NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.
- W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.
- RO - Read-only. The variable cannot be modified by system system software or console commands.

(continued on next page)

**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>System Dependent Environment Variables</b>			
	d_report		Specifies the level of information provided by diagnostic error reports.  SUMMARY (Default) FULL OFF
	d_softerr		Specifies the action taken following soft error detection.  CONTINUE (Default) HALT LOOP
	d_startup		Specifies whether or not to display the diagnostic startup message.  OFF (Default) - Disables the startup message. ON - Enables the startup message.
	d_trace		Specifies whether or not to display test trace messages.  OFF (Default) - Disables trace messages. ON - Enables trace messages.
	enable_servers	NV	Set to ON to run storage port loopback test. Default is OFF.
	etherneta		Specifies the Ethernet station address for port eza0.
	ethernetb		Specifies the Ethernet station address for port ezb0.
	exdep_data	RO	Specifies the data value referenced by the last examine or deposit command.
	exdep_location	RO	Specifies the location referenced by the last examine or deposit command.
	exdep_size	RO	Specifies the data size referenced by the last examine or deposit command.
	exdep_space	RO	Specifies the address space referenced by the last examine or deposit command.
	exdep_type	RO	Specifies the data type referenced by the last examine or deposit command.

Key to variable attributes:

NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.  
W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.  
RO - Read-only. The variable cannot be modified by system system software or console commands.

(continued on next page)

**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>System Dependent Environment Variables</b>			
	ez*0_arp_tries	NV	Sets the number of transmissions that are attempted before the ARP protocol fails. Values less than 1 cause the protocol to fail immediately. Default value is 3, which translates to an average of 12 seconds before failing. Interfaces on busy networks may need higher values.
	ez*0_bootp_file	NV	Supplies the generic file name to be included in a BOOTP request. The BOOTP server will return a fully qualified file name for booting. This can be left empty.
	ez*0_bootp_server	NV	Supplies the server name to be included in a BOOTP request. This can be set to the name of the server from which the machine is to be booted, or can be left empty.
	ez*0_bootp_tries	NV	Sets the number of transmissions that are attempted before the BOOTP protocol fails. Values less than 1 cause the protocol to fail immediately. Default value is 3, which translates to an average of 12 seconds before failing. Interfaces on busy networks may need higher values.
	ez*0_def_ginetaddr	NV	Supplies the initial value for ez*0_ginetaddr when the interface's internal internet database is initialized from nvram (ie, ez*0_inet_init is set to "nvram").
	ez*0_def_inetaddr	NV	Supplies the initial value for ez*0_inetaddr when the interface's internal internet database is initialized from nvram (ie, ez*0_inet_init is set to "nvram").
	ez*0_def_inetfile	NV	Supplies the initial value for ez*0_inetfile when the interface's internal internet database is initialized from nvram (ie, ez*0_inet_init is set to "nvram").
	ez*0_def_sinetaddr	NV	Supplies the initial value for ez*0_sinetaddr when the interface's internal internet database is initialized from nvram (ie, ez*0_inet_init is set to "nvram").

Key to variable attributes:

- NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.
- W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.
- RO - Read-only. The variable cannot be modified by system system software or console commands.

(continued on next page)



**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>System Dependent Environment Variables</b>			
	ez*0_driver_flags		Specifies the flags to be used by the driver. Current values are:  1 : NDLSM_ENA_BROADCAST will enable broadcast messages. 2 : NDLSM_ENA_HASH will enable hash filtering. 4 : NDLSM_ENA_INVF will enable inverse filtering. 8 : NDLSM_MEMZONE will allocate the message buffers from memzone.
	ez*0_ginetaddr		Accesses the gateway address field of the interface's internal internet database. This is normally the address of the local network's gateway to other networks.
	ez*0_inet_init	NV	Determines whether the interface's internal internet database is initialized from nvram or from a network server (via the BOOTP protocol). Legal values are "nvram" and "bootp"; default is "bootp".
	ez*0_inetaddr		the local address field of the interface's internal internet database.
	ez*0_inetfile		Accesses the file name field of the interface's internal internet database. This is normally the file to be booted from the TFTP server. This variable supplies the default remote file name for TFTP transactions.
	ez*0_loop_count		Specifies the number of time each message is looped.
	ez*0_loop_inc		Specifies the amount the message size is increased from message to message.
	ez*0_loop_patt		Specifies the type of data pattern to be used when doing loopback. Current patterns are accessed by the following:  0xffffffff = All the patterns 0 = all zero's 1 = all one's 2 = all fives 3 = all A's 4 = incrementing 5 = decrementing

Key to variable attributes:

NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.  
W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.  
RO - Read-only. The variable cannot be modified by system system software or console commands.

(continued on next page)

**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>System Dependent Environment Variables</b>			
	ez*0_loop_list_size		Specifies the size of the preallocated list used during loopback.
	ez*0_loop_size		Specifies the size of the loop data to be used.
	ez*0_lp_msg_node		Specifies the number of messages originally sent to each node.
	ez*0_mode		Specifies the value for the SGEC mode when the device is started. This value is a mirror of CSR6. It can be different from device to device.
	ez*0_msg_buf_size		Specifies the message size. Receive data chaining can be achieved by picking a small value for this variable.
	ez*0_msg_mod		Specifies the modulus for message alignment.
	ez*0_msg_rem		Specifies the remainder for message alignment.
	ez*0_protocols	NV	Determines which network protocols are enabled for booting and other functions. Legal values include "BOOTP", "MOP", and "BOOTP,MOP". A null value is equivalent to "BOOTP,MOP".
	ez*0_rcv_buf_no		Specifies the number of receive buffers
	ez*0_rcv_mod		Specifies the modulus for receive descriptor alignment.
	ez*0_rcv_rem		Specifies the remainder for receive descriptor alignment.
	ez*0_sinetaddr		Accesses the server address field of the interface's internal internet database. This is normally the address of the BOOTP and TFTP server. This variable supplies the default remote address for TFTP transactions.
	ez*0_tftp_tries	NV	Sets the number of transmissions that are attempted before the TFTP protocol fails. Values less than 1 cause the protocol to fail immediately. Default value is 3, which translates to an average of 12 seconds before failing. Interfaces on busy networks may need higher values.
	ez*0_xmt_buf_no		Specifies the number of transmit buffers.
	ez*0_xmt_int_msg		Specifies the number of transmit interrupts per message.
	ez*0_xmt_max_size		Specifies the maximum message size that can be transmitted. Transmit data chaining can be achieved by picking a small value for this variable.

Key to variable attributes:

NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.  
W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.  
RO - Read-only. The variable cannot be modified by system system software or console commands.

(continued on next page)

**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>System Dependent Environment Variables</b>			
	ez*0_xmt_mod		Specifies the modulus for transmit descriptor alignment.
	ez*0_xmt_msg_post		Specifies the number of messages before posting a transmit.
	ez*0_xmt_rem		Specifies the remainder for transmit descriptor alignment.
	ferr1		Quadword of error information that Futurebus+ modules can store.
	ferr2		Quadword of error information that Futurebus+ modules can store.
	fis_name	NV	Specifies a string indicating the factory installed software.
	interleave	NV	<p>Specifies the memory interleave configuration for the system. The value must be one of: "default", "none", or an explicit interleave list. The syntax for specifying the configuration is:</p> <p style="padding-left: 40px;">0,1,2,3 - Indicates the memory module (or slot) numbers.            : - Indicates that the adjacent memory modules are combined to form a logical module or single interleave unit.            + - Indicates that the adjacent memory modules or units are to be interleaved forming a set.            , - Indicates that the adjacent memory modules, units, or sets are not to be interleaved.</p> <p>For example, assume a system where memory module 0 and 1 are 64MB each, module 2 is 128MB, and module 3 is 32MB. Memory is configured memory such that module 0 and 1 are combined as a logical unit which is 128MB. This unit is interleaved with module 2 which is also 128MB to form an interleaved set which is 256MB. Module 3 is not interleaved, but configured as the next 32MB after the interleave set.</p> <p>set interleave 0:1+2,3            The system is shipped with interleave set to "default". With this value, the optimal interleave configuration for the memory modules will be set. Normally, there is no reason to change the interleave setting.</p>

Key to variable attributes:

- NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.
- W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.
- RO - Read-only. The variable cannot be modified by system system software or console commands.

(continued on next page)

**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>System Dependent Environment Variables</b>			
	mopv3_boot		Specifies whether to use MOP Version 3 format messages first in the boot requests sequence instead of MOP Version 4.
	ncr*_setup	NV	Here "*" may be one of 0, 1, 2, 3, or 4, corresponding to the storage bus adapters A, B, C, D, or E, respectively.
	pal	RO	Specifies the versions of VMS and OSF PALcode in the firmware. For instance, VMS PALcode X5.12B, OSF PALcode X1.09A.
	screen_mode	NV	Specifies whether to show the powerup display as a series of lines of text or in screen-mode. screen_mode can be ON or OFF. The default is ON.
	sys_serial_num	NV	The system serial number, set at the factory.
	tt_allow_login		Set to 0 if loopback testing is to be done on tta1. Default is 1.
	tta_merge	NV	Merges the input from tta0 to the input stream of tta1. Output from tta0 is duplicated to tta1. Output from tta1 is not displayed on tta0.
	tta*_baud	NV	Here "*" may be one of 0 or 1, corresponding to the primary console serial port, tta0 or the auxiliary console serial port, tta1. Specifies the Baud rate of the primary console serial port, tta0. Allowable values are 600, 1200, 2400, 4800, 9600, and 19200. The initial value for tta0 is read from the Baud rate switch on OCP.
	tta*_halts	NV	Specifies halt characters recognized on the console serial ports, tta0 and tta1. The value is an integer bitmap, where:  bit 0 - Enables(1) or disables(0) CTRL-P to init from the console. bit 1 - Enables(1) or disables(0) CTRL-P halts from the operating system. bit 2 - Enables(1) or disables(0) BREAK /halts from the operating system. Note, since tta1 is intended for modem support, this bit is ignored on tta1 (i.e. BREAKs are not permitted on the auxiliary port).  The default for tta0 is a 2, enabling CTRL-P halts from the operating system. The default for tta1 is a 0, disabling halts from the operating system.

Key to variable attributes:

NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.  
W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.  
RO - Read-only. The variable cannot be modified by system system software or console commands.

(continued on next page)

**Table E-1 (Cont.) Environment Variables**

#	Variable	Attributes	Function
<b>System Dependent Environment Variables</b>			
	version	RO	Specifies the version of the console code in the firmware. For instance, V2.3-2001 Aug 21 1992 14:25:19.

Key to variable attributes:

NV - Non-volatile. The last value saved by system software or set by console commands is preserved across system initializations, cold bootstraps, and long power outages.

W - Warm non-volatile. The last value set by system software is preserved across warm bootstraps and restarts.

RO - Read-only. The variable cannot be modified by system system software or console commands.

---

# Glossary

## **ANSI**

American National Standards Institute, an organization that develops and publishes standards for the computer industry.

## **arbiter**

The entity responsible for controlling a bus. It controls bus mastership and may field bus interrupt requests.

## **assert**

To cause a signal to change to its logical true state.

## **AST**

*See* asynchronous system trap.

## **AST atomic**

A characteristic of certain operations that indicates that the operation may be performed in an environment where ASTs may occur. From the standpoint of the AST service routine, the operation will always appear either to have not started or to have completed. *See also* asynchronous system trap.

## **asymptotic**

An equality statement about two functions, which states that the ratio of the two functions approaches 1 while the variable approaches some value, usually infinity. It is not a true equality. This term is sometimes used in reference to bus bandwidth (approaching, but never reaching, full theoretical bandwidth).

## **asynchronous system trap (AST)**

A software-simulated interrupt to a user-defined routine. ASTs enable a user process to be notified asynchronously, with respect to that process, of the occurrence of a specific event. If a user process has defined an AST routine for an event, the system interrupts the process and executes the AST routine when that event occurs. When the AST routine exits, the system resumes execution of the process at the point where it was interrupted.

## **atomic**

An operation or sequence of events that, once begun, completes without interruption.

## **atomic instructions**

Instructions that consist of one or more discrete operations, which are handled as a single operation by the hardware, without interruption.

**atomicity of modifications**

A characteristic of VAX memory systems that allows read-modify-write operations that cannot be interrupted by other system events. These operations appear to other processes to be a single operation. Once the operation starts, it always completes without interruption. Atomic memory modification may involve bus locking protocols or other hardware assistance to achieve the atomic operation.

**autoboot**

The process by which the system boots automatically.

**auxiliary serial port**

The EIA 232 serial port on the I/O module of the DEC 4000 AXP system. This port provides asynchronous communication with a device, such as a modem.

**availability**

The amount of scheduled time that a computing system provides application service during the year. Availability is typically measured as either a percentage of "uptime" per year or as system "unavailability," the number of hours or minutes of downtime per year.

**BA640**

The enclosure that houses the DEC 4000 AXP system. The BA640 is compatible with the departmental environment and is designed for maximum flexibility in system configuration. Employing an open system architecture, the BA640 incorporates a state-of-the-art Futurebus+ area, which allows for expansion of the DEC 4000 AXP system with options available from Digital and other vendors.

**backplane**

The main circuit board or panel that connects all of the modules in the system. In desktop systems, the backplane is analogous to the motherboard.

**backup cache**

A second, very fast memory that is used in combination with slower large-capacity memories.

**bandwidth**

Bandwidth is often used to express "high rate of data transfer" in an I/O channel. This usage assumes that a wide bandwidth may contain a high frequency, which can accommodate a high rate of data transfer.

**baud rate**

The speed at which data is transmitted over a data line; baud rates are measured in bits per second.

**benchmark**

A routine or program used in the evaluation of computer performance.

**bit**

Binary digit. The smallest unit of data in a binary notation system, designated as 0 or 1.

**BIU**

See bus interface unit.

**block exchange**

Memory feature that improves bus bandwidth by paralleling a cache victim write-back with a cache miss fill.

**boot**

Short for bootstrap. Loading an operating system into memory is called booting.

**bootblock**

The first logical block on the boot device. It contains information about the location of the primary bootstrap on the device.

**boot device**

The device from which the system bootstrap software is acquired.

**boot flags**

Boot flags contain information that is read and used by the bootstrap software during a system bootstrap procedure.

**boot primitives**

Device handler routines that read the bootblock and, subsequently, the primary bootstrap program, into memory from the boot device. *See also* bootblock.

**boot server**

A system that provides boot services to remote devices such as network routers and VAXcluster satellite nodes.

**bootstrap**

See boot.

**buffer**

An internal memory area used for temporary storage of data records during input or output operations.

**bugcheck**

A software condition, usually the response to software's detection of an "internal inconsistency," which results in the execution of the system bugcheck code.

**bus**

A group of signals that consists of many transmission lines or wires. It interconnects computer system components to provide communications paths for addresses, data, and control information.

**bus interface unit**

Logic designed to interface internal logic, a module or a chip, to a bus.

**bystander**

A system bus node that is not addressed by a current system bus commander transaction address.



**byte**

Eight contiguous bits starting on an addressable byte boundary. The bits are numbered right to left, 0 through 7.

**byte granularity**

Memory systems are said to have byte granularity if adjacent bytes can be written concurrently and independently by different processes or processors.

**C<sup>3</sup> chip**

An acronym for command, control, and communication chip. On the DEC 4000 AXP system, the ASIC gate array chip located on the CPU module. This chip contains CPU command, control, and communication logic, as well as the bus interface unit for the processor module.

**cache**

*See* cache memory.

**cache block**

The fundamental unit of manipulation in a cache. Also known as cache line.

**cache interference**

The result of an operation that adversely affects the mechanisms and procedures used to keep frequently used items in a cache. Such interference may cause frequently used items to be removed from a cache or incur significant overhead operations to ensure correct results. Either action hampers performance.

**cache line**

The fundamental unit of manipulation in a cache. Also known as cache block.

**cache memory**

A small, high-speed memory placed between slower main memory and the processor. A cache increases effective memory transfer rates and processor speed. It contains copies of data recently used by the processor and fetches several bytes of data from memory in anticipation that the processor will access the next sequential series of bytes.

**card cage**

A mechanical assembly in the shape of a frame that holds modules against the system and storage backplanes.

**CD-ROM**

Compact disc read-only memory. The optical removable media used in a compact disc reader mass storage device.

**central processing unit (CPU)**

The unit of the computer that is responsible for interpreting and executing instructions.

**channel**

A path along which digital information can flow in a computer.

**checksum**

A sum of digits or bits that is used to verify the integrity of a piece of data.

**CI**

*See* computer interconnect.

**CISC**

Complex instruction set computer. An instruction set consisting of a large number of complex instructions that are managed by microcode. *Contrast with* RISC.

**clean**

In the cache of a system bus node, refers to a cache line that is valid but has not been written.

**client-server computing**

An approach to computing that enables personal computer and workstation users—the “client”—to work cooperatively with software programs stored on a mainframe or minicomputer—the “server.”

**clock**

A signal used to synchronize the circuits in a computer system.

**cluster**

A group of systems and hardware that communicate over a common interface. *See also* VMScluster system.

**CMOS**

Complementary metal-oxide semiconductor. A silicon device formed by a process that combines PMOS and NMOS semiconductor material.

**command**

A field of the system bus address and command cycle (cycle 1), which encodes the transaction type.

**commander**

A system bus node that participates in arbitration and initiates a transaction. Also called a commander node.

**computer interconnect (CI)**

Digital's high-speed, fault-tolerant, dual-path bus, which has a bandwidth of 70 megabits per second. With the CI, any combination of processor nodes and intelligent I/O subsystem nodes—up to 16 in number—can be coupled loosely in a computer-room environment.

**concurrency**

Simultaneous operations by multiple agents on a shared object.

**conditional invalidation**

Invalidation of a cached location based upon a set of conditions, which are the state of other caches, or the source of the information causing the invalidate.

**console mode**

The state in which the system and the console terminal operate under the control of the console program.

**console password**

The password used to access privileged console commands.

**console program**

The code that the CPU executes during console mode.

**console subsystem**

The subsystem that provides the user interface for a system when operating system software is not running. The console subsystem consists of the following components:

- console program
- console terminal
- console terminal port
- remote access device
- remote access port
- Ethernet ports

**console terminal**

The terminal connected to the console subsystem. The console is used to start the system and direct activities between the computer operator and the computer system.

**console terminal port**

The connector to which the console terminal cable is attached.

**control and status register (CSR)**

A device or controller register that resides in the processor's I/O space. The CSR initiates device activity and records its status.

**CPU**

*See* central processing unit.

**CSR**

*See* control and status register.

**cycle**

One clock interval.

**data alignment**

An attribute of a data item that refers to its placement in memory (therefore its address).

**data bus**

A bus used to carry signals between two or more components of the system.

**D-bus**

On the DEC 4000 AXP system, the bus between the 21064 CPU chip and the "D-bus micro" and the serial ROMs.

**D-cache**

Data cache. A high-speed memory reserved for the storage of data. *Contrast with I-cache.*

**DC-DC converter**

A device that converts one DC voltage to another DC voltage.

**deassert**

To cause a signal to change to its logical false state.

**DECchip 21064 processor**

The CMOS-4, Alpha AXP architecture, single-chip processor used on Alpha AXP based computers.

**DECnet**

Networking software designed and developed by Digital. DECnet is an implementation of the Digital Network Architecture.

**DEC OSF/1 AXP operating system**

A general-purpose operating system based on the Open Software Foundation OSF/1 1.0 technology. DEC OSF/1 V1.2 runs on the range of Alpha AXP systems, from workstations to servers.

**DEC VET**

Digital's DEC Verifier and Exerciser Tool. DEC VET is a multipurpose system maintenance tool that performs exerciser-oriented maintenance testing.

**Dhrystone**

A benchmark for computer system performance that measures integer performance. Dhrystone is part of the suite of SPEC benchmarks.

**direct-mapping cache**

A cache organization in which only one address comparison is needed to locate any data in the cache, because any block of main memory data can be placed in only one possible position in the cache.

**direct memory access (DMA)**

Access to memory by an I/O device that does not require processor intervention.

**dirty**

Used in reference to a cache block in the cache of a system bus node. The cache block is valid and has been written so that it differs from the copy in system memory.

**dirty victim**

Used in reference to a cache block in the cache of a system bus node. The cache block is valid but is about to be replaced due to a cache block resource conflict. The data must therefore be written to memory.

**disk fragmentation**

The writing of files in noncontiguous areas on a disk. Fragmentation can cause slower system performance because of repeated read or write operations on fragmented data.

**disk mirroring**

*See* volume shadowing.

**distributed processing**

A processing configuration in which each processor has its own autonomous operating environment. The processors are not tightly coupled and globally controlled as they are with multiprocessing. A distributed processing environment can include multiprocessor systems, uniprocessor systems, and cluster systems. It entails the distribution of an application over more than one system. The application must have the ability to coordinate its activity over a dispersed operating environment. *Contrast with* symmetric multiprocessing.

**DRAM**

Dynamic random-access memory. Read/write memory that must be refreshed (read from or written to) periodically to maintain the storage of information.

**DSSI**

Digital's proprietary data bus that uses the System Communication Architecture (SCA) protocols for direct host-to-storage communications.

**DSSI VMScluster**

A VMScluster system that uses the DSSI bus as the interconnect between DSSI disks and systems.

**ECC**

Error correction code. Code and algorithms used by logic to facilitate error detection and correction. *See also* ECC error; EDC logic.

**ECC error**

An error detected by EDC logic, to indicate that data (or the protected "entity" has been corrupted. The error may be correctable (ECC error) or uncorrectable (ECCU error). *See also* EDC logic.

**EDC logic**

Error detection and correction logic. Used to detect and correct errors. *See also* ECC; ECC error.

**EEPROM**

Electrically erasable programmable read-only memory. A memory device that can be byte-erased, written to, and read from. *Contrast with* FEPRM.

**emitter-coupled logic**

A form of current-mode logic in which only one transistor conducts at a time. The emitters of the two transistors are tied together to a single current-carrying resistor.

**environment variable**

Global data structures that can be accessed from console mode. The setting of these data structures determines how a system powers up, boots operating system software, and operates.

**Ethernet**

A local area network that was originally developed by Xerox Corporation and has become the IEEE 802.3 standard LAN. Ethernet LANs use bus topology.

**Ethernet ports**

The connectors through which the Ethernet is connected to the system.

**extents**

The physical locations in a storage device allocated for use by a particular data set.

**Factory Installed Software (FIS)**

Operating system software that is loaded into a system disk during manufacture. On site, the FIS is bootstrapped in the system, prompting a predefined menu of questions on the final configuration.

**fast SCSI**

An optional mode of SCSI-2 that allows transmission rates of up to 10 MB/s. *See also* SCSI.

**FDDI**

Fiber Distributed Data Interface. A high-speed networking technology that uses fiber optics as the transmissions medium.

**FEPRM**

Flash-erasable programmable read-only memory. FEPRMs can be bank- or bulk-erased. *Contrast with* EEPROM.

**FIFO**

First in/first out; the order in which data is accessed.

**FIS**

*See* Factory Installed Software.

**firmware**

Software code stored in hardware.

**fixed-media compartments**

Compartments that house nonremovable storage media.

**floating-point**

Arithmetic operations in which the location of the decimal point will be place automatically in its correct position.

**front end unit (FEU)**

One of four modules in the DEC 4000 AXP system power supply. The FEU converts alternating current from a wall plug to 48 V DC that the rest of the power subsystem can use and convert.

**FRU**

Field-replaceable unit. Any system component that the service engineer is able to replace on-site.

**full-height device**

Standard form factor for 5 1/4-inch storage devices.

**Futurebus+**

A computer bus architecture that provides performance scalable over both time and cost. It is the IEEE 896 open standard.

**Futurebus+ Profile B**

A profile is a specification that calls out a subset of functions from a larger specification. Profile B satisfies the requirements for an I/O bus. *See also* Futurebus+.

**granularity**

A characteristic of storage systems that defines the amount of data that can be read and/or written with a single instruction, or read and/or written independently. VAX systems have byte or multibyte granularities, whereas disk systems typically have 512-byte or greater granularities. For a given storage device, a higher granularity generally yields a greater throughput.

**half-height device**

Standard form factor for storage devices that are not the height of full-height devices.

**halt**

The action of transferring control to the console program.

**hard error**

An error that has induced a nonrecoverable failure in a system.

**hexword**

Short for "hexadecimalword." Thirty-two contiguous bytes (256 bits) starting on an addressable byte boundary. Bits are numbered from right to left, 0 through 255.

**high-level language**

A language for specifying computing procedures or organization of data within a digital computer. High-level languages are distinguished from low-level languages, such as assembly and machine languages, by the omission of machine-specific details required for direct execution on a given computer. *See also* low-level language.

**hit**

Indicates that a valid copy of a desired memory location is currently in cache.

**I-cache**

Instruction cache. A high-speed memory reserved for the storage of instructions. *Contrast with D-cache.*

**image section**

A group of program sections with the same attributes (such as read-only access, read/write access, absolute, relocatable, and so on) that is the unit of virtual memory allocation for an image.

**initialization**

The sequence of steps that prepare the system to start. Initialization occurs after a system has been powered up.

**intelligent refresh control**

Scheduling DRAM “refresh” operations on an opportunistic basis. This process serves to reduce average memory latency.

**interleaving**

*See memory interleaving.*

**interlocked instruction**

An instruction that performs some action in a way that guarantees the complete result atomically in a multiprocessing environment. Since other potentially conflicting operations may be blocked while the interlocked instruction completes, interlocked instructions can negatively affect performance.

**internal processor register (IPR)**

A register internal to the CPU chip.

**interoperable**

Describes a characteristic of a programming function that allows it to work properly within a heterogeneous computing environment.

**LAN (local area network)**

A network that supports servers, PCs, printers, minicomputers, and mainframe computers that are connected over limited distances.

**latency**

The amount of time it takes the system to respond to an event.

**LED**

Light-emitting diode. A semiconductor device that glows when supplied with voltage.

**Linpack**

A benchmark for computer system performance that measures floating-point performance. Part of the suite of SPEC benchmarks.

**load/store architecture**

A characteristic of a machine architecture where data items are first loaded into a processor register, operated on, and then stored back to memory. No operations on memory other than load and store are provided by the instruction set.



**local area VMSccluster system**

Digital's VMSccluster configuration in which cluster communication is carried out over the Ethernet by software that emulates certain computer interconnect (CI) port functions.

**longword**

Four contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 31.

**loopback tests**

Diagnostic tests used to isolate a failure by testing segments of a particular control or data path.

**low-level language**

Any language that exposes the details of the hardware implementation to the programmer. Typically this refers to assembly languages that allow direct hardware manipulation. *See also* high-level language.

**machine check**

An operating system action triggered by certain system hardware-detected errors that can be fatal to system operation. Once triggered, machine check handler software analyzes the error.

**mailbox**

A memory data structure used to communicate between different components of the system.

**masked write**

A write cycle that only updates a subset of a nominal data block.

**mass storage device**

An input/output device on which data is stored. Typical mass storage devices include disks, magnetic tapes, and floppy disks.

**memory interleaving**

The process of assigning consecutive physical memory addresses across multiple memory controllers. Improves total memory bandwidth by overlapping system bus command execution across two or four memory modules.

**memory-like**

Refers to regions that have predictable behavior. For example, all locations are read/write; a write to a location followed by a read from that location returns precisely the bits written. *See also* non-memory-like.

**MFLOPS**

Millions of floating-point operations per second. An estimation of the theoretical peak performance of vector operations. A more useful means of measurement is to express MFLOPS in an industry-standard benchmark, such as Linpack MFLOPS.

**MIPS**

Millions of instructions per second.

**miss**

Indicates that a copy of a desired memory location is not in a cache.

**mixed-interconnect VMScluster system**

Digital's VMScluster system that uses multiple interconnect types between systems; for example, CI, Ethernet, DSSI, or FDDI.

**MOP**

Maintenance Operations Protocol. The transport protocol for network bootstraps and other network operations.

**multiprocessing system**

A system that executes multiple tasks simultaneously.

**multiplex**

To transmit several messages or signals simultaneously on the same circuit or channel.

**NAS**

*See* Network Applications Support.

**naturally aligned data**

Data stored in memory such that the address of the data is evenly divisible by the size of the data in bytes. For example, an aligned longword is stored such that the address of the longword is evenly divisible by 4.

**Network Applications Support**

A comprehensive set of software supplied by Digital Equipment Corporation that enables application integration across a distributed multivendor environment. NAS consists of well-defined programming interfaces, toolkits, and products that help developers build applications that are well-integrated and more easily portable across different systems.

**node**

A device that has an address on, is connected to, and is able to communicate with other devices on the bus. In a computer network, an individual computer system connected to the network that can communicate with other systems on the network.

**non-memory-like**

Regions that may have arbitrary behavior. For example, there may be unimplemented locations or bits anywhere; some locations or bits may be read-only and others write-only, and so on. *See also* memory-like.

**NVRAM**

Nonvolatile random-access memory. Memory that retains its information in the absence of power such as magnetic tape, drum, or core memory.

**octaword**

Sixteen contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 127.

**open system**

A system that implements sufficient open specifications for interfaces, services, and supporting formats to enable applications software to:

- Be ported across a wide range of systems with minimal changes
- Interoperate with other applications on local and remote systems
- Interact with users in a style that facilitates user portability

**Open Systems Interconnect standards**

Communications reference model defined by the ISO (International Organization for Standards). The OSI reference model consists of seven layers and defines protocols for the physical transmission of data, as well as the structuring and organization of data, so that it can be sent and received in a form that can be understood by conforming implementations. Conformance to the OSI standard will enable communication among computer systems from different vendors.

**OpenVMS AXP operating system**

Digital's open version of the VMS operating system, which runs on Alpha AXP machines. *See also* open system.

**operand**

The data or register upon which an operation is performed.

**operating system mode**

The state in which the system console terminal is under the control of the operating system software. Also called program mode.

**operator control panel**

The panel on the top right side of the DEC 4000 AXP system that contains the power, Reset, and Halt switches and system status lights.

**OSI standards**

*See* Open System Interconnect standards.

**page size**

A number of bytes, aligned on an address evenly divisible by that number, which a system's hardware treats as a unit for virtual address mapping, sharing, protection, and movement to and from secondary storage.

**PAL**

Programmable array logic (hardware), a device that can be programmed by a process that blows individual fuses to create a circuit.

**PALcode**

Alpha AXP Privileged Architecture Library code, written to support Alpha AXP processors. PALcode implements architecturally defined behavior.

**parity**

A method for checking the accuracy of data by calculating the sum of the number of ones in a piece of binary data. Even parity requires the correct sum to be an even number, odd parity requires the correct sum to be an odd number.

**pipeline**

A CPU design technique whereby multiple instructions are simultaneously overlapped in execution.

**portability**

Degree to which a software application can be easily moved from one computing environment to another.

**porting**

Adapting a given body of code so that it will provide equivalent functions in a computing environment that differs from the original implementation environment.

**POSIX**

Portable Operating System Interface. An IEEE standard that refers to a set of standards designed to enhance application portability. POSIX is also commonly used to refer specifically to the first of a series of such standards to be ratified: the 1003 standard that defines an operating system interface specification.

**power-down**

The sequence of steps that stops the flow of electricity to a system or its components.

**power system controller (PSC)**

One of four units in the DEC 4000 AXP power supply subsystem. The H7851AA PSC monitors signals from the rest of the system including temperature, fan rotation, and DC voltages, as well as provides power-up and power-down sequencing to the DC-DC converters and communicates with the system CPU across the serial control bus.

**power-up**

The sequence of events that starts the flow of electrical current to a system or its components.

**prefetch**

Refers to read lookahead activity in which DEC 4000 AXP memory modules fetch DRAM data prior to an actual read request for that data. *See also* read stream buffers.

**primary cache**

The cache that is the fastest and closest to the processor.

**probe**

The act of using the current operation address on the DEC 4000 AXP bus or processor to perform a cache line lookup to determine if the line is valid and/or must be invalidated, updated, or returned.

**processor module**

Module that contains the CPU chip.

**program counter**

That portion of the CPU that contains the virtual address of the next instruction to be executed. Most current CPUs implement the program counter (PC) as a register. This register may be visible to the programmer through the instruction set.

**program mode**

*See operating system mode.*

**quadword**

Eight contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 63.

**R400X mass storage expander**

A Digital enclosure used for mass storage expansion.

**RAID**

Redundant array of inexpensive disks. A technique that organizes disk data to improve performance and reliability. RAID has three attributes:

1. It is a set of physical disks viewed by the user as a single logical device.
2. The user's data is distributed across the physical set of drives in a defined manner.
3. Redundant disk capacity is added so that the user's data can be recovered even if a drive fails.

*Contrast with striping.*

**read data wrapping**

Memory feature that reduces apparent memory latency by allowing octawords within a selected hexword block to be accessed in reverse order.

**read-merge**

Indicates that an item is read from a responder/bystander, and new data is then added to the returned read data. This occurs when a masked write cycle is requested by the processor or when unmasked cycles occur and the CPU is configured to allocate on full block write misses.

**read-modify-write operation**

A hardware operation that involves the reading, modifying, and writing of a piece of data in main memory as a single, uninterruptible operation.

**read stream buffers**

Arrangement whereby each memory module independently prefetches DRAM data prior to an actual read request for that data. Reduces average memory latency while improving total memory bandwidth.

**read-write ordering**

Refers to the order in which memory on one CPU becomes visible to an execution agent (a different CPU or device within a tightly coupled system).

**register**

A temporary storage or control location in hardware logic.

**redundant**

Describes duplicate or extra computing components that protect a computing system from failure.

**reliability**

The probability a device or system will not fail to perform its intended functions during a specified time interval when operated under stated conditions.

**remote access device**

Hardware other than the local console terminal that can access a system's console user interface. The remote device is connected to the system through the system's auxiliary serial port or Ethernet.

**removable-media compartment**

Compartment in the enclosure that houses removable media.

**responder**

A system bus node that accepts or supplies data in response to an address and command from a system bus commander. Also called a responder node.

**reset**

A hardware condition where by a switch is pushed causing the processor to start and initialize.

**ring topology**

A circular LAN configuration that connects a group of nodes.

**RISC**

Reduced instruction set computer. A computer with an instruction set that is reduced in complexity.

**scalability**

The ability to add computing and storage resources to an existing system configuration without making software modifications or application conversions, and without shutting down the system.

**scratchpad memory**

A small memory for holding instructions and data that can be accessed quickly. Similar to cache memory. Also called scratch memory.

**script**

A data structure that points to various tests and exercisers and defines the order in which they are run.

**SCSI**

Small Computer System Interface. An ANSI-standard interface for connecting disks and other peripheral devices to computer systems. *See also* fast SCSI.

**SDD**

*See* symptom-directed diagnostics.

**self-test**

A test that is invoked automatically when the system powers up.

**serial control bus**

A two-conductor serial interconnect that is independent of the system bus. This bus links the processor modules, the I/O, the memory, the power subsystem, and the operator control panel. It reports any failed devices to the processor module so the processor module can illuminate LEDs on the operator control panel.

**shadowing**

*See* volume shadowing.

**shadow set**

In volume shadowing, the set of disks on which the data is duplicated. Access to a shadow set is achieved by means of a virtual disk unit. After a shadow set is created, applications and users access the virtual disk unit as if it were a physical disk. *See also* volume shadowing.

**shared**

With reference to a cache block in the cache of a system bus node, the cache block is valid, and it is valid in at least one other cache of another system bus node.

**SMP**

*See* symmetric multiprocessing.

**snoop**

For a cached node, the act of monitoring system bus transactions to determine whether the node has a copy of a cache line.

**snooping protocol**

A cache coherence protocol whereby all nodes on a common system bus monitor all bus activity. This allows a node to keep its copy of a particular datum up-to-date and/or supply data to the bus when it has the newest copy.

**soft error**

Soft errors are errors that are dynamically correctable.

**SPEC**

System Performance Evaluation Cooperative. A nonprofit organization that creates and maintains a suite of benchmarks to compare the performance of computer systems across vendors.

**SPECmark**

The geometric mean of the normalized results from the benchmarks defined by SPEC. Measures the performance of a single CPU. *See also* SPEC.

**SPECthruput**

A measure of how well a multiprocessing system scales with additional CPUs.

**SROM**

Serial read-only memory.

**stack**

An area of memory set aside for temporary data storage or for procedure and interrupt service linkages. A stack uses the last-in/first-out concept. As items are added to (pushed on) the stack, the stack pointer decrements. As items are retrieved from (popped off) the stack, the stack pointer increments.

**star topology**

A LAN configuration in which nodes are connected through a central point.

**storage array**

A group of mass storage devices, frequently configured as one logical disk.

**storage assembly**

All the components necessary to configure storage devices into a DEC 4000 AXP storage compartment. These components include the storage device, brackets, screws, shock absorbers, and cabling.

**storage backplane**

One of two backplanes in the BA640 enclosure. Fixed and removable media devices plug into this backplane. *See also* backplane.

**stripe set**

A group of physical disks that are used for disk striping. *See also* striping.

**striping**

A storage option that increases I/O performance. With disk striping, a single file is split between multiple physical disks. Read and write disk performance is increased by sharing input/output operations between multiple spindles, which allows an I/O rate greater than that of any one disk member of the stripe set. In striping, the loss of any one member of the stripe set causes loss of the set. Striping is particularly useful for applications that move large amounts of disk-based information, for example, graphic imaging. *Contrast with* RAID.

**superpipelined**

Describes a pipelined machine that has a larger number of pipe stages and more complex scheduling and control. *See also* pipeline.

**superscalar**

Describes a machine that issues multiple independent instructions per clock cycle.

**symmetric multiprocessing (SMP)**

A processing configuration in which multiple processors in a system operate as equals, dividing and sharing the workload. OpenVMS SMP provides two forms of multiprocessing: multiple processes can execute simultaneously on different CPUs, thereby maximizing overall system performance; and single-stream application programs can be partitioned into multistream jobs, minimizing the processing time for a particular program. *Contrast with* distributed processing.



**symptom-directed diagnostics (SDD)**

Online analysis of system errors to locate potential system faults. SDD helps isolate system problems.

**synchronization**

A method of controlling access to some shared resource so that predictable, well-defined results are obtained when operating in a multiprocessing environment.

**system backplane**

One of two backplanes in the BA640 enclosure. CPU, memory, I/O, Futurebus+, and power modules plug into this backplane. *See also* backplane.

**system bus**

The private interconnect used on the DEC 4000 AXP CPU subsystem. This bus connects the B2001 processor module, the B2002 memory module, and the B2101 I/O module.

**system disk**

The device on which operating system software resides.

**system fatal error**

An error that is fatal to the system operation, because the error occurred in the context of a system process or the context of an error cannot be determined.

**system integration**

The assembly of various hardware and software components into a single operating environment (usually to solve some business need).

**TCP/IP**

Transmission Control Protocol/Internet Protocol. A set of software communications protocols widely used in UNIX operating environments. TCP delivers data over a connection between applications on different computers on a network; IP controls how packets (units of data) are transferred between computers on a network.

**thickwire**

An IEEE standard 802.3-compliant Ethernet network made of standard Ethernet cable, as opposed to ThinWire Ethernet cable. Also called standard Ethernet. *Contrast with* ThinWire.

**ThinWire**

Digital's proprietary Ethernet products used for local distribution of data communications. *Contrast with* thickwire.

**TPS**

Transactions per second. Measurement of CPU and I/O performance of a system.

**UETP**

User Environment Test Package. An OpenVMS AXP software package designed to test whether the OpenVMS operating system is installed correctly. UETP puts the system through a series of tests that simulate a typical user environment, by making demands on the system that are similar to demands that might occur in everyday use.

**uninterruptible power supply (UPS)**

A battery-backup option that maintains AC power if a power failure occurs.

**unmasked write**

In memory, a write cycle that updates all locations of a nominal data block. That is, a hexword update to a cache block.

**UPS**

*See* uninterruptible power supply.

**victim**

Used in reference to a cache block in the cache of a system bus node. The cache block is valid but is about to be replaced due to a cache block resource conflict.

**victim processing**

The process of replacing the victim in the cache. *See also* victim.

**VMScluster system**

A highly integrated organization of Digital's VMS systems that communicate over a high-speed communications path. VMScluster configurations have all the functions of single-node systems, plus the ability to share CPU resources, queues, and disk storage.

**volume shadowing**

The process of maintaining multiple copies of the same data on two or more disk volumes. When data is recorded on more than one disk volume, you have access to critical data even when one volume is unavailable. Also called disk mirroring.

**warm swap**

The shutdown and removal and replacement of a failing DSSI disk from an active bus.

**Whetstone**

A benchmark for computer system performance that measures floating-point performance. Part of the suite of SPEC benchmarks.

**word**

Two contiguous bytes (16 bits) starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 15.

**writable global section**

A data structure (for example, FORTRAN global common) or shareable image section potentially available to all processes in the system.

**write back**

A cache management technique in which data from a write operation to cache is written into main memory only when the data in cache must be overwritten. This results in temporary inconsistencies between cache and main memory. *Contrast with* write through.

**write-enabled**

A device is write-enabled when data can be written to it. *Contrast with* write-protected.

**write-protected**

A device is write-protected when transfers are prevented from writing information to it. *Contrast with* write-enabled.

**write through**

A cache management technique in which data from a write operation is copied to both cache and main memory. Cache and main memory data is always consistent. *Contrast with* write back.

## A

---

- A32 Futurebus+ address space, 15-1
- A64 Futurebus+ address space, 15-2
- A-box control register, 4-23
- A-box IPRs, 4-19
- A-BOX\_CTL, 4-23
- Access doors, 1-5
- AC power connector port, 1-48
- ADDER, 18-47
- Address decoding
  - SCSI controller, 16-1
- Addresses
  - D-bus microcontroller, 1-43
  - I/O port to serial control bus, 1-43
  - memory module serial bus, 1-43
  - OCP serial bus, 1-43
  - PSC 8-bit register, 1-43
  - vterm 8-bit register, 1-43
  - 256X8 NVRAM, 1-43
- Address field, 19-11
  - for 1-MB cache, 19-11
  - for 4-MB cache, 19-11
- Addressing, 3-1
  - Lbus, 16-1
- Address mapping, 13-1
- Address space, 19-1
- Algorithms
  - bootstrap, 23-2
  - update vs. invalidate, 5-19
- alloc command, D-2
- Alpha AXP architecture, 3-1
- Alternate processor mode register, 4-24
- ALT\_MODE, 4-24
- Arbitration signals, 19-9
- ASTER, 4-17
- ASTRR, 4-15
- Asynchronous system trap enable register, 4-17
- Asynchronous trap request register, 4-15
- ASYNC\_RESET L, 19-17
- ASYNC\_RESET\_L, 21-5
- AUTO\_ACTION, 24-2
- Auxiliary serial line, 16-2
- Auxiliary serial port, 16-2
- AUX\_CTRL, 16-4
- AUX\_Data register, 16-4

## B

---

- BA640 enclosure
  - description, 1-5
- Backplane assembly, 1-10
- Backup cache, 5-1
  - access time, 5-17
  - behavior on tag control parity error, 9-7
  - behavior on tag parity errors, 9-7
  - initialization, 11-1
  - memory devices, 1-21
- Backup cache control register, 5-3
- Backup cache correctable error address register, 5-11
- Backup cache correctable error register, 5-8
- Backup cache tag register, 4-36
- Backup cache uncorrectable error address register, 5-15
- Backup cache uncorrectable error register, 5-12
- BARRIER, 7-1
- BARRIER transaction, 7-6
- Baud rates, 1-9
- Bcache
  - See Backup cache
- BCCEA, 5-11
- BCREQ L, 19-10
- BC\_TAG, 4-36
- bin command, D-3
- BIP flag, 24-2
- BIU, 19-1
- BIU\_ADDR, 4-32
- BIU\_CTL, 4-26
- BIU\_STAT, 4-31
- Block count, 23-5
- Block diagrams
  - I/O expansion area, 1-37
  - I/O module, 1-22
  - initialization, 21-2
  - KN430 CPU subsystem, 1-15
  - KN430 processor module, 1-18
  - mass storage area, 1-34
  - operator control panel, 1-7
  - power supply, 1-39
- Boot block calling interface, 23-4
- boot command, D-4
- Boot devices, 23-5

- Booting
  - BOOTP, 23–8
  - disk, 23–6
  - Ethernet, 23–6
  - MOP, 23–7
  - tape, 23–6
- Boot message register format, 17–26
- Boot parameters, 23–5
- BOOTP booting, 23–8
- BOOTP bootstrap, 23–10
- Boot processor, 23–2
- Bootstrap, 23–2
  - BOOTP, 23–10
  - cold start flag, 23–2
- Bootstrap in progress flag, 23–2
- Bus
  - Futurebus+, 1–36
  - secondary, 1–33
  - system, 1–29
- Bus interface control unit register, 4–26
- Bus interface unit, 19–1
- Bus interface unit address register, 4–32
- Bus interface unit status register, 4–31
- Bus master
  - serial control bus, 1–43
- Byte, 3–1
- Byte offset
  - as address argument, 25–8
- Byte stream, 25–7
  - See device*
- Byte swapping
  - using the eval command (example), 25–15

## C

- C<sup>3</sup> revision register, 6–18
- Cache block merge buffer, 5–17
- Cache block prefetch, lack of, 5–22
- Cache invalidate management, 8–1
- Cache line buffers, 1–22
- Cache line format, 19–3
- Cache protocol, 19–3
- CAD<127:0>, 19–10
- CAD parity coverage, 19–13
- CA L, 19–14
- Callbacks, 23–3
- cat command, D–6
- cbcc command, D–7
- CC, 4–25
- CCITT V.10, 16–2
- CCITT V.28, 16–2
- CC\_CTL, 4–25
- CDIRTY L, 19–15
- cdp command, D–9
- CD register, 16–5
- CERR1, 13–11
- CERR2, 13–17
- CERR3, 13–18

- CHALT L, 19–18
- check command, D–11
- Checksum, 23–5
- Chips
  - 85C30, 16–2
    - accessing internal registers, 16–3
  - 87C652
    - startup, 21–6
    - system bus interface, 6–1
- chmod command, D–12
- chown command, D–13
- CID<2:0>, 19–13
- CINT\_TIM, 19–17
- CIRQ L<1:0>, 19–17
- clear command, D–14
- Clock frequencies
  - 151.51 Mhz, 1–33
- Clocking signals, 19–18
- Clocks, 1–33
  - differential clock signals, 1–33
  - ECL, 1–33
  - generation, 1–33
  - interval timer interrupt, 1–31
  - PECL, 1–21
  - power/detect, 1–21
  - system bus, 1–21, 6–22
  - TOY registers, 1–25
- cmp command, D–15
- Command address cycle format, 19–6
- Command and address format, 19–9
- Command CID assignments, 19–13
- Commander, 1–29, 19–19
- Commands
  - alloc, D–2
  - bin, D–3
  - boot, D–4
  - cat, D–6
  - cbcc, D–7
  - cdp, D–9
  - check, D–11
  - chmod, D–12
  - chown, D–13
  - clear, D–14
  - cmp, D–15
  - continue, D–17
  - crc, D–18
  - date, D–19
  - deposit, D–20
  - dynamic, D–23
  - echo, D–25
  - edit, D–26
  - eval, D–29
  - examine, D–31
  - exer, D–34
  - exer\_read, D–40
  - exer\_write, D–41
  - exit, D–42
  - fbus\_diag, D–43

## Commands (Cont.)

- fbus\_sizer, D-45
- find\_field, D-46
- free, D-47
- grep, D-48
- hd, D-50
- help, D-51
- initialize, D-53
- io\_diag, D-54
- kill, D-55
- kill\_diags, D-56
- line, D-57
- ls, D-58
- memexer, D-59
- memexer\_mp, D-60
- memtest, D-61
- net, D-64
- nettest, D-67
- ps, D-70
- psc, D-71
- rm, D-72
- sa, D-73
- semaphore, D-74
- set, D-75, D-77
- sh, D-79
- show, D-80, D-83, D-84, D-85, D-87, D-89,  
D-90, D-91, D-92
- show\_status, D-93
- sleep, D-94
- sort, D-95
- sp, D-96
- start, D-97
- stop, D-98
- sw, D-99
- test, D-100
- tr, D-102
- uniq, D-103
- update, D-104
- used as abbreviations, 25-8
- wc, D-106
- Conditional branching
  - in if, while, until loops, 25-15
- Connectors
  - CPU clock, 19-8
  - memory, 1-29
- Console
  - preparing for BOOTP boot, 23-8
  - services, 1-46
- Console command option -n
  - specifying repeat count, 25-9
- Console commands
  - contrasted with VAX, 25-4
  - frequently used (table), 25-4
  - summarized, 25-18
- Console diagnostics
  - structure of, 1-45
- Console EEPROM, 1-46
- Console serial line, 16-2
- Console serial ports, 16-2
- Console shell operators (table), 25-16
- Console tasks
  - examining and depositing stuff, 25-7
  - examining registers, 25-9
- continue command, D-17
- Continuous system bus transactions, 1-30
- Control store, 5-1
- CON\_CTRL, 16-5
- CPU 2ID L, 19-18
- CPU clock connector
  - initialization signal, 19-17
- CPUG L, 19-10
- CPU LEDs
  - and CRESET L, 21-4
- CPU module
  - subsystems, 2-2
  - transactions, 7-1
- CPUREQ L, 19-10
- CPU specific buffer, 5-17
- CPU system bus register definitions, 6-1
- crc command, D-18
- Creating scripts
  - using the output creation operator (>), 25-13
- CRESET L, 6-21, 19-18
  - and the system bus clock, 6-22
- CRESET\_L, 21-5
- CSHARED L, 19-15
- CSR0, 5-3
- CSR1, 5-8
- CSR10, 6-13
- CSR11, 6-14
- CSR2, 5-11
- CSR3, 5-12
- CSR4, 5-15
- CSR5, 5-20
- CSR6, 6-1
- CSR7, 6-4
- CSR8, 6-10
- CSR9, 6-11
- CSR map, 17-1
- CSR read time, 1-33
- CSTALL0 L and CSTALL1 L, 19-14
- CSYS\_EVENT L, 19-17
- CTEST0, 18-24
- CTEST1, 18-25
- CTEST2, 18-26
- CTEST3, 18-27
- CTEST4, 18-28
- CTEST5, 18-29
- CTEST6, 18-31
- CTEST7, 18-32
- CTEST8, 18-37
- CUCERR L, 19-17
- CXACK L, 19-14
- Cycle counter control register, 4-25
- Cycle counter register, 4-25
- C\_ERR, 14-2

C\_ERR L, 19-16

## D

---

Data cache status register, 4-29  
Data integrity, 5-22  
Data store, 5-2  
Data translation buffer ASM register, 4-22  
Data translation buffer invalidate signal register, 4-22  
Data translation buffer page table entry register, 4-19  
Data translation buffer page table entry temporary register, 4-20  
Data translation buffer zap register, 4-22  
Data types, 3-1  
Data width options, 25-8  
Data wrap, 1-30  
Data write, 1-43  
date command, D-19  
DBC, 18-39  
D-bus, 6-19  
DC3, 1-42  
DC5, 1-42  
DCMD, 18-39  
DCNTL, 18-45  
DC on/off switch, 1-8  
DC\_STAT, 4-29  
DECchip 21064 microprocessor  
  A-box, 4-19  
  cycle types, 7-1  
  features, 4-1  
  startup, 21-6  
  supported data types, 4-1  
  transactions  
    allocate-invalid, 7-8  
    BARRIER, 7-6  
    cacheable, 7-7  
    fast external cache read hit, 7-2  
    fast external cache write hit, 7-3  
    FETCH, 7-7  
    FETCHM, 7-7  
    LDxL, 7-6  
    non-cacheable, 7-7  
    READ\_BLOCK, 7-4  
    STxC, 7-6  
    WRITE\_BLOCK, 7-5  
DECchip™  
  startup code, 1-45  
deposit command, D-20  
  accessing physical memory (example), 25-8  
Descriptor lists, 17-28  
Descriptors and buffers format, 17-28  
Device, 25-7  
Devices  
  platform-specific (list), 25-8  
DFIFO, 18-33  
DIAGCSR, 13-20

Diagnostic mode address registers, 13-5  
Diagnostic mode data registers, 13-4  
Diagnostics  
  structure of, 1-45  
DIEN, 18-43  
Disk power source voltages, 1-34  
DMA watchdog timer register, 18-44  
DMODE, 18-42  
DNA CSMA/CD counters and events support, 17-51  
DNAD, 18-40  
Documentation conventions, xxxii  
Driver initialization, 21-15  
Drivers  
  as access mechanisms, 25-7  
  for Alpha AXP devices (list), 25-7  
DSA, 18-23  
DSP, 18-40  
DSPS, 18-41  
DSTAT, 18-17  
DTBASM, 4-22  
DTBIS, 4-22  
DTBZAP, 4-22  
DTB\_PTE, 4-19  
DTB\_PTE\_TEMP, 4-20  
Duplicate primary data cache tag store, 5-18  
Duplicate tag error register, 5-20  
Duplicate tag store initialization, 11-3  
DWT, 18-45  
DWT register, 18-44  
dynamic command, D-23  
D\_floating, 3-5

## E

---

ECHADR<15:0>, 19-11  
echo command, D-25  
ECL clock, 1-33  
EDC  
  See Error Detection and Correction  
edit command, D-26  
EEPROM, 16-2  
  console, 1-46  
  for serial control bus, 1-47  
  write cycle time, 1-43  
EIA 423, 16-2  
Environment variables, 23-2  
Error detection and correction, 4-37  
Error logging EEPROM, 6-23  
Errors  
  backup cache tag control parity, 9-7  
  bcache data single bit EDC, 9-7  
  bcache data uncorrectable EDC, 9-7  
  bcache single bit EDC, 9-15  
  bcache tag or tag control parity, 9-8  
  bcache tag parity, 9-15  
  current CPU BCTC parity error processor detected, 9-7

## Errors (Cont.)

- DECchip 21064 microprocessor
  - data bus uncorrectable EDC, 9-8
- DECchip 21064 microprocessor
  - data bus single bit EDC, 9-8
- EDC, 4-29
- EDC uncorrectable, 9-9
- hardware 0, 9-15
- hardware interrupt, 14-2
- invalidate system bus address, 9-9
- miscellaneous CPU, 9-9
- parity, 4-29
- system bus parity, 9-9, 9-16
- tag control store, 9-8
- tag store, 9-8
- TGEC heartbeat collision, 17-34
- uncorrectable EDC, 9-15

Ethernet

- booting, 23-6
- interface selection, 17-1

Ethernet adapter, 17-1

Ethernet controllers, 1-24

Ethernet ports, 1-48

Ethernet station address ROM, 1-46, 18-48

eval command, D-29

examine command, D-31

- accessing pmem device (example), 25-8
- referencing registers, 25-9
- used as abbreviation (example), 25-8
- with address implied, 25-8
- with explicit address, 25-8

Exception address register, 4-8

Exception handling, 9-2

Exceptions, 9-1

- causes, 9-1
- machine checks, 9-2
- multiple, 9-1
- PALcode 0020 entry characteristics, 9-3
- PAL priority level, 9-3
- PAL routine behavior, 9-7
- parse tree PALcode entry 0020, 9-3

Exception summary register, 4-11

EXC\_ADDR, 4-8

EXC\_SUM, 4-11

exer command, D-34

exer\_read command, D-40

exer\_write command, D-41

exit command, D-42

## F

- F0-F31, 3-12
- F31, 3-12
- Fan startup, 21-4
- Fast External Cache Write Hit, 7-3
- Fault management, 10-1
- fbus\_diag command, D-43
- fbus\_sizer, D-45

- FEPROM, 1-24, 18-48
  - code structure, 1-45
  - defined, 1-45
- FEPROMS, 1-45
- FEPROM unloading, 21-10
- FERR1, 13-22
- FERR2, 13-23
- FETCH, 5-22
- FETCH cycle, 7-1
- FETCHM, 5-22
- FETCHM cycle, 7-1
- FETCHM transaction, 7-7
- FETCH transaction, 7-7
- FEU, 1-41
- FHVECT, 13-22
- FIFO queue, 14-1
- Fill address register, 4-33
- Fill syndrome register, 4-34
- FILL\_ADDR, 4-33
- FILL\_SYNDROME, 4-34
- Filtering output
  - using pipes and grep, 25-10
- find\_field command, D-46
- Firmware
  - console emulation and diagnostics, 1-45
  - defined, 1-43
  - overview, 1-43
  - upgrade time, 1-45
- FIVECT, 13-21
- Flags, 23-5
  - BIP, 24-2
  - bootstrap in progress, 23-2
  - cold start, 23-2
  - inverse filtering, 17-40
- Flash-erase PROM, 18-48
- Flash PROM
  - See FEPROM
- Floating-point registers, 3-12
- Flow control
  - syntax for constructs, 25-14
- Flush instruction cache ASM register, 4-22
- Flush instruction cache register, 4-22
- FLUSH\_IC, 4-22
- FLUSH\_IC\_ASM, 4-22
- FMBPR, 13-19
- free command, D-47
- Futurebus+
  - routing, 1-13
- Futurebus+ adapters, 15-1
- Futurebus+ address space, 15-1, 15-2
- Futurebus+ interrupts, 14-1
- Futurebus+ mailbox command field, 13-31
- Futurebus+ mailbox status field, 13-33
- Futurebus+ slots, 1-36
- F\_floating, 3-3
- F\_floating load exponent mapping, 3-3



## G

---

grep command, 25–10, D-48  
    *See also* pipe( | ) command  
G\_floating, 3–4

## H

---

H7178, 1–42  
H7179, 1–42  
H7851, 1–41  
H7853, 1–41  
Halt switch, 1–9  
    and system interrupts, 14–3  
Hard errors  
    defined, 1–31  
Hardware error interrupts, 14–2  
Hardware interrupt enable register, 4–16  
Hardware interrupt request register, 4–13  
Hardware invalidates, 5–22  
Hardware restart parameter block, 23–3  
hd command, D-50  
Heartbeat collision check, 17–34  
help command, D-51  
hex dump (example), 25–9  
hex dump command  
    dumping memory, 25–9  
Hexword read transactions  
    non-cacheable, 1–29  
Hexword write transactions  
    non-cacheable, 1–29  
HIER, 4–16  
HIRR, 4–13  
HWRPB, 23–3  
HWRPBSQ\_RESTART, 24–2

## I

---

I/O  
    for removable media, 1–2  
I/O expansion slots, 1–36  
I/O module  
    diagnostic address register map, 13–5  
    functional diagram, 1–22  
    Futurebus+ diagnostic register map, 13–3  
    hardware error interrupt conditions, 14–2  
    lbus diagnostic register map, 13–3  
    register map, 13–2  
I/O redirection  
    to other devices, 25–11  
I-box internal processor registers, 4–2  
ICCSR, 4–4  
ID, 18–8  
    for environment variables, 23–2  
Idle bus, 19–13  
Idle cycles, 1–30  
Imperfect filtering, 17–42

Initialization, 21–6  
    and reset, 11–3  
    and the backup cache, 11–1  
    and the CPU clocks, 11–3  
    of the duplicate tag store, 11–3  
    of the system bus interface, 11–3  
Initialization flow diagram, 21–7, 21–9, 21–11  
initialization signal, 19–17  
initialize, D-53  
Instruction cache control and status register, 4–4  
Instruction formats, 3–12  
Instruction translation buffer ASM, 4–10  
Instruction translation buffer IS register, 4–10  
Instruction translation buffer page table entry register, 4–3  
Instruction translation buffer page table entry temporary register, 4–7  
Instruction translation buffer zap register, 4–10  
Interleave, 1–30  
Internal processor register format, 4–8  
Internal processor registers  
    and initialization, 11–1  
Interprocessor interrupt request register, 6–14  
Interrupt encoding, 14–1  
Interrupt handling, 9–10  
Interrupts, 1–31, 9–1, 14–1  
    and the interrupt vector register, 14–2  
    CIRQ\_L[0], 14–2  
    entry into PALcode, 9–10  
    Futurebus+, 14–1  
    hardware error, 14–2  
    local I/O, 14–2  
    non-processor generated, 9–9  
    processor generated, 9–1  
    TGEC, 17–45  
Interrupts and exceptions  
    backup cache tag parity error, 9–7  
Invalidates  
    system bus caused, 8–1  
Inverse\_filtering flag, 17–40  
IOCSR, 13–6  
IOG L, 19–10  
IOREQ L, 19–10  
io\_diag, D-54  
ISTAT, 18–34, 18–35  
ITBASM, 4–10  
ITBIS, 4–10  
ITBZAP, 4–10  
ITB\_PTE, 4–3  
ITB\_PTE\_TEMP, 4–7

## K

---

KFA40 I/O module, 1–22  
kill command, D-55  
kill\_diags command, D-56  
KN430 CPU subsystem, 1–14  
KN430 processor, 2–2

## L

---

- Lbus addressing, 16-1
- Lbus mailbox command field, 13-35
- Lbus mailbox status field, 13-36
- LCRC, 18-38
- LDxL cycle, 7-2
- LDxL transaction, 7-6
- LERR1, 13-26
- LERR2, 13-27
- line command, D-57
- LINT, 13-24
- Listing contents of a script
  - using cat command, 25-12
- LMBPR, 13-18
- Local I/O bus
  - routing, 1-13
- Local I/O devices, 16-1
- Local I/O interrupts, 14-2
- Lock registers, 3-12
- Longword, 3-2
- Longword integer format in FPU, 3-9
- Loopback
  - and the TGEC, 17-50
- ls command, D-58

## M

---

- Machine checks, 9-2
- Mailbox, 13-28
  - defined, 1-33
  - format, 13-29
  - mapping, 16-2
- Mailbox pointer register, 13-29
- Mass storage
  - front panel, 1-36
- Mass storage area, 1-34
- memexer command, D-59
- memexer\_mp command, D-60
- Memory
  - bad on power-up, 11-3
  - configuring, 26-9
- Memory exchange transactions, 19-26
- Memory-like locations
  - defined, 7-7
- Memory management control and status register, 4-21
- Memory move
  - and SFBR, 18-15
- Memory prefetch register, 3-12
- Memory slot ID code, 19-19
- Memory testing, 21-13
- Memory write transactions, 19-28
- memtest command, D-61
- Microcontroller
  - startup firmware, 1-45
- MID<1:0>, 19-19

- Miss address register, 6-17
- MM\_CSR, 4-21
- Module order, 1-17
- Modules
  - front end unit, 1-41
  - KFA40, 1-22
  - KN430 processor, 1-17, 1-19
  - memory, 1-26
  - power system controller, 1-41
  - storage backplane, 1-11
  - system backplane, 1-13
  - 5 volt converter, 1-42
  - 3 volt converter, 1-42
- Monarch processor, 15-1
- Monitoring status
  - using ps command (example), 25-12
  - using show-status command (example), 25-12
- MOP
  - must transact operaton, 23-7
- Mop booting, 23-7
- Multiple exceptions, 9-1
- Multiprocessor configuration CSR definitions, 6-13

## N

---

- net command, D-64
- nettest command, D-67
- Network listening, 23-7
- Noncacheable address sapce write transaction, 19-30
- Non-memory like locations
  - defined, 7-7
- Non-stalled transaction times, 19-2
- Nonvolatile EEPROM, 6-23
- Null, 19-24
- Numbering Conventions, xxxii
- NUT, 19-24

## O

---

- OCP
  - See* Operator control panel
- OCP halt request buffer, 6-23
- On-line help
  - available topics, 25-6
  - brief, 25-5
    - | more, 25-7
  - multiple topics, 25-6
  - screen display, 25-5
  - wildcarding (example), 25-7
- Operating system restart, 24-2
- Operator control panel, 1-6
- Ownership flag, 1-31

## P

---

- PALcode entry characteristics, 9–10
- PAL priority level, 9–10
- PAL\_BASE, 4–13
- PAL\_TEMP, 4–28
- Parse trees
  - PALcode entry, 9–11
- PC, 3–11
- PCD8584, 18–48
- PECL, 1–21
- Perfect filtering, 17–40
- PHI1 and PHI1 L, 19–18
- PHI3 and PHI3 L, 19–18
- Physical address space, 1–32, 1–33
- Physical cache, 8–1
- Pipe (|) command, 25–10
  - See also* grep command
- pmem:
  - physical memory, 25–8
- Ports
  - AC power, 1–48
  - Ethernet, 1–48
  - SCSI storage expansion, 1–48
  - serial line, 1–47
- Power-down mode, 18–25
- Power supply
  - and system event, 14–3
  - indicators, 1–38
- Power-up
  - with bad main memory, 11–3
- Power-up baud rate switch, 1–9
- Power-up sequence, 11–3
- Primary processor functions, 1–18
- Privileged architecture library base register, 4–13
- Processor caused invalidates, 8–1
- Processor initiated transactions, 7–2
- Processor mailbox register, 6–13
- Processor module
  - configurations, 1–14
  - differences, 1–14
- Processor registers
  - symbolic reference, 25–10
- Processor status register, 4–11
- Process registers
  - pc, sp, ps (example), 25–10
- Protocol signals, 19–10
- PS, 3–11, 4–11
- PSC, 1–41
  - startup firmware, 1–44
- psc command, D–71
- ps command, D–70

## Q

---

- Quadword, 3–2
- Quadword integer format in FPU, 3–9

## R

---

- R30, 3–11
- R31, 3–11
- RAM
  - for SCSI controllers, 1–46
- RDES0 word, 17–29
- RDES1 word, 17–31
- RDES2 word, 17–32
- RDES3 word, 17–32
- Read
  - to secondary bus, 13–36
- Read and write transactions, 19–20
- READ\_BLOCK, 7–1, 7–4
- Reboot
  - limitations on a secondary processor, 24–2
- Receive descriptors, 17–28
- Receive descriptor status validity, 17–33
- Redirecting output, 25–11
  - using append operator (>>), 25–14
  - using redirection operator (>), 25–11
- Redirecting output (example), 25–11
- Registers
  - A-box control, 4–23
  - adder sum output, 18–47
  - Alpha IPRs, 3–12
  - Alpha optional, 3–12
  - alternate processor mode, 4–24
  - asynchronous system trap enable, 4–17
  - asynchronous trap request, 4–15
  - AUX\_CTRL, 16–4
  - AUX\_Data, 16–4
  - backup cache control, 5–3
  - backup cache correctable error, 5–8
  - backup cache correctable error address, 5–11
  - backup cache tag, 4–36
  - backup cache uncorrectable error, 5–12
  - backup cache uncorrectable error address, 5–15
  - bus interface control unit, 4–26
  - bus interface unit address, 4–32
  - bus interface unit status, 4–31
  - C<sup>3</sup> revision, 6–18
  - chip test eight, 18–37
  - chip test five, 18–29
  - chip test four, 18–28
  - chip test one, 18–25
  - chip test seven, 18–32
  - chip test six, 18–31
  - chip test three, 18–27
  - chip test two, 18–26
  - chip test zero, 18–24
  - console control, 16–5

## Registers (Cont.)

- console data, 16-5
- cycle counter, 4-25
- cycle counter control, 4-25
- data cache status, 4-29
- data structure address, 18-23
- data translation buffer ASM, 4-22
- data translation buffer invalidate signal, 4-22
- data translation buffer page table entry, 4-19
- data translation buffer page table entry
  - temporary, 4-20
- data translation buffer zap, 4-22
- descriptor list addresses register, 17-9
- diagnostic control and status, 13-20
- DMA byte counter, 18-39
- DMA command, 18-39
- DMA control, 18-45
- DMA FIFO, 18-33
- DMA interrupt enable, 18-43
- DMA mode, 18-42
- DMA next data address, 18-40
- DMA scripts pointer save, 18-41
- DMA status, 18-17
- DMA watchdog timer, 18-45
- DMS scripts pointer, 18-40
- duplicate tag error, 5-20
- exception address, 4-8
- exception summary, 4-11
- explicit reference, 25-9
- fill address, 4-33
- fill syndrome, 4-34
- floating-point, 3-12
- flush instruction cache, 4-22
- flush instruction cache ASM, 4-22
- Futurebus+ error 1, 13-22
- Futurebus+ error 2, 13-23
- Futurebus+ halt vector, 13-22, 14-4
- Futurebus+ interrupt vector, 13-21
- Futurebus+ mailbox pointer, 13-19
- GTEC command and mode, 17-17
- hardware interrupt enable, 4-16
- hardware interrupt request, 4-13
- I/O control and status, 13-6, 14-4
- I/O diagnostic mode address, 13-5
- implicit reference, 25-9
- instruction cache control and status, 4-4
- instruction translation buffer ASM, 4-10
- instruction translation buffer page table entry, 4-3
- instruction translation buffer page table entry
  - temporary, 4-7
- instruction translation buffer zap, 4-10
- integer, 3-11
- interprocessor interrupt, 1-31
- interprocessor interrupt request, 6-14
- interprocessor mailbox, 1-31
- interrupt status, 18-34, 18-35
- intr-flag, 3-12

## Registers (Cont.)

- instruction translation buffer IS, 4-10
- Lbus error 1, 13-26
- lbus error 2, 13-27
- Lbus mailbox pointer, 13-18
- local interrupt, 13-24, 14-2
- lock, 3-12
- longitudinal parity, 18-38
- mailbox pointer, 13-29
- memory management control and status, 4-21
- memory prefetch, 3-12
- miss address, 6-17
- multiprocessor configuration CSR, 6-13
- noncacheable address space silo, 1-31
- privileged architecture library base, 4-13
- processor mailbox, 6-13
- processor status, 3-11, 4-11
- program counter, 3-11
- readable noncacheable address space device
  - request, 1-31
- receive polling register, 17-8
- scratch register, 18-41
- SCSI bus control lines, 18-16
- SCSI bus data lines, 18-16
- SCSI chip ID, 18-9
- SCSI control, 18-4
- SCSI control 1, 18-7
- SCSI Destination ID 02 (01), 18-8
- SCSI first byte received, 18-15
- SCSI input data latch, 18-15
- SCSI interrupt enable, 18-9
- SCSI output control latch, 18-13
- SCSI output data latch, 18-13
- SCSI status 0, 18-19
- SCSI status 1, 18-21
- SCSI status 2, 18-22
- SCSI transfer, 18-10
- serial line clear, 4-9
- serial line receive, 4-10
- serial line transmit, 4-18
- silo, 1-31
- software interrupt enable, 4-17
- software interrupt request, 4-15
- symbolic reference, 25-9
- system bus, 6-1
- system bus control, 6-1
- system bus error, 6-4
- system bus error 1, 13-11
- system bus error 2, 13-17
- system bus error 3, 13-18
- system bus error address high, 6-11
- system bus error address low, 6-10
- system interrupt clear, 6-14
- temporary stack, 18-33
- TGEC boot message, 17-26
- TGEC command and status, 17-3
- TGEC diagnostic, 17-28

## Registers (Cont.)

- TGEC identification and missed frame count, 17-25
- TGEC reserved, 17-23
- TGEC status, 17-12
- TGEC system base, 17-22
- TGEC vector address, 17-5
- TGEC watchdog timer, 17-23
- TOY, 1-25, 1-46
- TOY A (TOYA), 16-7
- TOY B (TOYB), 16-9
- TOY C (TOYC), 16-11
- TOY clock, 16-6
- TOY D (TOYD), 16-12
- translation buffer control, 4-19
- Transmit polling demand, 17-7
- VAX compatibility registers, 3-12
- virtual address, 4-22
- Relative addresses
  - symbolic reference (example), 25-9
- Remote boot
  - and system event, 14-3
- Removable-media storage, 1-2
- Removable-media storage I/O, 1-2
- REQ\_PROGRAM, 23-7
- Reset, 21-5
  - and the OCP, 21-5
  - of the TGEC, 17-44
- Reset switch, 1-8
- Responder, 19-19
- Response and interrupt signals, 19-14
- Restart, 24-2
- Restart and boot code, 1-46
- Restart failure, 24-2
- Restart parameters, 24-3
- rm command, D-72
- ROM
  - Ethernet station address, 1-46, 18-48
- Round-robin, 1-30
- RS-232, 16-2
- Running tasks
  - in background mode "&", 25-11
  - in background mode "&" (example), 25-11

## S

---

- sa command, D-73
- SBCL, 18-16
- SBDL, 18-16
- SCID, 18-9
- SCL signal, 19-19
- SCNTL0, 18-4
- SCNTL1, 18-7
- SCRATCH, 18-41
- Script RAM Buffer, 18-47
- SCSI controller
  - revision level, 18-37
- SCSI controller address decoding, 16-1

- SCSI controllers static RAM, 1-46
- SCSI driver
  - sizing, 26-23
  - startup, 26-23
- SCSI/DSSI Adapters, 18-1
- SCSI/DSSI initialization, 21-15
- SCSI storage expansion ports, 1-48
- SDA signal, 19-19
- SDID, 18-8
- Secondary buses, 1-33
- Secondary bus read, 13-36
- Secondary bus write, 13-37
- Secondary I/O space, 13-28
- semaphore command, D-74
- Serial bus bus
  - interface register map, 18-48
- Serial control bus
  - EEPROM, 1-47
  - interface, 1-21, 18-48
  - node addresses, 1-43
  - routing, 1-13
- Serial control bus signals, 19-19
- Serial line
  - units, 1-25
- Serial line clear register, 4-9
- Serial line receive register, 4-10
- Serial lines
  - differences, 1-47
  - features, 1-47
  - ports, 1-47
- Serial line transmit register, 4-18
- Serial line unit register map, 16-2
- Serial line units, 16-2
- set command, D-75, D-77
- SFBR, 18-15
- sh, D-79
- show command, D-80, D-83, D-84, D-85, D-87, D-89, D-90, D-91, D-92
- show\_status command, D-93
- SIDL, 18-15
- SIEN, 18-9
- SIER, 4-17
  - and corresponding SIRR bits, 4-15
- SIRR, 4-15
- sleep command, D-94
- SL\_CLR, 4-9
- SL\_RCV, 4-10
- SL\_XMIT, 4-18
- Snooping protocol, 1-29, 19-4
- SOCL, 18-13
- SODL, 18-13
- Soft errors
  - defined, 1-31
- Software interrupt enable register, 4-17
- Software interrupt request register, 4-15
- sort command, D-95
- sp command, D-96
- SSTAT0, 18-19

- SSTAT1, 18–21
- SSTAT2, 18–22
- Stall signals, 19–21
- start command, D–97
- Start LBN, 23–5
- Startup
  - 87C652, 21–6
- Status LEDs, 1–10
- stop command, D–98
- Stopping a process
  - using the kill command (example), 25–13
- Storage backplane, 1–11
- STQC cycle, 7–2
- STxC cycle, 7–2
- STxC transaction, 7–6
- sw command, D–99
- Switches
  - DC on/off, 1–8
  - halt, 1–9
  - power-up baud rate, 1–9
  - reset, 1–8
- SXFER, 18–10
- System backplane, 1–13
- System bus, 1–29, 19–1
  - arbiter, 6–22, 19–2
  - arbitration, 1–30
  - clock, 1–21
  - CRESET L generation, 6–22
  - cycles, 5–17
  - cycle time, 19–6
  - idle cycles, 1–30
  - interface, 1–21
  - interface chip, 6–1
    - initialization, 11–3
  - routing, 1–13
  - signals, 19–6, 19–7
  - supported transactions, 1–29
  - test, 26–14
  - transactions, 19–1, 19–2
- System bus address map, 13–1
- System bus control register, 6–1
- System bus error address high register, 6–11
- System bus error address low register, 6–10
- System bus error register, 6–4
- System bus stall, 13–28
- System bus transactions, 7–1, 7–9
  - CPU as bystander, 7–10
  - CPU as commander, 7–10
  - CPU as responder, 7–10
  - null, 1–30
  - reasons for monitoring, 7–1
- System event
  - causes, 1–31
- System event interrupts, 14–3
- System features, 1–1
- System features summary, 1–3
- System I/O, 1–2
- System I/O space, 13–2

- System initialization, 21–6
- System interrupt clear register, 6–14
- System memory space, 13–1
- S\_floating, 3–6

## T

---

- Tag store, 5–2
- Tape booting, 23–6
- TB\_CTL, 4–19
- TB\_TAG, 4–2
- TDES0 word, 17–34
- TDES1 word, 17–35
- TDES2 word, 17–37
- TDES3 word, 17–37
- TEMP, 18–33
- test command, D–100
- TGEC, 17–1
  - See Ethernet controllers
    - virtual to physical address translation, 17–28
- TGEC command and status registers, 17–3
- TGEC CSR0, 17–5
- TGEC CSR1, 17–7
- TGEC CSR10, 17–25
- TGEC CSR11, 17–26
- TGEC CSR12, 17–26
- TGEC CSR13, 17–26
- TGEC CSR14, 17–28
- TGEC CSR15, 17–28
- TGEC CSR2, 17–8
- TGEC CSR3, 17–9
- TGEC CSR4, 17–9
- TGEC CSR5, 17–12
- TGEC CSR5 and interrupts, 17–45
- TGEC CSR5 status report, 17–15
- TGEC CSR6, 17–17
- TGEC CSR7, 17–22
- TGEC CSR8, 17–23
- TGEC CSR9, 17–23
- TGEC CSR read, 17–4
- TGEC CSR write, 17–4
- TGEC loopback, 17–50
- TGEC physical CSRs, 17–4
- TGEC programming, 17–2
- TGEC reception process, 17–46
- TGEC register map, 17–1
- TGEC reset, 17–44
- TGEC setup frame, 17–38
- TGEC startup procedure, 17–46
- TGEC states, 17–2
- TGEC transmission process, 17–48
- TGEC transmit descriptors, 17–33
- TGEC virtual CSRs, 17–4
- Time domains, 1–33
- Time-of-year clock, 16–6
- TOYA, 16–7
- TOYB, 16–9
- TOYC, 16–11

TOY clock, 16–6  
TOYD, 16–12  
TOY registers, 1–46  
TRANS<2:0>, 19–12  
Transaction counter, 1–30  
Transactions  
  CPU module, 7–1  
  memory exchange, 19–26  
  memory write, 19–28  
  noncacheable address space write, 19–30  
  null, 19–24  
  processor initiated, 7–2  
  read and read exclusive, 19–20  
Transaction timing, 19–19  
Translation buffer control register, 4–19  
Transmit descriptor status validity, 17–37  
tr command, D–102  
two-conductor interconnect, 1–43  
T\_floating, 3–7

## U

---

uniq command, D–103  
update, D–104  
Using pipes (|) and grep

  to filter output, 25–10  
Using quotes  
  to write longer scripts, 25–14

## V

---

VA, 4–22  
VAX compatibility registers, 3–12  
Virtual address register, 4–22  
Virtual cache, 8–1  
Voltages, 1–42  
  +24 and -24 VDC, 1–41  
  disk power source, 1–34

## W

---

wc command, D–106  
Word, 3–1  
Word tearing, 5–17  
Write  
  to non-cacheable addresses, 7–7  
  to secondary bus, 13–37  
WRITE\_BLOCK cycle, 7–1  
WRITE\_BLOCK transaction, 7–5