

# **VAX 6000 Model 600 System Technical User's Guide**

Order Number: EK-660EA-TM.001

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

**Digital Equipment Corporation**

---

**First Printing, January 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, if any, 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 or equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

---

Copyright ©1992 by Digital Equipment Corporation

All Rights Reserved.  
Printed in U.S.A.

---

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	PDP	VAXcluster
DEC LANcontroller	ULTRIX	VAXELN
DECnet	UNIBUS	VMS
DECUS	VAX	XMI
DWMVA	VAXBI	<b>digital</b> <sup>™</sup>

This document was prepared using VAX DOCUMENT, Version 1.2

---

# Contents

---

PREFACE	xv
---------	----

---

## CHAPTER 1 THE VAX 6000 MODEL 600 SYSTEM 1-1

---

1.1	SYSTEM ARCHITECTURE	1-2
1.2	SAMPLE SYSTEM	1-4
1.3	SYSTEM FRONT VIEW	1-6
1.4	SYSTEM REAR VIEW	1-8
1.5	SUPPORTED ADAPTERS	1-10

---

## CHAPTER 2 KA66A CPU MODULE 2-1

---

2.1	OVERVIEW AND BLOCK DIAGRAM	2-2
2.1.1	NVAX CPU Chip	2-3
2.1.1.1	Ibox • 2-4	
2.1.1.2	Ebox and Microsequencer • 2-5	
2.1.1.3	Fbox • 2-5	
2.1.1.4	Mbox • 2-5	
2.1.1.5	Cbox • 2-6	
2.1.2	Backup Cache	2-6
2.1.3	NEXMI Chip, System Support, and XMI Interface	2-7
2.2	CPU SECTION	2-8
2.2.1	Data Types	2-8
2.2.2	Instruction Set	2-9
2.2.3	Physical Address Space	2-10
2.2.4	Memory Management	2-11
2.2.4.1	Translation Buffer • 2-12	
2.2.4.2	Memory Management Control Registers • 2-13	

# Contents

2.2.5	<b>Exceptions and Interrupts</b> _____	2-14
2.2.5.1	Interrupts • 2-15	
2.2.5.2	Exceptions • 2-17	
2.2.5.3	Unique Exceptions • 2-18	
2.2.5.4	Console Halt • 2-23	
2.2.6	<b>System Control Block</b> _____	2-25
2.2.7	<b>Process Structure</b> _____	2-28
<hr/>		
2.3	<b>CACHE OVERVIEW</b> _____	2-30
2.3.1	<b>Writeback Cache and Ownership Concepts</b> _____	2-30
2.3.2	<b>Virtual Instruction Cache</b> _____	2-31
2.3.3	<b>Primary Cache</b> _____	2-32
2.3.4	<b>Backup Cache</b> _____	2-32
2.3.4.1	Backup Cache Operating Modes • 2-33	
2.3.4.2	Cbox Internal Processor Registers • 2-33	
2.3.4.3	Tag Store and Data RAM Control • 2-36	
2.3.4.4	Backup Cache Is OFF • 2-36	
2.3.4.5	Backup Cache Is in Force Hit Mode • 2-37	
2.3.4.6	Backup Cache Is in Error Transition Mode • 2-37	
2.3.4.7	How to Turn the B-Cache Off • 2-38	
2.3.4.8	How to Turn the B-Cache On • 2-39	
2.3.5	<b>Cache Initialization</b> _____	2-40
<hr/>		
2.4	<b>NVAX BOX DESCRIPTIONS</b> _____	2-42
2.4.1	<b>Ibox</b> _____	2-42
2.4.1.1	Effects of Ibox Pipelining • 2-42	
2.4.1.2	Branch Prediction Unit • 2-43	
2.4.2	<b>Ebox</b> _____	2-43
2.4.3	<b>Fbox</b> _____	2-44
2.4.4	<b>Mbox</b> _____	2-44
2.4.4.1	Translation Buffer Tag Fills • 2-45	
2.4.4.2	Translation Buffer PTE Fills • 2-45	
2.4.4.3	Recording Mbox Errors • 2-47	
2.4.5	<b>Cbox</b> _____	2-47
<hr/>		
2.5	<b>KA66A TOY CLOCK AND INTERVAL TIMER</b> _____	2-49
2.5.1	<b>Time-of-Day Register (TODR)</b> _____	2-49
2.5.2	<b>Programmable Interval Clock</b> _____	2-49
2.5.3	<b>Time-of-Year Clock</b> _____	2-50
<hr/>		
2.6	<b>XMI INTERFACE</b> _____	2-54
2.6.1	<b>XMI Address Space</b> _____	2-54
2.6.1.1	XMI Memory Space • 2-54	
2.6.1.2	XMI I/O Space • 2-54	

2.6.2	<b>XMI Transaction Generation/Response Tables</b>	2-56
2.6.3	<b>Invalidates</b>	2-57
2.6.4	<b>Writeback Queues</b>	2-58
2.6.5	<b>Lockout Avoidance</b>	2-58
2.6.6	<b>Interrupts and IDENTs</b>	2-59
2.6.6.1	Responding to XMI Interrupts • 2-59	
2.6.6.2	Generating the IDENT • 2-59	
2.6.6.3	XMI Device Interrupt Priority • 2-60	
2.6.6.4	Implied Vector Interrupts (IVINTR) • 2-60	
2.6.6.4.1	IVINTR Mask Generation • 2-60	
2.6.6.4.2	Interprocessor IVINTR (IP IVINTR) Response • 2-60	
2.6.6.4.3	Write Error IVINTR (WE IVINTR) Response • 2-61	
2.6.7	<b>XMI Registers</b>	2-61
<hr/>		
2.7	<b>KA66A CPU MODULE REGISTERS</b>	2-63
2.7.1	<b>IPR and Cache Addressing</b>	2-63
2.7.2	<b>Internal Processor Registers</b>	2-67
	CPU IDENTIFICATION REGISTER (CPUID)	2-71
	INTERVAL CLOCK CONTROL AND STATUS REGISTER (ICCS)	2-72
	NEXT INTERVAL COUNT REGISTER (NICR)	2-74
	INTERVAL COUNT REGISTER (ICR)	2-75
	CONSOLE RECEIVER CONTROL AND STATUS REGISTER (RXCS)	2-76
	CONSOLE RECEIVER DATA BUFFER REGISTER (RXDB)	2-78
	CONSOLE TRANSMITTER CONTROL AND STATUS REGISTER (TXCS)	2-80
	CONSOLE TRANSMITTER DATA BUFFER REGISTER (TXDB)	2-82
	MACHINE CHECK ERROR SUMMARY REGISTER (MCSR)	2-83
	CONSOLE SAVED PROGRAM COUNTER REGISTER (SAVPC)	2-84
	CONSOLE SAVED PROCESSOR STATUS LONGWORD (SAVPSL)	2-85
	I/O RESET REGISTER (IORESET)	2-90
	SYSTEM IDENTIFICATION REGISTER (SID)	2-91
	PATCHABLE CONTROL STORE CONTROL REGISTER (PCSCR)	2-93
	EBOX CONTROL REGISTER (ECR)	2-96
	CBOX CONTROL REGISTER (CCTL)	2-99
	BACKUP CACHE DATA ECC REGISTER (BCDECC)	2-103
	BACKUP CACHE ERROR TAG STATUS REGISTER (BCETSTS)	2-105
	BACKUP CACHE ERROR TAG INDEX REGISTER (BCETIDX)	2-108
	BACKUP CACHE ERROR TAG REGISTER (BCETAG)	2-109
	BACKUP CACHE ERROR DATA STATUS REGISTER (BCEDSTS)	2-111
	BACKUP CACHE ERROR DATA INDEX REGISTER (BCEDIDX)	2-114

# Contents

	BACKUP CACHE ERROR DATA ECC REGISTER (BCEDECC)	2-115	
	CBOX ERROR FILL ADDRESS REGISTER (CEFADR)	2-117	
	CBOX ERROR FILL STATUS REGISTER (CEFSTS)	2-118	
	NDAL ERROR STATUS REGISTER (NESTS)	2-123	
	NDAL ERROR OUTPUT ADDRESS REGISTER (NEOADR)	2-126	
	NDAL ERROR OUTPUT COMMAND REGISTER (NEOCMD)	2-127	
	NDAL ERROR DATA HIGH REGISTER (NEDATHI)	2-130	
	NDAL ERROR DATA LOW REGISTER (NEDATLO)	2-132	
	NDAL ERROR INPUT COMMAND REGISTER (NEICMD)	2-133	
	VIC MEMORY ADDRESS REGISTER (VMAR)	2-135	
	VIC TAG REGISTER (VTAG)	2-137	
	VIC DATA REGISTER (VDATA)	2-139	
	IBOX CONTROL AND STATUS REGISTER (ICSR)	2-140	
	PHYSICAL ADDRESS MODE REGISTER (PAMODE)	2-142	
	MEMORY MANAGEMENT EXCEPTION ADDRESS REGISTER (MMEADR)	2-143	
	MEMORY MANAGEMENT EXCEPTION PTE ADDRESS REGISTER (MMEPTE)	2-144	
	MEMORY MANAGEMENT EXCEPTION STATUS REGISTER (MMESTS)	2-145	
	TB PARITY ADDRESS REGISTER (TBADR)	2-148	
	TB PARITY STATUS REGISTER (TBSTS)	2-149	
	P-CACHE PARITY ADDRESS REGISTER (PCADR)	2-153	
	P-CACHE STATUS REGISTER (PCSTS)	2-154	
	P-CACHE CONTROL REGISTER (PCCTL)	2-157	
2.7.3	<b>XMI Registers</b>		2-160
	NDAL CONTROL AND STATUS REGISTER (NCSR)	2-162	
	NEXMI INPUT PORT REGISTER (IPORT)	2-168	
	NEXMI OUTPUT PORT0 REGISTER (OPORT0)	2-170	
	NEXMI OUTPUT PORT1 REGISTER (OPORT1)	2-172	
	DEVICE REGISTER (XDEV)	2-173	
	BUS ERROR REGISTER (XBER)	2-174	
	FAILING ADDRESS REGISTER (XFADR)	2-181	
	XMI GENERAL PURPOSE REGISTER (XGPR)	2-185	
	NODE-SPECIFIC CONTROL AND STATUS REGISTER (NSCSR)	2-186	
	XMI CONTROL REGISTER (XCR)	2-188	
	FAILING ADDRESS EXTENSION REGISTER (XFAER)	2-194	
	BUS ERROR EXTENSION REGISTER (XBEER)	2-197	
	WRITEBACK 0 FAILING ADDRESS REGISTER (WFADR0)	2-201	
	WRITEBACK 1 FAILING ADDRESS REGISTER (WFADR1)	2-202	
2.8	<b>KA66A CPU MODULE INITIALIZATION, SELF-TEST, AND BOOTING</b>		2-203
2.8.1	<b>Initialization Overview</b>		2-203

<b>2.8.2</b>	<b>Detailed Initialization Description</b>	<b>2-205</b>
2.8.2.1	NVAX CPU Hardware/Microcode Initialization • 2-208	
2.8.2.2	Console Initialization • 2-208	
2.8.2.3	Unnecessary Explicit Initialization • 2-210	
2.8.2.4	Warm Start Initialization • 2-210	
2.8.2.5	Node Reset • 2-210	
2.8.2.6	Boot Processor Determination • 2-211	
2.8.2.7	Memory Configuration • 2-211	
2.8.2.7.1	Selection of Interleave • 2-211	
2.8.2.7.2	Memory Testing and the Bitmap • 2-212	
2.8.2.8	DWMBB Configuration • 2-213	
2.8.2.9	DWMVA Configuration • 2-213	
<b>2.8.3</b>	<b>Bootstrapping or Restarting the Operating System</b>	<b>2-214</b>
2.8.3.1	Operating System Restart • 2-214	
2.8.3.2	Failing Restart • 2-215	
2.8.3.3	Restart Parameters • 2-216	
2.8.3.4	Operating System Bootstrap • 2-216	
2.8.3.5	Boot Algorithm • 2-217	
2.8.3.6	Boot Parameters • 2-218	
2.8.3.7	Bootstrap Software Sequence • 2-219	
<hr/>		
<b>2.9</b>	<b>INTERPROCESSOR COMMUNICATION THROUGH THE CONSOLE PROGRAM</b>	<b>2-220</b>
2.9.1	Required Communications Paths	2-220
2.9.2	Console Communications Area	2-221
2.9.3	Sending a Message to Another Processor	2-229
<hr/>		
<b>2.10</b>	<b>ERROR HANDLING</b>	<b>2-231</b>
2.10.1	Error State Collection	2-233
2.10.2	Error Analysis	2-236
2.10.3	Error Recovery	2-237
2.10.3.1	Special Considerations when Memory Management Is Off • 2-238	
2.10.3.2	Cache Coherence in Error Handling • 2-239	
2.10.3.2.1	Disabling and Flushing the Caches (Leaving the B-Cache in ETM) • 2-240	
2.10.3.2.2	Enabling the Caches • 2-241	
2.10.3.3	Special Writeback Cache Recovery • 2-241	
2.10.3.3.1	B-Cache Uncorrectable Error During Writeback • 2-241	
2.10.3.3.2	Memory State • 2-241	
2.10.3.3.2.1	Accessing Memory State • 2-242	
2.10.3.3.2.2	Repairing Memory State (Fill Errors) • 2-242	
2.10.3.3.2.3	Repairing Memory State (Tagged-Bad Locations) • 2-243	
2.10.3.3.3	Extracting Data from the B-Cache • 2-243	
2.10.3.3.4	Address Determination Procedure for Recovery from Uncorrectable B-Cache Data RAM Errors • 2-243	
2.10.3.3.5	Special Address Determination Procedure for Recovery from Uncorrectable B-Cache Tag Store Errors • 2-244	
2.10.3.4	Cache and TB Test Procedures • 2-245	

# Contents

2.10.3.5	NEXMI Error Handling • 2–245	
<b>2.10.4</b>	<b>Error Retry</b> _____	<b>2–246</b>
2.10.4.1	General Multiple Error Handling Philosophy • 2–246	
2.10.4.2	Retry Special Cases • 2–247	
<b>2.10.5</b>	<b>Console Halt and Halt Interrupt</b> _____	<b>2–247</b>
<b>2.10.6</b>	<b>Machine Check Exception</b> _____	<b>2–249</b>
2.10.6.1	MCHK_UNKNOWN_MSTATUS • 2–259	
2.10.6.2	MCHK_INT.ID_VALUE • 2–259	
2.10.6.3	MCHK_CANT_GET_HERE • 2–259	
2.10.6.4	MCHK_MOVC.STATUS • 2–259	
2.10.6.5	MCHK_ASYNC_ERROR • 2–259	
2.10.6.5.1	TB Parity Errors • 2–260	
2.10.6.5.2	Ebox Stage 3 STALL Timeout Error • 2–260	
2.10.6.6	MCHK_SYNC_ERROR • 2–260	
2.10.6.6.1	VIC Parity Errors • 2–261	
2.10.6.6.2	B-Cache Data RAM Uncorrectable ECC Errors and Addressing Errors • 2–262	
2.10.6.6.3	B-Cache Lost Data RAM Access Error • 2–263	
2.10.6.6.4	NDAL I-Stream or D-Stream Read or D-Stream Ownership Read Timeout Errors • 2–263	
2.10.6.6.5	NDAL I-Stream or D-Stream Read or D-Stream Ownership Read Data Errors • 2–265	
2.10.6.6.6	Lost B-Cache Fill Error • 2–267	
2.10.6.6.7	Unacknowledged NDAL I-Stream or D-Stream Read or D-Stream Ownership Read • 2–268	
2.10.6.6.8	Lost NDAL Output Error • 2–269	
2.10.6.6.9	PTE Read Errors • 2–269	
2.10.6.6.9.1	PTE Read Errors in Interruptable Instructions • 2–270	
2.10.6.6.9.2	B-Cache Data RAM Uncorrectable ECC Errors and Addressing Errors on PTE Reads • 2–271	
2.10.6.6.9.3	NDAL PTE Read Timeout Errors • 2–272	
2.10.6.6.9.4	NDAL PTE Read Data Errors • 2–273	
2.10.6.6.9.5	Unacknowledged NDAL PTE Read • 2–274	
2.10.6.6.9.6	Multiple Errors That interfere with Analysis of PTE Read Errors • 2–274	
2.10.6.7	Inconsistent Status in Machine Checks • 2–275	
<b>2.10.7</b>	<b>Power Fail Interrupt</b> _____	<b>2–276</b>
<b>2.10.8</b>	<b>Hard Error Interrupt</b> _____	<b>2–277</b>
2.10.8.1	Uncorrectable Data Errors and Addressing Errors During Write or Write Unlock Processing • 2–282	
2.10.8.2	Lost B-Cache Data RAM Hard Errors • 2–283	
2.10.8.3	Read Data Error in Quadword OREAD Fill After Write Data Merged • 2–284	
2.10.8.4	Timeout in Quadword OREAD Fill After Write Data Merged • 2–285	
2.10.8.4.1	Unexpected Fill Error • 2–286	
2.10.8.4.2	Lost B-Cache Fill Error • 2–286	
2.10.8.5	NDAL NO ACK During WRITE or WDISOWN • 2–287	
2.10.8.6	Lost NDAL NO ACK Hard Errors • 2–287	
2.10.8.7	Read Data Timeout with Potential Soft Error Cause • 2–288	
2.10.8.8	Read Data Error with Potential Soft Error Cause • 2–288	
2.10.8.9	NEXMI Hard Error Interrupts • 2–289	

2.10.8.10	Inconsistent Status in Hard Error Interrupts • 2–293	
<b>2.10.9</b>	<b>Soft Error Interrupt</b> _____	<b>2–294</b>
2.10.9.1	VIC Parity Errors • 2–308	
2.10.9.2	P-Cache Parity Errors • 2–308	
2.10.9.3	B-Cache Tag Store Uncorrectable ECC Errors • 2–308	
2.10.9.3.1	Case: BCETSTS<TS CMD>=W UNLOCK • 2–309	
2.10.9.3.2	Case: BCETSTS<TS CMD>=DREAD, IREAD, OREAD • 2–309	
2.10.9.3.3	Case: BCETSTS<TS CMD>=R INVAL, O INVAL, IPR DEALLOCATE • 2–309	
2.10.9.4	Lost B-Cache Tag Store Errors • 2–310	
2.10.9.5	B-Cache Tag Store Correctable ECC Errors • 2–310	
2.10.9.6	Lost B-Cache Tag Store Correctable ECC Errors • 2–310	
2.10.9.7	B-Cache Data RAM Correctable ECC Errors • 2–311	
2.10.9.8	Lost B-Cache Data RAM Correctable ECC Errors • 2–311	
2.10.9.9	B-Cache Data RAM Uncorrectable ECC Errors and Addressing Errors on I-Stream or D-Stream Reads • 2–311	
2.10.9.10	B-Cache Data RAM Uncorrectable ECC Errors and Addressing Errors on Writebacks • 2–312	
2.10.9.11	Lost B-Cache Data RAM Errors with Possible Lost Writebacks • 2–313	
2.10.9.12	Lost B-Cache Data RAM Errors Without Lost Writebacks • 2–314	
2.10.9.13	NDAL I-Stream or D-Stream Read or D-Stream Ownership Read Timeout Errors • 2–314	
2.10.9.14	NDAL I-Stream or D-Stream Read or D-Stream Ownership Read Data Errors • 2–317	
2.10.9.15	Lost B-Cache Fill Error • 2–319	
2.10.9.16	Unacknowledged NDAL I-Stream or D-Stream Read or D-Stream Ownership Read • 2–320	
2.10.9.17	Lost NDAL Output Error • 2–321	
2.10.9.18	PTE Read Errors • 2–321	
2.10.9.18.1	B-Cache Data RAM Uncorrectable ECC Errors and Addressing Errors on PTE Reads • 2–321	
2.10.9.18.2	NDAL PTE Read Timeout Errors • 2–322	
2.10.9.18.3	NDAL PTE Read Data Errors • 2–323	
2.10.9.18.4	Unacknowledged NDAL PTE Read • 2–324	
2.10.9.18.5	Multiple Errors That Interfere with Analysis of PTE Read Errors • 2–325	
2.10.9.19	NDAL Parity Errors • 2–325	
2.10.9.20	Lost Parity Errors • 2–328	
2.10.9.21	Inconsistent Parity Errors • 2–328	
2.10.9.22	NEXMI Soft Error Interrupts • 2–328	
2.10.9.23	Inconsistent Status in Soft Error Interrupts • 2–330	
<b>2.10.10</b>	<b>Note on Tagged-Bad Data Mechanisms</b> _____	<b>2–330</b>
<b>2.10.11</b>	<b>Kernel Stack Not Valid Exception</b> _____	<b>2–331</b>

<b>CHAPTER 3 MS65A MEMORY MODULE</b>		<b>3-1</b>
<b>3.1</b>	<b>MODULE DESCRIPTION</b>	<b>3-2</b>
<b>3.2</b>	<b>SELF-TEST AND INITIALIZATION</b>	<b>3-4</b>
3.2.1	Starting and Ending Addresses _____	3-5
3.2.2	Interleaving _____	3-5
<b>3.3</b>	<b>CONTROL AND STATUS REGISTERS</b>	<b>3-6</b>
	DEVICE REGISTER (XDEV)	3-8
	BUS ERROR REGISTER (XBER)	3-10
	MEMORY CONTROL REGISTER 1 (MCTL1)	3-14
	MEMORY ECC ERROR REGISTER (MECER)	3-17
	MEMORY ECC ERROR ADDRESS REGISTER (MECEA)	3-21
	MEMORY CONTROL REGISTER 2 (MCTL2)	3-22
	TCY TESTER REGISTER (TCY)	3-24
	BLOCK STATE ECC ERROR REGISTER (BECER)	3-25
	BLOCK STATE ECC ADDRESS REGISTER (BECEA)	3-26
	STARTING ADDRESS REGISTER (STADR)	3-27
	ENDING ADDRESS REGISTER (ENADR)	3-28
	SEGMENT/INTERLEAVE REGISTER (INTLV)	3-30
	MEMORY CONTROL REGISTER 3 (MCTL3)	3-32
	MEMORY CONTROL REGISTER 4 (MCTL4)	3-34
	BLOCK STATE CONTROL REGISTER (BSCTL)	3-37
	BLOCK STATE ADDRESS REGISTER (BSADR)	3-39
	EEPROM CONTROL REGISTER (EECTL)	3-40
	TIMEOUT CONTROL/STATUS REGISTER (TMOER)	3-42
<b>3.4</b>	<b>ERROR HANDLING</b>	<b>3-43</b>

**INDEX**

**EXAMPLES**

2-1	Error State Collection _____	2-235
2-2	Backup Cache Flushing and Error State Collection _____	2-236

**FIGURES**

1-1	System Architecture _____	1-2
1-2	Sample System _____	1-4
1-3	System Front View _____	1-6
1-4	System Rear View _____	1-8
1-5	Adapters _____	1-10
2-1	KA66A CPU Module Block Diagram _____	2-2
2-2	Physical Address Space Layout _____	2-10
2-3	PTE Format (21-Bit PFN) _____	2-11
2-4	PTE Format (25-Bit PFN) _____	2-12
2-5	Minimum Stack Frame _____	2-14
2-6	Large Stack Frame _____	2-15
2-7	Arithmetic Exception Stack Frame _____	2-18
2-8	Memory Management Exception Stack Frame _____	2-19
2-9	Emulated Instruction Trap _____	2-20
2-10	Emulated Instruction Fault _____	2-21
2-11	Machine Check Stack Frame _____	2-22
2-12	System Control Block Vectors _____	2-25
2-13	Process Control Block _____	2-29
2-14	PTE Fills from MME Latch _____	2-46
2-15	PTE Fills from EM Latch _____	2-46
2-16	Cbox in the System _____	2-48
2-17	Watch Chip CSR A (E018 300A) _____	2-51
2-18	Watch Chip CSR B (E018 300B) _____	2-52
2-19	Watch Chip CSR D (E018 300D) _____	2-52
2-20	KA66A CPU Module Private I/O Address Space Map _____	2-55
2-21	Mask Generation Diagram _____	2-61
2-22	IPR Address Space Decoding _____	2-64
2-23	Initialization Flowchart _____	2-205
2-24	Restart Parameter Block Format _____	2-215
2-25	CCA Layout _____	2-223
2-26	Layout of XMI Node Buffers _____	2-227
2-27	Machine Check Exception Parse Tree _____	2-250
2-28	Hard Error Interrupt Parse Tree _____	2-277
2-29	Soft Error Interrupt Parse Tree _____	2-295
3-1	Error Bit Hierarchy _____	3-13

---

**TABLES**

1	VAX 6000 Series Documentation _____	xv
2	VAX 6000 Model Level Documentation _____	xvi
3	Associated Documents _____	xvi
1-1	System Components _____	1-5
1-2	Adapters _____	1-11
2-1	NVAX CPU Chip Functional Units _____	2-4
2-2	30-Bit Mapping of Program Addresses to 32-Bit Hardware Addresses _____	2-10
2-3	KA66A CPU Module Interrupts _____	2-16
2-4	KA66A CPU Module Exceptions _____	2-17
2-5	Arithmetic Exceptions Type Codes _____	2-19
2-6	Memory Management Exceptions _____	2-19
2-7	Emulated Instruction Trap Stack Frame Parameters _____	2-21
2-8	Machine Check Stack Frame Fields _____	2-22
2-9	Machine Check Codes _____	2-23
2-10	CPU State Initialized on Console Halt _____	2-24
2-11	Console Halt Codes _____	2-25
2-12	System Control Block Layout _____	2-26
2-13	VIC Attributes _____	2-31
2-14	IPR Address Space Decoding _____	2-33
2-15	Backup Cache Behavior During ETM _____	2-38
2-16	Backup Cache State Changes During ETM _____	2-39
2-17	Interval Clock Register Addresses _____	2-49
2-18	Watch Chip Data _____	2-50
2-19	Watch Chip Example _____	2-51
2-20	Watch Chip Control Registers _____	2-51
2-21	NEXMI Transaction Generation/Response for NVAX Chip-to-XMI Operations _____	2-56
2-22	NEXMI Transaction Generation/Response for XMI-to-NVAX Chip Operations _____	2-57
2-23	Transaction Priority Table _____	2-58
2-24	IPR Address Space Decoding _____	2-65
2-25	I/O Space Registers _____	2-66
2-26	KA66A CPU Module Internal Processor Registers _____	2-67
2-27	Types of Registers and Bits _____	2-70
2-28	Interpretation of TS CMD _____	2-105
2-29	Interpretation of DR CMD _____	2-112
2-30	Data Length Code _____	2-127
2-31	CMD Definitions _____	2-150
2-32	KA66A CPU Module Registers in XMI Private Space _____	2-160
2-33	XMI Registers for the KA66A CPU Module _____	2-161
2-34	Boot Parameters Loaded into GPRs _____	2-218

<b>2-35</b>	<b>CCA Fields</b> _____	<b>2-224</b>
<b>2-36</b>	<b>Buffer Fields</b> _____	<b>2-227</b>
<b>2-37</b>	<b>Hardware-Detected Errors</b> _____	<b>2-231</b>
<b>2-38</b>	<b>NVAX Chip Internally Generated SCB Entry Points</b> _____	<b>2-232</b>
<b>2-39</b>	<b>Error Summary Notification by Entry Point</b> _____	<b>2-232</b>
<b>2-40</b>	<b>S_CEFSTS Cycle Type Decode</b> _____	<b>2-314</b>
<b>3-1</b>	<b>MS65A Memory Module Control and Status Registers</b> _____	<b>3-6</b>

---

# Preface

---

## Intended Audience

This manual is written for Digital customer service engineers doing field-level repairs or programming and for OEMs who are writing specialized applications, such as their own operating systems. The *VAX 6000 Platform Technical User's Guide* is also useful for such purposes.

---

## Document Structure

This manual has three chapters.

- **Chapter 1** introduces the VAX 6000 Model 600 system and its parts.
- **Chapter 2** explains the KA66A CPU module.
- **Chapter 3** explains the MS65A memory module.
- The **Index** provides additional reference support.

---

## VAX 6000 Series Documents

There are two sets of documentation: manuals that apply to all VAX 6000 series systems and manuals that are specific to one VAX 6000 model. Table 1 lists the manuals in the VAX 6000 series documentation set.

**Table 1 VAX 6000 Series Documentation**

Title	Order Number
<b>Operation</b>	
<i>VAX 6000 Series Owner's Manual</i>	EK-600EB-OM
<i>VAX 6000 Series Vector Processor Owner's Manual</i>	EK-60VAA-OM
<i>VAX 6000 Vector Processor Programmer's Guide</i>	EK-60VAA-PG
<b>Service and Installation</b>	
<i>VAX 6000 Platform Technical User's Guide</i>	EK-600EA-TM
<i>VAX 6000 Series Installation Guide</i>	EK-600EB-IN
<i>VAX 6000 Installationsanleitung</i>	EK-600GB-IN
<i>VAX 6000 Guide d'installation</i>	EK-600FB-IN
<i>VAX 6000 Guia de instalacion</i>	EK-600SB-IN
<i>VAX 6000 Platform Service Manual</i>	EK-600EA-MG

**Table 1 (Cont.) VAX 6000 Series Documentation**

<b>Title</b>	<b>Order Number</b>
<b>Options and Upgrades</b>	
<i>VAX 6000: XMI Conversion Manual</i>	EK-650EB-UP
<i>VAX 6000: Installing MS65A Memories</i>	EK-MS65A-UP
<i>VAX 6000: Installing the H7236-A Battery Backup Option</i>	EK-60BBA-IN
<i>VAX 6000: Installing the FV64A Vector Option</i>	EK-60VEA-IN
<i>VAX 6000: Installing the VAXBI Option</i>	EK-60BIA-IN

Manuals specific to models are listed in Table 2.

**Table 2 VAX 6000 Model Level Documentation**

<b>Title</b>	<b>Order Number</b>
<b>Model 600</b>	
<i>VAX 6000 Model 600 Mini-Reference</i>	EK-660EA-HR
<i>VAX 6000 Model 600 Service Manual</i>	EK-660EA-MG
<i>VAX 6000 Model 600 System Technical User's Guide</i>	EK-660EA-TM
<i>VAX 6000: Installing Model 600 Processors</i>	EK-660EA-UP
<b>Model 500</b>	
<i>VAX 6000 Model 500 Mini-Reference</i>	EK-650EA-HR
<i>VAX 6000 Model 500 Service Manual</i>	EK-650EA-MG
<i>VAX 6000 Model 500 System Technical User's Guide</i>	EK-650EA-TM
<i>VAX 6000: Installing Model 500 Processors</i>	EK-KA65A-UP
<b>Models 200/300/400</b>	
<i>VAX 6000 Model 300 and 400 Service Manual</i>	EK-624EA-MG
<i>VAX 6000: Installing Model 200/300/400 Processors</i>	EK-6234A-UP

## Associated Documents

Table 3 lists other documents that you may find useful.

**Table 3 Associated Documents**

<b>Title</b>	<b>Order Number</b>
<b>System Hardware Options</b>	
<i>VAXBI Expander Cabinet Installation Guide</i>	EK-VBIEA-IN
<i>VAXBI Options Handbook</i>	EB-32255-46

**Table 3 (Cont.) Associated Documents**

<b>Title</b>	<b>Order Number</b>
<b>System I/O Options</b>	
<i>CIBCA User Guide</i>	EK-CIBCA-UG
<i>CIXCD Interface User Guide</i>	EK-CIXCD-UG
<i>DEC LANcontroller 200 Installation Guide</i>	EK-DEBNI-IN
<i>DEC LANcontroller 400 Installation Guide</i>	EK-DEMNA-IN
<i>DSSI VAXcluster Installation Guide</i>	EK-DVCLU-IN
<i>InfoServer Installation Guide</i>	EK-DIS1K-IN
<i>KDB50 Disk Controller User's Guide</i>	EK-KDB50-UG
<i>KDM70 Controller User Guide</i>	EK-KDM70-UG
<i>KFMSA Module Installation and User Manual</i>	EK-KFMSA-IM
<i>KFMSA Module Service Guide</i>	EK-KFMSA-SV
<i>RRD42 Disc Drive Owner's Manual</i>	EK-RRD42-OM
<i>RA90/RA92 Disk Drive User Guide</i>	EK-ORA90-UG
<i>RF31/RF72 Integrated Storage Element Installation Manual for BA200-Series Enclosures</i>	EK-RF72D-IM
<i>RF31/RF72 Integrated Storage Element User Guide</i>	EK-RF72D-UF
<i>RF31/RF72 Integrated Storage Element Service Guide</i>	EK-RF72D-SV
<i>SA70 Enclosure User Guide</i>	EK-SA70E-UG
<i>SF2xx Storage Array Installation Guide</i>	EK-SF200-IG
<i>SF7x Storage Enclosure and SF2xx Storage Array Cabinet Service Guide</i>	EK-SF72S-SG
<i>TF85 Cartridge Tape Subsystem Owner's Manual</i>	EK-OTF85-OM
<i>TF857 Magazine Tape Subsystem Service Manual</i>	EK-TF857-OM
<i>VAX 6000/SF2xx Embedded Storage Installation Guide</i>	EK-EMBED-IN
<b>Operating System Manuals</b>	
<i>Guide to Maintaining a VMS System</i>	AA-LA34B-TE
<i>Guide to Setting Up a VMS System</i>	AA-LA25A-TE
<i>Introduction to VMS System Management</i>	AA-LA24A-TE
<i>ULTRIX-32 Guide to System Exercisers</i>	AA-ME96B-TE
<i>VMS Networking Manual</i>	AA-LA48A-TE
<i>VMS System Manager's Manual</i>	AA-LA00B-TE
<i>VMS Upgrade and Installation Supplement: VAX 6000 Series</i>	AA-LB36C-TE
<i>VMS Version 5.5 Upgrade and Installation Manual</i>	AA-NG61D-TE

**Table 3 (Cont.) Associated Documents**

<b>Title</b>	<b>Order Number</b>
<b>VAXclusters and Networking</b>	
<i>DECbridge 500 Installation Guide</i>	EK-DEFEB-IN
<i>DEMFA Installation Guide</i>	EK-DEMFA-IN
<i>Fiber Distributed Data Interface Description</i>	EK-DFSLD-SD
<i>Guidelines for VAXcluster System Configurations</i>	EK-VAXCS-CG
<i>H4000 Digital Ethernet Transceiver Installation Manual</i>	EK-H4000-IN
<i>HSC Installation Manual</i>	EK-HSCMN-IN
<i>VAXcluster Principles</i>	EK-VAXCP-TM
<i>VMS VAXcluster Manual</i>	AA-LA27B-TE
<b>Peripherals</b>	
<i>Installing and Using the VT420 Video Terminal</i>	EK-VT420-UG
<i>RV20 Optical Disk Owner's Manual</i>	EK-ORV20-OM
<i>SC008 Star Coupler User's Guide</i>	EK-SC008-UG
<i>TA78 Magnetic Tape Drive User's Guide</i>	EK-OTA78-UG
<i>TA90 Magnetic Tape Subsystem Owner's Manual</i>	EK-OTA90-OM
<i>TK70 Streaming Tape Drive Owner's Manual</i>	EK-OTK70-OM
<i>TU81/TA81 and TU/81 PLUS Subsystem User's Guide</i>	EK-TUA81-UG
<b>VAX Manuals</b>	
<i>VAX Architecture Reference Manual</i>	EY-3459E-DP
<i>VAX Systems Hardware Handbook — VAXBI Systems</i>	EB-31692-46
<i>VAX Vector Processing Handbook</i>	EC-H0739-46

# 1

---

## The VAX 6000 Model 600 System

The VAX 6000 Model 600 computer system is designed for growth and can be configured for many different applications. Like other VAX systems, the VAX 6000 Model 600 system can support many users in a time-sharing environment. This system does the following:

- Supports a full range of VAX applications and operating systems
- Supports writeback caching which enhances system performance
- Functions as a standalone system, a member of a VAXcluster, a boot node of a local area VAXcluster, or as a VAX file server for workstations
- Allows for expansion of processors, memory, and I/O on the XMI bus
- Implements symmetric multiprocessing where all processors have equal access to memory
- Uses a high-bandwidth system bus designed for multiprocessing
- Performs automatic self-test on power-up, reset, reboot, or system initialization
- Supports I/O devices on the VAXBI bus and provides access to the VME bus

This chapter describes the system packages and introduces the location of components in the cabinet. Sections include:

- System Architecture
- Sample System
- System Front View
- System Rear View
- Supported Adapters

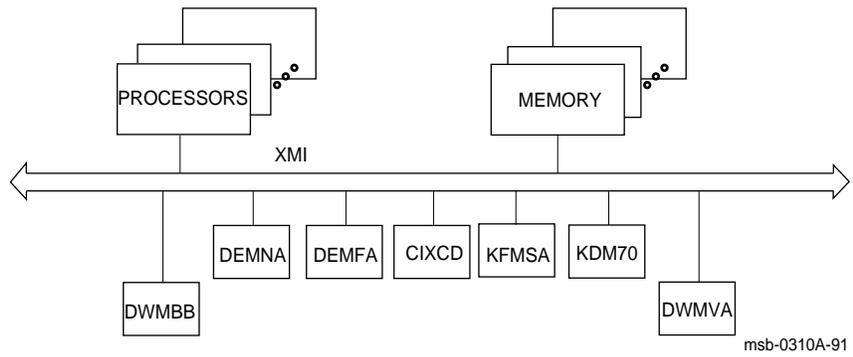
---

## 1.1 System Architecture

The high-speed XMI bus is used to interconnect processors, memory modules, and I/O adapters.

Figure 1-1 System Architecture

---



The XMI is the 64-bit system bus that interconnects the processors, memory modules, and I/O adapters.

The XMI bus uses the concept of a **node**. A node is a single functional unit that consists of one or more modules. The XMI has three types of nodes: processor nodes, memory nodes, and I/O adapters.

A **processor node**, called a KA66A CPU module, is a single-board processor containing a central processor unit (CPU) that executes instructions and manipulates data. A writeback cache subsystem improves system performance.

The VAX 6000 Model 600 system supports multiprocessing with up to six processors. Symmetric multiprocessing is supported, allowing a program to execute on any processor. In a multiprocessing system one processor becomes the boot processor during power-up, and that processor loads the operating system and handles communication with the operator console. The other processors become secondary processors and receive system information from the boot processor.

A **memory node** is one memory module. Memory is a global resource equally accessible by all processors on the XMI. A memory module can have 32, 64, or 128 Mbytes of memory and associated ECC and control logic. The memories are automatically interleaved. An optional battery backup unit protects memory in case of power failure. The system supports up to eight MS65A memories.

**I/O adapters** are installed on the XMI bus (see Section 1.5). If your system has a VAXBI, then the DWMBB adapter is used to connect VAXBI I/O adapters to the XMI bus. The DWMVA adapter provides an interface to the VMEbus.

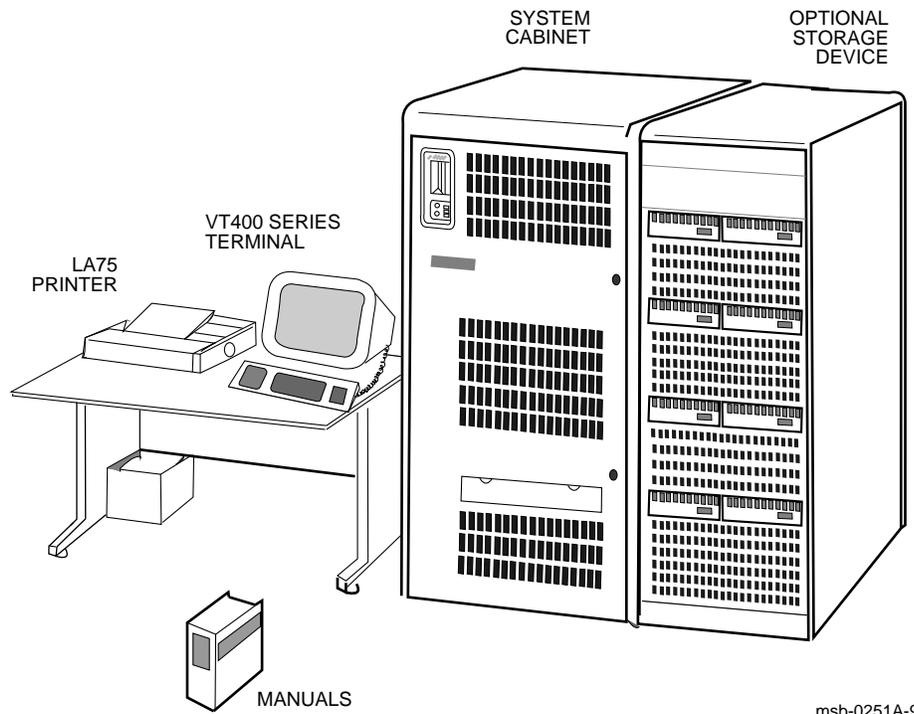
---

## 1.2 Sample System

A sample system has a system cabinet, a console load device—either a tape drive or a compact disk server on the Ethernet—a console terminal and printer, an accessories kit, and a documentation set. The system may have additional storage devices and may be a member of a VAXcluster.

Figure 1-2 Sample System

---



msb-0251A-91

**Table 1-1 System Components**

<b>Component</b>	<b>Function</b>
System cabinet	Houses system components and optional storage
Console load device	Software distribution; stores and transfers data
Console terminal	Manages system and its resources
Console printer	Provides hardcopy of console transactions
Documentation	See the Preface for a full list of documentation related to VAX 6000 Model 600 systems
Storage cabinet	Provides additional storage capacity

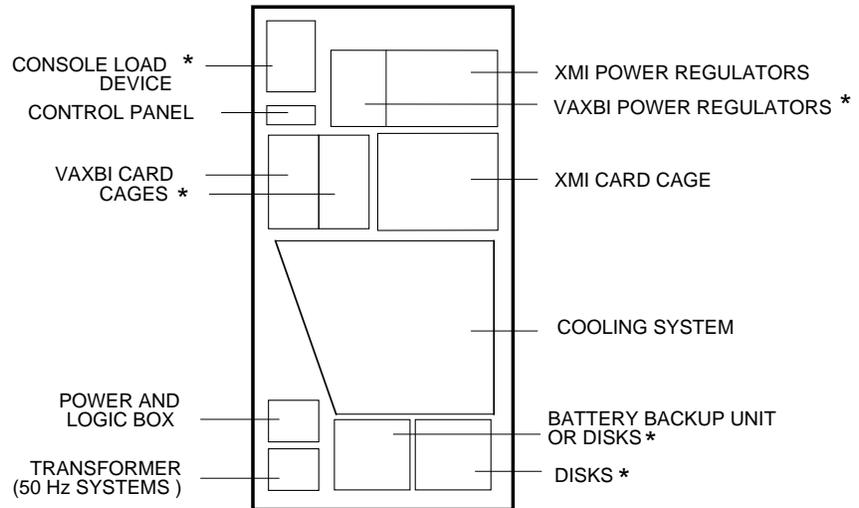
The VAX 6000 Model 600 components include:

- **The system cabinet** houses the XMI card cage (which contains the processors, memories, and I/O adapters) and the control panel with status indicators. Optional hardware in the cabinet includes a console load device, a VAXBI backplane, disk drives, and a battery backup unit.
- **The console load device** is used for installing operating systems, software, and some diagnostics. The console load device can be a tape drive, either in the cabinet or in the SF200 storage array, or it can be an Ethernet-based compact disk server.
- **A storage cabinet** provides local storage and archiving capability.
- **The console terminal** is used for booting and for system management operations.
- **A system documentation kit**

### 1.3 System Front View

The control panel and optional console load device and disk control panel are on the front of the system cabinet, accessible with the doors closed. With the front door open, Digital customer service engineers can access the power regulators, the XMI card cage and optional VAXBI card cages, the cooling system, and the optional battery backup unit.

Figure 1-3 System Front View



\* OPTIONAL

msb-0311A-91

**WARNING: The inside of the system cabinet is not designed to be accessed by the customer. The cabinet doors are to be opened only by Digital customer service engineers.**

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

- Control panel
- XMI power regulators
- XMI card cage
- Cooling system  
One of the two blowers is visible from the front of the cabinet.
- Power and logic box
- Transformer (on 50 Hz systems only)

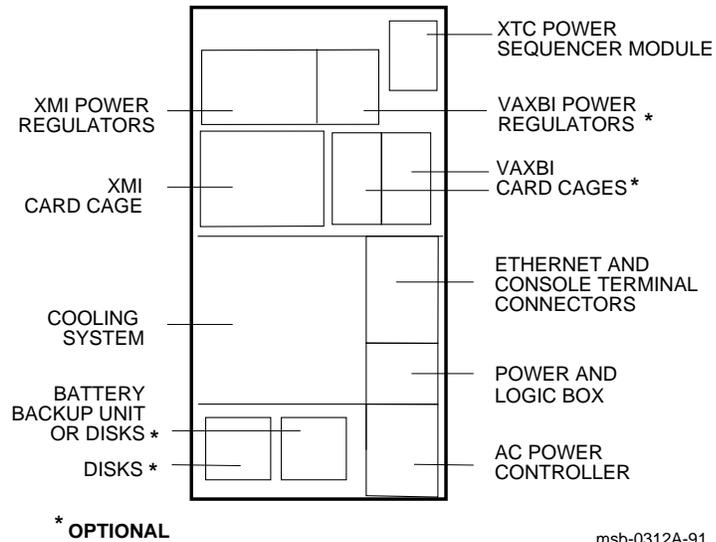
Optional components:

- Console load device
- VAXBI power regulators
- Two VAXBI card cages configured as one 12-slot channel
- Battery backup unit
- Disks

## 1.4 System Rear View

With the rear door open, Digital customer service engineers can access the power sequencer module (XTC); the power regulators; the I/O bulkhead space behind the card cages; Ethernet and console terminal connectors; cooling system; power and logic box; battery backup unit and disks, if present; and the AC power controller.

Figure 1-4 System Rear View



**WARNING: The inside of the system cabinet is not designed to be accessed by the customer. The cabinet doors are to be opened only by Digital customer service engineers.**

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

- Power sequencer module (XTC) located on the back of the system control assembly
- XMI power regulators
- I/O bulkhead space  
The panel covering the XMI and VAXBI areas is the I/O bulkhead panel and provides space for additional I/O connections.
- XMI backplane and cables
- Ethernet and console terminal connectors
- Cooling system
- Power and logic box
- AC power controller

Optional components:

- VAXBI power regulators
- VAXBI backplane and cables
- Battery backup unit
- Disks

## 1.5 Supported Adapters

VAX 6000 systems provide interfaces to other buses and to the Ethernet. Systems can be clustered and storage can be added and shared among systems. The system supports the following adapters: CIXCD, DEC LANcontroller 400 (DEMNA), DEMFA, DWMBB, DWMVA, KDM70, and KFMSA.

Figure 1-5 Adapters

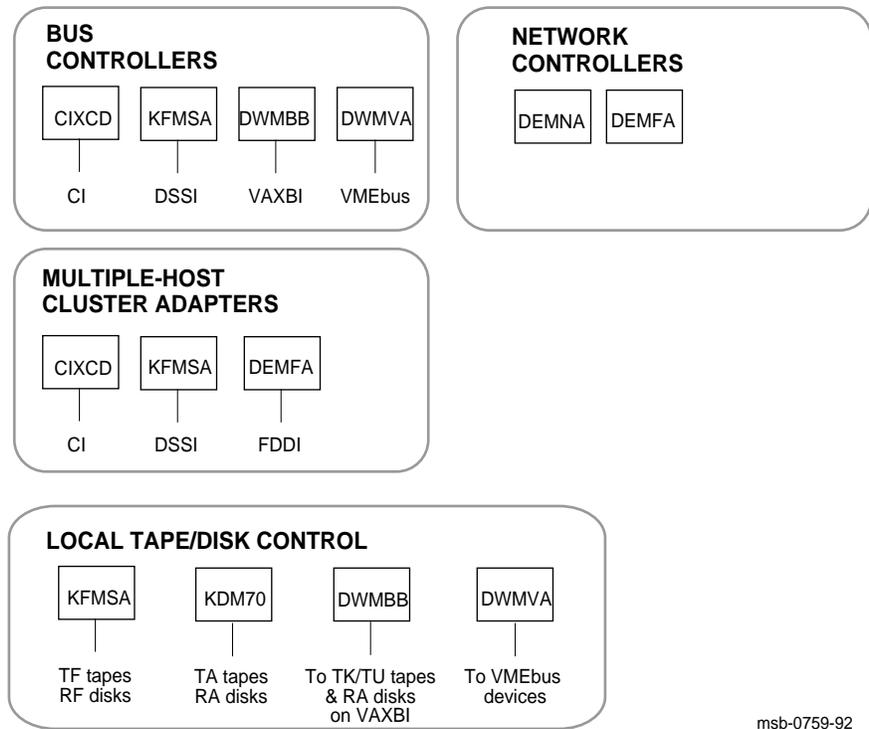


Table 1-2 describes the adapters supported by the system. For more information on adapters, see Digital's *Systems and Options Catalog* or the *VAX 6000 Platform Service Manual*.

**Table 1-2 Adapters**

<b>Adapter</b>	<b>XMI Slots</b>	<b>Function</b>
CIXCD	1	CI port interface; connects the system to a Star Coupler.
DEMFA	1	FDDI (fiber optic) port interface; connects a system to a local area network.
DEMNA	1	Ethernet port interface; connects a system to a local area network.
DWMBB	1	XMI-to-VAXBI interface, a two-module set. The DWMBB/A is in the XMI card cage; the DWMBB/B is installed in the VAXBI card cage.
DWMVA	1	XMI-to-VMEbus interface, a two-module set. The DWMVA/A is in the XMI card cage; the DWMVA/B is installed in a VMEbus expansion cabinet.
KDM70	2	Disk adapter; enables connection to RA disk drives.
KFMSA	1	DSSI adapter; enables connection to TF tape drives and to RF disk drives.

# The VAX 6000 Model 600 System

# 2

---

## KA66A CPU Module

This chapter describes the KA66A CPU module, the processor for the VAX 6000 Model 600 system.

This chapter includes the following sections:

- Overview and Block Diagram
- CPU Section
- Cache Overview
- NVAX Box Descriptions
- KA66A Toy Clock and Interval Timer
- XMI Interface
- KA66A CPU Module Registers
- Initialization, Self-Test, and Booting
- Interprocessor Communication Through the Console Program
- Error Handling

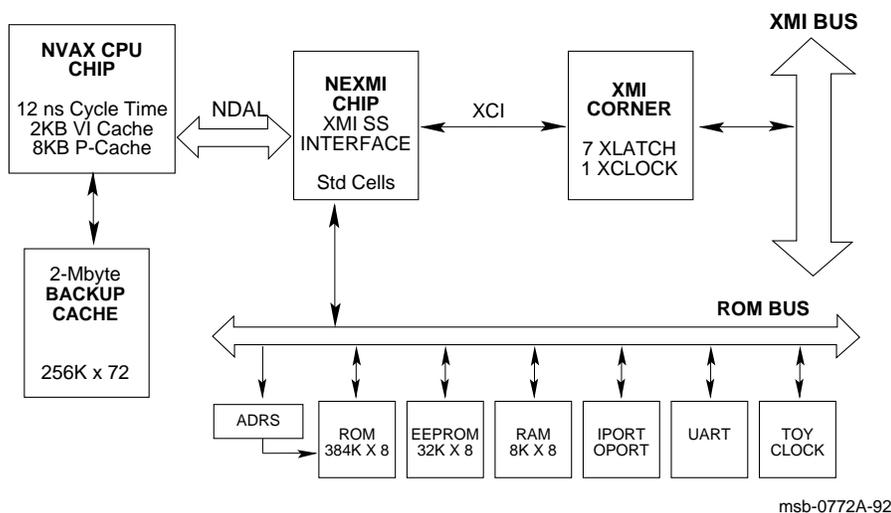
## 2.1 Overview and Block Diagram

The KA66A CPU module consists of three major sections:

- NVAX CPU chip
- Backup cache
- NEXMI chip, system support, and XMI interface

Figure 2–1 shows the KA66A CPU module block diagram.

Figure 2–1 KA66A CPU Module Block Diagram



The NVAX and NEXMI chips implement a VAX with the following features:

- Support for the 242-instruction VAX base instruction group, associated data types, full VAX memory management, and a 4-Gbyte virtual address space.
- Support for 3.5 Gbytes of physical memory and 512 Mbytes of I/O space, when the system is in 32-bit addressing mode.
- A floating-point accelerator that improves the execution of the F\_, D\_, and G\_format floating-point instructions and the longword variants of integer multiply.

- A three-level cache subsystem. A 2-Kbyte instruction cache and an 8-Kbyte I- and D-stream primary cache are in the NVAX chip. A 2-Mbyte backup cache is implemented in RAMs.
- A writeback cache system that allows multiple read and write operations to be serviced which reduces XMI bus traffic.
- A VAX-compatible macrocoded console program.
- A set of processor clock registers that support the following:
  - A VAX-standard time-of-year (TOY) clock with battery backup
  - An interval timer with 10 millisecond interrupts
  - A programmable timer
- A bootstrap and diagnostic facility that provides:
  - Full microcode and macrocode power-up self-testing
  - Node initialization
  - Booting from various XMI and VAXBI devices
- An XMI interface that includes:
  - A writeback buffer with four hexword entries
  - Hexword cache fill logic that loads the backup cache with eight longwords of data on each cache miss
  - XMI read/write monitoring logic
  - Full error recovery and logging capabilities

### 2.1.1 **NVAX CPU Chip**

The NVAX CPU chip is a single-chip CMOS-4 macropipelined implementation of the VAX base instruction group. Included in the chip are:

- CPU: Instruction fetch and decode, microsequencer, and execution unit
- Control store: 1600, 61-bit microwords
- Primary cache: 8-Kbyte, 2-way set associative, physically addressed, write through, mixed instruction and data stream
- Virtual instruction cache: 2-Kbyte, direct-mapped, virtually addressed, instruction stream only
- Translation buffer: 96 entries, fully associative
- Floating-point unit: four-stage, pipelined, integrated floating-point unit
- Backup cache interface: supports a 2-Mbyte B-cache

The NVAX has a macroinstruction pipelined design. Pipelining allows significant processing overlap not possible with other designs. The design allows several macro instructions to be decoded and operands fetched prior to their execution. The pipeline queues instruction information and operand values for later use by the execution unit. Thus, when the macropipeline is running smoothly, the instruction unit (Ibox), which parses instructions and fetches operands, is running several macroinstructions ahead of the execution unit (Ebox). Branch predictions are made prior to knowing whether a particular branch will be taken allowing quicker execution of loops in software. Outstanding writes to registers or memory locations are kept in a scoreboard to ensure that data is not read before it has been written.

The chip is partitioned into functional units or boxes. The boxes and their functions are listed in Table 2–1.

**Table 2–1 NVAX CPU Chip Functional Units**

Ibox	The instruction box prefetches, decodes, parses, and queues VAX instructions for execution.
Ebox	The execution box and the microsequencer execute instructions passed to it by the Ibox.
Fbox	The floating-point accelerator box executes floating-point and integer multiply VAX instructions passed to it by the Ebox.
Mbox	The memory management box handles all virtual to physical address translation for both the Ibox and Ebox. It also handles Cbox requests for cache fills and invalidates for the primary cache.
Cbox	The cache control box initiates access to the the backup cache, or B-cache, issues memory requests, and controls cache coherency by handling invalidates.

#### 2.1.1.1

##### Ibox

The Ibox supports four main functions:

- **Instruction Stream Prefetching**  
The Ibox attempts to maintain sufficient instruction stream data to decode instructions or operand specifiers prior to execution of the previous instruction by the Ebox.
- **Instruction Parsing**  
The Ibox identifies the instruction opcodes and operand specifiers, and extracts the information necessary for further processing.
- **Operand Specifier Processing**  
The Ibox processes the operand specifiers, initiates the required memory references, and provides the Ebox with the information necessary to access the instruction's operands.
- **Branch Prediction**  
Upon identification of a branch opcode, the Ibox hardware predicts the direction of the branch. Should the prediction be incorrect, the Ibox redirects the instruction prefetching and parsing logic to the correct branch destination, where instruction processing resumes.

---

**2.1.1.2 Ebox and Microsequencer**

The Ebox is the instruction execution unit in the NVAX chip. It is a three-stage pipeline that supports the following functions:

- **Instruction Execution**  
The Ebox is responsible for carrying out the execution portion of each VAX instruction under control of a microflow whose initial address is provided by the Ibox.
- **Instruction Coordination**  
The Ebox is a major source of control to coordinate instruction processing in the Ibox, Mbox, and Fbox. It ensures that Ebox and Fbox macroinstructions retire in the proper order, and it provides controls to the Mbox and Ibox which help manage certain macroinstruction interdependencies. The Ebox cooperates with the Ibox in handling mispredicted branches.
- **Trap, Fault, and Exception Handling**  
The Ebox coordinates trap, fault, and interrupt handling. It delays the condition until all preceding macroinstructions complete properly. It then collects information about the condition and ensures that the correct architectural state is reached.
- **CPU Control**  
Most CPU control is provided by the Ebox. Ebox control functions include CPU initialization, controlling the Ibox, Fbox, and Mbox activities, and setting control bits during major CPU state changes like taking an interrupt or executing a change mode instruction.

---

**2.1.1.3 Fbox**

The Fbox executes floating-point and integer multiply instructions using a four-stage pipeline. Up to 64-bit operands are supplied per cycle by the Ebox on the A-bus and B-bus. Results are returned to the Ebox 32 bits per cycle on the result bus. The Ebox sends the Fbox result to the Mbox.

---

**2.1.1.4 Mbox**

The Mbox performs three tasks:

- **VAX Memory Management**  
The Mbox, in conjunction with the operating system memory management software, is responsible for the allocation and use of physical memory. It performs translations of virtual addresses to physical addresses, checks for access violation on all memory references, and calls upon software memory management code when necessary.
- **Reference Processing**  
The Ibox, Ebox, and Cbox can each access memory simultaneously. Since the Mbox is responsible for acquiring data from memory, it must prioritize, sequence, and process such references and then transfer the data to its correct destination.

- **Primary Cache Control**

The Mbox maintains an 8-Kbyte physical address cache of I-stream and D-stream data called the primary cache or P-cache. The P-cache provides a two-cycle pipeline latency for most I- and D-stream data requests. It is the fastest D-stream storage medium for NVAX and represents the first level of D-stream memory hierarchy and the second level of I-stream memory hierarchy.

---

### 2.1.1.5 Cbox

The Cbox is the interface to the backup cache, or B-cache. Both the tags and data for the B-cache are stored in RAMs. In conjunction with the Mbox, the Cbox is used for invalidates and writeback instructions.

The Cbox is also the interface to the memory subsystem. When a cache miss is detected, the Cbox sends a read transaction onto the NDAL for XMI conversion by the NEXMI. The NDAL is the bus containing the data and address lines from the NVAX chip to the NEXMI chip.

The Cbox packs sequential writes to the same quadword to minimize B-cache write accesses. This is a function of the write packer. Multiple write commands are held in the 8-entry write queue.

---

## 2.1.2 Backup Cache

The backup cache is implemented in RAMs external to the NVAX chip. The size of the cache is 2 Mbytes with a block and subblock size of 32 bytes. Since the data bus to the cache is 8 bytes wide, four accesses are required to read out an entire block. ECC protection is provided on each quadword in the cache and for the tag store.

Each block of data (hexword) has a tag, a valid bit, and an owned bit associated with it. When set, the valid bit indicates that the data stored in the B-cache is accurate data. When set, the owned bit indicates that the data stored in the B-cache may be the only accurate data for this particular address in the system.

### 2.1.3 **NEXMI Chip, System Support, and XMI Interface**

The NEXMI chip provides the interface between the NVAX chip and the XMI. The NVAX data and address lines, the NDAL, is the bus between the NEXMI chip and the NVAX chip, and the XMI is the system bus that connects CPUs, main memory, and I/O adapters. In addition to providing this interface, the NEXMI provides the interface for various system support functions.

The NEXMI operates from two asynchronous clock sources, the XMI six-phase 64-ns cycle clock and the NDAL four-phase 36-ns cycle clock. Synchronization is done internally. Different NEXMI interface drivers and receivers accommodate either 3.3V or 5V bus signal levels.

The NEXMI contains the system support for a common core of functions. The chip provides an interface between the NDAL and a serial-line interface that serves as console, and a ROM interface to support self-test, console command parsing, and boot functions. The ROM bus (see Figure 2-1) is a nonpended multiplexed address and data interconnect to various slow-speed memory devices such as ROM, EEPROM, stack RAM, I/O registers, and a battery-backed-up real time clock. The NEXMI also contains an interval timer and support for I/O reset.

The NEXMI provides bus protocol translation between the NVAX NDAL and the XMI system bus. The NDAL is a pended bus that operates at 3 times the system cycle time. To memory it supports hexword reads and quadword and hexword writes; to I/O space it supports longword read and write operations.

The NEXMI drives correct parity on the NDAL at all times to prevent interrupts when the bus is idle. Multiple interrupts from the XMI and system support functions are merged by the NEXMI and issued to the NVAX on the four interrupt request lines. An interrupt acknowledge, an IACK (I/O Read), from the NVAX is automatically converted to an XMI IDENT command and sent to the highest priority interrupting node. System support and interprocessor interrupts, however, are handled by the NEXMI.

---

## 2.2 CPU Section

**The CPU section of the KA66A CPU module executes the VAX base instruction group and provides full VAX memory management. All of these functions are carried out by the NVAX chip. For more information, see the *VAX Architecture Reference Manual*.**

The CPU description that follows includes the following topics:

- Data Types
- Instruction Set
- Physical Address Space
- Memory Management
- Exceptions and Interrupts
- System Control Block
- Process Structure

---

### 2.2.1 Data Types

The KA66A CPU module supports the following subset of VAX data types:

- Byte
- Word
- Longword
- Quadword
- Character string
- Variable-length bit field
- Absolute queues
- Self-relative queues
- D\_floating
- F\_floating
- G\_floating

The remaining VAX data types are supported by macrocode emulation.

## 2.2.2 Instruction Set

---

The KA66A CPU module supports the following instruction classes:

- Integer arithmetic and logical
- Address
- Variable-length bit field
- Control
- Procedure call
- Miscellaneous
- Queue
- Character string
- Operating system support
- D\_floating
- F\_floating
- G\_floating

The KA66A CPU module has special microcode to aid the macrocode emulation of the following instruction groups:

- MATCHC, MOVTC, MOVTUC
- Decimal string
- CRC
- EDITPC

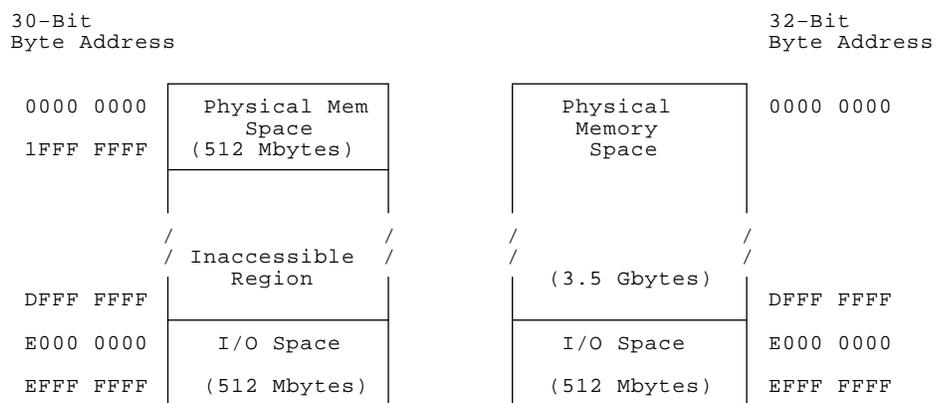
The following instruction groups are not implemented but are emulated by macrocode:

- Octaword
- H\_floating
- POLYF, POLYD, and POLYG
- EMOF, EMODD, and EMODG
- ACBF, ACBD, and ACBG
- Compatibility-mode instructions

### 2.2.3 Physical Address Space

The KA66A CPU generates a 32-bit address that corresponds to 4 gigabytes of physical address space. However, the CPU can run in 30-bit mode as well. Figure 2–2 shows the layout of both memory and I/O space. I/O space occupies the last one-eighth (512 Mbytes) of the physical address space and can be distinguished from memory space by the fact that bits <31:29> of the physical address are all ones.

**Figure 2–2 Physical Address Space Layout**



msb-p501-91

The translation from a 30-bit address to a 32-bit address is accomplished by sign-extending physical address <29> to physical address <31:29>. In this mode the programmer sees a 1-Gbyte address space, split evenly between memory and I/O space. A 30-bit address is mapped to the 32-bit physical address as shown in Table 2–2.

**Table 2–2 30-Bit Mapping of Program Addresses to 32-Bit Hardware Addresses**

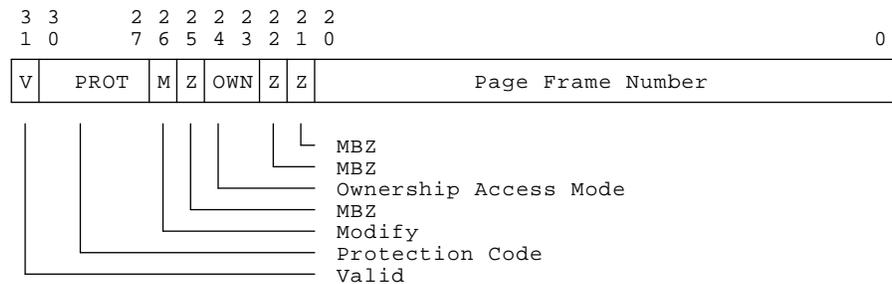
Program Address	Hardware Address
00000000–1FFFFFFF	00000000–1FFFFFFF
20000000–3FFFFFFF	E0000000–FFFFFFF

During power-up, microcode configures the KA66A CPU module to generate 30-bit physical addresses. Operating system initialization code can reconfigure the CPU to generate either 30-bit or 32-bit physical addresses by writing to the MODE bit <0> in the Physical Address Mode Register (IPR231). For full details on physical address space, see the *VAX 6000 Platform Technical User's Guide*.

## 2.2.4 Memory Management

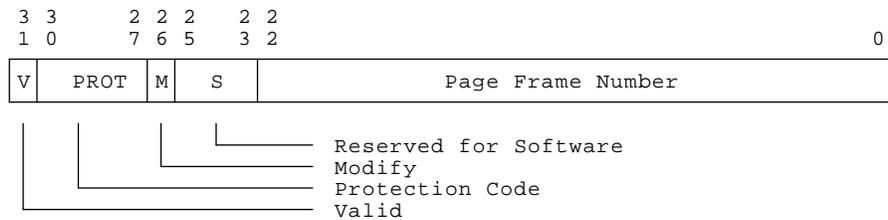
The KA66A CPU module implements full VAX memory management. System space addresses are mapped through single-level page tables, and process space addresses are mapped through two-level page tables. The KA66A CPU module supports two physical addressing modes: a 30-bit mode and a 32-bit mode. When the KA66A CPU module is in 30-bit mode, page table entries (PTEs) are interpreted in a 21-bit page frame number (PFN) format, and when the KA66A CPU module is in 32-bit mode, the PTEs are interpreted in a 25-bit page frame number (PFN) format. See Figure 2-3 and Figure 2-4. In the 32-bit mode, bits <24:23> of the 25-bit PFN are ignored by the NVAX chip and may be used by the software. Refer to the *VAX Architecture Reference Manual* for descriptions of the virtual-to-physical address translation process and the format for VAX page table entries (PTEs).

**Figure 2-3 PTE Format (21-Bit PFN)**



msb-p507-91

Figure 2-4 PTE Format (25-Bit PFN)



msb-p508-91

### 2.2.4.1 Translation Buffer

The NVAX chip includes a 96-entry, fully associative, translation buffer to reduce the overhead associated with translating virtual addresses to physical addresses. The translation buffer caches VAX PTEs. Each entry stores a PTE for translating virtual addresses in either VAX process space or VAX system space. Each entry is divided into two parts: a 26-bit tag register and a 29-bit PTE register.

The tag register is used to store the virtual page number (VPN) of the virtual page that the corresponding PTE register maps. The tag register also stores a valid bit (TB.V) that indicates a valid VPN in the tag. The PTE register stores bits <22:0> of the page frame number (PFN) field, a parity bit for the PFN, the PTE.M bit, and the 4-bit protection (PROT) field from the corresponding VAX PTE. The page table entry valid bit is not stored in the TB because only valid PTEs are stored. Bits <24:23> of the PFN are ignored and along with bit <25> are reserved for software.

During virtual-to-physical address translation, the contents of the 96 tag registers are compared with the VPN field (bits <31:9> of the virtual address of the reference). If there is a match with one of the tag registers and the TB.V bit indicates that the entry is valid, a translation buffer "hit" has occurred, and the contents of the corresponding PTE register are used for the translation.

If there is no match, the translation buffer does not contain the necessary VAX PTE information to translate the address of the reference. The PTE that maps the page is fetched from memory and the translation buffer is updated by replacing the entry at the location indicated by the replacement pointer. The replacement pointer points to the next sequential location after the one last used during translation. The pointer is called the not last used pointer or NLU.

---

#### 2.2.4.2 Memory Management Control Registers

Four internal processor registers (IPRs) control memory management:

- IPR56, Memory Management Enable Register (MAPEN)
- IPR57, Translation Buffer Invalidate All Register (TBIA)
- IPR58, Translation Buffer Invalidate Single Register (TBIS)
- IPR63, Translation Buffer Check Register (TBCHK)

Three pairs of IPRs specify the base and length of P0, P1, and S0 space:

- IPR8, P0 Base Register (P0BR)
- IPR9, P0 Length Register (P0LR)
- IPR10, P1 Base Register (P1BR)
- IPR11, P1 Length Register (P1LR)
- IPR12, System Base Register (SBR)
- IPR13, System Length Register (SLR)

Memory management is enabled/disabled using MAPEN, IPR56. Writing zero to MAPEN with a Move To Processor Register (MTPR) instruction disables memory management; a one enables. MAPEN is read with a Move From Processor Register (MFPR) instruction to determine if memory management is enabled.

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

Translation buffer entries that map a particular virtual address are invalidated by writing the virtual address to TBIS (IPR58) using the MTPR instruction.

**CAUTION: All affected process pages MUST be invalidated in the translation buffer whenever software changes a valid PTE for the system or the current process region, or whenever software changes a system PTE that maps any part of the current process page table. If the pages are not invalidated, the pages used will point to the wrong locations in memory.**

The entire translation buffer is invalidated by writing a zero to TBIA (IPR57) using the MTPR instruction.

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

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

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

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

## 2.2.5 Exceptions and Interrupts

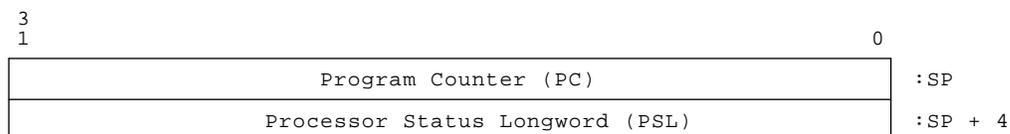
Sometimes events require execution of software routines outside the explicit flow of instructions.

An **exception** is an event caused by the currently executing process that invokes a software routine in the context of the currently executing process. Exception handlers are often system routines, not process routines.

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

The CPU chip reports exceptions and interrupts by constructing a frame on the stack and then dispatching to the service routine through an event-specific vector in the system control block (SCB). The minimum stack frame for any interrupt and exception is a program counter/processor status longword (PC/PSL) pair, as shown in Figure 2–5.

**Figure 2–5 Minimum Stack Frame**



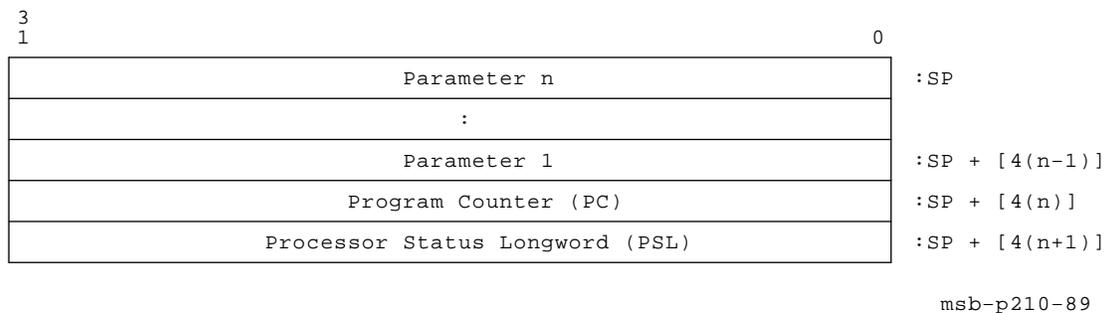
msb-p209-89

---

This minimum stack frame is used for all interrupts. Certain exceptions expand the stack frame by pushing additional parameters on the stack above the PC/PSL pair, as shown in Figure 2–6.

The parameters that are pushed on the stack above the PC/PSL pair, if any, depend on the exception being reported.

**Figure 2–6 Large Stack Frame**



### 2.2.5.1 Interrupts

A subset of the 31 VAX interrupt priority levels (IPLs) is implemented by the NVAX chip. When an interrupt request is generated, the NVAX chip compares the request with the current IPL of the CPU. If the new request is of higher priority, an internal request is generated. At the completion of the current instruction, or at selected points during the execution of interruptable instructions, a microcode interrupt handler is invoked to process the request. The microcode handler, determines the highest priority interrupt, updates the IPL, pushes a PC/PSL pair on the stack, and dispatches to a macrocode interrupt handler through the appropriate location in the SCB.

The interrupt system is controlled by three IPRs:

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

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

Table 2–3 lists the IPLs implemented by the KA66A CPU module.

## KA66A CPU Module

**Table 2–3 KA66A CPU Module Interrupts**

Interrupt Level (hex)	Interrupt Condition	SCB Vector (hex)
1F – Forced console entry or machine check	CTRL/P typed at the console, Node HALT bit (XBER<29>) set, node reset, or system reset	None; the console is entered using the console halt procedure and is nonmaskable.
1F – Machine check	Machine check	04
1E – Power Fail	XMI AC LO L assertion	0C
1D – "Hard" error notification	Causes reported in XBER, XBEER, NCSR, and NSCSR	60
1C – 1B	Unused	
1A – "Soft" error notification	Causes reported in ECSR, PCSTS, BCETSTS, BCEDSTS, NESTS, CEFSTS, XBER, and NCSR	54
19 – 18	Unused	
17 – Device interrupt	XMI Level 7 interrupt (INTR)	Supplied by the device
16 – Device or special interrupt	XMI interprocessor interrupt (IVINTR) <sup>1</sup> XMI level 6 interrupt (INTR) Interval timer interrupt	80 Supplied by the device C0
15 – Device or special interrupt	Console terminal receive interrupt <sup>1</sup> Console terminal transmit interrupt Programmable timer interrupt (timer 0 takes priority over timer 1) XMI level 5 interrupt (INTR)	F8 FC Programmable by writing to the ICCS register Supplied by the device
14 – Device interrupt	XMI level 4 interrupt (INTR)	Supplied by the device
13 – 10	Unused	
0F – 01	Software interrupt request	80 to 9C indexed by the level

<sup>1</sup>At this IPL, the priority of interrupts is shown in descending order.

---

### 2.2.5.2 Exceptions

Exceptions fall into one of three types:

- Traps
- Faults
- Aborts

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

A **fault** occurs during an instruction that leaves the registers and memory in a consistent state so that eliminating the fault condition and restarting the instruction gives correct results. After the instruction faults, the PC saved on the stack points to the instruction that was executing when the fault occurred.

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

Table 2–4 lists the KA66A CPU module-specific instances of the seven classes of exceptions in the VAX.

**Table 2–4 KA66A CPU Module Exceptions**

Exception Class	Instances
Arithmetic traps/faults	Integer overflow trap
	Integer divide-by-zero trap
	Subscript range trap
	Floating overflow fault
	Floating divide by zero fault
	Floating underflow fault
Memory management exceptions	Access control violation fault
	Translation not valid fault
Operand reference exceptions	Reserved addressing mode fault
	Reserved operand fault or abort
Instruction execution exceptions	Reserved/privileged instruction fault
	Emulated instruction fault
	Extended function (XFC) fault
	Change-mode trap
	Breakpoint fault
Tracing exceptions	Trace fault
System failure exceptions	Kernel stack not valid abort
	Interrupt stack not valid abort

**Table 2-4 (Cont.) KA66A CPU Module Exceptions**

Exception Class	Instances
Machine-check exceptions (aborts)	Console error halt
	Unknown memory management fault
	Illegal interrupt ID value
	Illegal microcode dispatch
	Illegal state during string instruction
	Asynchronous hardware error
	Synchronous hardware error

**2.2.5.3 Unique Exceptions**

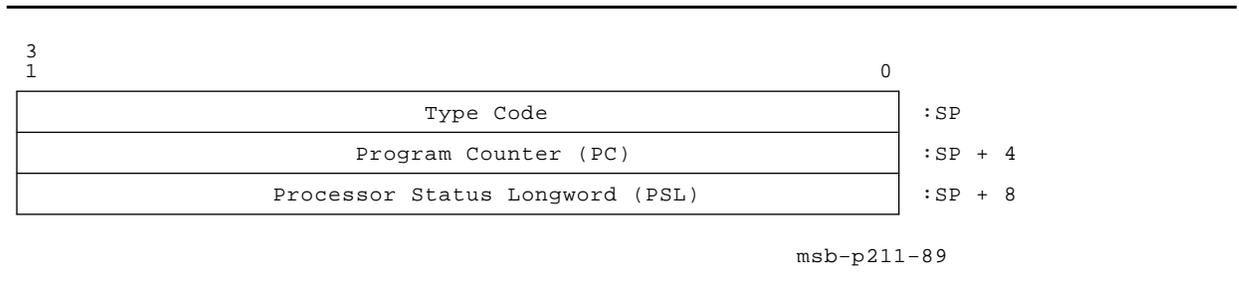
The following exceptions are unique to the NVAX chip. The standard exceptions are described in the *VAX Architecture Reference Manual*.

**Arithmetic Exceptions**

Arithmetic exceptions are detected during the execution of integer or floating-point arithmetic instructions. The exception is reported as either a trap or a fault, depending on the specific event. Figure 2-7 shows the arithmetic exception stack frame.

The exceptions are reported in the manner shown in Table 2-5.

**Figure 2-7 Arithmetic Exception Stack Frame**



**Table 2-5 Arithmetic Exceptions Type Codes**

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

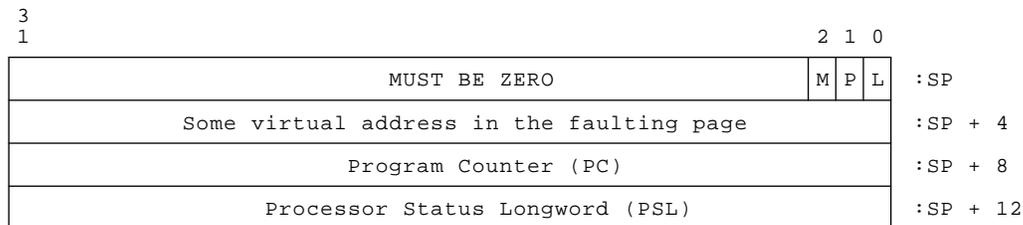
**Memory Management Exceptions**

Memory management exceptions are detected during a memory reference and are always reported as faults. The memory management exceptions are listed in Table 2-6.

**Table 2-6 Memory Management Exceptions**

SCB Vector (hex)	Exception
20	Access control violation
24	Translation not valid

All memory management exceptions push the same frame on the stack, as shown in Figure 2-8.

**Figure 2-8 Memory Management Exception Stack Frame**

msb-p573-91

The M, P, and L bits (bits<2:0>) of the parameter pointed to by the stack pointer are described in the *VAX Architecture Reference Manual* under Memory Management Faults and Parameters.

**Emulated Instruction Exceptions**

The NVAX chip implements the VAX base instruction group and provides microcode that supports the macrocode emulation of certain other instructions. Two types of emulation exceptions depend on the state of PSL<27> (First Part Done, FPD). If FPD is zero at the beginning of the instruction, the instruction has no microcode assistance and the exception is reported through SCB vector C8 (hex) as a trap with the stack frame shown in Figure 2-9 and the stack frame's parameters listed in Table 2-7.

If PSL<FPD> is a one at the beginning of the instruction, the instruction has microcode assistance and the exception is reported through SCB vector CC (hex) as a fault with the stack frame shown in Figure 2-10. In this case, PC is the opcode of the emulated instruction.

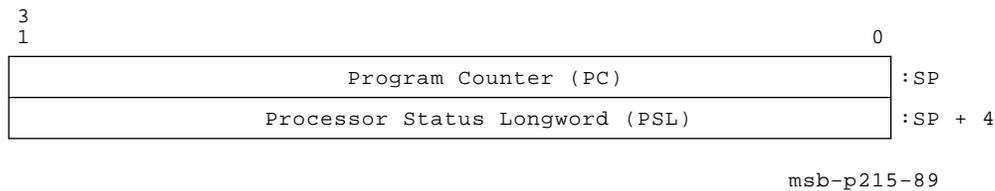
**Figure 2-9 Emulated Instruction Trap**

3		0
1	Opcode	:SP
	Old Program Counter	:SP + 4
	Specifier #1	:SP + 8
	Specifier #2	:SP + 12
	Specifier #3	:SP + 16
	Specifier #4	:SP + 20
	Specifier #5	:SP + 24
	Specifier #6	:SP + 28
	Specifier #7	:SP + 32
	Specifier #8	:SP + 36
	New Program Counter	:SP + 40
	Processor Status Longword (PSL)	:SP + 44

msb-p214-89

**Table 2–7 Emulated Instruction Trap Stack Frame Parameters**

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

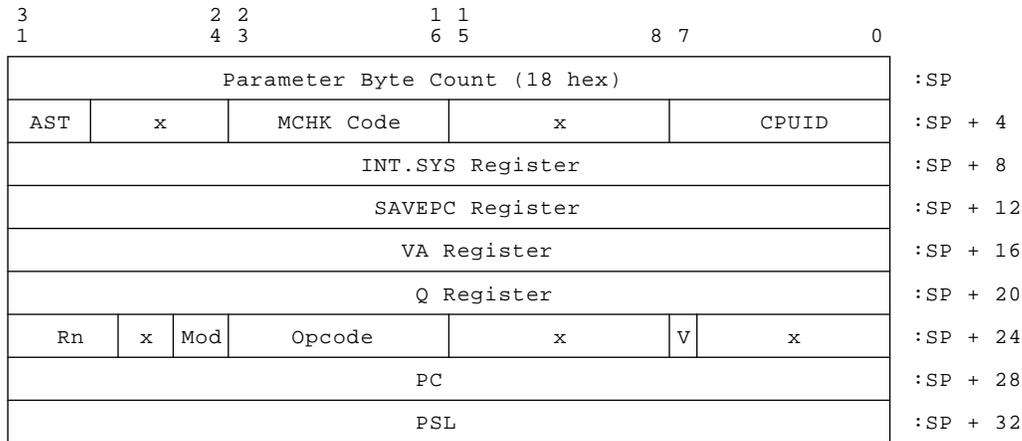
**Figure 2–10 Emulated Instruction Fault**

### Machine Check Exceptions

A machine check exception is reported through SCB vector 04 (hex) when the NVAX chip detects an error condition. The frame pushed on the stack for a machine check indicates the type of error and provides internal state information that may help identify the cause of the error. The machine check stack frame is shown in Figure 2–11 and its parameters are described in Table 2–8. Table 2–9 lists and describes the machine check codes.

Software must acknowledge machine checks by writing a zero to IPR38, MCESR, as a second machine check causes a machine check during machine check processing console halt.

Figure 2–11 Machine Check Stack Frame



msb-p503-91

Table 2–8 Machine Check Stack Frame Fields

Longword Bits	Contents
SP+0 <31:0>	Byte count— The size of the stack frame in bytes, not including the PC, PSL, or the byte count longword. Stack frame PC and PSL values should always be referenced using this count as an offset from the stack pointer.
SP+4 <31:29>	AST LVL— The current value of the register.
<23:16>	Machine check code— The reason for the machine check, as listed in Table 2–9.
<7:0>	CPUID—This field contains the current value of the CPUID register.
SP+8 <31:0>	INT.SYS register— The value of the INT.SYS register, which is read onto the A-bus by the microcode.
SP+12 <31:0>	SAVEPC register— The SAVEPC register which is loaded by microcode with the PC value in certain circumstances. It is used in error handling for PTE read errors with PSL<FPD> set in this stack frame.
SP+16 <31:0>	VA register— The contents of the Ebox VA register, which may be loaded from the output of the ALU.
SP+20 <31:0>	Q register— The contents of the Ebox Q register, which may be loaded from the output of the shifter.
SP+24 <31:28>	Rn— The value of the Rn register, which is used to obtain the register number for the CVTPL and EDIV instructions. In general, the value of this field is UNPREDICTABLE.
<25:24>	Mod— A copy of the current mode field, PSL<CUR MOD>.
<23:16>	Opcode— Bits <7:0> of the instruction opcode. The FD bit is not included.
<7>	VR— The VAX Restart bit, which is used to communicate restart information between the microcode and the operating system. If this bit is set, no architectural state has been changed by the instruction that was executing when the error was detected. If this bit is not set, architectural state was modified by the instruction.
SP+28 <31:0>	PC— The value of the program counter at the time of the fault.
SP+32 <31:0>	PSL— The value of the processor status longword at the time of the fault.

Table 2–9 Machine Check Codes

Code (hex)	Mnemonic	Description	Restart Condition
01	MCHK_UNKNOWN_MSTATUS	Unknown memory management fault parameter returned by Mbox	(VR = 1) or (PSL<FPD> = 1)
02	MCHK_INIT.ID_VALUE	Illegal interrupt ID value returned in INT.SYS	(VR = 1) or (PSL<FPD> = 1)
03	MCHK_CANT_GET_HERE	Illegal microcode dispatch occurred	(VR = 1) or (PSL<FPD> = 1)
04	MCHK_MOVC.STATUS	Illegal combination of state bits detected during string instruction	(PSL<FPD> = 1)
05	MCHK_ASYNC_ERROR	Asynchronous hardware error occurred	Recovery generally not possible
06	MCHK_SYNC_ERROR	Synchronous hardware error occurred	See Section 2.10.6.6 for various recovery conditions.

#### 2.2.5.4 Console Halt

A console halt is a transfer of control by the NVAX CPU microcode directly into console macrocode at the boot ROM address E004 0000 (hex). The restart sequence begins at this location. Console halts occur:

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

No exception stack frame is associated with a console halt. Instead, the SAVPC (IPR42) and SAVPSL (IPR43) processor registers provide the necessary information.

**NOTE: In certain error conditions detected during the execution of a string instruction, the state pack sequence leaves the FPD bit set in the SAVPSL register, but the SAVPC register pointing at the instruction following the string instruction, rather than at the string instruction itself.**

If the FPD bit is not set in the SAVPSL register, SAVPC is correct. Since error halts can normally be restarted, this is not a problem. For a console halt due to the assertion of HALT L, normally the only console halt that can be restarted, SAVPC is always correct, even if the halt interrupt was detected during the execution of a string instruction.

The hardware restart sequence is as follows:

- 1 The NVAX microcode saves the current CPU state.
  - The stack pointer is saved in the appropriate stack pointer IPR
    - IPR0, Kernel Stack Pointer
    - IPR1, Executive Stack Pointer
    - IPR2, Supervisor Stack Pointer
    - IPR3, User Stack Pointer
    - IPR4, Interrupt Stack Pointer
- 2 The current PC is saved in IPR42, SAVPC.
- 3 The PSL, halt code, MAPEN<0>, and a validity bit are saved in IPR43, SAVPSL.
  - SAVPSL<31:16> and <7:0> are loaded from PSL<31:16> and <7:0>.
  - SAVPSL<15> is set to MAPEN<0>.
  - SAVPSL<14> is set to zero if the PSL is valid and is set to one if the PSL is not valid. If the halt is due to a system reset, SAVPSL<14> is undefined.
  - SAVPSL<13:8> is set to the console halt code. The console halt codes are listed in Table 2–11.
- 4 The NVAX microcode then initializes the following CPU state to the values shown in Table 2–10. The remainder of the machine state is undefined.

**Table 2–10 CPU State Initialized on Console Halt**

State	Initialized Value
SP	IPR4 (IS), Interrupt Stack Pointer
PSL	041F 0000 (hex)
PC	E004 0000 (hex)
MAPEN	0
ICCS	0 (after reset, halt code = 3, only)
SISR	0 (after reset, halt code = 3, only)
ASTLVL	4 (after reset, halt code = 3, only)
PAMODE	0 (after reset, halt code = 3, only)
BPCR<31:16>	FECA (hex) (after reset, code = 3, only)
CPUID	0 (after reset, halt code = 3, only)

- 5 Control passes to the console program code at E004 0000 (hex).

Table 2–11 Console Halt Codes

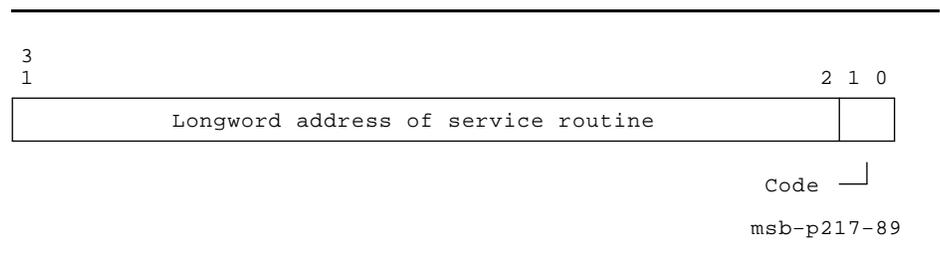
Code (Hex)	Mnemonic	Description
02	ERR_HLTPIN	CTRL/P, break, or external halt
03	ERR_PWRUP	Initial power-up
04	ERR_INTSTK	Interrupt stack not valid during exception processing
05	ERR_DOUBLE	Machine check during exception processing
06	ERR_HLTINS	HALT instruction executed in kernel mode
07	ERR_ILLVEC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

## 2.2.6 System Control Block

The system control block (SCB) is a page containing vectors for servicing traps, faults, software interrupts, and exceptions. IPR17, the System Control Block Base Register (SCBB), points to the SCB.

Each SCB vector is longword aligned in the SCB. The NVAX chip microcode dispatches interrupts and exceptions through the SCB vector, as shown in Figure 2–12.

Figure 2–12 System Control Block Vectors



## KA66A CPU Module

Bits <31:2> of each vector supply the virtual address of the service routine for the interrupt or exception. The routine is longword aligned, as the microcode forces the lower two bits of the address to 00 (hex).

Bits <1:0> of each vector are a code:

00 – The event is to be serviced on the kernel stack unless the CPU is already on the interrupt stack. If the CPU is already on the interrupt stack, the event is to be serviced on the interrupt stack.

01 – The event is to be serviced on the interrupt stack. If the event is an exception, the IPL is raised to 1F (hex).

10 – Unimplemented; results in a console error halt.

11 – Unimplemented; results in a console error halt.

Table 2–12 shows the SCB layout.

**Table 2–12 System Control Block Layout**

Vector (hex)	Name	Type	Number of Parameters	Notes
00	Passive release	Interrupt	None	IPL is raised to requested IPL
04	Machine check	Abort	6	Parameters reflect machine state; must be serviced on the interrupt stack
08	Kernel stack not valid	Abort	None	Must be serviced on the interrupt stack
0C	Power fail	Interrupt	None	IPL is raised to 1E (hex)
10	Reserved/privileged instruction	Fault	None	
14	Customer reserved instruction	Fault	None	XFC instruction
18	Reserved operand	Fault /abort	None	Not always recoverable
1C	Reserved addressing mode	Fault	None	
20	Access control violation/vector alignment fault	Fault	2	Parameters are virtual address, status code
24	Translation not valid	Fault	2	Parameters are virtual address, status code
28	Trace pending (TP)	Fault	0	
2C	Breakpoint instruction	Fault	0	
30	Unused	–	–	Compatibility mode in other VAXes
34	Arithmetic	Trap/fault	1	Parameter is type code
38 – 3C	Unused	–	–	

Table 2-12 (Cont.) System Control Block Layout

Vector (hex)	Name	Type	Number of Parameters	Notes
40	CHMK	Trap	1	Parameter is sign-extended operand word
44	CHME	Trap	1	Parameter is sign-extended operand word
48	CHMS	Trap	1	Parameter is sign-extended operand word
4C	CHMU	Trap	1	Parameter is sign-extended operand word
50	Unused	—	—	
54	Soft error notification	Interrupt	None	IPL is 1A (hex)
58	Reserved	—	—	
59 – 5C	Unused	—	—	
60	Hard error notification	Interrupt	None	IPL is 1D (hex)
64 – 7C	Unused	—	—	
80	Interprocessor interrupt	Interrupt	None	IPL is 16 (hex)
84	Software level 1	Interrupt	None	
88	Software level 2	Interrupt	None	Usually used for AST delivery
8C	Software level 3	Interrupt	None	Usually used for process scheduling
90 – BC	Software levels 4 through 15	Interrupt	None	
C0	Interval timer	Interrupt	None	IPL is 16 (hex)
C4	Unused	—	—	
C8	Emulation start	Fault	10	Same mode exception, FPD=0; parameters are opcode, PC, specifiers
CC	Emulation continue	Fault	None	Same mode exception, FPD=1; no parameters
D0 – F4	Unused	—	—	
F8	Console receiver	Interrupt	None	IPL is 15 (hex)
FC	Console transmitter	Interrupt	None	IPL is 15 (hex)
100 – FFFC	Device vectors	Interrupt	None	Device interrupt vectors

---

## 2.2.7 Process Structure

A process is a single thread of execution. The context of the current process is contained in the process control block (PCB).

The physical address of the current PCB is changed by writing to the Process Control Block Base Register (PCBB), IPR16. The PCB may be located anywhere in memory space and is pointed to by the address contained in the PCBB. The LDPCTX instruction loads a process context from the PCB as described in the *VAX Architecture Reference Manual*. LDPCTX flushes only the process-space entries from the translation buffer; system-space entries are preserved. When the CPU is in 30-bit addressing mode, PCBB<31:30> are ignored.

Figure 2–13 shows the PCB layout.

Other process structure functions are implemented as described in the *VAX Architecture Reference Manual*.

Figure 2-13 Process Control Block

3			1		
			1		
KSP					: (PCBB)
ESP					+4
SSP					+8
USP					+12
R0					+16
R1					+20
R2					+24
R3					+28
R4					+32
R5					+36
R6					+40
R7					+44
R8					+48
R9					+52
R10					+56
R11					+60
AP(R12)					+64
FP(R13)					+68
PC					+72
PSL					+76
POBR					+80
MBZ	AST	MBZ	POLR		+84
P1BR					+88
MBZ		P1LR			+92

└ Performance Monitor Enable (PME)

msb-p218-89

---

## 2.3 Cache Overview

The KA66A CPU module has three caches.

- **A virtual instruction cache (VIC): 2 Kbytes, direct mapped, virtually addressed, instruction stream only**
- **A primary cache: 8 Kbytes, 2-way set associative, physically addressed, write through, mixed instruction and data**
- **A backup cache: 2 Mbytes, direct mapped, physically addressed, writeback, mixed instruction and data**

These three caches are designed to improve the performance of VAX 6000 Model 600 systems. Access to data in caches is much faster than access to data in memory and therefore the speed with which work can be done is much improved by cache designs.

KA66A caches are hierarchical. The VIC, which is virtually addressed, holds I-stream data only. It is a subset of the primary cache, which is a subset of the backup cache. Under such conditions cache coherency becomes an issue, and the sequence of turning caches on and off is important. The B-cache is a writeback cache, the P-cache is a write-through cache, and the VIC is not written to memory at all.

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

---

### 2.3.1 Writeback Cache and Ownership Concepts

The fundamental difference between a writeback cache and a write-through cache is that in a write-through cache data is always written to memory. In a writeback cache, data is always written into the cache but is not necessarily forwarded to memory. The data is written back to memory only if another device in the system needs that data, or if the block is displaced (deallocated) from the cache.

A block of data in the NVAX writeback cache can be in one of three states: invalid, valid-unowned, and valid-owned. A valid-unowned block is a read-only copy of memory data. A valid-owned block may be written, and if it has been written since being put into the cache, it is the only up-to-date copy of the data in the system. The NVAX cache makes no distinction between valid-owned blocks it has written and those that it has not written. An invalid block is one that was once valid but has been invalidated because some other device modified it in memory.

A valid-unowned copy of a given cache block may reside in one or more backup caches in a multiprocessor system. No backup cache may contain a valid cache block that is valid-owned by another backup cache in the system.

Memory is implemented with an ownership bit associated with each hexword of data. When memory receives an Ownership Read (OREAD) for a hexword it owns, ownership and data is passed to the requesting CPU. If another Ownership Read arrives for that hexword from a second CPU, memory does not return the data since the hexword is not owned by memory but by the first CPU. The first CPU recognizes the OREAD as a cache coherence transaction and writes back the data from its cache, using the Disown Write Mask command. The data is then available for the second CPU from memory.

## 2.3.2 Virtual Instruction Cache

The virtual instruction cache (VIC) is a subsection of the Ibox which supports instruction prefetching. This 2-Kbyte, direct-mapped, I-stream cache becomes the primary source of instruction stream data for the Ibox. The VIC attributes are summarized in Table 2–13.

**Table 2–13 VIC Attributes**

Characteristic	Implementation
Cache size	2 Kbytes
Access type	Direct-mapped
Block size	32 bytes
Subblock size	8 bytes
Valid bits	4 valid bits/cache block = 1 per subblock
Data parity bits	4 even parity bits/cache block = 1 per subblock
Number of tags	64 tags
Tag parity bit	1 even parity bit per tag
Fill algorithm	Fill forward
Access size	8 bytes
Bus size	8 bytes
Prefetching	None
Data stored	I-stream only
Virtual/physical	Virtual

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

- **VIC Memory Address Register, VMAR (IPR208).** Used as an index to the cache arrays.
- **VIC Tag Register, VTAG (IPR209).** Provides read/write access to the cache tag array.

- **VIC Data Register, VDATA (IPR210).** Provides read/write access to the cache data array.
- **Ibox Control and Status Register, ICSR (IPR211).** Provides control and status functions of the Ibox.

---

### 2.3.3 Primary Cache

The primary cache (P-cache) is a two-way set associative, read allocate, no-write allocate, write-through, physical address cache of I- and D-stream data. With 256, 20-bit tags it stores 256 hexword blocks or 8 Kbytes of data. Each tag corresponds to bits <31:12> of the physical address. There are four quadword subblocks per block with a valid bit associated with each subblock. The access size for both P-cache reads and writes is one quadword. Byte parity is maintained for each byte of data (32 bits per block). One bit of parity is maintained for every tag.

The P-cache represents the first level of D-stream memory hierarchy and the second level of I-stream memory hierarchy. P-cache entries must be invalidated to maintain cache coherency with higher levels of the memory hierarchy.

The P-cache is used as the second source of I-stream data and the first source of D-stream data for the CPU. If an instruction is not found in the VIC, it is looked for first in the P-cache, then in the B-cache, and finally in memory. Data is sought first from the P-cache, then the B-cache, and finally from memory.

---

### 2.3.4 Backup Cache

The backup cache (B-cache) is implemented in RAMs external to the NVAX chip and is controlled by the Cbox. The size of the cache is 2 Mbytes with a block and subblock size of 32 bytes. Since the data bus to the cache is 8 bytes wide, four accesses are required to read out an entire block. ECC protection is provided on each quadword in the cache and for the tag store.

Each block of data (hexword) has a tag, a valid bit, and an owned bit associated with it. When set, the valid bit indicates that the data stored in the B-cache is accurate. When set, the owned bit indicates that the data stored in the B-cache may be the only accurate data for this particular address in the system and control of the data belongs to this CPU. Data may be valid and not owned. Data cannot be owned and not valid.

### 2.3.4.1 Backup Cache Operating Modes

The backup cache has four modes of operation:

- Cache On. Normal operation.
- Cache Off. Reset puts the backup cache into the Off state. The backup cache may be enabled/disabled (turned on/off) by software through the Cbox Control Register (CCTL). **No data can be owned before turning off the B-cache.** Cache off mode is described in Section 2.3.4.4.
- Force Hit. The Cbox forces all memory space reads and writes to hit in the backup cache. This mode is used for testing and initialization. Force hit mode is described in Section 2.3.4.5.
- Error Transition Mode. The Cbox enters error transition mode upon recognition of some error conditions or when put into ETM explicitly by an IPR write. Error transition mode is described in Section 2.3.4.6.

### 2.3.4.2 Cbox Internal Processor Registers

The Cbox controls the B-cache. The processor registers implemented by the Cbox can be divided into three groups, as follows:

- Normal—Those IPRs that address registers in the NVAX chip or system environment
- B-cache tag IPRs—Read/write IPRs that allow direct access to the B-cache tags
- B-cache deallocate IPRs—Write-only IPRs by which a B-cache block can be deallocated

The numeric range for each group is shown in Table 2–14.

**Table 2–14 IPR Address Space Decoding**

IPR Group	Mnemonic	IPR Address Range (hex)	Contents
Normal	None	00000000–000000FF	256 individual IPRs
B-Cache Tag	BCTAG	01000000–011FFFE0 <sup>1</sup>	64K B-cache tag IPRs, each separated by 20 (hex)
B-Cache Deallocate	BCFLUSH	01400000–015FFFE0 <sup>1</sup>	64K B-cache tag deallocate IPRs, each separated by 20 (hex)

<sup>1</sup>Unused fields in the IPR addresses for these groups should be zero. Neither hardware nor microcode detects and faults on an address in which these bits are non-zero.

**NOTE: The address ranges shown in Table 2–14 are those used by the programmer. When processing normal IPRs, the microcode shifts the IPR number left by 2 bits for use as an IPR command address. This positions the IPR number to bits <9:2> and modifies the address range as seen by the hardware to 0–3FC, with bits <1:0> = 00. No shifting is performed for the other groups of IPR addresses.**

Since there are more addresses for the BCTAG group and the BCFLUSH group than are needed, valid IPR addresses are separated by 20 (hex) rather than by one as they are for the normal group. For example, the IPR address for B-cache tag 0 is 0100 0000 (hex), and the IPR address for B-cache tag 1 is 0100 0020 (hex). In this group, bits <4:0> of the IPR address are ignored, so IPR numbers 0100 0001 through 0100 001F all address B-cache tag 0.

Processor registers in all groups except the normal group are processed by the NVAX chip and do not appear on the NDAL. This is also true for a number of the IPRs in the normal group. IPRs in the normal group that are not processed by the NVAX chip are converted into I/O space references and passed to the system environment by a read or write command on the NDAL.

IPRs in the system and in the Cbox are accessed through IPR reads and IPR writes from the Mbox to the Cbox. IPR reads and IPR writes are generated from MFPR and MTPR instructions. When the Cbox recognizes a valid IPR read, it loads the read into the data read latch (DREAD latch) to be processed. The Mbox allows only one data read (DREAD) or IPR read to be outstanding at a time, so that the DREAD latch will not be overwritten. A valid IPR write is loaded into the write packer and proceeds immediately to the write queue.

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

If the IPR read addresses one of the Cbox registers, the Cbox returns the data from the register the same way it would return data for a read hit except it returns just one quadword or less of data, rather than the usual four quadwords. The Cbox then signals the Mbox so it expects no more fills.

If a write-only Cbox register is read, the Cbox returns unpredictable data. Reading an unimplemented Cbox register returns unpredictable data. If an unimplemented register is written, the write is discarded by the Cbox and normal operation continues.

If the Cbox receives an IPR access to an address that is not within the Cbox block of IPR addresses, it converts it to an I/O space read or write. The Cbox merges the IPR address with E100 0000 (hex), effectively adding the base I/O space address of the IPR block to the IPR address. This is done in hardware by forcing bits <31:29> and bit <24> to one. (The other upper bits are expected to be received as zero.)

From this point on, the transaction is treated as an I/O space transaction by the Cbox. It sends the request to the NDAL through the non-writeback queue. When the fill data returns, the data is returned to the Mbox but is not cached by the Cbox. I/O space reads and writes are never cached in the primary cache or the backup cache.

Eighteen registers are used for various purposes by the Cbox. Several are used simultaneously to record errors or for testing purposes. The following lists these registers and describes their interdependence.

- **Cbox Control Register, CCTL (IPR160).** Contains bits that control the behavior of the Cbox.
- **Backup Cache Data ECC Register, BCDECC (IPR162).** Used by diagnostics to test the Cbox error detection logic. Control bits for this register are in the CCTL register.
- Backup Cache Tag Store Error Registers
 

These three registers record errors detected in the B-cache tag store.

  - **Backup Cache Error Tag Status Register, BCETSTS (IPR163).** Holds status information on the error.
  - **Backup Cache Error Tag Index Register, BCETIDX (IPR164).** Holds the address of the location accessed that resulted in the error.
  - **Backup Cache Error Tag Register, BCETAG (IPR165).** Holds data read from the tag store that resulted in error.
- Backup Cache Data RAM Error Registers
 

These three registers record errors detected in the B-cache data RAMs.

  - **Backup Cache Error Data Status Register, BCEDSTS (IPR166).** Holds the status of the error.
  - **Backup Cache Error Data Index Register, BCEDIDX (IPR167).** Holds the address of the location accessed that resulted in the error.
  - **Backup Cache Error Data ECC Register, BCEDECC (IPR168).** Holds the ECC check bits calculated on the B-cache data and check bits.
- Fill Error Registers
 

These two registers contain information about errors related to reads to memory.

  - **Cbox Error Fill Address Register, CEFADR (IPR171).** Holds the address of the memory location that resulted in an error.
  - **Cbox Error Fill Status Register, CEFSTS (IPR172).** Holds the status of a read to memory.
- NDAL Error Registers
 

The NDAL error registers hold information related to NDAL errors.

  - **NDAL Error Status Register, NESTS (IPR174).** Holds error status relating to any problems encountered.
  - **NDAL Error Output Address Register, NEOADR (IPR176).** Holds the address corresponding to the cycle in error.

- **NDAL Error Output Command Register, NEOCMD (IPR178).** Holds the command bits corresponding to the cycle in error.
- **NDAL Error Data High Register, NEDATHI (IPR180) and NDAL Error Data Low Register, NEDATLO (IPR182).** Hold the data from an NDAL cycle where the NVAX detected a parity error on the bus.
- **NDAL Error Input Command Register, NEICMD (IPR184).** Holds the command bits corresponding to a cycle with a parity error.
- **Backup Cache Tag Store Registers, BCTAG (01000000–011FFFE0).** Provide software direct access to the B-cache tag store to aid in error recovery and for diagnostics.
- **Backup Cache Flush Registers, BCFLUSH (01400000–015FFFE0).** Used to deallocate cache blocks.

---

#### 2.3.4.3 Tag Store and Data RAM Control

The four operating states of the B-cache are controlled by four bits in the Cbox Control (CCTL) Register: Enable, Force Hit, SW ETM, and HW ETM. The four states — on, off, force hit, and error transition mode — are determined as follows:

- 1 If the Enable bit is clear, the B-cache is off and cannot be accessed regardless of the state of any other control bits.
- 2 When the Enable bit is set and Force Hit is set, the B-cache is in force hit mode regardless of the state of any other control bits.
- 3 When the Enable bit is set and Force Hit is clear, and either SW ETM or HW ETM is set, the cache is in ETM mode.
- 4 When the Enable bit is set and force hit, SW ETM, and HW ETM are clear, the cache is on.

The On state is the normal operating condition of the cache. Off, force hit, and ETM modes are described in the following sections.

---

#### 2.3.4.4 Backup Cache Is OFF

The backup cache may be off for two reasons: the chip has just powered up, or software has disabled the cache by clearing the Enable bit in the Cbox Control Register (CCTL<0>).

When the cache is off, no accesses to the backup cache are done. Errors are not detected and cache state is unchanged unless explicitly changed by software through IPR reads and writes.

When the backup cache is off, all cache coherency requests that arrive are forwarded as invalidates to the Mbox, as the data may be valid in the P-cache. Fills that return are sent directly to the Mbox without incurring the overhead of cache access.

When the cache is off, a DREAD Lock/Write Unlock pair from the Mbox becomes hexword Ownership Read/quadword Disown Write on the NDAL.

Any writes issued when the B-cache is off are of quadword length.

A DREAD Modify command from the Mbox normally becomes an OREAD on the NDAL when it misses in the cache. However, when the cache is off, a normal DREAD is used on the NDAL.

---

#### 2.3.4.5 Backup Cache Is in Force Hit Mode

Force hit mode is for testing purposes only.

When Force Hit and Enable are set, all memory space reads and writes are forced to hit in the B-cache. Tag store state is not changed at all; the data RAMs are accessed as if the tag store access produced an owned-valid hit. Cache coherency transactions are treated as if the B-cache were off: no lookups are done and the transactions are forwarded to the Mbox.

When the B-cache is in force hit mode, deallocates are not done. Even if the tag matches and the valid and owned bits are set, the block is not written back. The implication of this is that if force hit mode is used in a multiprocessor environment, the B-cache must first be flushed of all owned blocks.

Tag store and data RAM ECC errors are detected in force hit mode if the Disable ECC Errors bit in the CCTL register is not set, resulting in the usual error handling.

Force hit mode is used to test the ECC logic for the data RAMs, as follows: Set SW ECC in the Cbox Control Register. Write the desired ECC into BCDECC. Do a D-stream write to the desired location, and the location will be written using ECC from BCDECC rather than from Cbox-generated ECC. Assume the ECC value written is incorrect; an ECC error will be flagged when the data is read. Perform a read of the location while Force Hit is still set. The read will result in an ECC error, showing that the logic is working correctly. The data RAM error registers may be read and will correspond to the induced error.

---

#### 2.3.4.6 Backup Cache Is in Error Transition Mode

When the Cbox detects certain errors, it puts itself into error transition mode. The CPU remains in ETM until software explicitly disables or enables the cache. To ensure cache coherency, the cache must be completely flushed of valid blocks before it is reenabled because some data can become stale while the cache is in ETM.

Table 2–15 describes how the backup cache behaves when in ETM.

Any reads or writes that do not hit valid-owned data during ETM are sent to memory: read data is retrieved from memory, and writes are written to memory, bypassing the cache entirely.

In ETM mode the cache behaves normally supplying data for IREADs, DREADs, and Read Modifs that hit valid-owned data.

If a write hits a valid-owned block in the cache, the block is written back to memory and the write is also sent to memory. If a Read Lock hits valid-owned data in the cache, a writeback of the block is forced and the Read Lock is sent to memory (as an OREAD on the NDAL). This behavior enforces write ordering between previous writes that may have missed in the cache and the Write Unlock that will follow the Read Lock.

**Table 2–15 Backup Cache Behavior During ETM**

Cache Transaction	Cache Response		
	Miss	Valid Hit	Owned Hit
IREAD, DREAD, Read Modify	Read from memory	Read from memory	Read from cache
CPU Read Lock	Read from memory	Read from memory	Force block writeback, read from memory
CPU Write	Write to memory	Write to memory	Force block writeback, write to memory
CPU Write Unlock	Write to memory	Write to memory	Write to cache
Fill (from read started before ETM)	Normal cache behavior		
Fill (from read started during ETM)	Do not update backup cache; return data to Mbox		
NDAL cache coherency request	Normal cache behavior except that Inval always goes to P-cache		

Table 2–15 shows that during ETM, cache coherency requests are treated as they are during normal operation. Fills caused by any type of read originating before the cache entered ETM are processed in the usual fashion. If the fill is the result of a write miss, the write data is merged, as usual, as the requested fill returns. Fills caused by any type of read originating during ETM are not written into the cache or validated in the tag store.

While the B-cache is in ETM mode, changes to the cache state are kept to a minimum. Table 2–16 shows how each transaction modifies the state of the cache.

### 2.3.4.7 How to Turn the B-Cache Off

Care must be taken to maintain cache coherency when turning the B-cache off. If the cache is running normally and software wishes to turn it off, it must do the following:

- 1 Put the B-cache in ETM by setting CCTL<SW ETM>. In this mode the B-cache will not allocate any new blocks and will send all cache coherency requests to the Mbox as invalidates.
- 2 Use the BCFLUSH register to flush all owned blocks from the cache.
- 3 Write CCTL to clear Enable and SW ETM simultaneously. If an error was encountered during the deallocate process, HW ETM may be set and if so, should be cleared as well.

If the B-cache encounters an uncorrectable ECC error, the Cbox sets HW ETM in the CCTL register. If software wishes to turn off the cache, it must do the following:

- 1 Use the BCFLUSH register to flush all owned blocks from the cache.

2 Write CCTL to clear Enable and HW ETM simultaneously.

**Table 2–16 Backup Cache State Changes During ETM**

Cache Transaction	Cache State Modified		
	Miss	Valid Hit	Owned Hit
IREAD, DREAD, Read Modify	None	None	None
Read Lock	None	None	Clear Valid & Owned; change TS ECC accordingly.
Write	None	None	Clear Valid & Owned; change TS ECC accordingly.
Write Unlock	None	None	Write new data, change DR ECC accordingly.
Fill (from read started before ETM)		Write new TS TAG, TS VALID, TS OWNED, TS ECC, DR DATA, DR ECC	
Fill (from read started during ETM)		None	
NDAL cache coherency request		Clear Valid & Owned; change TS ECC accordingly	

#### 2.3.4.8 How to Turn the B-Cache On

On power-up, undefined data is stored in the B-cache tags and data. Should the cache be turned on immediately, ECC errors would result. Therefore, the cache must be initialized.

Through a series of IPR writes, every B-cache tag store entry must be written with cleared owned and valid bits. The value written to the tag is irrelevant, as long as correct ECC is written to the tag store.

Once the tag store has been initialized, the cache may be enabled by setting B-Cache Enable in the CCTL register.

It is necessary to initialize the B-cache data RAMs with correct ECC on power-up. ECC errors in the data RAMs are not ignored.

Force hit mode may be used to initialize the B-cache data RAMs with correct ECC. If full quadword writes are used, no data RAM errors will be detected during this process, since the RAMs are written without being read first. If partial quadword writes are used, errors will be detected because of the Read-Modify-Write that is necessary. If the programmer sets the Disable ECC Errors bit in the CCTL register, the Cbox will ignore these errors.

If the B-cache is in ETM, it may be incoherent with respect to other CPUs and memory. (Table 2–15 shows how writes that hit valid but not owned cached data are not written into the cache.) In addition, the P-cache, if enabled, is no longer a subset of the B-cache.

The programmer must ensure that when the B-cache is reenabled, all the owned and valid bits are cleared.

## 2.3.5 Cache Initialization

A combination of the console code and the operating system does the final initialization.

### 1 Initialize the VIC.

```

;          This code initializes the VIC by writing all 128
;          tags with good parity and all valid bits clear.
;
movl    #^x00000020, r0    ; tag index increment = 1 hexword
movl    #0, r1             ; block tag init value
movl    #0, r2             ; VIC tag starting address
movl    #^x00000800, r3    ; VIC tag ending address + 1 block
movl    #PR19$_VMAR, r4    ; VIC memory address register
movl    #PR19$_VTAG, r5    ; VIC tag register (VTAG)

vic_loop:
mtpr    r2, r4             ; write current index to VMAR
mtpr    r1, r5             ; write the tag via VTAG
addl2   r0, r2             ; increment index by the block size
cmpl    r3, r2             ; check if done
bneq    vic_loop          ;

```

### 2 Enable the VIC.

```
mtpr    #<icsr$m_enable+icsr$m_lock>, #PR19$_ICSR
```

### 3 Initialize the B-cache.

```

;          This code initializes the B-cache by writing all tags with good
;          ECC and all valid and owned bits clear. This example initializes
;          a 512Kb B-cache. This code can be changed to init the other legal
;          B-cache sizes by changing the value in R3. SW_ECC in CCTL is clear,
;          so the CBOX will generate correct ECC for the tag/valid/owned bits.
;
movl    #^x00000020, r0    ; tag index increment = 1 hexword
movl    #0, r1             ; block tag init value
movl    #^x01000000, r2    ; B-cache tag starting address
movl    #^x01080000, r3    ; B-cache tag ending address + 1 block
;                               ; for 512Kb B-cache

B-cache_loop:
mtpr    r1, r2             ; write tag to current tag address
addl2   r0, r2             ; increment index by the block size
cmpl    r3, r2             ; check if done
bneq    B-cache_loop      ;

```

### 4 Initialize the P-cache.

```

;          This code initializes the P-cache by writing all 256 tags with
;          good parity and all valid bits clear.
;
movl    #^x00000020, r0    ; tag index increment = 1 hexword
movl    #0, r1             ; block tag init value
movl    #^x01800000, r2    ; P-cache tag starting address
movl    #^x01802000, r3    ; P-cache tag ending address + 1 block

```

```

P-cache_loop:
    mtpc    r1, r2            ; write tag to current tag address
    addl2   r0, r2            ; increment index by the block size
    cmpl    r3, r2            ; check if done
    bneq    P-cache_loop     ;

```

## 5 Enable the B-cache and the P-cache.

Cache coherency requires that the P-cache is always a subset of the B-cache. The code below enabling the caches ensures that this is true. The B-cache is enabled first, and an REI is executed between the B-cache enable and the P-cache enable. The purpose of the REI is to synchronize data prefetching so that the P-cache will not perform any fills to addresses that were not also filled in the B-cache.

```

mfpr    #PR19$_CTL, r6      ; get current value in Cbox CTL IPR
bisl2   #<cctl$m_enable>, r6 ; set the B-cache enable bit
mtpc    r6, #PR19$_CTL     ; write the new Cbox CTL IPR

movpsl  -(sp)               ; push the psl
moval   init_cont, -(sp)    ; and the next PC
rei     ; branch to the next PC
        ; flushing the VIC and aborting
        ; all previous IREADS

        ; Now that state is synchronized,
        ; enable the P-cache

init_cont:
    mtpc    #<pcctl$m_d_enable+pcctl$m_i_enable+pcctl$m_p_enable>, -
            #PR19$_PCCTL

```

---

## 2.4 NVAX Box Descriptions

**The macropipelined VAX design is implemented through a group of tightly coupled pieces of logic, called boxes, that perform VAX functions relatively independently of each other. This section describes each of these boxes.**

---

### 2.4.1 Ibox

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

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

The Ibox has both read and write ports to the general purpose register (GPR) and memory data (MD) portions of the Ebox register file, which are used to process the operand specifiers. The Ibox maintains a scoreboard to ensure that reads and writes to the register file are always performed in synchronization with the Ebox. The Ibox stops processing instructions and operands upon issuing certain complex instructions, like CALL, RET, or character string instructions, so that proper read/write ordering is maintained while the Ebox alters large amounts of VAX state.

Since the Ibox is often parsing several macroinstructions ahead of the Ebox, the correct value for the PSL condition codes is not known at the time the Ibox executes a conditional branch instruction. Rather than emptying the pipe, the Ibox predicts which direction the branch will take, and passes this information on to the Ebox via the branch queue. The Ebox later signals if there was a misprediction, and the hardware backs out of the path. The branch prediction algorithm uses a 512-entry RAM, which caches four bits of branch history per entry.

---

#### 2.4.1.1 Effects of Ibox Pipelining

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

Instruction prefetching also provides a buffer 4 bytes wide by 4 elements deep that isolates the instruction parser from the bursts of data coming in from cache and memory. The result is that the instruction fetching and instruction parsing can be done in parallel.

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

---

#### 2.4.1.2 Branch Prediction Unit

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

Since branch direction relies on Ebox condition codes, the Ibox has no prior knowledge of branch direction. Branch prediction logic makes a prediction on which way the branch will go and forces the Ibox to take that path. The program counter pointing to the alternate branch path is saved should the prediction prove wrong. If the prediction was wrong, the Ibox is redirected to the correct path.

---

## 2.4.2 Ebox

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

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

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

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

---

### 2.4.3 Fbox

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

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

---

### 2.4.4 Mbox

The Mbox manages data coming in and going out of the NVAX chip. The Mbox receives read requests from the Ibox (both instruction stream and data stream) and from the Ebox (data stream only). It receives write/store requests from the Ebox, and the Cbox sends the Mbox fill data and invalidates for the P-cache. The Mbox arbitrates between these requesters,

and queues requests that cannot currently be handled. Once a request is started, the Mbox performs address translation and cache lookup in two cycles, assuming there are no misses or other delays. The two-cycle Mbox operation is pipelined.

The Mbox uses the translation buffer (96 fully associative entries) to map virtual to physical addresses. In the case of a TB miss, the memory management hardware in the Mbox will read the page table entry and fill the TB. The Mbox performs all access checks, TNV checks, and quadword unaligned data processing.

The Mbox contains the primary cache (P-cache), which is described in Section 2.3.3.

To prevent the Ibox from using data that the Ebox should have written before use, the Mbox scoreboards physical addresses. This memory "scoreboarding" is done using the physical address queue, a small list of physical addresses that have a pending Ebox store.

---

#### 2.4.4.1 Translation Buffer Tag Fills

The NVAX TB tag fill command is used with the TB PTE fill command to cache a PTE in the translation buffer. The data associated with the TB tag fill command corresponds to a virtual byte address in some virtual page. The TB tag fill command causes the page address of the TB tag fill data to be written into the tag field of the TB entry pointed to by the NLU TB allocation pointer. The TB valid bit (TBV) of the entry is cleared.

There are two sources: the memory management exception latch (MME latch) and the latch that holds commands from the Ebox, called the EM latch, from which the virtual page number (VPN) is driven into the TB as a tag. The formats of the two differ. Bit <0> is a zero when sourced from the MME latch and is an even parity bit when sourced from the EM latch.

When sourced from the EM latch, even tag parity corresponding to the VPN is specified in bit <0> of the data field for the TB tag fill. This mechanism allows correct or incorrect parity to be written into the TB tag array for testing purposes by invoking the TB tag fill operation through the appropriate MTPR instruction.

---

#### 2.4.4.2 Translation Buffer PTE Fills

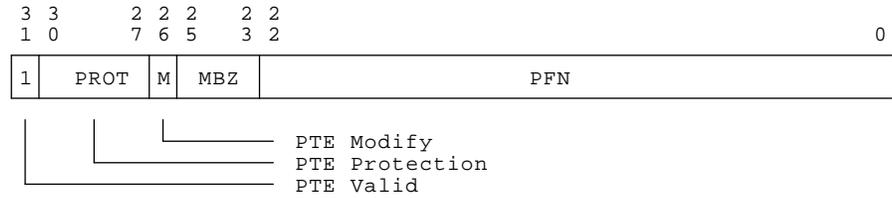
The TB PTE fill operation drives the PTE data to be written into the data array of the translation buffer. The data is written into the entry pointed to by the NLU TB allocation pointer. The TB valid bit (TBV) of the entry is set. (Note that a TB TAG fill command will not be issued by the Mbox if PTE<31> is clear in order to guarantee that only validated PTEs are cached in the TB.) The NLU TB allocation pointer is incremented after the fill is done.

TB PTE fill operations can also be sourced from either the MME latch or the EM latch. Their formats differ depending upon the source.

When TB PTE fills occur from the MME latch, the PTE data is driven in the format shown in Figure 2-14. Only bits <30:26>, <22:0>, and the corresponding PTE parity bit are written into the TB array during a TB PTE fill. TB PTE fills from the MME latch will only be issued for validated PTEs. Therefore, PTE<31> will always be set. The TB

logic will automatically generate even parity to be written during the fill corresponding to PTE<31:0>. Note that the parity generator includes PTE<31> in this calculation even though this bit is not written into the TB. Since PTE<31> is always set during a TB PTE fill, the stored parity can be thought of as odd parity on bits <30:0>.

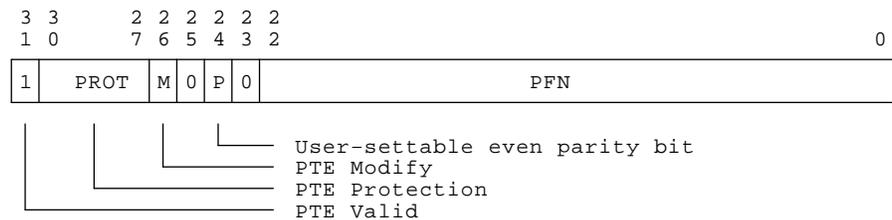
**Figure 2–14 PTE Fills from MME Latch**



msb-p571-91

When TB PTE fills occur from the EM latch, the PTE data is in the format shown in Figure 2–15. Bits <30:26> and <22:0> are written into the TB array during a TB PTE fill. Bit <24> is interpreted as the corresponding PTE parity and is directly written into the TB as such. This gives the user the flexibility of writing correct or incorrect PTE parity for testing purposes. Note, however, that while PTE<31> is not written into the TB, it must be assumed that this bit is set when the user calculates even parity on PTE<31:0>. Similarly, PTE<25> and PTE<23> must be cleared for proper parity calculation.

**Figure 2–15 PTE Fills from EM Latch**



msb-p572-91

---

#### 2.4.4.3 Recording Mbox Errors

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

- **TB Parity Address Register, TBADR (IPR236)**. Holds the virtual address associated with a translation buffer parity error.
- **TB Parity Status Register, TBSTS (IPR237)**. Holds the error status associated with errors occurring in the Mbox.
- **P-Cache Parity Address Register, PCADR (IPR242)**. Holds the physical address associated with a P-cache parity error.
- **P-Cache Status Register, PCSTS (IPR244)**. Holds the error status of errors that occur in the P-cache.

When the operating system error handler routine is invoked from a microtrap or interrupt, the handler determines what errors were present by reading the state of all the error registers using IPR read operations. IPR read operations are invoked by MFPR instructions.

---

### 2.4.5 Cbox

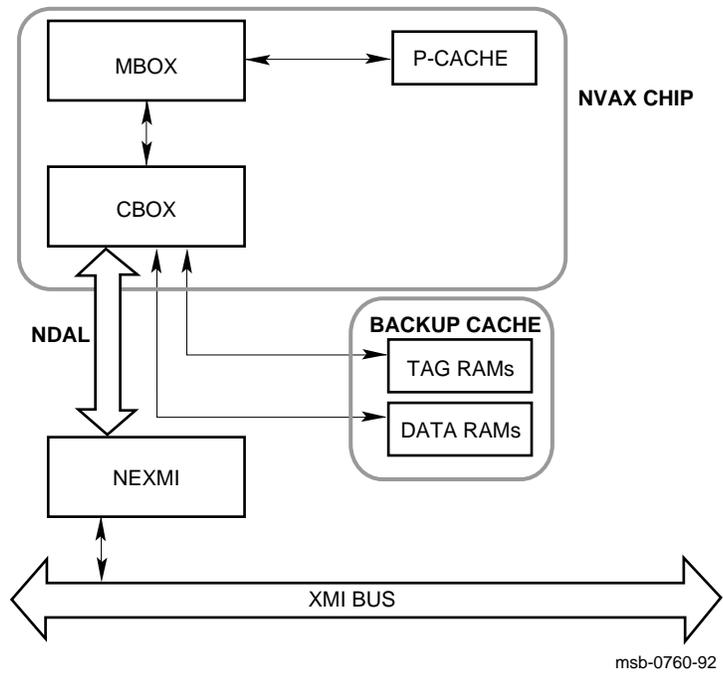
The Cbox is tightly coupled to the Mbox. It has three external buses that communicate with the backup cache tag RAMs, the backup cache data RAMs, and the NEXMI chip. Figure 2–16 is a block diagram of the Cbox.

The Cbox controls the backup cache and interfaces to the NDAL bus. The backup cache is a writeback cache. Cache tags and cache data are stored in static RAMs on the module. The Cbox implements the control for the cache tags and data and control for the NDAL.

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

The Cbox communicates with the memory subsystem (everything beyond the backup cache) through the NDAL. The Cbox generates reads and receives fills; it receives cache coherence transactions from the NDAL to which it responds with invalidates and writebacks, as appropriate.

Figure 2-16 Cbox in the System



## 2.5 KA66A TOY Clock and Interval Timer

The KA66A module includes a Time-of-Day Register (TODR), Time-of-Year (TOY) watch chip, and an interval clock (ICCS, ICR, NICR). The implementation of TODR and the interval clock are as defined in the *VAX Architecture Reference Manual*.

### 2.5.1 Time-of-Day Register (TODR)

The KA66A Time-of-Day Register forms an unsigned 32-bit binary counter that is driven from a 100-Hz oscillator, so that the least significant bit of the clock represents a resolution of 10 milliseconds. The R/W register counts only when it contains a non-zero value.

### 2.5.2 Programmable Interval Clock

The interval clock provides an interrupt at IPL 16 (hex) at programmed intervals. To use the Programmable Interval Timer (PIT), the ECR<ICCS EXT> bit must be set. The counter is incremented at 1 microsecond intervals, with at least .01% accuracy. References to the ICCS must be made using the IPR address 18 (hex) and not to the I/O address E100 0060 (hex), otherwise unpredictable results will occur. The IPR addresses should be used for all three registers, the ICCS, NICR, and ICR (see Table 2–17).

**Table 2–17 Interval Clock Register Addresses**

Register	IPR Address Decimal (Hex)	I/O Address (Hex)
Interval Clock Control and Status	24 (18)	E100 0060
Next Interval Count	25 (19)	E100 0064
Interval Count	26 (1A)	E100 0068

The interval clock consists of three registers:

- **Interval Count Register, ICR (IPR26).** The interval count register is a read-only register incremented every microsecond. Upon a carry out (overflow) from bit <31>, it is automatically loaded from NICR, and an interrupt is generated if the interrupt is enabled.
- **Next Interval Count Register, NICR (IPR25).** This reload register is a write-only register that holds the value to be loaded into ICR when ICR overflows. The value is retained when ICR is loaded.

- **Interval Clock Control and Status Register, ICCS (IPR24).** The ICCS register contains control and status information for the interval clock.

To use the interval clock, load the negative (2's complement) of the desired interval into the Next Interval Count Register. Then, writing 51 (hex) to the ICCS will enable interrupts, load the next interval into the Interval Count Register, and set ICCS<0>. An interrupt will then occur every "interval count" microseconds. The interrupt routine should write C1 (hex) to the ICCS to clear the interrupt. If Interrupt has not been cleared (the interrupt has not been handled) by the time of the next ICR overflow, ICCS<ERR> will be set.

If NICR is written while the clock is running, the clock may lose or add a few ticks. If the interval clock interrupt is enabled, this may cause the loss of an interrupt.

The interrupt bit (ICCS<INT>) sets, and an interrupt is posted if ICCS<IE> is set, when the interval counter overflows. The Interval Count Register is then loaded from the Next Interval Count Register and continues incrementing.

### 2.5.3 Time-of-Year Clock

The time-of-year (TOY) clock consists of a "watch" chip located on the ROM bus. The watch chip enables the KA66A to keep time through a power outage or system shutdown that lasts up to 100 hours. Three byte-wide control and status registers (CSRA, CSRB, and CSRD) on the watch chip allow software to write the time in the watch chip registers during installation. Then, in normal operation, software reads the watch chip during the bootstrap operation. Because the watch chip stores time information in units of seconds, minutes, hours, days, and months, the operating system must convert this data to a 32-bit format before using it.

The TOY clock is maintained during power-fail conditions by supplying battery backup to the watch chip and to the low frequency external oscillator which the watch uses for this purpose.

Table 2–18 gives addresses of registers containing watch chip data.

**Table 2–18 Watch Chip Data**

Address	Units	Decimal Range	Hex Range
E018 3000	Seconds	0–59	00–3B
E018 3002	Minutes	0–59	00–3B
E018 3004	Hours	0–23	00–17
E018 3007	Day of Month	1–31	01–1F
E018 3008	Month	1–12	01–0C
E018 3009	Year	0–99	00–63

**NOTE: Bits <6:0> of the watch chip's Seconds Register (E018 3000) are read/write, making it possible to write a value greater than 59 (dec). Writing a value outside the acceptable range (0–59) gives unpredictable results.**

Table 2–19 shows examples of how data stored in registers is converted to time. Table 2–20 gives the addresses of watch chip control registers.

**Table 2–19 Watch Chip Example**

Time	Output (bin)	Output (hex)
21 seconds	00010101	15
58 minutes	00111010	3A
5 hours	00000101	05
15th day	00001111	0F
February	00000010	02
92nd year	01011100	5C

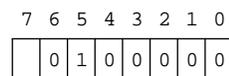
**Table 2–20 Watch Chip Control Registers**

Address	Function
E018 300A	CSR A–Busy bit
E018 300B	CSR B–Off bit
E018 300C	CSR C–Reserved
E018 300D	CSR D–Valid bit
E018 300E–E018 303F	50 bytes RAM

The control and status registers on the watch chip allow software to:

- 1 Check the validity of the date and time registers.
- 2 Set the time.
- 3 Stop and start the chip.

**Figure 2–17 Watch Chip CSR A (E018 300A)**



└ Busy

msb-p564-91

- Bit<7>, Busy (read only)

Busy = 1: The watch chip is busy with an update cycle; date and time registers are undefined.

Busy = 0: The watch chip is not busy; date and time registers are valid.

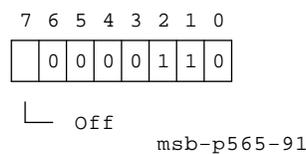
Software should check Busy before reading the date and time registers.

The watch chip sets Busy for 2 milliseconds every second. If software finds Busy set, it should try again in 2 milliseconds.

If Busy is cleared, software has at least 244 microseconds to read the date and time registers before the next update cycle.

- Bits <6:0>, Miscellaneous setup bits (read/write) - Software must write these bits as shown in Figure 2–17 before it sets the time in the watch chip.

**Figure 2–18 Watch Chip CSR B (E018 300B)**



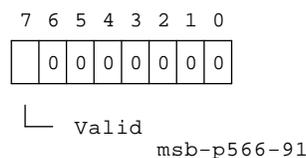
- Bit <7>, Off (read/write)

Software must stop the watch chip by setting Off before it loads the date and time registers.

Software can start the watch chip after setting the time by clearing Off.

- Bits <6:0>, Miscellaneous setup bits (read/write) - Software must write these bits as shown in Figure 2–18, when it sets the Off bit and before loading the date and time registers.

**Figure 2–19 Watch Chip CSR D (E018 300D)**



- Bit<7>, Valid (Read only)

The Valid bit indicates that the time in the watch chip registers is correct. If the battery backup voltage falls below the required level, a sensing circuit clears the Valid bit.

Valid = 1: Watch chip registers are valid.

Valid = 0: Watch chip registers are invalid.

**NOTE: The watch chip sets Valid to one after software reads CSR D. Therefore, when software reads Valid as zero, it should immediately update the date and time registers in the watch chip. Otherwise, the state of the Valid bit will be misleading.**

To read time from the watch chip, code should:

- 1 Check the Busy bit.
- 2 If Busy is clear, perform the read data operations.
- 3 If Busy is set, wait for Busy to clear and then perform read operations.

To write time into the watch chip, code should:

- 1 Set the Off bit.
- 2 Check the Busy bit.
- 3 If Busy is clear, perform the write operations.
- 4 If Busy is set, wait for Busy to clear and then perform the write operations.
- 5 Clear the Off bit.

---

## 2.6 XMI Interface

**The KA66A CPU module uses the NEXMI chip to interface between the XCI bus and the CPU module's NDAL bus. The XCI bus is the interface between the NEXMI and the XMI Corner.**

**The XMI is the interconnect for the VAX 6000 family of machines. There are two XMI protocols; one that supports writeback cache designs and one that does not. The VAX 6000 Model 600 uses the XMI that supports the writeback cache design.**

The primary tasks of the XMI interface are to:

- Translate NVAX memory and I/O space references to the appropriate XMI transactions.
- Implement a writeback buffer to handle writeback requests from the NVAX Cbox.
- Support control of cache fills and cache invalidates.
- Support XMI-required interrupt logic.
- Implement all XMI-required registers.
- Provide ROM bus and system support.

---

### 2.6.1 XMI Address Space

Although the XMI has a 40-bit address space divided equally between memory and I/O space, the KA66A CPU module implements a 30-bit and a 32-bit (4 Gbytes) address space. See the *VAX 6000 Platform Technical User's Guide* for full details on memory space.

---

#### 2.6.1.1 XMI Memory Space

XMI memory space covers addresses 0000 0000 to DFFF FFFF (hex). NVAX references in this range are initially accessed in XMI memory. However, since read references that miss both caches normally result in a cache fill of both caches, future references may be serviced by the caches. All XMI writes and reads to memory are monitored by the KA66A CPU module for "hits" in the NVAX caches. If a hit occurs, then an invalidate or writeback request is posted by the NVAX Cbox.

---

#### 2.6.1.2 XMI I/O Space

XMI I/O space covers addresses E000 0000 to FFFF FFFF. Data in this range is never cached by the KA66A CPU module. All I/O addresses with the exception of the first 24 Mbytes are transmitted on the XMI. The first 24 Mbytes of I/O space (called XMI private space) are for implementation of registers that are not accessible from the XMI.

Figure 2–20 KA66A CPU Module Private I/O Address Space Map

Byte Address E000 0000	NCSR	
E000 0004 E003 FFFF	RESERVED	256 Kbytes
E004 0000 E009 FFFF	Self-Test/Console/Boot Code (halt protected) three (3) 128 Kbytes X 8 PROMs	384 Kbytes
E00A 0000 E00B FFFF	Self-Test/Console/Boot Code (halt protected) one (1) 128 Kbytes X 8 PROM (Expansion)	128 Kbytes
E00C 0000 E00C 7FFF	Self-Test/Console/Boot Code (halt protected) one (1) 32 Kbytes X 8 EEPROM	32 Kbytes
E00C 8000 E00D FFFF	RESERVED	96 Kbytes
E00E 0000 E013 FFFF	Self-Test/Console/Boot Code (not halt protected, PROM)	384 Kbytes
E014 0000 E015 FFFF	Self-Test/Console/Boot Code (not halt protected, PROM) (Expansion)	128 Kbytes
E016 0000 E016 7FFF	Self-Test/Console/Boot Code (not halt protected, EEPROM)	32 Kbytes
E016 8000 E017 FFFF	RESERVED	96 Kbytes
E018 0000 E018 9FFF	System Support Address Space Watch Chip, BBU RAM, Stack RAM IPort, OPort, UART	40 Kbytes
E018 A000 E0FF FFFF	RESERVED	14.8 Mbytes
E100 0000 E100 03FF	IPR Address Space	1 Kbyte
E100 0400 E100 FFFF	RESERVED	63 Kbytes
E101 0000 E101 FFFF	Interprocessor IVINTR Generation "Virtual" Registers	64 Kbytes
E102 0000 E102 FFFF	Write Error IVINTR Generation "Virtual" Registers	64 Kbytes
E103 0000 E17F FFFF	RESERVED	~7.75 Mbytes

msb-p550-91

## 2.6.2 XMI Transaction Generation/Response Tables

The NEXMI can generate the following XMI transaction types:

- Hexword memory Reads
- Hexword memory Ownership Reads
- Hexword memory Disown Write Masks
- Quadword memory Write Masks
- Quadword memory Disown Write Masks
- Longword I/O Reads
- Longword I/O Write Masks
- Write Error IVINTRs
- Interprocessor IVINTRs
- IDENTs (in response to NVAX Interrupt Acknowledge)

The NEXMI responds to the following XMI transaction types:

- Longword nodespace Reads
- Longword nodespace Write Masks
- Interrupts
- Memory Reads and Writes for cache invalidates

**Table 2–21 NEXMI Transaction Generation/Response for NVAX Chip-to-XMI Operations**

NVAX Chip Operation	Resulting XMI Operation
Memory Space References	
I-stream Read (hexword)	Hexword Read
D-stream Read (hexword)	Hexword Read
D-stream Read Ownership (hexword)	Hexword Ownership Read
Write Disown (hexword)	Hexword Disown Write Mask
Write Mask (quadword) cache off	Quadword Write Mask
Write Disown (quadword) cache off	Quadword Disown Write Mask
I/O Space References (outside XMI Private Space)	
I-stream Read (quadword)	Longword Read
D-stream Read (quadword)	Longword Read
Write Mask (quadword)	Longword Write Mask
Miscellaneous References	
I/O space Read to Interrupt Acknowledge space	XMI IDENT (assuming that an XMI interrupt is pending and no NEXMI or IP IVINTR interrupts are pending)

**Table 2–21 (Cont.) NEXMI Transaction Generation/Response for NVAX Chip-to-XMI Operations**

NVAX Chip Operation	Resulting XMI Operation
I/O space write to IVINTR generation space	XMI IVINTR
Clear write buffer	On a read, return null data with a read data response (RDR). On a write, ACK the transaction.

**Table 2–22 NEXMI Transaction Generation/Response for XMI-to-NVAX Chip Operations**

XMI Transaction	Resulting NVAX Chip Operation
XMI memory Writes (all types except Disown Writes) from other nodes	Load responder queue, perform NDAL transaction.
XMI memory Reads (all types) from other nodes	Load responder queue, perform NDAL transaction.
XMI Writes to XMI nodespace	Write the appropriate CSR.
XMI Read to XMI nodespace	Respond with appropriate CSR data.
XMI INTR	When this processor is the destination, set the appropriate interrupt-pending bit and post interrupt request to NVAX chip.
XMI interprocessor IVINTR	When this processor is the destination, set the IP IVINTR pending bit and post IPL 16 (hex) interrupt request.
XMI write error IVINTR	Set XBER<WEI> and post a hard error interrupt.
XMI parity error detected	Set XBER<PE> and post a soft error interrupt. If inconsistent, also set XBER<IPE> and assert H ERR L signal (hard error).
XMI IDENT	Clear the appropriate interrupt-pending bit.

### 2.6.3 Invalidates

The NEXMI monitors all read and write traffic by other nodes to memory space to maintain cache coherency between the KA66A CPU module caches and main memory and to allow other XMI nodes access to memory locations owned by the KA66A CPU module. The NEXMI forwards these addresses over the NDAL to the NVAX Cbox. The Cbox "looks up" the address in the tag store and determines if the corresponding cache subblock needs to be invalidated or written back. There is no filtering mechanism for invalidates forcing the NDAL to be used for every potential invalidate.

When the NEXMI detects a memory reference by another node on the XMI, it places the address into the responder queue. This address is driven onto the NDAL, and the NEXMI requests the NVAX Cbox to do a cache lookup.

The NEXMI's responder queue is 12 entries deep. The NEXMI uses the XMI suppress line (XMI SUP L) to suppress XMI transactions to keep the responder queue from overflowing. If four or more entries in the responder queue are valid, the NEXMI asserts the suppress line. At most, one more XMI write or two XMI reads can occur once the NEXMI asserts the

suppress signal. The suppression of XMI commands allows the NEXMI and NVAX Cbox to catch up on invalidate processing and to open up queue entries for future invalidate addresses. This also ensures that four entries remain available for an outstanding hexword read transaction. When the number of valid entries drops below three, the NEXMI deasserts the suppress line.

A potential problem exists if an invalidate address is received that is in the same cache subblock as an outstanding cacheable memory read. The Cbox tag lookup will produce a cache miss since that subblock has not yet been validated. Since the XMI request that generated this invalidate request may have occurred after the KA66A's command went out on the XMI, this invalidate must be processed. The Cbox maintains an internal state that will force this cache subblock to be invalidated or written back to memory once the cache fill completes. The Cbox will process further invalidates normally while waiting for the cache fill to complete.

## 2.6.4 Writeback Queues

The NEXMI contains two two-entry writeback queues to hold cache subblocks that need to be written back from cache to memory. It is separated into queue 0 and queue 1.

The NEXMI processes the following types of requests and issues the associated XMI transactions based upon the internal priority shown in Table 2–23. These priorities apply if these transactions are pending internally in the NEXMI.

**Table 2–23 Transaction Priority Table**

Transaction Type	Priority
CSR read response	1
Writebacks	2
NVAX chip requests	3

## 2.6.5 Lockout Avoidance

The NEXMI supports a lockout avoidance function that prevents it from being denied access to an owned or interlocked location on the XMI. The NEXMI commander asserts LOCKOUT when either:

- Its Ownership Read has received at least one LOC response and this transaction's LOCKOUT assertion time has expired.
- Its I/O space access or IDENT has received at least one NO ACK response and this transaction's LOCKOUT assertion time has expired.

## 2.6.6 Interrupts and IDENTs

The XMI supports an interrupt response protocol using INTR and IDENT commands. The device interrupts are INTRs and implied vector interrupts (IVINTRs). Any I/O device is permitted to generate interrupts (INTRs) to one or more CPU nodes, as designated by a destination mask. IVINTRs are either interprocessor interrupts or write error interrupts. The NEXMI has a set of 56 interrupt-pending bits, four wide for IPL<17:14> and fourteen deep for each I/O device possible.

### 2.6.6.1 Responding to XMI Interrupts

The XMI receiver logic monitors each XMI bus cycle. If it detects an Interrupt command targeted to its node ID, it sets the interrupt-pending bit corresponding to the interrupt level (IPL 17, 16, 15, or 14) and the interrupting node's node ID. The interrupt logic then posts an interrupt request at the appropriate level. Since the CPU only has four interrupt request lines (one for each level), multiple interrupts from the XMI and system support are merged by the NEXMI. The 14 interrupt-pending bits at each level, plus the system support and the IVINTRs, are ORed together to form a set of four composite interrupt requests (one for each level). VAX 6000 Model 600 systems support ten interrupting I/O nodes (nodes 1–4 and 9–14), even though the NEXMI can support more (nodes 5–8 are reserved for memory).

### 2.6.6.2 Generating the IDENT

When the NVAX drops its IPL low enough to recognize the interrupt, it issues an interrupt acknowledge (I/O Read) to the I/O address associated with the interrupt at the given IPL. The I/O read is converted to an XMI IDENT and sent to the highest priority interrupting node. Exceptions to this are system support and interprocessor (IP IVINTR) interrupts which are handled by the NEXMI. If multiple interrupts are pending at the same level, the NEXMI gives the following priority to the read interrupt vector command:

- System support interrupts (console terminal), only if the read interrupt vector is at IPL 15 (hex)
- Interprocessor (IP IVINTR) interrupts (IPL 16 (hex) only)
- XMI interrupts
- Interval timer

Should there be more than one interrupt pending at a given IPL the NEXMI gives priority to the node with the highest ID.

Each CPU in the system monitors the XMI for IDENT transactions. When an IDENT is detected, the interrupt-pending bit at the corresponding level and node ID is cleared. This assures that several CPUs in multiprocessing systems will not attempt to service the same interrupt.

Sometime after the transmission of the IDENT, the interrupting device returns an interrupt vector to the CPU. The CPU then executes the appropriate interrupt service routine.

---

### 2.6.6.3 XMI Device Interrupt Priority

The KA66A CPU module has a fixed priority scheme for XMI devices within the same interrupt level. If more than one XMI device interrupt is outstanding at a given interrupt level, then the interrupts are serviced in node ID order (from highest to lowest node ID).

---

### 2.6.6.4 Implied Vector Interrupts (IVINTR)

The Implied Vector Interrupt (IVINTR) is a single-cycle XMI transaction used to implement VAX Interprocessor Interrupts (IP) and Write Error (WE) Interrupts. For both types of interrupt the interrupt priority and interrupt vector are implied.

The KA66A CPU module can generate and respond to IP and WE IVINTRs. WE IVINTRs are issued by I/O nodes that are unable to complete an I/O write transaction (these are "disconnected" transfers on the XMI).

---

#### 2.6.6.4.1 IVINTR Mask Generation

Since the VAX instruction set does not include a primitive for IVINTR generation, the NEXMI has defined a fixed range of I/O space addresses (XMI private space) that when written will cause the generation of an XMI IVINTR transaction. During error reporting, the NEXMI handles such a transaction as if it were a write.

**NOTE: The generation of an IVINTR instruction must be done with a byte-type macro instruction. MOV<sub>B</sub> is the recommended instruction.**

For both types of IVINTRs, the lower 16 bits of the address are used as the XMI destination mask (the destination mask is used to select which node(s) will be targeted by the IVINTR). The addresses for these IVINTR-generation registers are:

- E101 0000 to E101 FFFF for IP IVINTRs
- E102 0000 to E102 FFFF for WEIs

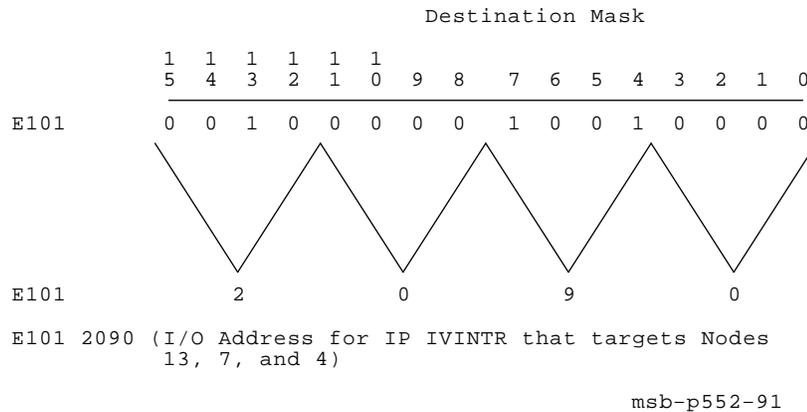
See Figure 2-21 for a diagram showing how to generate the mask.

---

#### 2.6.6.4.2 Interprocessor IVINTR (IP IVINTR) Response

The receipt of an IP IVINTR with a destination mask that has a corresponding node ID bit set, causes the NEXMI logic to set an internal "IP IVINTR pending" bit and generate an IPL 16 device interrupt to the NVAX. When the NVAX acknowledges an interrupt at IPL 16, the NEXMI checks for a pending IP IVINTR and returns a vector of 80 (hex) and resets the IP IVINTR pending bit.

Figure 2–21 Mask Generation Diagram



#### 2.6.6.4.3 Write Error IVINTR (WE IVINTR) Response

The receipt of a WE IVINTR with a destination mask that has corresponding node ID bit set causes the NEXMI logic to set the XBER<WEI> bit and generate a hard error interrupt to the NVAX. The NVAX does not issue an interrupt acknowledge for hard error interrupts but instead vectors directly to 60 (hex) in the SCB. The XBER<WEI> bit should be cleared by the hard error interrupt service routine prior to servicing the write error interrupt. Software then polls all XMI devices to determine which device sent the WE IVINTR.

## 2.6.7 XMI Registers

The NEXMI has two sets of registers, one set located in XMI nodespace and another in XMI private space. The NEXMI registers located within the KA66A's nodespace are directly accessible by all XMI nodes, while the private space registers are not.

The NEXMI can initiate an XMI transaction while another transaction is still outstanding. To identify the transaction that causes an error, the NEXMI implements separate failing registers for each writeback queue.

Addresses of XMI registers in the NEXMI are as follows:

- Device Register, XDEV (BB + 00)
- Bus Error Register, XBER (BB + 04)
- Failing Address Register, XFADR (BB + 08)
- General Purpose Register, XGPR (BB + 0C)
- Node-Specific Control and Status Register, NSCSR (BB + 1C)
- XMI Control Register, XCR (BB + 24)
- Failing Address Extension Register, XFAER (BB + 2C)
- Bus Error Extension Register, XBEER (BB + 34)

## KA66A CPU Module

- Writeback 0 Failing Address Register, WFADR0 (BB + 40)
- Writeback 1 Failing Address Register, WFADR1 (BB + 44)

The NEXMI's registers have the following characteristics:

- The mask bits are ignored on writes to the NEXMI's control and status registers. A full longword write is always performed.
- Interlocks are not supported. Interlock Read and Unlock Write Mask commands are treated like Read and Write Mask commands, respectively.
- The XMI responder queue is only one deep so the NEXMI NO ACKs subsequent CSR references until the read data for the queued CSR read has been returned.
- Write transactions directed at read-only registers will be accepted and acknowledged, but no action will be taken and the value of the register will not be affected.
- Access to the registers is strictly on a longword basis; word and byte addresses are ignored and the full longword is returned on any register read.

In addition to the required XMI registers listed here, the KA66A module also has several registers in XMI private space. See Table 2–32.

---

## 2.7 KA66A CPU Module Registers

The KA66A CPU module registers consist of internal processor registers, KA66A CPU module registers in XMI private space, and XMI required registers.

---

### 2.7.1 IPR and Cache Addressing

The processor registers in the NVAX chip, and those required of the system environment, are divided into five groups and are distinguished by particular bit patterns in the IPR address, as shown in Figure 2–22.

The five groups are:

- 1 Normal — IPRs that address individual registers in the NVAX chip or system environment.
- 2 B-cache tag IPRs — The read/write block of IPRs that allow direct access to the B-cache tags.
- 3 B-cache deallocate IPRs — The write-only block of IPRs used to deallocate a B-cache block.
- 4 P-cache tag IPRs — The read/write block of IPRs that allow direct access to the P-cache tags.
- 5 P-cache data parity IPRs — The read/write block of IPRs that allow direct access to the P-cache data parity bits.



The numeric range for each of the five groups is shown in Table 2–24.

**Table 2–24 IPR Address Space Decoding**

IPR Group	Mnemonic	IPR Address Range <sup>1</sup> (hex)	Contents
Normal	None	00000000–000000FF	256 individual IPRs
B-Cache Tag	BCTAG	01000000–011FFFE0	64K B-cache tag IPRs, each separated by 20 (hex)
B-Cache Deallocate	BCFLUSH	01400000–015FFFE0	64K B-cache tag deallocate IPRs, each separated by 20 (hex)
P-Cache Tag	PCTAG	01800000–01801FE0	256 P-cache tag IPRs, 128 for each P-cache set, each separated by 20 (hex)
P-Cache Data Parity	PCDAP	01C00000–01C01FF8	1024 P-cache data parity IPRs, 512 for each P-cache set, each separated by 8 (hex)

<sup>1</sup>Unused fields in the IPR addresses for these groups should be zero. However, if these bits are not zero, neither hardware nor microcode detects and faults on the address. Although noncontiguous address ranges are shown for these groups, the entire IPR address space maps into one of these groups. If these fields are non-zero, the operation of the CPU is UNDEFINED.

**NOTE: The address ranges shown in Table 2–24 are those used by the programmer. When processing normal IPRs, the microcode shifts the IPR number left by two bits for use as an IPR command address. This positions the IPR number to bits <9:2> and modifies the address range as seen by the hardware to 0–3FC, with bits <1:0> = 00. No shifting is performed for the other groups of IPR addresses.**

Because of the sparse addressing used for IPRs in groups other than the normal group, valid IPR addresses are not separated by one. Rather, valid IPR addresses are separated by either 8 or 20 (hex). For example, the IPR address for B-cache tag 0 is 0100 0000 (hex), and the IPR address for B-cache tag 1 is 0100 0020 (hex). In this group, bits <4:0> of the IPR address are ignored, so IPR numbers 0100 0001 through 0100 001F all address B-cache tag 0. Similarly, the IPR address for the first subblock of P-cache data parity is 01C0 0000 (hex), and the IPR address for the second subblock of P-cache data parity is 01C0 0008 (hex).

Processor registers in all groups except the normal group are processed entirely by the NVAX chip and never appear on the NDAL. This is also true for a number of the IPRs in the normal group. IPRs in the normal group that are not processed by the NVAX chip are converted into I/O space references and passed to the system environment by a read or write command on the NDAL.

Each of the IPRs in the normal group is of longword length, so a 1-Kbyte block of I/O space is required to convert each possible IPR to a unique I/O space longword. This block starts at address E100 0000 (hex). Conversion of an IPR address to an I/O space address in this block is done by shifting the IPR address left into bits <9:2>, filling bits <1:0> with zeros, and

merging in the base address of the block. This can be expressed by the equation

$$IO\ ADDRESS = E1000000 + (IPR\ NUMBER * 4)$$

The actual hardware implementation of this is different in that the IPR number is shifted left by 2 bits, and bits <31:30,24> are set. No multiply or add is done as one might conclude from the equation.

Because many of the IPRs in the normal group are processed entirely by the NVAX chip, the corresponding I/O space location in the 1-Kbyte block should not be referenced. A programmer can reference these locations by an explicit I/O space reference using a MOVL instruction. However, referencing these registers with instructions other than MTPR/MFPR instructions can result in UNDEFINED behavior.

**NOTE: Many of the internal processor registers are used internally by the microcode during normal operation of the CPU and are not intended to be referenced by software except during test or diagnosis of the system.**

Processor registers not implemented in the NVAX chip are converted to I/O space reads or writes. The I/O space registers that are implemented by the system environment on the KA66A CPU module are shown in Table 2–25.

**Table 2–25 I/O Space Registers**

I/O Space Address (Hex)	Type	Definition
E100 0100	RO	Interrupt acknowledge for IPL 14 (hex) interrupt requested on IRQ L<0> (BR4)
E100 0104	RO	Interrupt acknowledge for an IPL 15 (hex) interrupt requested on IRQ L<1> (BR5)
E100 0108	RO	Interrupt acknowledge for an IPL 16 (hex) interrupt requested on IRQ L<2> (BR6)
E100 010C	RO	Interrupt acknowledge for an IPL 17 (hex) interrupt requested on IRQ L<3> (BR7)
E100 0110	R/W	Location that invokes a write buffer flush in the NVAX Cbox. When this location is read, the CPU is waiting for confirmation that the flush has completed. The NEXMI responds to a Cbox write buffer read on the NDAL by returning an RDR with null data. A Cbox write buffer write is ACKed.

## 2.7.2 Internal Processor Registers

The processor state is stored in internal processor registers rather than in memory. See Table 2–26 and Table 2–27. The processor state is composed of 16 general purpose registers (GPRs), the processor status longword (PSL), and internal processor registers (IPRs).

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

**Table 2–26 KA66A CPU Module Internal Processor Registers**

Address Decimal (Hex)	Register	Mnemonic	Type <sup>1</sup>	Class <sup>2</sup>	I/O Address
0 (0)	Kernel Stack Pointer	KSP	R/W	1	
1 (1)	Executive Stack Pointer	ESP	R/W	1	
2 (2)	Supervisor Stack Pointer	SSP	R/W	1	
3 (3)	User Stack Pointer	USP	R/W	1	
4 (4)	Interrupt Stack Pointer	ISP	R/W	1	
8 (8)	P0 Base	P0BR	R/W	1	
9 (9)	P0 Length	P0LR	R/W	1	
10 (A)	P1 Base	P1BR	R/W	1	
11 (B)	P1 Length	P1LR	R/W	1	
12 (C)	System Base	SBR	R/W	1	
13 (D)	System Length	SLR	R/W	1	
14 (E)	CPU Identification	CPUID	R/W	2 Init	
16 (10)	Process Control Block Base	PCBB	R/W	1	
17 (11)	System Control Block Base	SCBB	R/W	1	
18 (12)	Interrupt Priority Level	IPL	R/W	1 Init	
19 (13)	AST Level	ASTLVL	R/W	1 Init	
20 (14)	Software Interrupt Request	SIRR	WO	1	

<sup>1</sup>See Table 2–27.

<sup>2</sup>Key to Classes:

1 = Implemented by the KA66A CPU module as specified in the *VAX Architecture Reference Manual*.

2 = Implemented uniquely by the KA66A CPU module.

3 = Accessible, but not fully implemented; accesses when the system is in console mode are appropriate, accesses when the system is in user mode yield UNPREDICTABLE results.

*n* Init = The register is initialized on a KA66A CPU module reset (power-up, system reset, and node reset).

**NOTE:** Per-process registers, loaded by LDPCTX (load process context instruction), are the following IPRs (in decimal): 0, 1, 2, 3, 8, 9, 10, 11, 19, and 61. The remainder of the registers are not affected by LDPCTX.

## KA66A CPU Module

**Table 2–26 (Cont.) KA66A CPU Module Internal Processor Registers**

Address Decimal (Hex)	Register	Mnemonic	Type <sup>1</sup>	Class <sup>2</sup>	I/O Address
21 (15)	Software Interrupt Summary	SISR	R/W	1 Init	
24 (18)	Interval Clock Control and Status <sup>3</sup>	ICCS	R/W	1 Init	E100 0060
25 (19)	Next Interval Count <sup>3</sup>	NICR	WO	2	E100 0064
26 (1A)	Interval Count <sup>3</sup>	ICR	RO	2	E100 0068
27 (1B)	Time-of-Day <sup>4</sup>	TODR	R/W	1	E100 006C
28 (1C)	Console Storage Receiver Status	CSRS	R/W	3 Init	E100 0070
29 (1D)	Console Storage Receiver Data	CSRD	RO	3 Init	E100 0074
30 (1E)	Console Storage Transmitter Status	CSTS	R/W	3 Init	E100 0078
31 (1F)	Console Storage Transmitter Data	CSTD	WO	3 Init	E100 007C
32 (20)	Console Receiver Control and Status	RXCS	R/W	2 Init	E100 0080
33 (21)	Console Receiver Data Buffer	RXDB	RO	2 Init	E100 0084
34 (22)	Console Transmitter Control and Status	TXCS	R/W	2 Init	E100 0088
35 (23)	Console Transmitter Data Buffer	TXDB	WO	2 Init	E100 008C
38 (26)	Machine Check Error Summary	MCESR	WO	2	
42 (2A)	Console Saved Program Counter	SAVPC	RO	2	
43 (2B)	Console Saved Processor Status Longword	SAVPSL	RO	2	
55 (37)	I/O Reset	IORESET	WO	2	E100 00DC
56 (38)	Memory Management Enable	MAPEN	R/W	1 Init	
57 (39)	Translation Buffer Invalidate All	TBIA	WO	1	
58 (3A)	Translation Buffer Invalidate Single	TBIS	WO	1	
62 (3E)	System Identification	SID	RO	2	
63 (3F)	Translation Buffer Check	TBCHK	WO	1	
64 (40)	IPL 14 Interrupt ACK	IAK14	RO	1	E100 0100
65 (41)	IPL 15 Interrupt ACK	IAK15	RO	1	E100 0104
66 (42)	IPL 16 Interrupt ACK	IAK16	RO	1	E100 0108
67 (43)	IPL 17 Interrupt ACK	IAK17	RO	1	E100 010C
68 (44)	Clear Write Buffer	CWB	R/W	1	E100 0110
122 (7A)	Interrupt System Status	INTSYS	R/W	2	

<sup>1</sup>See Table 2–27.

<sup>2</sup>Key to Classes:

1 = Implemented by the KA66A CPU module as specified in the *VAX Architecture Reference Manual*.

2 = Implemented uniquely by the KA66A CPU module.

3 = Accessible, but not fully implemented; accesses when the system is in console mode are appropriate, accesses when the system is in user mode yield UNPREDICTABLE results.

*n* Init = The register is initialized on a KA66A CPU module reset (power-up, system reset, and node reset).

<sup>3</sup>Interval timer requests are posted at IPL 16 with a vector of C0 (hex). The interval timer is the lowest priority device at the IPL. A subset of ICCS is implemented in the NVAX chip. NICR and ICR can be used, depending on the settings in the Ebox Control Register.

<sup>4</sup>TODR is maintained during power failure by the XMI TOY BBU PWR line on the XMI backplane.

Table 2–26 (Cont.) KA66A CPU Module Internal Processor Registers

Address Decimal (Hex)	Register	Mnemonic	Type <sup>1</sup>	Class <sup>2</sup>	I/O Address
124 (7C)	Patchable Control Store Control	PCSCR	R/W	2	
125 (7D)	Ebox Control Register	ECR	R/W	2	
160 (A0)	Cbox Control	CCTL	R/W	2 Init	
162 (A2)	Backup Cache Data ECC	BCDECC	WO	2 Init	
163 (A3)	Backup Cache Error Tag Status	BCETSTS	R/W	2	
164 (A4)	Backup Cache Error Tag Index	BCETIDX	RO	2	
165 (A5)	Backup Cache Error Tag	BCETAG	RO	2	
166 (A6)	Backup Cache Error Data Status	BCEDSTS	R/W	2	
167 (A7)	Backup Cache Error Data Index	BCEDIDX	RO	2	
168 (A8)	Backup Cache Error Data ECC	BCEDECC	RO	2	
171 (AB)	Cbox Error Fill Address	CEFADR	RO	2	
172 (AC)	Cbox Error Fill Status	CEFSTS	R/W	2	
174 (AE)	NDAL Error Status	NESTS	R/W	2	
176 (B0)	NDAL Error Output Address	NEOADR	RO	2	
178 (B2)	NDAL Error Output Command	NEOCMD	RO	2	
180 (B4)	NDAL Error Data High	NEDATHI	RO	2	
182 (B6)	NDAL Error Data Low	NEDATLO	RO	2	
184 (B8)	NDAL Error Input Command	NEICMD	RO	2	
208 (D0)	VIC Memory Address	VMAR	R/W	2	
209 (D1)	VIC Tag	VTAG	R/W	2	
210 (D2)	VIC Data	VDATA	R/W	2	
211 (D3)	Ibox Control and Status	ICSR	R/W	2	
212 (D4)	Ibox Branch Prediction Control	BPCR	R/W	2	
214 (D6)	Ibox Backup PC	BPC	RO	2	
215 (D7)	Ibox Backup PC with RLOG Unwind	BPCUNW	RO	2	
231 (E7)	Physical Address Mode	PAMODE	R/W	2	
232 (E8)	Memory Management Exception Address	MMEADR	RO	2	
233 (E9)	Memory Management Exception PTE Address	MMEPTE	RO	2	
234 (EA)	Memory Management Exception Status	MMESTS	RO	2	
236 (EC)	TB Parity Address	TBADR	RO	2	
237 (ED)	TB Parity Status	TBSTS	R/W	2	

<sup>1</sup>See Table 2–27.

<sup>2</sup>Key to Classes:

1 = Implemented by the KA66A CPU module as specified in the *VAX Architecture Reference Manual*.

2 = Implemented uniquely by the KA66A CPU module.

3 = Accessible, but not fully implemented; accesses when the system is in console mode are appropriate, accesses when the system is in user mode yield UNPREDICTABLE results.

*n* Init = The register is initialized on a KA66A CPU module reset (power-up, system reset, and node reset).

## KA66A CPU Module

**Table 2–26 (Cont.) KA66A CPU Module Internal Processor Registers**

Address		Register	Mnemonic	Type <sup>1</sup>	Class <sup>2</sup>	I/O Address
Decimal	(Hex)					
242	(F2)	P-Cache Parity Address	PCADR	RO	2	
244	(F4)	P-Cache Status	PCSTS	R/W	2	
248	(F8)	P-Cache Control	PCCTL	R/W	2	

<sup>1</sup>See Table 2–27.

<sup>2</sup>Key to Classes:

1 = Implemented by the KA66A CPU module as specified in the *VAX Architecture Reference Manual*.

2 = Implemented uniquely by the KA66A CPU module.

3 = Accessible, but not fully implemented; accesses when the system is in console mode are appropriate, accesses when the system is in user mode yield UNPREDICTABLE results.

*n* Init = The register is initialized on a KA66A CPU module reset (power-up, system reset, and node reset).

**Table 2–27 Types of Registers and Bits**

Type	Description
0	Initialized to logic level zero
1	Initialized to logic level one
X	Initialized to either logic level
RO	Read only
R/W	Read/write
R/W1C	Read/cleared by writing a one
WO	Write only
MBZ	Must be zero
SBZ	Should be zero

# KA66A CPU Module Internal Processor Registers

## CPU Identification Register (CPUID)

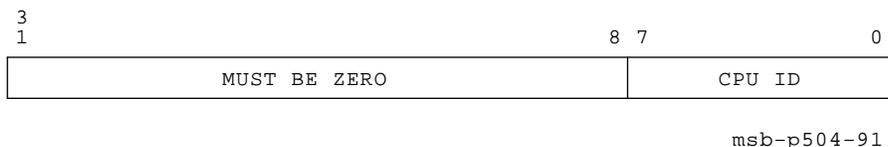
---

### CPU Identification Register (CPUID)

CPUID contains the node identification number of the KA66A CPU module. Software can determine which CPU it is operating on by reading this register. The console initializes this register.

---

**ADDRESS**      *IPR14 (NVAX chip)*



---

**bits<31:8>**

Name:      Reserved  
Mnemonic:   None  
Type:      -  
Reserved; must be zero.

---

**bits<7:0>**

Name:      CPU Identification  
Mnemonic:   CPU ID  
Type:      R/W  
During power-up the console writes this register with the node ID number.

# KA66A CPU Module Internal Processor Registers

## Interval Clock Control and Status Register (ICCS)

---

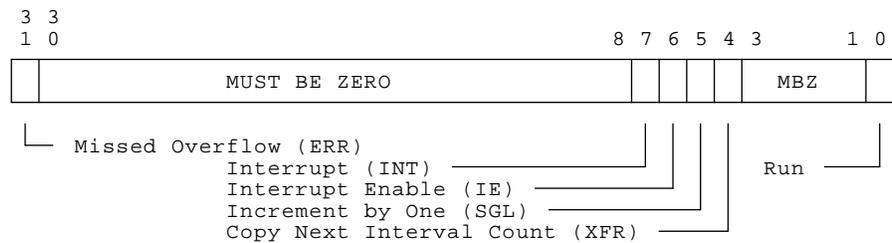
### Interval Clock Control and Status Register (ICCS)

ICCS contains control information for the interval clock. The interval clock is used for accounting, for time-dependent events, and for maintaining the software date and time. Interval timer requests are posted at IPL 16 (hex) through SCB vector C0 (hex). The interval timer is the lowest priority device at IPL 16.

---

#### ADDRESS

*IPR24 (NVAX chip)*



msb-p561-91

---

#### bit<31>

Name: Missed Overflow  
Mnemonic: ERR  
Type: R/W1C, 0

When set, ERR indicates that the Interval Count Register (ICR) overflowed while INT was set. Thus an overflow was missed.

---

#### bit<30:8>

Name: Reserved  
Mnemonic: None  
Type: -  
Reserved; must be zero.

---

#### bit<7>

Name: Interrupt  
Mnemonic: INT  
Type: R/W1C, 0

When set, INT indicates that the Interval Count Register (ICR) overflowed. If IE is also set, an interrupt is posted.

## KA66A CPU Module Internal Processor Registers

### Interval Clock Control and Status Register (ICCS)

---

#### bit<6>

Name: Interrupt Enable

Mnemonic: IE

Type: R/W, 0

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

---

#### bit<5>

Name: Increment by One

Mnemonic: SGL

Type: WO, 0

When Run is set, writing a one to SGL causes the Interval Count Register to increment by one.

---

#### bit<4>

Name: Copy Next Interval Count

Mnemonic: XFR

Type: WO, 0

When XFR is set, the contents of the Next Interval Count Register is transferred into the Interval Count Register.

---

#### bits<3:1>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

---

#### bit<0>

Name: Run

Mnemonic: –

Type: R/W, 0

When Run is set, the Interval Count Register is incremented once per microsecond. When Run is clear, the Interval Count Register does not increment automatically.







## KA66A CPU Module Internal Processor Registers Console Receiver Control and Status Register (RXCS)

---

**bits<5:0>**

Name: Reserved

Mnemonic: None

Type: -

Reserved; must be zero.



# KA66A CPU Module Internal Processor Registers

## Console Receiver Data Buffer Register (RXDB)

---

### bit<13>

Name: Framing Error

Mnemonic: FRM ERR

Type: RO, 0

FRM ERR is set if the present character did not have a valid stop bit and remains set until the register is read.

---

### bit<12>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

---

### bit<11>

Name: Received Break

Mnemonic: RCV BRK

Type: RO, 0

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

---

### bits<10:8>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

---

### bits<7:0>

Name: Received Data

Mnemonic: None

Type: RO

Received Data contains the last character received from the console.

# KA66A CPU Module Internal Processor Registers

## Console Transmitter Control and Status Register (TXCS)

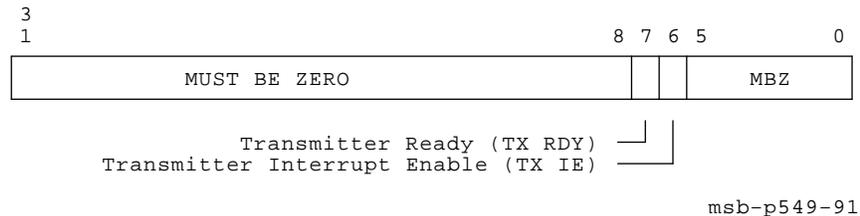
---

# Console Transmitter Control and Status Register (TXCS)

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

---

**ADDRESS**      *IPR34 (NEXMI chip) I/O Address E100 0088*



msb-p549-91

---

### bits<31:8>

Name:      Reserved  
Mnemonic: None  
Type:      -  
Reserved; must be zero.

---

### bit<7>

Name:      Transmitter Ready  
Mnemonic: TX RDY  
Type:      RO, 1

TX RDY sets when TXDB<7:0> (Transmit Data) can receive a character. It clears when TXDB<7:0> is loaded with a character, and it remains clear until the character is transferred to the serialization buffer.

---

### bit<6>

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

TX IE enables transmitter interrupts. If TX IE is set and the transmitter interrupt becomes ready (that is, TX RDY sets), then a transmitter interrupt is requested. If TX RDY is set and software sets TX IE, then a transmitter interrupt is requested. The interrupt request is cleared when it is serviced or if TX IE is cleared by software.

## KA66A CPU Module Internal Processor Registers Console Transmitter Control and Status Register (TXCS)

---

**bits<5:0>**

Name: Reserved

Mnemonic: None

Type: -

Reserved; must be zero.



**KA66A CPU Module Internal Processor Registers**  
**Machine Check Error Summary Register (MCESR)**

---

## Machine Check Error Summary Register (MCESR)

Software acknowledges the receipt of a machine check from the hardware by clearing this register.

---

**ADDRESS**

*IPR38 (NVAX chip)*

3  
1

0

Machine Check Error Summary Register (MCESR)
--

msb-p270-90

---

**bits<31:0>**

Name: Machine Check Error Summary Register

Mnemonic: MCESR

Type: WO

MCESR allows software to acknowledge receipt of a machine check. When the microcode invokes the software machine check handler, it sets a "machine check in progress" flag. If a machine check or memory management exception occurs when this flag is set, the microcode initiates a console double error halt. Machine check handler software needs to clear the "machine check in progress" flag as soon as possible by writing a zero to MCESR to reenables normal machine check and memory management exception reporting.





## KA66A CPU Module Internal Processor Registers

### Console Saved Processor Status Longword (SAVPSL)

---

#### bit<30>

Name: Trace Pending  
Mnemonic: TP  
Type: RO

Forces a trace fault when set at the beginning of any instruction. Set by the processor if T is set at the beginning of an instruction.

---

#### bit<29>

Name: Virtual Machine Mode  
Mnemonic: VM  
Type: RO

When set, the processor is executing a virtual machine, and the VMPSL register contains parts of the PSL of the virtual machine. When clear, the processor is running a real machine.

---

#### bit<28>

Name: Reserved  
Mnemonic: None  
Type: RO, 0

Reserved; must be zero.

---

#### bit<27>

Name: First Part Done  
Mnemonic: FPD  
Type: RO

When set, execution of the instruction addressed by PC cannot simply be started at the beginning but must be restarted at some other implementation-dependent point in its operation.

---

#### bit<26>

Name: Interrupt Stack  
Mnemonic: IS  
Type: RO

When set, the processor is executing on the interrupt stack. Any mechanism that sets IS also clears current mode and raises IPL above zero. When IS is clear, the processor is executing on the stack specified by current mode.

# KA66A CPU Module Internal Processor Registers

## Console Saved Processor Status Longword (SAVPSL)

---

### bits<25:24>

Name: Current Mode  
Mnemonic: CUR MOD  
Type: RO

The access mode of the currently executing process.

0—Kernel  
1—Executive  
2—Supervisor  
3—User

---

### bits<23:22>

Name: Previous Access Mode  
Mnemonic: PRV MOD  
Type: RO

Loaded from current mode by exceptions and change mode x instructions, cleared by interrupts, and restored by the REI instruction.

---

### bit<21>

Name: Reserved  
Mnemonic: None  
Type: RO, 0

Reserved; must be zero.

---

### bits<20:16>

Name: Interrupt Priority Level  
Mnemonic: IPL  
Type: RO

The current processor priority, in the range 0 to F (hex). The processor will accept interrupts only on levels greater than the current level.

---

### bit<15>

Name: Memory Management Enable<0>  
Mnemonic: MAPEN<0>  
Type: RO

At the time of a console halt, MAPEN<0> is loaded into SAVPSL<15>.

## KA66A CPU Module Internal Processor Registers

### Console Saved Processor Status Longword (SAVPSL)

---

#### bit<14>

Name: Invalid  
Mnemonic: None  
Type: RO

At the time of a console halt, a zero is loaded into the Invalid bit if the PSL is valid and a one is loaded if it is not valid. The Invalid bit is undefined after a halt due to a system reset.

---

#### bits<13:8>

Name: Halt Code  
Mnemonic: None  
Type: RO

At the time of a console halt, console halt code is loaded into SAVPSL<13:8>.

---

#### bit<7>

Name: Decimal Overflow Enable  
Mnemonic: DV  
Type: RO

When set, forces a decimal overflow trap after execution of an instruction that produced an overflowed decimal result or had a conversion error. When DV is clear, no trap occurs (however, the condition code V bit is set).

---

#### bit<6>

Name: Floating Underflow Enable  
Mnemonic: FU  
Type: RO

When set, FU forces a floating underflow exception after execution of an instruction that produced an underflowed result. (That is, FU forces an exception when a result exponent, after normalization and rounding, is less than the smallest representable exponent for the data type.) When FU is clear, no exception occurs.

---

#### bit<5>

Name: Integer Overflow Enable  
Mnemonic: IV  
Type: RO

When set, IV forces an integer overflow trap after execution of an instruction that produced an integer result that overflowed or had a conversion error. When IV is clear, no integer overflow trap occurs (however, the condition code V bit is set).

## KA66A CPU Module Internal Processor Registers

### Console Saved Processor Status Longword (SAVPSL)

---

#### bit<4>

Name: Trace  
Mnemonic: T  
Type: RO

When T is set at the beginning of an instruction, TP gets set. Most programs should treat T as UNPREDICTABLE because it is set by debuggers and trace programs for tracing and for proceeding from a breakpoint.

---

#### bit<3>

Name: Negative  
Mnemonic: N  
Type: RO

When set, indicates that the last instruction that affected N produced a result that was negative. When N is clear, the result was positive or zero.

---

#### bit<2>

Name: Zero  
Mnemonic: Z  
Type: RO

When set, indicates that the last instruction that affected Z produced a result that was zero. When Z is clear, the result was non-zero.

---

#### bit<1>

Name: Overflow  
Mnemonic: V  
Type: RO

When set, indicates that the last instruction that affected V produced a result that was too large to be properly represented in the operand that received the result if there was a conversion error. When V is clear, there was no overflow or conversion error.

---

#### bit<0>

Name: Carry  
Mnemonic: C  
Type: RO

When set, indicates that the last instruction that affected C had a carry out of the most significant bit of the result or a borrow into the most significant bit. When C is clear, there was no carry or borrow.



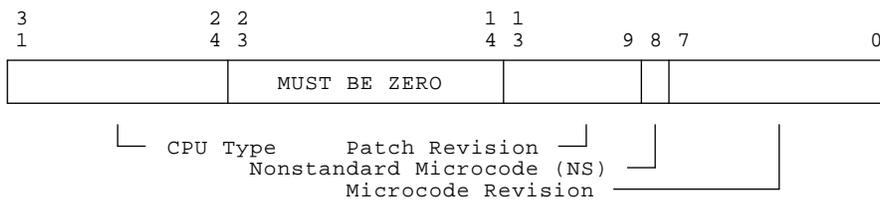
# KA66A CPU Module Internal Processor Registers

## System Identification Register (SID)

### System Identification Register (SID)

SID specifies the processor type and its microcode revision level. It can only be accessed locally. Other devices on the XMI determine the nature of a node by reading its XMI Device Register (XDEV).

**ADDRESS**      *IPR62 (NVAX chip)*



msb-p505-91

**bits<31:24>**

Name:      CPU Type  
Mnemonic: None  
Type:        RO

This field is always 13 (hex), indicating the KA66A CPU module's NVAX chip.

**bits<23:14>**

Name:      Reserved  
Mnemonic: None  
Type:        –  
Reserved; must be zero.

**bits<13:9>**

Name:      Patch Revision  
Mnemonic: None  
Type:        RO, 0

This field specifies the microcode patch revision of the NVAX chip. If the field is zero, no microcode patch is loaded. The data comes from the Patchable Control Store Control Register (PCSCR).

## KA66A CPU Module Internal Processor Registers

### System Identification Register (SID)

---

#### bit<8>

Name: Nonstandard Microcode

Mnemonic: NS

Type: RO, 0

This bit specifies whether a microcode patch revision is standard. If the bit is clear (0) and the Patch Revision field is nonzero, then the microcode patch is standard. If the bit is set (1), then a nonstandard microcode patch is loaded. The data comes from the Patchable Control Store Control Register (PCSCR).

---

#### bits<7:0>

Name: Microcode Revision

Mnemonic: None

Type: RO

This field specifies the microcode revision of the NVAX chip and is hardwired in the NVAX chip.



## KA66A CPU Module Internal Processor Registers

### Patchable Control Store Control Register (PCSCR)

---

#### bit<12>

Name: Data  
Mnemonic: None  
Type: R/W

Data to be shifted into the PCS shift scan chain. The data bit is shifted into the PCS chain whenever the RWL SHIFT bit is set. By repeatedly setting or clearing the data bit and setting the RWL SHIFT bit, any data pattern can be written into the PCS scan chain.

---

#### bit<11>

Name: Read/Write Shift  
Mnemonic: RWL SHIFT  
Type: WO

Writing a one to RWL SHIFT causes the PCS scan chain to shift by one, picking up the Data bit in the process. The control signal that enables the shift returns to the inactive state automatically; software need not clear this bit. RWL SHIFT always reads as zero.

---

#### bit<10>

Name: PCS Write  
Mnemonic: None  
Type: WO, 0

Writing a one to PCS Write causes the contents of the PCS scan chain to be written into the patchable control store. The control signal that enables the write returns to the inactive state automatically; software need not clear this bit. PCS Write always reads as zero.

---

#### bit<9>

Name: PCS Enable  
Mnemonic: PCS ENB  
Type: R/W, 0

When set, this bit enables the patchable control store outputs so that patches supersede the control store ROM.

---

#### bit<8>

Name: Parallel Port Disable  
Mnemonic: PAR PORT DIS  
Type: R/W, 0

Writing a one to PAR PORT DIS disables control by the test section of the parallel port used in loading the control store. When using this register to load the control store, disabling the parallel port is necessary.

# KA66A CPU Module Internal Processor Registers

## Patchable Control Store Control Register (PCSCR)

---

**bits<7:0>**

Name: Reserved

Mnemonic: None

Type: -

Reserved; must be zero.

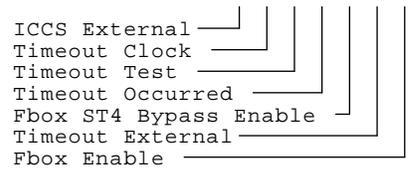
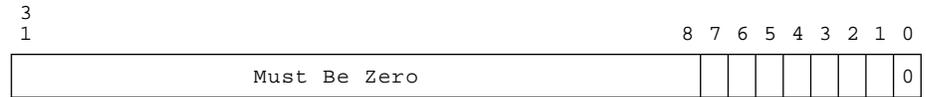
# KA66A CPU Module Internal Processor Registers

## Ebox Control Register (ECR)

### Ebox Control Register (ECR)

ECR is used to configure certain Ebox functions.

**ADDRESS**      *IPR125 (NVAX chip)*



msb-p514-91

#### bits<31:8>

Name:      Reserved  
 Mnemonic: None  
 Type:      —  
 Reserved; Must be zero.

#### bit<7>

Name:      ICCS External  
 Mnemonic: ICCS EXT  
 Type:      R/W, 0

ICCS EXT selects the interval timer. When clear, the CPU implements a subset interval timer maintained on the chip. When set, the CPU implements a full interval timer with ICCS, NICR, and ICR processor registers implemented off chip. The KA66A CPU module uses an off-chip interval timer.

# KA66A CPU Module Internal Processor Registers

## Ebox Control Register (ECR)

---

### bit<6>

Name: Timeout Clock  
Mnemonic: None  
Type: RO, 0

This is the most significant bit of the timeout base counter. It is used as an indication that the TIMEOUT ENABLE H signal is functioning. It should be clear half the time and set half the time. The period of oscillation is 65536 times the cycle time of the chip or of the external clock depending upon the condition of the ECR<Timeout External> bit. If ECR<Timeout External> is clear, this period is approximately 786 microseconds.

---

### bit<5>

Name: Timeout Test  
Mnemonic: None  
Type: RW, 0

When set, the stage 3 pipe stall circuit counts every cycle instead of only those when the TIMEOUT ENABLE H signal is asserted. In this test mode the STALL timeout time is approximately 50 microseconds instead of 3 seconds.

---

### bit<4>

Name: Timeout Occurred  
Mnemonic: None  
Type: R/W1C, 0

When set, this bit indicates that a STALL timeout occurred. The read timeout is approximately 200 ms. Writing Timeout Occurred with a one clears it.

---

### bit<3>

Name: Fbox ST4 Bypass Enable  
Mnemonic: None  
Type: R/W, 0

When set, this bit enables the Fbox stage 4 bypass.

---

### bit<2>

Name: Timeout External  
Mnemonic: None  
Type: R/W, 0

This bit is set by configuration code to select an external clock for the STALL timeout timer.

## KA66A CPU Module Internal Processor Registers

### Ebox Control Register (ECR)

**bit<1>**

---

Name: Fbox Enable

Mnemonic: None

Type: R/W, 0

When set, the Fbox is enabled.

---

**bit<0>**

Name: Reserved

Mnemonic: None

Type: -

Reserved; must be zero.



# KA66A CPU Module Internal Processor Registers

## Cbox Control Register (CCTL)

---

### bits<29:17>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

---

### bit<16>

Name: Force NDAL Parity Error  
Mnemonic: FORCE NDAL PERR  
Type: R/W, 0

When set, the Force NDAL Parity Error bit causes a single NDAL parity error.

Software uses this bit as follows: Setting the bit causes one NDAL parity error. The parity error does not occur until the NVAX is granted the NDAL for its next outgoing transaction. If software sets FORCE NDAL PERR and clears it before the NVAX is granted the bus, the NVAX will still force a parity error on the next transaction. The bit must be cleared and then set again to cause another NDAL parity error.

---

### bits<15:11>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

---

### bit<10>

Name: Disable Pack  
Mnemonic: None  
Type: R/W, 0

When Disable Pack is set, the Cbox does not pack quadword writes together. Instead, the write packer passes every write it receives directly into the write queue. When the bit is clear, the Cbox write packer operates normally. Disable Pack is for testing purposes only. Reset clears this bit.

---

### bit<9>

Name: Timeout Test  
Mnemonic: None  
Type: R/W, 0

When Timeout Test is set, the Cbox uses the internal clock to clock its read timeout counter. When Timeout Test is clear, the Cbox uses the external base clock to clock its timeout counters. Reset clears this bit.

# KA66A CPU Module Internal Processor Registers

## Cbox Control Register (CCTL)

---

### bit<8>

Name: Software ECC Enable  
Mnemonic: SW ECC  
Type: R/W, 0

When SW ECC is clear, the Cbox generates correct ECC check bits for all writes to the tag store and data RAMs. When SW ECC is set, the Cbox does not generate the check bits when the backup cache is written with data, but uses the check bit values specified by software and written in the BCDECC register. Note that if a read or write reference misses in the B-cache when SW ECC is set, all four fills will be written with the ECC given in BCDECC when they return.

When SW ECC is set and the tag store is written using an IPR write to BCTAG, the Cbox uses the check bits for the tag store as given through the IPR write. The value of SW ECC does not affect tag store transactions other than IPR writes.

Reset clears this bit.

---

### bit<7>

Name: Disable ECC Errors  
Mnemonic: None  
Type: R/W, 0

When Disable ECC Errors is set, all ECC errors from the B-cache are ignored. The B-cache data syndrome is loaded into BCEDECC on every cache access; the behavior of BCETSTS, BCETIDX, BCETAG, BCEDSTS, and BCEDIDX is UNPREDICTABLE. This feature allows operation of the B-cache even if the error detection and correction logic is faulty. It also allows access to the B-cache syndrome for the purposes of testing the ECC logic. Reset clears this bit.

---

### bit<6>

Name: Force Hit  
Mnemonic: None  
Type: R/W, 0

When Force Hit is set, all memory references, both D-stream and I-stream reads and writes, are forced to hit in the backup cache. The tag store state is not changed, but data is always read or written. Reset clears this bit.

The backup cache must be enabled when the cache is used in force hit mode. This mode is for testing purposes only.

## KA66A CPU Module Internal Processor Registers

### Cbox Control Register (CCTL)

---

#### bits<5:4>

Name: B-Cache Size  
Mnemonic: –  
Type: R/W, 0

Four backup cache sizes are possible using the NVAX chip. In the KA66A CPU module the backup cache size is 2 Mbytes and both bits are set.

---

#### bits<3:2>

Name: Reserved  
Mnemonic: None  
Type: R, 01

Reserved; default is 01.

---

#### bit<1>

Name: Reserved  
Mnemonic: None  
Type: R, 1

Reserved; default is 1.

---

#### bit<0>

Name: B-Cache Enable  
Mnemonic: None  
Type: R/W, 0

When B-Cache Enable is set, the backup cache is enabled for operation. When B-Cache Enable is clear, the B-cache is off and all references are treated as misses. When the B-cache is off, CCTL<Force Hit>, CCTL<SW ETM>, and CCTL<HW ETM> are ignored. Reset clears this bit so that the B-cache is off when the chip is reset.

# KA66A CPU Module Internal Processor Registers

## Backup Cache Data ECC Register (BCDECC)

### Backup Cache Data ECC Register (BCDECC)

BCDECC is used for testing. Software writes bad ECC into the data RAMs to test Cbox error detection logic. BCDECC will be used as the source of the ECC syndrome bits during any write to the backup cache data RAMs, including those done for fills from memory.

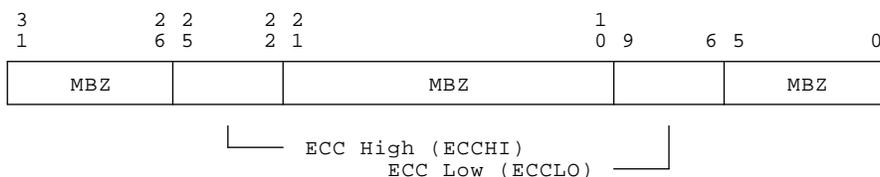
The hardware will use data in this register only if the CCTL<SW ECC> bit is set; otherwise, it will generate the ECC check bits.

Cache transactions must be carefully controlled while this register is being used. BCDECC will probably be most useful when used in force hit mode, so that no fills are generated.

Reset does not affect this register.

#### ADDRESS

*IPR162 (NVAX chip)*



msb-p559-91

#### bits<31:26>

Name: Reserved  
Mnemonic: None  
Type: -  
Reserved; must be zero.

#### bits<25:22>

Name: ECC High  
Mnemonic: ECCHI  
Type: WO

The ECCHI field corresponds to ECC syndrome bits <7:4> and to the upper longword of B-cache data.

#### bits<21:10>

Name: Reserved  
Mnemonic: None  
Type: -  
Reserved; must be zero.

## KA66A CPU Module Internal Processor Registers

### Backup Cache Data ECC Register (BCDECC)

---

#### bits<9:6>

Name: ECC Low

Mnemonic: ECCL0

Type: WO

The ECCL0 field corresponds to ECC syndrome bits <3:0> and to the lower longword of B-cache data.

---

#### bits<5:0>

Name: Reserved

Mnemonic: None

Type: -

Reserved; must be zero.

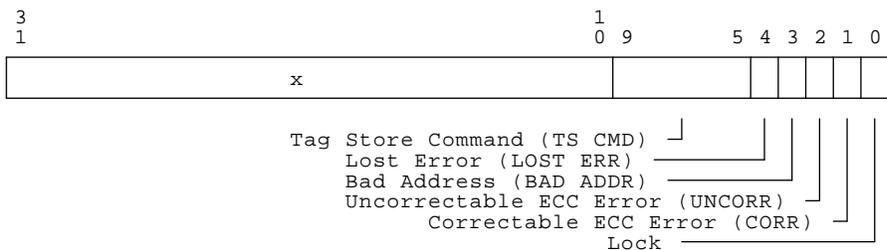
# KA66A CPU Module Internal Processor Registers

## Backup Cache Error Tag Status Register (BCETSTS)

### Backup Cache Error Tag Status Register (BCETSTS)

BCETSTS gives the status of an error in the B-cache tag store. For a given error the hardware writes the register, the software reads the register and must clear the error bits by using a write-one-to-clear write to the lowest five bits of the register.

**ADDRESS**      *IPR163 (NVAX chip)*



msb-p533-91

**bits<31:10>**

Name:      Reserved  
Mnemonic:   None  
Type:        -  
Reserved; initialized to either logic level.

**bits<9:5>**

Name:      Tag Store Command  
Mnemonic:   TS CMD  
Type:        RO

The TS CMD field indicates the operation of the tag store when the error was detected. Its values are listed in Table 2-28.

**Table 2-28 Interpretation of TS CMD**

TS CMD	NAME	Tag Store Operation
00111	DREAD	Data-stream tag lookup
00011	IREAD	Instruction-stream tag lookup
00010	OREAD	Ownership-Read tag lookup for a write or a Read Lock
01000	WUNLOCK	Ownership-Read tag lookup for a Write Unlock (lookup done only in ETM)

# KA66A CPU Module Internal Processor Registers

## Backup Cache Error Tag Status Register (BCETSTS)

Table 2–28 (Cont.) Interpretation of TS CMD

TS CMD	NAME	Tag Store Operation
01101	R_INVALID	Cache coherency tag lookup as the result of NDAL DREAD or IREAD
01001	O_INVALID	Cache coherency tag lookup as the result of NDAL OREAD or WRITE
01010	IPR_DEALLOC	Tag lookup for an explicit IPR deallocate operation

Three tag store operations do not cause any sort of errors: tag store update after a fill, IPR write of the tag store, and IPR read of the tag store. These commands will not appear in BCETSTS.

### bit<4>

Name: Lost Error  
Mnemonic: LOST ERR  
Type: R/W1C

LOST ERR indicates that after the first uncorrectable error was recorded in the tag store error registers, an additional uncorrectable error occurred for which state was not saved. LOST ERR is set by hardware and cleared by software.

### bit<3>

Name: Bad Address  
Mnemonic: BAD ADDR  
Type: R/W1C

BAD ADDR is set when the tag store ECC decoder detects an error in the address bit, indicating some problem with the address lines going to the tag RAMs. This is an uncorrectable error, and the B-cache tag store error registers are loaded and locked when it occurs.

The UNCORR bit and the BAD ADDR bit are exclusive: only one of them is set for a given error which sets the Lock bit. If the other type of error occurs later, the related bit is not set since the register is already locked.

BAD ADDR is set by hardware and cleared by software.

## KA66A CPU Module Internal Processor Registers Backup Cache Error Tag Status Register (BCETSTS)

---

### bit<2>

Name:       Uncorrectable ECC Error  
Mnemonic:   UNCORR  
Type:        R/W1C

UNCORR is set when the tag store ECC decoder detects an uncorrectable error. When this occurs, the B-cache tag store error registers are loaded and locked.

The UNCORR bit and the BAD ADDR bit are exclusive: only one of them is set for a given error which sets the Lock bit. If the other type of error occurs later, the related bit is not set since the register is already locked.

The UNCORR bit is set by hardware and cleared by software.

---

### bit<1>

Name:       Correctable ECC Error  
Mnemonic:   CORR  
Type:        R/W1C

CORR is set when the tag store ECC decoder detects a correctable error. When this occurs, the B-cache tag store error registers are loaded and locked against further correctable errors. They are not locked against an uncorrectable error.

If a correctable error is followed by an uncorrectable error, the CORR bit remains set.

The CORR bit is set by hardware and cleared by software.

---

### bit<0>

Name:       Lock  
Mnemonic:   None  
Type:        R/W1C

The Lock bit is set when an uncorrectable error is detected. Either UNCORR or BAD ADDR is also set to indicate the type of uncorrectable error. When the Lock bit is set, all three tag store error registers, the BCETSTS, BCETIDX, and BCETAG registers are also locked. Clearing the Lock bit unlocks all three registers. The Lock bit is set by hardware and cleared by software.

# KA66A CPU Module Internal Processor Registers

## Backup Cache Error Tag Index Register (BCETIDX)

---

### Backup Cache Error Tag Index Register (BCETIDX)

BCETIDX is loaded and locked when a B-cache tag store error occurs. If a correctable error is followed by an uncorrectable error, the register is loaded with information from the second, more serious error. Except for this case, once the register is locked, it is not changed until software explicitly unlocks the register. This register is written by hardware and read by software.

BCETIDX contains the complete hexword address pointed to by the tag store request that resulted in the error. Since the full address is saved, both the cache index and the cache tag of the request are known. Thus, this register shows both the index accessed and the tag of the request that was in error. Software can compare this tag with the actual tag read from the RAMs, which is saved in BCETAG.

---

#### ADDRESS

*IPR164 (NVAX chip)*



msb-p534-91

---

#### bits<31:0>

Name: Backup Cache Tag Store Address  
Mnemonic: None  
Type: RO

BCETIDX contains the hexword address pointed to by the tag store request that resulted in the error. The last five bits of the address are forced to zero.

# KA66A CPU Module Internal Processor Registers

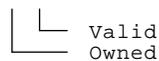
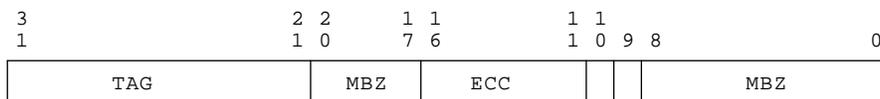
## Backup Cache Error Tag Register (BCETAG)

### Backup Cache Error Tag Register (BCETAG)

BCETAG is loaded when a tag store error occurs. It is locked when an uncorrectable error occurs on a tag store access. Once locked, it is not overwritten until unlocked by software. BCETAG is written by hardware and read by software.

The register holds the data that was read from the tag store and which produced the error.

**ADDRESS**      *IPR165 (NVAX chip)*



msb-p535-91

**bits<31:21>**

Name:      Tag  
Mnemonic:   None  
Type:        RO

The TAG field is the cache tag as read from the tag RAMs.

**bits<20:17>**

Name:      Reserved  
Mnemonic:   None  
Type:        –

Reserved; must be zero.

**bits<16:11>**

Name:      ECC  
Mnemonic:   None  
Type:        –

The ECC field contains the check bits as read from the tag RAMs during the tag access that produced the error.

## KA66A CPU Module Internal Processor Registers

### Backup Cache Error Tag Register (BCETAG)

---

#### bit<10>

Name: Owned  
Mnemonic: None  
Type: RO

The Owned bit read from the tag RAMs indicates whether the B-cache owns the block in question.

---

#### bit<9>

Name: Valid  
Mnemonic: None  
Type: RO

The Valid bit read from the tag RAMs indicates whether the block is valid in the B-cache.

---

#### bits<8:0>

Name: Reserved  
Mnemonic: None  
Type: –

Reserved; must be zero.



# KA66A CPU Module Internal Processor Registers

## Backup Cache Error Data Status Register (BCEDSTS)

Table 2–29 Interpretation of DR CMD

DR CMD <11:8>	Name	Data RAM Operation
0111	DREAD	Data lookup for a D-stream Read
0011	IREAD	Data lookup for an I-stream Read
0100	WBACK	Data lookup for a Writeback
0010	RMW	Data lookup for a Read-Modify-Write (done for normal Writes and Write Unlock)

### bits<7:5>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

### bit<4>

Name: Lost Error  
Mnemonic: LOST ERR  
Type: R/W1C

LOST ERR indicates that after the first uncorrectable error was recorded in the data error registers, an additional uncorrectable error occurred for which state was not saved. LOST ERR is set by hardware and cleared by software.

### bit<3>

Name: Bad Address  
Mnemonic: BAD ADDR  
Type: R/W1C

BAD ADDR is set when the data ECC decoder detects an error in the address bit, indicating some problem with the address lines going to the data RAMs. This is an uncorrectable error, and when it occurs, the B-cache data error registers are loaded and locked. The BAD ADDR bit is set by hardware and cleared by software.

## KA66A CPU Module Internal Processor Registers Backup Cache Error Data Status Register (BCEDSTS)

---

### bit<2>

Name:       Uncorrectable Error  
Mnemonic:   UNCORR  
Type:        R/W1C

UNCORR is set when the data ECC decoder detects an uncorrectable error. When this occurs, the B-cache data error registers are loaded and locked. The UNCORR bit is set by hardware and cleared by software.

---

### bit<1>

Name:        Correctable Data Error  
Mnemonic:   CORR  
Type:        R/W1C

CORR is set when the data ECC decoder detects a correctable error. When this occurs, the B-cache data error registers are loaded and locked. The CORR bit is set by hardware and cleared by software.

---

### bit<0>

Name:        Lock  
Mnemonic:   None  
Type:        R/W1C

The Lock bit is set when an uncorrectable error occurs. Either UNCORR or BAD ADDR is also set to indicate the type of uncorrectable error. When the Lock bit is set, the BCEDSTS, BCEDIDX, and BCEDECC registers are all locked. Clearing the Lock bit unlocks all three registers. The Lock bit is set by hardware and cleared by software.

If the CORR bit is set, the data RAM error registers are locked unless an uncorrectable error occurs.

# KA66A CPU Module Internal Processor Registers

## Backup Cache Error Data Index Register (BCEDIDX)

---

### Backup Cache Error Data Index Register (BCEDIDX)

BCEDIDX holds the index of a data RAM transaction. It is loaded when an error is detected on a data RAM access. The index loaded due to a correctable error is not overwritten unless an uncorrectable error occurs. If an uncorrectable error occurs, BCEDIDX is loaded and locked. BCEDIDX is unlocked by software; the Lock bit is in BCEDSTS.

BCEDIDX is read only from software's point of view.

---

**ADDRESS** *IPR167 (NVAX chip)*



msb-p537-92

---

**bits<31:21>**

Name: Reserved

Mnemonic: None

Type: -

Reserved; must be zero.

---

**bits<20:0>**

Name: Data RAM Index

Mnemonic: None

Type: RO

The index of the data RAM transaction. The last three bits are forced to zero.

# KA66A CPU Module Internal Processor Registers

## Backup Cache Error Data ECC Register (BCEDECC)

### Backup Cache Error Data ECC Register (BCEDECC)

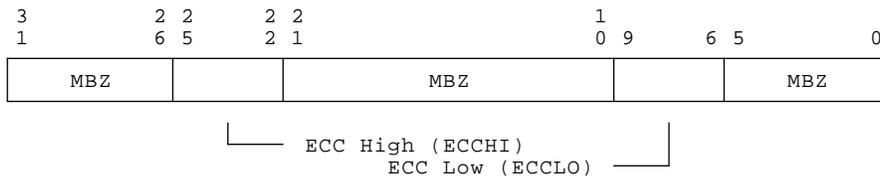
BCEDECC holds the syndrome as calculated on the backup cache data and check bits. It is loaded when an error occurs on a data RAM access. Then it follows the same lock rules that the other B-cache data error registers follow. It is unlocked by software. The Lock bit is in BCEDSTS. The contents of BCEDECC are not affected by reset.

When Disable ECC Errors is set, BCEDECC is loaded on every quadword read from the cache. This provides a way of testing the ECC logic by reading the results of the syndrome calculation. Software can use BCEDECC to write known check bits to the data RAM. When the RAMs are read, the syndrome is captured by BCEDECC. Once the syndrome is known, the check bits that were calculated by the ECC logic can be deduced, because the check bits read from the RAMs are known. The syndrome is the XOR of the calculated bits and those read from RAM.

BCEDECC is read only from software's point of view.

#### ADDRESS

*IPR168 (NVAX chip)*



msb-p574-91

#### bits<31:26>

Name: Reserved  
Mnemonic: None  
Type: -  
Reserved; must be zero.

#### bits<25:22>

Name: ECC High  
Mnemonic: ECCHI  
Type: RO

The ECCHI field corresponds to syndrome bits <7:4> and to the high order longword.

## KA66A CPU Module Internal Processor Registers

### Backup Cache Error Data ECC Register (BCEDECC)

---

#### bits<21:10>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

---

#### bits<9:6>

Name: ECC Low  
Mnemonic: ECCL0  
Type: RO  
The ECCL0 field corresponds to syndrome bits <3:0> and to the low order longword.

---

#### bits<5:0>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

KA66A CPU Module Internal Processor Registers  
Cbox Error Fill Address Register (CEFADR)

---

## Cbox Error Fill Address Register (CEFADR)

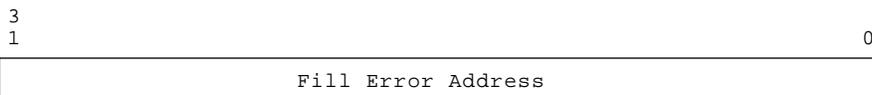
CEFADR contains the address of a read operation that resulted in an error. It is loaded when an error is detected on a fill.

CEFADR is locked when CEFSTS is locked.

---

### ADDRESS

*IPR171 (NVAX chip)*



msb-p539-91

---

### bits<31:0>

Name: Fill Error Address

Mnemonic: None

Type: RO

Contains the address of a fill read operation that resulted in an error. The last three bits of the address are forced to zero.



# KA66A CPU Module Internal Processor Registers

## Cbox Error Fill Status Register (CEFSTS)

---

### bits<20:17>

Name: Reserved

Mnemonic: None

Type: –

Reserved; initialized to either logic level.

---

### bits<16:15>

Name: Count

Mnemonic: None

Type: RO

These two bits indicate how many of the expected four quadwords have been returned successfully from memory for this read. The bits are a binary count of the quadwords returned. If the entry was for a quadword read, the count bits are set to 11 (binary) when the reference is sent out.

---

### bit<14>

Name: Requested Fill Done

Mnemonic: REQ FILL DONE

Type: RO

REQ FILL DONE is set when the requested quadword of data was successfully received from the NDAL. By examining REQ FILL DONE software can determine whether the error occurred before or after the fill completed.

---

### bit<13>

Name: Read Lock Fill Done

Mnemonic: RDLK FILL DONE

Type: RO

RDLK FILL DONE is set when a Read Lock hits in the B-cache or the last fill arrives for a Read Lock. Once RDLK FILL DONE is set, the corresponding Write Unlock is allowed to proceed overriding the lock on the address.

---

### bit<12>

Name: Do Not Fill

Mnemonic: DNF

Type: RO

DNF is set when data for a read is not to be written into the B-cache. This is the case when the cache is off, in ETM, or when the read is to I/O space. The assertion of this bit prevents the block from being validated in the cache.

# KA66A CPU Module Internal Processor Registers

## Cbox Error Fill Status Register (CEFSTS)

---

### bit<11>

Name: OREAD Invalidate Pending  
Mnemonic: OIP  
Type: RO

OIP is set when a cache coherency transaction due to an OREAD or a Write on the NDAL is requested for a block that has OREAD fills outstanding at the time. This triggers a writeback and invalidate of the block when the fill data arrives.

---

### bit<10>

Name: Read Invalidate Pending  
Mnemonic: RIP  
Type: RO

RIP is set when a cache coherency transaction due to a Read on the NDAL is requested for a block that has OREAD fills outstanding at the time. This triggers a writeback of the block when the fill data arrives; a valid copy of the data is kept in the cache.

---

### bit<9>

Name: Data Sent to Mbox  
Mnemonic: TO MBOX  
Type: RO

TO MBOX indicates that data returning for the Read was to be sent to the Mbox.

---

### bit<8>

Name: Read Done for a Write  
Mnemonic: WRITE  
Type: RO

WRITE indicates that the transaction in error was an OREAD done because of a write request.

---

### bit<7>

Name: Ownership Read  
Mnemonic: OREAD  
Type: RO

OREAD indicates that the transaction in error was an OREAD. The OREAD may have been done for a Write, a Read Lock, or a Read Modify.

# KA66A CPU Module Internal Processor Registers

## Cbox Error Fill Status Register (CEFSTS)

---

### bit<6>

Name: I-stream Read

Mnemonic: IREAD

Type: RO

IREAD indicates that the transaction in error was an IREAD.

---

### bit<5>

Name: NDAL ID

Mnemonic: ID0

Type: RO

ID0 indicates which one of two entries was used to save information about the failed transaction. Corresponds to XMI ID0 and NDAL ID0.

---

### bit<4>

Name: Lost Error

Mnemonic: LOST ERR

Type: R/W1C, 0

LOST ERR is set when CEFSTS is already locked and another RDE or timeout error occurs. This indicates to software that multiple errors have happened and state has not been saved for every error.

---

### bit<3>

Name: Read Data Error

Mnemonic: RDE

Type: R/W1C, 0

RDE is set when a memory read transaction terminates in RDE. When the RDE bit is set, the Lock bit is also set.

---

### bit<2>

Name: Timeout

Mnemonic: None

Type: R/W1C, 0

Timeout is set when a memory read transaction times out. When Timeout is set, the Lock bit is also set.

## KA66A CPU Module Internal Processor Registers

### Cbox Error Fill Status Register (CEFSTS)

---

#### bit<1>

Name: Lock  
Mnemonic: None  
Type: R/W1C, 0

The Lock bit is set when a read transaction sent to memory terminates in a Read Data Error or in a timeout. At the same time, all information about the read is loaded into the CEFSTS register. When the Lock bit is set, either Timeout or RDE is also set. Once the Lock bit is set, only the LOST ERR bit can be modified in either CEFSTS or CEFADR until the Lock bit is cleared.

---

#### bit<0>

Name: Read Lock  
Mnemonic: RDLK  
Type: R/W1C, 0

RDLK is set to show that a Read Lock is in progress. When performing a write-one-to-clear to this bit, the Valid bit for an entry that had its RDLK bit set is also cleared. The same action is taken when a Write Unlock is received.

This bit is normally not read as a one by software, because the microcode ensures that the Read Lock–Write Unlock sequence is an indivisible operation. If, however, the first quadword of a Read Lock is returned successfully and then the transaction either times out or is terminated in RDE, CEFSTS is loaded with the RDLK bit set.

# KA66A CPU Module Internal Processor Registers

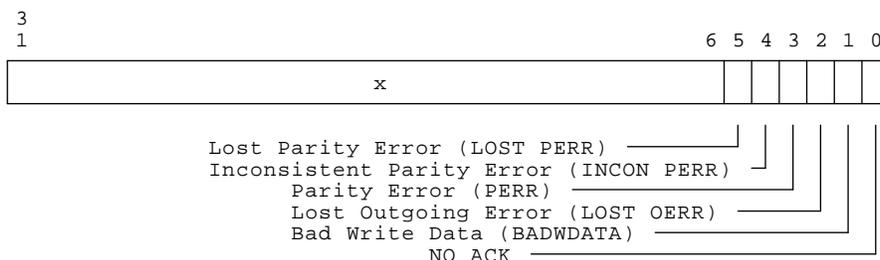
## NDAL Error Status Register (NESTS)

### NDAL Error Status Register (NESTS)

NESTS holds information about errors on the NDAL. All six bits in this register are write-one-to-clear. Reset does not affect this register. Power-up does not initialize the register.

#### ADDRESS

*IPR174 (NVAX chip)*



msb-p540-91

#### bits<31:6>

Name: Reserved

Mnemonic: None

Type: -

Reserved; initialized to either logic level.

#### bit<5>

Name: Lost Parity Error

Mnemonic: LOST PERR

Type: R/W1C

LOST PERR is set when PERR is already set and another NVAX transfer fails the parity check. LOST PERR notifies software that multiple NVAX transfers have failed the parity check; state is saved for the first.

#### bit<4>

Name: Inconsistent Parity Error

Mnemonic: INCON PERR

Type: R/W1C

INCON PERR is set when an NDAL parity error is detected on a cycle which is also acknowledged. This means that the NVAX detected a parity error but some other device acknowledged the transfer. INCON PERR is set in conjunction with PERR.

## KA66A CPU Module Internal Processor Registers

### NDAL Error Status Register (NESTS)

---

#### bit<3>

Name: Parity Error  
Mnemonic: PERR  
Type: R/W1C

PERR is set when NVAX detects a parity error on the NDAL. When PERR is set, NEDATHI, NEDATLO, and NEICMD are locked so that software can determine what was on the NDAL when the error occurred.

Since the NVAX calculates parity on every cycle, PERR will be set on both outgoing and incoming transfers that fail the parity check.

---

#### bit<2>

Name: Lost Outgoing Error  
Mnemonic: LOST OERR  
Type: R/W1C

LOST OERR is set when NO ACK or BADWDATA are already set and another one of those errors occurs. Only the state of the first outgoing error is saved. Subsequent error state is lost.

---

#### bit<1>

Name: Bad Write Data  
Mnemonic: BADWDATA  
Type: R/W1C

BADWDATA is set when a writeback from the cache had an uncorrectable ECC error. Therefore, the data is issued on the NDAL with the BADWDATA command. When BADWDATA is set, both NEOADR and NEOCMD are locked so that software can determine both the address and command associated with the failure.

BADWDATA is not set if there was a previous NO ACK. If a BADWDATA cycle is NO ACKed, both BADWDATA and NO ACK are set.

## KA66A CPU Module Internal Processor Registers NDAL Error Status Register (NESTS)

---

**bit<0>**

Name: NO ACK

Mnemonic: None

Type: R/W1C

NO ACK is set when NVAX detects that ACK was not asserted on the NDAL for an outgoing NVAX cycle. When NO ACK is set, NEOADR and NEOCMD are locked so that software can determine both the address and the command of the transaction when the error occurred.

NO ACK is not set if there was a previous BADWDATA. If a BADWDATA cycle is NO ACKed, both BADWDATA and NO ACK are set.

# KA66A CPU Module Internal Processor Registers

## NDAL Error Output Address Register (NEOADR)

---

### NDAL Error Output Address Register (NEOADR)

NEOADR contains the address driven by the Cbox onto the NDAL. Unless NEOADR is locked, it is loaded for every address cycle. It is loaded during the cycle when the corresponding ACK should be asserted on the NDAL.

It is locked when the NO ACK bit in the NESTS register is set.

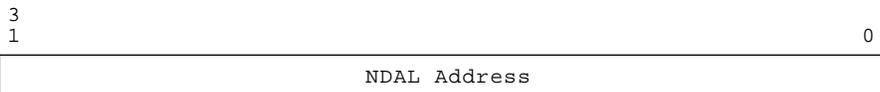
When NEOADR is locked, it contains the address information for the first transaction that failed. If it is read when it is not locked, it contains information from the last address cycle that was acknowledged on the NDAL.

The format of NEOADR matches the low longword of the NDAL during an address cycle.

NEOADR is not affected by reset.

---

**ADDRESS** *IPR176 (NVAX chip)*



msb-p541-91

---

**bits<31:0>**

Name: NDAL Address

Mnemonic: None

Type: RO

This register contains the address of a failing NDAL transaction.

# KA66A CPU Module Internal Processor Registers

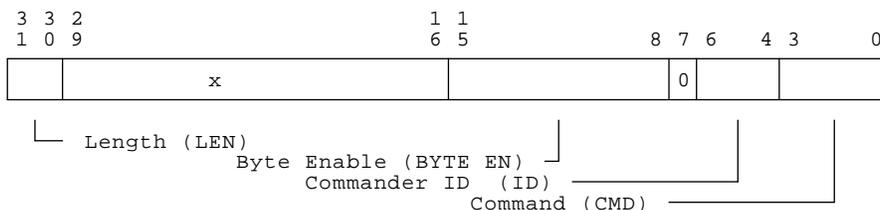
## NDAL Error Output Command Register (NEOCMD)

### NDAL Error Output Command Register (NEOCMD)

NEOCMD contains data similar to that of the high longword of the NDAL during an address cycle and is loaded on every address cycle the Cbox drives onto the NDAL, unless the NEOCMD is locked. The high quadword byte enable positions are NOT included, since NVAX only uses quadword byte-enabled transactions. The NDAL ID and command are added in the lower seven bits of the longword.

NEOCMD is not affected by reset.

**ADDRESS** *IPR178 (NVAX chip)*



msb-p542-91

#### bits<31:30>

Name: Length  
Mnemonic: LEN  
Type: RO

LEN gives the length of the NDAL transaction. Table 2-30 shows the value of LEN and the corresponding data type.

**Table 2-30 Data Length Code**

LEN	Data Type
00	Hexword
01	Unused
10	Quadword
11	Unused

#### bits<29:16>

Name: Reserved  
Mnemonic: None  
Type: -

Reserved; initialized to either logic state.

# KA66A CPU Module Internal Processor Registers

## NDAL Error Output Command Register (NEOCMD)

---

### bits<15:8>

Name: Byte Enable  
Mnemonic: BYTE EN  
Type: RO

BYTE EN is a mask field driven by the NVAX during the transaction.

---

### bit<7>

Name: Reserved  
Mnemonic: None  
Type: –

Reserved; must be zero.

---

### bits<6:4>

Name: Commander ID  
Mnemonic: ID  
Type: RO

The ID defines the node ID of the device on the NDAL and is driven by the NVAX during the transaction. The values are as follows:

---

ID	Node Name
0	NVAX Cmd 0
1	NVAX Cmd 1
2	NEXMI

---

---

### bits<3:0>

Name: Command  
Mnemonic: CMD  
Type: RO

CMD is the NDAL command driven by the NVAX during the transaction and is defined below.

---

CMD	NDAL Function
0	NOP
1	Reserved
2	WRITE (Write)
3	WDISOWN (Write-Disown; Writeback)
4	IREAD (Instruction-Stream Read)
5	DREAD (Data-Stream Read)
6	OREAD (Ownership Read)

---

## KA66A CPU Module Internal Processor Registers NDAL Error Output Command Register (NEOCMD)

<b>CMD</b>	<b>NDAL Function</b>
7	Reserved
8	Reserved
9	RDE (Read Data Error)
A	WDATA (Write Data Cycle)
B	BADWDATA (Bad Write Data)
C	RDR0 (Read Data0 Return/fill)
D	RDR1 (Read Data1 Return/fill)
E	RDR2 (Read Data2 Return/fill)
F	RDR3 (Read Data3 Return/fill)

# KA66A CPU Module Internal Processor Registers

## NDAL Error Data High Register (NEDATHI)

### NDAL Error Data High Register (NEDATHI)

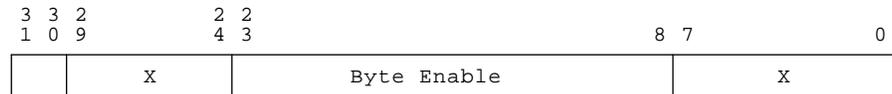
NEDATHI captures the high longword on the NDAL during an NDAL parity error.

The format of NEDATHI is dependent on the command found in NEICMD. If the command shows a data cycle, NEDATHI contains the high longword of data. If the command shows an address cycle, NEDATHI contains data.

NEDATHI is not affected by reset.

#### ADDRESS

*IPR180 (NVAX chip)*



└─ Length (LEN)

msb-p544-91

**NOTE: The format shown here is for an address command cycle. During a data cycle this register contains the high longword of data on the NDAL during a parity error.**

#### bits<31:30>

Name: Length  
Mnemonic: None  
Type: RO

Length contains a code for the length of the transaction that contains the parity error.

Length Code	Size
00	Hexword
01	Unused
10	Quadword
11	Octaword

#### bits<29:24>

Name: Reserved  
Mnemonic: None  
Type: -

Reserved; initialized to either logic level.

# KA66A CPU Module Internal Processor Registers

## NDAL Error Data High Register (NEDATHI)

---

**bits<23:8>**

Name: Byte Enable

Mnemonic: BYTE EN

Type: RO

BYTE EN is a mask field driven by the NVAX during the transaction.

---

**bits<7:0>**

Name: Reserved

Mnemonic: None

Type: –

Reserved; initialized to either logic level.

# KA66A CPU Module Internal Processor Registers

## NDAL Error Data Low Register (NEDATLO)

---

### NDAL Error Data Low Register (NEDATLO)

NEDATLO captures the low longword on the NDAL, during an NDAL parity error.

The format of NEDATLO is dependent on the command found in NEICMD. If the command shows a data cycle, NEDATLO contains the low longword of data. If the command shows an address cycle, NEDATLO contains data.

NEDATLO is not affected by reset.

---

#### ADDRESS

*IPR182 (NVAX chip)*



msb-p545-91

---

#### bits<31:0>

Name: Address  
Mnemonic: None  
Type: RO

NEDATLO contains either the address or the low data longword associated with an NDAL parity error.

# KA66A CPU Module Internal Processor Registers

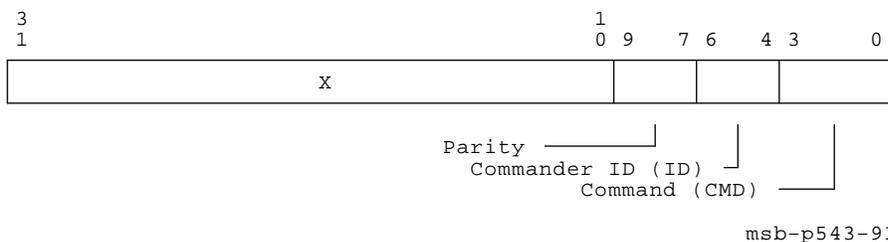
## NDAL Error Input Command Register (NEICMD)

### NDAL Error Input Command Register (NEICMD)

NEICMD, NEDATHI, and NEDATLO are loaded and locked simultaneously when a parity error occurs. At this time the PERR bit, which locks the three registers, is set in NESTS. If a second NDAL parity error occurs, the registers are not loaded. Software must clear the PERR bit before these registers are available for use again.

NEICMD is not affected by reset.

**ADDRESS**      *IPR184 (NVAX chip)*



**bits<31:10>**

Name:      Reserved  
Mnemonic: None  
Type:      -

Reserved; initialized to either logic level.

**bits<9:7>**

Name:      Parity  
Mnemonic: PAR  
Type:      RO

PAR corresponds to the NDAL lines PARITY H<2:0>.

# KA66A CPU Module Internal Processor Registers

## NDAL Error Input Command Register (NEICMD)

### bits<6:4>

Name: Commander ID  
Mnemonic: ID  
Type: RO

The ID defines the node ID of the device on the NDAL and is driven during the transaction. The values are as follows:

---

ID	Node Name
0	NVAX Cmd 0
1	NVAX Cmd 1
2	NEXMI

---

### bits<3:0>

Name: Command  
Mnemonic: CMD  
Type: RO

CMD is the NDAL command driven by the NVAX during the transaction and is defined below.

---

CMD	NDAL Function
0	NOP
1	Reserved
2	WRITE (Write)
3	WDISOWN (Write-Disown; Writeback)
4	IREAD (Instruction-Stream Read)
5	DREAD (Data-Stream Read)
6	OREAD (Ownership Read)
7	Reserved
8	Reserved
9	RDE (Read Data Error)
A	WDATA (Write Data Cycle)
B	BADWDATA (Bad Write Data)
C	RDR0 (Read Data0 Return/fill)
D	RDR1 (Read Data1 Return/fill)
E	RDR2 (Read Data2 Return/fill)
F	RDR3 (Read Data3 Return/fill)

---

# KA66A CPU Module Internal Processor Registers

## VIC Memory Address Register (VMAR)

### VIC Memory Address Register (VMAR)

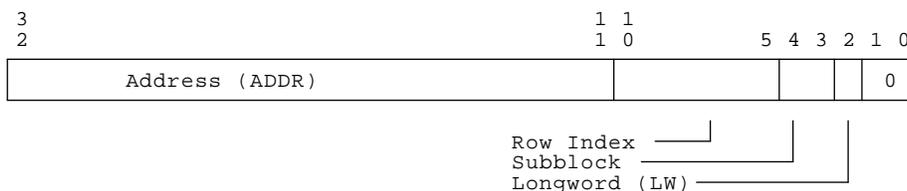
VMAR supplies cache row index, cache subblock, and quadword pointer for accessing the virtual instruction cache (VIC). When the VIC is disabled, this register is used as an index for direct IPR access to the cache arrays.

On VIC parity errors, the VMAR latches and holds VIBA <31:3>.

**Macrocode Restriction:**

**ICSR<VIC Enable> must be cleared before writing to the VIC IPRs: VMAR, VDATA, or VTAG. ICSR<VIC Enable> must be cleared before reading from VIC IPRs: VDATA, VTAG. In functional operation, an REI must precede the MTPR that enables the VIC.**

**ADDRESS**      *IPR208 (NVAX chip)*



msb-p509-92

**bits<31:11>**

Name:      Address  
Mnemonic:   ADDR  
Type:        RO

ADDR latches the tag portion of the virtual instruction buffer address (VIBA) on VIC parity errors.

**bits<10:5>**

Name:      Row Index  
Mnemonic:   None  
Type:        R/W

During read and write access to the VIC, this field is used to select a cache row. On a VIC parity error, these bits latch VIBA <10:5>.

## KA66A CPU Module Internal Processor Registers

### VIC Memory Address Register (VMAR)

---

#### bits<4:3>

Name: Subblock  
Mnemonic: None  
Type: R/W

During read and write access to the VIC, this field is used to select the subblock of data being accessed. On VIC parity errors these bits latch VIBA <4:3>.

---

#### bit<2>

Name: Longword  
Mnemonic: LW  
Type: WO

LW indicates which longword in the subblock to access in the cache array. 1 = upper LW, 0 = lower LW.

---

#### bits<1:0>

Name: Reserved  
Mnemonic: None  
Type: –

Reserved; must be zero.

## VIC Tag Register (VTAG)

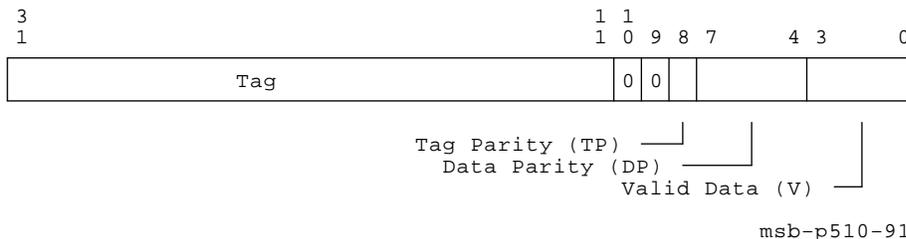
VTAG provides read and write access to the VIC tag array. An IPR write to VTAG writes the tag, parity, and valid bits for the row indexed by VMAR <10:5>. VTAG <31:11> are written to the cache tag. VTAG <8> is written to the associated tag parity bit. VTAG <7:4> are used to write the four data parity bits associated with the indexed cache row. Similarly, VTAG <3:0> writes the four data valid bits associated with the cache row.

Bits <7:4> and <3:0> are the Data Parity and Valid Data bits, respectively, for the four quadwords of data in the same row.

**Macrocode Restriction:**

**ICSR<VIC Enable> must be cleared before writing to the VIC IPRs: VMAR, VDATA, or VTAG. ICSR<VIC Enable> must be cleared before reading from VIC IPRs: VDATA, VTAG. In functional operation, an REI must precede the MTPR that enables the VIC.**

**ADDRESS**      *IPR209 (NVAX chip)*



**bits<31:11>**

Name:      Tag  
Mnemonic: None  
Type:      R/W

Supplies the cache tag on tag array read/writes.

**bits<10:9>**

Name:      Reserved  
Mnemonic: None  
Type:      -

Reserved; must be zero.

## KA66A CPU Module Internal Processor Registers

### VIC Tag Register (VTAG)

---

#### bit<8>

Name: Tag Parity  
Mnemonic: TP  
Type: R/W

TP supplies parity on the tag of tag array read/writes.

---

#### bits<7:4>

Name: Data Parity  
Mnemonic: DP  
Type: R/W

DP supplies parity on the four corresponding quadwords of data in the same row. DP <0> corresponds to the quadword of data addressed when address bits <4:3> = 00, DP <1> corresponds to the quadword of data addressed when address bits <4:3> = 01, and so forth.

---

#### bits<3:0>

Name: Valid Data  
Mnemonic: V  
Type: R/W

V supplies valid bits on the four corresponding quadwords of data in the same row. V <0> corresponds to the quadword of data addressed when address bits <4:3> = 00, V <1> corresponds to the quadword of data addressed when address bits <4:3> = 01, and so forth.

## VIC Data Register (VDATA)

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

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

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

**Macrocode Restriction:**

**ICSR<VIC Enable> must be cleared before writing to the VIC IPRs: VMAR, VDATA, or VTAG. ICSR<VIC Enable> must be cleared before reading from VIC IPRs: VDATA, VTAG. In functional operation, an REI must precede the MTPR that enables the VIC.**

**ADDRESS**

*IPR210 (NVAX chip)*



msb-p511-91

**bits<31:0>**

Name: Data  
Mnemonic: None  
Type: R/W

Data for data array reads and writes.

# KA66A CPU Module Internal Processor Registers

## Ibox Control and Status Register (ICSR)

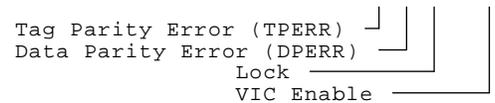
### Ibox Control and Status Register (ICSR)

ICSR provides control and status functions for the Ibox. VIC tag and data parity errors are latched in bits <4> and <3> respectively. Bit <2> is set when a tag or data parity error occurs and keeps the error status bits and the VMAR register from being modified further. Writing a one to ICSR <2> clears the Lock bit and allows the error status to be updated. Bit <0> provides IPR control of the VIC enable. It is cleared on reset.

When ICSR<2> is clear, the values in ICSR<4:3> are meaningless. When ICSR<2> is set, a VIC parity error has occurred, and either ICSR<4> or ICSR<3> will be set indicating a tag parity error or a data parity error. Bits <4:3> cannot be cleared by software.

#### ADDRESS

*IPR211 (NVAX chip)*



msb-p512-91

#### bits<31:5>

Name: Reserved  
 Mnemonic: None  
 Type: —  
 Reserved; must be zero.

#### bit<4>

Name: Tag Parity Error  
 Mnemonic: TPERR  
 Type: RO

When TPERR and Lock are set, a tag parity error occurred in the tag array.

## KA66A CPU Module Internal Processor Registers

### Ibox Control and Status Register (ICSR)

---

#### bit<3>

Name: Data Parity Error

Mnemonic: DPERR

Type: RO

When DPERR and Lock are set, a data parity error occurred in the data array.

---

#### bit<2>

Name: Lock

Mnemonic: None

Type: R/W1C

When Lock is set, the error status bits in ICSR and the error address in the VMAR are valid and cannot be modified. When the Lock bit is clear, no VIC parity error has been recorded and ICSR and VMAR can be written.

---

#### bit<1>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

---

#### bit<0>

Name: VIC Enable

Mnemonic: None

Type: R/W, 0

When VIC Enable is set, the virtual instruction cache can be accessed. Reset clears this bit.

# KA66A CPU Module Internal Processor Registers

## Physical Address Mode Register (PAMODE)

---

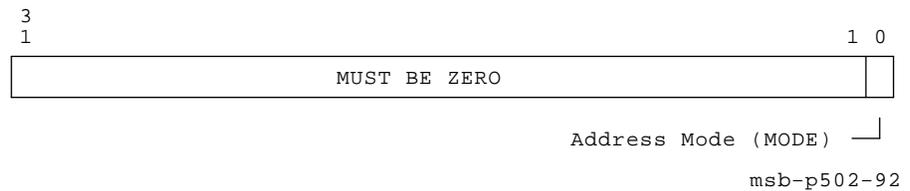
### Physical Address Mode Register (PAMODE)

PAMODE controls whether the system is in 30-bit or 32-bit physical address mode. During power-up, microcode configures the CPU to generate 30-bit physical addresses. The operating system can reconfigure the CPU to generate 32-bit physical addresses by writing to the MODE bit in the PAMODE register.

---

#### ADDRESS

*IPR231 (NVAX chip)*



---

#### bits<31:1>

Name: Reserved  
Mnemonic: None  
Type: -  
Reserved; must be zero.

---

#### bit<0>

Name: Address Mode  
Mnemonic: MODE  
Type: R/W, 0

When MODE is clear, addresses are mapped from 30-bit physical address space. When MODE is set, addresses are mapped from 32-bit physical address space.





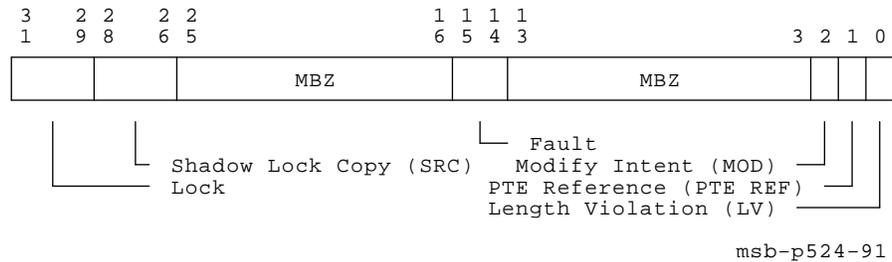
# KA66A CPU Module Internal Processor Registers

## Memory Management Exception Status Register (MMESTS)

### Memory Management Exception Status Register (MMESTS)

MMESTS contains information about a memory management exception fault.

**ADDRESS** *IPR234 (NVAX chip)*



**bits<31:29>**

Name: Lock  
Mnemonic: None  
Type: RO

The Lock field indicates the lock status of MMESTS and is decoded as follows:

Lock Values	Definitions
000	MMESTS, MMEADR, and MMEPTE are unlocked.
001	Valid IREAD fault stored. MMESTS, MMEADR, and MMEPTE are locked to other IREAD faults.
011	Valid Ibox specifier fault stored. Only Ebox reference fault can overwrite MMESTS, MMEADR, and MMEPTE.
111	Valid Ebox fault is stored. MMESTS, MMEADR, and MMEPTE are completely locked.

**bits<28:26>**

Name: Shadow Lock Copy  
Mnemonic: SRC  
Type: RO

The SRC field is a complemented copy of the Lock field. However, the SRC bits do not get cleared when the Lock field is cleared.

The value of SRC for the most severe MME is 000.

# KA66A CPU Module Internal Processor Registers

## Memory Management Exception Status Register (MMESTS)

---

### bits<25:16>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

---

### bits<15:14>

Name: Fault  
Mnemonic: None  
Type: RO

The Fault bits indicate the nature of a memory management exception, the priority given simultaneous faults, and are decoded as follows:

---

Fault Code	Priority	Definitions
01	Highest	ACV fault
10	Lowest	TNV fault

---

---

### bits<13:3>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

---

### bit<2>

Name: Modify Intent  
Mnemonic: MOD  
Type: RO

When set, the Modify bit indicates a corresponding reference had write or modify intent.

---

### bit<1>

Name: PTE Reference  
Mnemonic: PTE REF  
Type: RO

When set, PTE REF indicates either an access violation or translation not valid (ACV/TNV) fault occurred on a page table entry reference corresponding to MMEADR.

## KA66A CPU Module Internal Processor Registers Memory Management Exception Status Register (MMESTS)

---

**bit<0>**

Name: Length Violation

Mnemonic: LV

Type: RO, 0

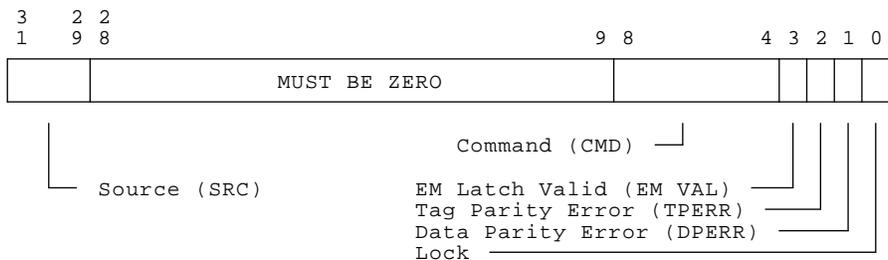
When set, LV indicates an access violation fault occurred due to a length violation.



## TB Parity Status Register (TBSTS)

TBSTS contains the error status of translation buffer parity errors and records hard error state associated with fatal errors occurring on Mbox PTE DREAD operations. These errors have nothing to do with TB parity errors, but are recorded here.

**ADDRESS**      *IPR237 (NVAX chip)*



msb-p526-91

### bits<31:29>

Name:      Source  
Mnemonic:   SRC  
Type:        RO

The SRC field indicates the original source of the reference causing the translation buffer parity error. The source field is decoded as follows:

SRC Values	Definition
110	Valid IREAD error is stored.
100	Valid lbox specifier reference error is stored.
000	Valid Ebox reference error is stored.

### bits<28:9>

Name:      Reserved  
Mnemonic:   None  
Type:        -

Reserved; must be zero.

# KA66A CPU Module Internal Processor Registers

## TB Parity Status Register (TBSTS)

bits<8:4>

Name: Command  
 Mnemonic: CMD  
 Type: RO

The CMD field indicates the command corresponding to the TB parity error.

**Table 2–31 CMD Definitions**

Name	Value (hex)	Reference Source	Description
IREAD	0E	Ibox	Aligned quadword I-stream read
DREAD	1C	Ibox, Ebox, Mbox	Variable-length D-stream read
DREAD Modify	1D	Ibox	Variable-length D-stream read with modify intent as a result of Ibox-decoded modify specifiers
DREAD Lock	1F	Ebox	Variable-length D-stream read with atomic memory lock
Write Unlock	1A	Ebox	Variable-length write with atomic memory unlock
WRITE	1B	Ebox	Variable-length write
DEST ADDR	0D	Ibox	Supplies address of a write-only destination specifier
STORE	19	Ebox	Supplies write data corresponding to a previously translated destination specifier address
IPR WR	06	Ebox	IPR Write
IPR RD	07	Ebox	IPR Read
IPR DATA	04	Mbox	Transfers Mbox IPR data to Ebox
LOAD PC	05	Ebox	Transfers a PC value to Ibox
PROBE	09	Ebox	Mbox returns ACV/TNV status of specified address to Ebox.
MME CHK	08	Ebox, Mbox	Performs ACV/TNV check on specified address and invokes the appropriate memory management exception
TB TAG FILL	0C	Ebox, Mbox	Writes a TB tag into a TB entry
TB PTE FILL	14	Ebox, Mbox	Writes PTE data into a TB entry
TBIS	10	Ebox	Invalidates a specific PTE entry in the TB
TBIA	18	Ebox, Mbox	Invalidates all entries in TB
TBIP	11	Ebox	Invalidates all PTE entries in TB corresponding to process-space translations.
D CF	03	Cbox	D-stream quadword P-cache fill

# KA66A CPU Module Internal Processor Registers

## TB Parity Status Register (TBSTS)

**Table 2–31 (Cont.) CMD Definitions**

Name	Value (hex)	Reference Source	Description
I CF	02	Cbox	I-stream quadword P-cache fill
INVAL	01	Cbox	Hexword invalidate of a P-cache entry
STOP SPEC Q	0F	Ibox	Stops processing of specifier references.
NOP	00	Ibox, Ebox, Mbox	No operation

### bit<3>

Name: EM Latch Valid  
Mnemonic: EM VAL  
Type: RO

When set, EM VAL indicates that the EM latch was valid at the time the TB parity error was detected. This information is helpful in determining whether a write operation was lost due to the TB parity error.

### bit<2>

Name: Tag Parity Error  
Mnemonic: TPERR  
Type: RO

When set, TPERR indicates that a TB tag parity error occurred.

### bit<1>

Name: Data Parity Error  
Mnemonic: DPERR  
Type: RO

When set, DPERR indicates that a TB data parity error occurred.

## KA66A CPU Module Internal Processor Registers

### TB Parity Status Register (TBSTS)

---

bit<0>

Name: Lock  
Mnemonic: None  
Type: R/W1C, 0

When set, Lock validates TBSTS contents and prevents any other field from further modification. When clear, the Lock bit indicates that no TB parity error or PTE error has been recorded and allows TBSTS and TBADR to be updated.

When a TB parity error is detected with Lock = 0, TBADR is loaded with the virtual address that caused the TB parity error, and all fields of TBSTS are updated to record the nature of the TB parity error. TPERR and DPERR can be set at the same time if these two error conditions occurred during the same cycle.

When a PTE read error is detected with Lock = 0, all fields of TBSTS are updated to record the nature of the error. Note that the contents of TBADR is invalid when TBSTS records a PTE read error. Also note that it is impossible to record a simultaneous TB parity error with a PTE read error. When an error is recorded, the Lock bit is set to validate the contents of TBSTS. When Lock is set, all bits of TBSTS are frozen and cannot be changed until the Lock bit is cleared.

Once the error handler has read these registers, it reenables TBSTS to record any new errors by clearing the Lock bit.

---

## P-Cache Parity Address Register (PCADR)

PCADR contains the physical address of a location causing a parity error.

---

### ADDRESS

*IPR242 (NVAX chip)*



msb-p527-91

---

### bits<31:0>

Name: Physical Address

Mnemonic: None

Type: -

PCADR is latched with the quadword physical address of the location that caused a parity error. Bits <2:0> are forced to zero.

# KA66A CPU Module Internal Processor Registers

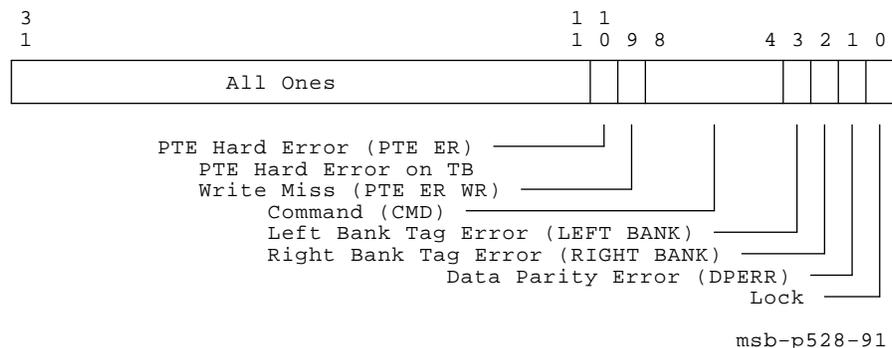
## P-Cache Status Register (PCSTS)

### P-Cache Status Register (PCSTS)

PCSTS and PCADR both record P-Cache tag and data parity errors. The function and operation of these registers is identical to the TBSTS and TBADR registers except that the PCADR stores physical quadword addresses rather than virtual byte addresses. Note that when PCSTS <0> is set, P-Cache memory reads, writes, and invalidates are disabled.

#### ADDRESS

*IPR244 (NVAX chip)*



#### bits<31:11>

Name: Reserved

Mnemonic: None

Type: -

Reserved; initialized to all ones.

#### bit<10>

Name: PTE Hard Error

Mnemonic: PTE ER

Type: R/W1C, 0

When set, PTE ER indicates a hard error on a PTE data read.

#### bit<9>

Name: PTE Hard Error on a TB Write Miss

Mnemonic: PTE ER WR

Type: R/W1C

When set, PTE ER WR indicates a hard error on a PTE data read caused by a TB miss on a write.

# KA66A CPU Module Internal Processor Registers

## P-Cache Status Register (PCSTS)

---

### bits<8:4>

Name: Command  
Mnemonic: CMD  
Type: RO

The CMD field contains the command corresponding to a P-cache parity error and is defined as follows:

---

CMD	NDAL Function
0	NOP
1	Reserved
2	WRITE (Write)
3	WDISOWN (Write-Disown; Writeback)
4	IREAD (Instruction-Stream Read)
5	DREAD (Data-Stream Read)
6	OREAD (Ownership Read)
7	Reserved
8	Reserved
9	RDE (Read Data Error)
A	WDATA (Write Data Cycle)
B	BADWDATA (Bad Write Data)
C	RDR0 (Read Data0 Return/fill)
D	RDR1 (Read Data1 Return/fill)
E	RDR2 (Read Data2 Return/fill)
F	RDR3 (Read Data3 Return/fill)

---

---

### bit<3>

Name: Left Bank Tag Error  
Mnemonic: LEFT BANK  
Type: RO

When set, LEFT BANK indicates that a P-cache tag parity error occurred on the left bank of the P-cache.

---

### bit<2>

Name: Right Bank Tag Error  
Mnemonic: RIGHT BANK  
Type: RO

When set, RIGHT BANK indicates that a P-cache tag parity error occurred on the right bank of the P-cache.

## KA66A CPU Module Internal Processor Registers

### P-Cache Status Register (PCSTS)

---

#### bit<1>

Name: Data Parity Error

Mnemonic: DPERR

Type: RO

When set, DPERR indicates a P-cache data parity error.

---

#### bit<0>

Name: Lock

Mnemonic: None

Type: R

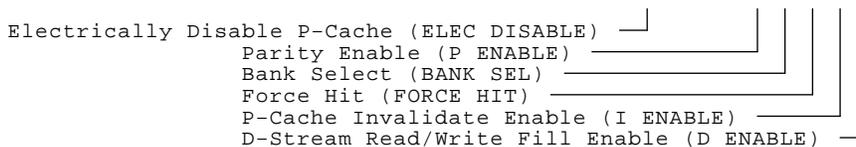
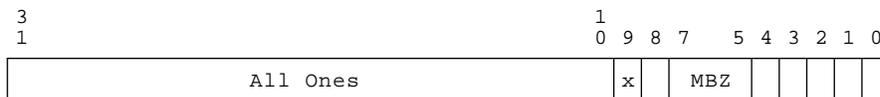
When set, Lock validates PCSTS contents and prevents modification of any other field. When clear, Lock indicates that no P-cache parity error has been recorded and allows PCSTS and PCADR to be updated.

## P-Cache Control Register (PCCTL)

PCCTL controls functions of the primary cache.

### ADDRESS

*IPR248 (NVAX chip)*



msb-p529-92

### bits<31:10>

Name: Reserved  
 Mnemonic: None  
 Type: –  
 Reserved; initialized to ones.

### bit<9>

Name: Reserved  
 Mnemonic: None  
 Type: –  
 Reserved; initialized to either logic level.

### bit<8>

Name: Electrically Disable the P-Cache  
 Mnemonic: ELEC DISABLE  
 Type: R/W, 0

When set, the P-cache is disabled electrically to reduce power dissipation. This bit should only be set when the P-cache is turned off by clearing both I ENABLE and D ENABLE; unpredictable operation results if either bit is set. P-cache tag and parity IPRs will not function properly when this bit is set.

# KA66A CPU Module Internal Processor Registers

## P-Cache Control Register (PCCTL)

---

### bits<7:5>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

---

### bit<4>

Name: Parity Enable  
Mnemonic: P ENABLE  
Type: R/W, 0  
When set, P ENABLE enables detection of P-cache tag and data parity errors.

---

### bit<3>

Name: Bank Select  
Mnemonic: BANK SEL  
Type: R/W, 0  
When set with Force Hit set, selects the "right bank" of the addressed P-cache index. When clear with Force Hit set, selects the "left bank" of the addressed P-cache index. BANK SEL is a do not care when Force Hit is clear. BANK SEL never affects bank selection during IPR reads and IPR writes to the P-cache tags or P-cache data parity bits; bank selection for these commands is determined by the specified IPR address.

---

### bit<2>

Name: Force Hit  
Mnemonic: None  
Type: R/W, 0  
When set, Force Hit causes a P-cache hit on all reads and writes when P-cache is enabled for I- or D-stream operation.

---

### bit<1>

Name: P-Cache Invalidate Enable  
Mnemonic: I ENABLE  
Type: R/W, 0  
When set, enables P-cache processing of INVALID, IREAD, and I CF commands. When clear, forces a P-cache miss on IREAD operations and prevents state modification due to an I CF operation. An ACV /TNV condition overrides a clear I ENABLE and a hit is forced in the P-cache.

# KA66A CPU Module Internal Processor Registers

## P-Cache Control Register (PCCTL)

---

**bit<0>**

Name: D-Stream Read/Write Fill Enable

Mnemonic: D ENABLE

Type: R/W, 0

When set, D ENABLE enables the P-cache for INVALID and D-stream read/write/fill operations, qualified by other control bits. When clear, forces a P-cache miss on all P-cache D-stream read/write/fill operations. An ACV/TNV condition overrides a clear D ENABLE and a hit is forced in the P-cache.

### 2.7.3 XMI Registers

In addition to the internal processor registers, the KA66A CPU module contains registers in XMI private space and XMI required registers in XMI nodespace. These registers are listed in Table 2–32 and Table 2–33.

The KA66A CPU module's XMI registers have the following characteristics:

- The mask bits are ignored on writes to the KA66A CPU module's control and status registers. The CPU always performs a full longword write.
- Interlocks are not supported. Interlock Read and Unlock Write Mask transactions are treated as Read and Write Mask transactions, respectively.
- The XMI responder queue is only one deep so the KA66A CPU module will NO ACK subsequent CSR references until the read data for the queued CSR read has been returned.
- Write transactions directed at read-only registers are accepted and acknowledged but no action is taken and the register's value is not changed.

Attempts to read or write unimplemented nodespace regions result in a NO ACK for the XMI transaction

**Table 2–32 KA66A CPU Module Registers in XMI Private Space**

Register	Mnemonic	Address	Location
NDAL Control and Status	NCSR	E000 0000	NEXMI chip
TOY Clock Registers		E018 3000 – E018 300D	Watch chip
BBU RAM		E018 300E – E018 303F	Watch chip
NEXMI Input Port	IPOINT	E018 4000	NEXMI chip
NEXMI Output Port0	OPOINT0	E018 5000	NEXMI chip
NEXMI Output Port1	OPOINT1	E018 6000	NEXMI chip
UART Registers		E018 7000 – E018 700F	UART chip
IPR Address Space		E100 0000 – E100 03FF	
IP IVINTR Generation	IPINTR	E101 0000 – E101 FFFF	NEXMI chip
WE IVINTR Generation	WEINTR	E102 0000 – E102 FFFF	NEXMI chip

**Table 2–33 XMI Registers for the KA66A CPU Module**

<b>Register</b>	<b>Mnemonic</b>	<b>Address</b>	<b>Location</b>
Device Register	XDEV	BB <sup>1</sup> + 0000 0000	NEXMI chip
Bus Error	XBER	BB + 0000 0004	NEXMI chip
Failing Address	XFADR	BB + 0000 0008	NEXMI chip
XMI General Purpose	XGPR	BB + 0000 000C	NEXMI chip
Node-Specific Control and Status	NSCSR	BB + 0000 001C	NEXMI chip
XMI Control Register	XCR	BB + 0000 0024	NEXMI chip
Failing Address Extension	XFAER	BB + 0000 002C	NEXMI chip
Bus Error Extension	XBEER	BB + 0000 0034	NEXMI chip
Writeback 0 Failing Address	WFADR0	BB + 0000 0040	NEXMI chip
Writeback 1 Failing Address	WFADR1	BB + 0000 0044	NEXMI chip

<sup>1</sup>BB = base address of a node, which is the address of the first location in nodespace.

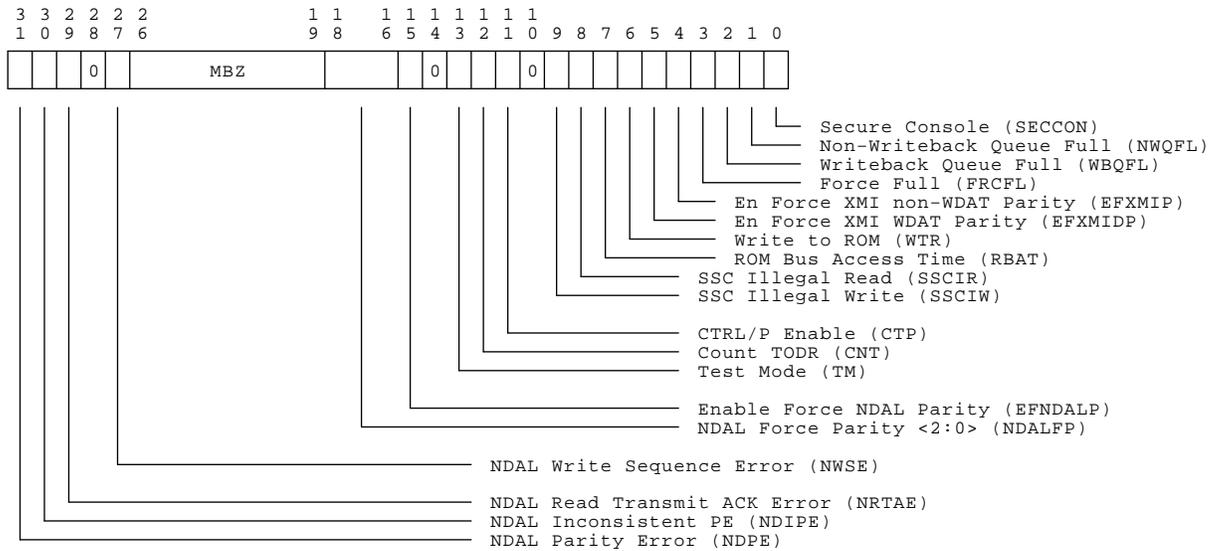
# KA65A CPU Module XMI Private Space Registers

## NDAL Control and Status Register (NCSR)

### NDAL Control and Status Register (NCSR)

NCSR holds information used to test or control various NDAL functions.

**ADDRESS**      *E000 0000 (NEXMI chip)*



msb-p558-91

#### bit<31>

Name:      NDAL Parity Error  
Mnemonic:   NDPE  
Type:        R/W1C, 0

During every NDAL cycle, the NEXMI computes and checks NDAL parity. If a parity error is detected, NDAL Parity Error is set.

#### bit<30>

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

NDAL Inconsistent Parity Error is set when an NDAL parity error is detected, and the cycle is ACKed by another node. This indicates that another node detected good parity while this node detected a parity error. Parity checking is performed for all NDAL transactions, and not just for those generated by this node.

# KA65A CPU Module XMI Private Space Registers

## NDAL Control and Status Register (NCSR)

---

### bit<29>

Name: NDAL Read Transmit ACK Error  
Mnemonic: NRTAE  
Type: R/W1C, 0

The NEXMI verifies that all its NDAL read transmissions are ACKed. NDAL Read Transmit ACK Error is set if ACK is not returned after NEXMI has driven read data onto the NDAL.

---

### bit<28>

Name: Reserved  
Mnemonic: None  
Type: –

Reserved; must be zero.

---

### bit<27>

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

When set, the NDAL Write Sequence Error indicates that a WRITE or WDISOWN command was received and was not followed by WDATA or BADWDATA command.

---

### bits<26:19>

Name: Reserved  
Mnemonic: None  
Type: RO, 0

Reserved; must be zero.

---

### bits<18:16>

Name: NDAL Force Parity <2:0>  
Mnemonic: NDALFP  
Type: R/W, 0

The bits in this field correspond to the NDAL parity bits PARITY<2:0> which will be driven with bad parity when set. Bad parity will be transmitted on command/address cycles and data cycles from this node in place of generated parity when the NCSR<EFNDALP> Enable Force NDAL Parity bit is set.

## KA65A CPU Module XMI Private Space Registers

### NDAL Control and Status Register (NCSR)

---

#### bit<15>

Name: Enable Force NDAL Parity  
Mnemonic: EFNDALP  
Type: R/W, 0

If Enable Force NDAL Parity is written with a one, NEXMI uses the bit field NDALFP<2:0> to selectively enable bad parity on the NDAL PARITY<2:0> lines. For example, if NCSR<18:16> = 010, then  
NDAL P<2> = good parity  
NDAL P<1> = bad parity  
NDAL P<0> = good parity

---

#### bit<14>

Name: Reserved  
Mnemonic: None  
Type: RO, 0

Reserved; must be zero.

---

#### bit<13>

Name: Test Mode  
Mnemonic: TM  
Type: R/W, 0

When the Test Mode (TM) bit is set, the Time-of-Day Register (TODR) is clocked by writing one to the CNT field. This bit is for diagnostic purposes only.

---

#### bit<12>

Name: Count TODR  
Mnemonic: CNT  
Type: WO, 0

Writing a one to this bit causes the TODR to increment by one. This bit is for diagnostic purposes only.

---

#### bit<11>

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

Setting CTRL/P Enable to a one causes CTRL/P to be recognized as a break in the console terminal UART. When CTP is a zero, only the break character is recognized as give me a break. CTP is cleared when the chip is reset. The console program sets this bit.

# KA65A CPU Module XMI Private Space Registers

## NDAL Control and Status Register (NCSR)

---

### bit<10>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

---

### bit<9>

Name: SSC Illegal Write  
Mnemonic: SSCIW  
Type: R/W1C, 0

When the SSC Illegal Write bit is set, an illegal write was attempted to SSC space. Both nonexistent address errors and illegal SSC commands will cause this bit to set. The write attempt results in a hard error interrupt.

---

### bit<8>

Name: SSC Illegal Read  
Mnemonic: SSCIR  
Type: R/W1C, 0

When the SSC Illegal Read bit is set, an illegal read was attempted to SSC space. Both nonexistent address errors and illegal SSC commands will cause this bit to set. The read attempt results in a soft error interrupt and an RDE.

---

### bit<7>

Name: ROM Bus Access Time  
Mnemonic: RBAT  
Type: R/W, 0

Access times on the ROM bus are based on the NDAL clock cycle time. Setting this bit increases the number of NDAL cycles to wait on a ROM bus access. This ensures that the minimum ROM bus access time is sufficient.

---

### bit<6>

Name: Write to ROM  
Mnemonic: WRT  
Type: R/W1C, 0

When set, WRT indicates that an attempt to write the ROM has occurred.

# KA65A CPU Module XMI Private Space Registers

## NDAL Control and Status Register (NCSR)

---

### bit<5>

Name: Enable Force XMI WDAT Parity  
Mnemonic: EFXMIDP  
Type: R/W, 0

If Enable Force XMI WDAT Parity is set, NEXMI uses the bit field XCR:XMIFP<2:0> to selectively enable bad parity on the XMI P<2:0> lines during WDAT cycles only. For example, if XCR<21:19> = 101, then

XMI P<2> = bad parity  
XMI P<1> = good parity  
XMI P<0> = bad parity

---

### bit<4>

Name: Enable Force XMI non-WDAT Parity  
Mnemonic: EFXMIP  
Type: R/W, 0

If Enable Force XMI non-WDAT Parity is set, NEXMI uses the bit field XCR:XMIFP<2:0> to selectively enable bad parity on the XMI P<2:0> lines during non-WDAT cycles only. For example, if XCR0<21:19> = 101, then

XMI P<2> = bad parity  
XMI P<1> = good parity  
XMI P<0> = bad parity

---

### bit<3>

Name: Force Full  
Mnemonic: FRCFL  
Type: R/W, 0

If Force Full is set, the NEXMI will not be granted the NDAL. This forces the Responder Queue to back up with invalidate writes. This bit is for diagnostic purposes only, to test the XMI Suppress logic and should be cleared during normal operation.

---

### bit<2>

Name: Writeback Queue Full  
Mnemonic: WBQFL  
Type: R/W1C, 0

The Writeback Queue Full bit is set when a legal NDAL Writeback cycle occurs but the WBQ is full and cannot accept additional data.

## KA65A CPU Module XMI Private Space Registers NDAL Control and Status Register (NCSR)

---

### bit<1>

Name: Non-Writeback Queue Full  
Mnemonic: NWQFL  
Type: R/W1C, 0

The Non-Writeback Queue Full bit is set when a legal NDAL Non-Writeback cycle occurs but the NWQ is full and cannot accept additional data.

---

### bit<0>

Name: Secure Console  
Mnemonic: SECCON  
Type: RO

This bit indicates the state of the front-panel console enable switch. The input to this bit is XMI CON SECURE L, which is fed directly into the NEXMI chip. A zero indicates that console halts are disabled and the console is secure, and a one indicates that console halts are enabled.

# KA65A CPU Module XMI Private Space Registers

## NEXMI Input Port Register (IPORT)

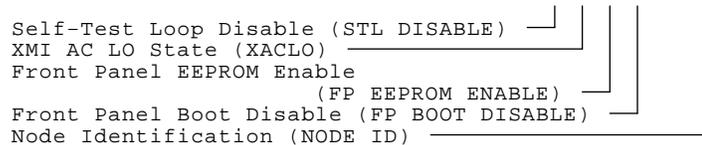
---

## NEXMI Input Port Register (IPORT)

The NEXMI chip reads eight input signals (IPORT H<7:0>) from the ROM bus. The IPORT gives KA66A CPU module state information.

---

**ADDRESS**      *E018 4000 (NEXMI chip)*



msb-p569-91

---

### bits<30:8>

Name:      Reserved  
Mnemonic:   None  
Type:      -  
Reserved; must be zero.

---

### bit<7>

Name:      Self-Test Loop Disable  
Mnemonic:   STL DISABLE  
Type:      RO, 1  
STL DISABLE indicates the state of the IO SELF-TEST LOOP L signal. A zero indicates that the console loops on self-test, and a one, the default value, indicates that the console performs self-test only once.

---

### bit<6>

Name:      XMI AC LO  
Mnemonic:   XACLO  
Type:      RO  
XACLO shows the state of the XMI AC LO L signal. A zero indicates that XMI AC LO L is asserted, and a one indicates that the line is deasserted. The console does not attempt to reference memory until XACLO is a one.

# KA65A CPU Module XMI Private Space Registers

## NEXMI Input Port Register (IPORT)

---

### bit<5>

Name: Front Panel EEPROM Enable  
Mnemonic: FP EEPROM ENABLE  
Type: RO

FP EEPROM ENABLE shows the state of the control (front) panel lower key switch as a reflection of the XMI UPDATE EN H signal. When FP EEPROM ENABLE is a zero, the EEPROM cannot be written or is disabled; a one indicates that the EEPROM is enabled; that is, the lower key switch is in the Update position and the EEPROM can be written.

---

### bit<4>

Name: Front Panel Boot Disable  
Mnemonic: FP BOOT DISABLE  
Type: RO

FP BOOT DISABLE indicates the state of the control (front) panel lower key switch. The input to the bit is the XMI BOOT EN L signal. A zero indicates that booting is enabled (that is, the lower key switch is in the Auto Start position), and a one indicates that booting is disabled.

---

### bits<3:0>

Name: Node Identification  
Mnemonic: NODE ID  
Type: RO

NODE ID contains the node identification of the XMI backplane slot.

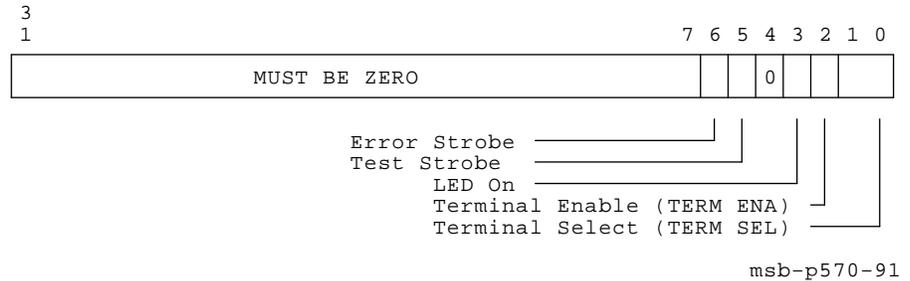
# KA65A CPU Module XMI Private Space Registers

## NEXMI Output Port0 Register (OPORT0)

### NEXMI Output Port0 Register (OPORT0)

OPORT0 is used by diagnostics and for controlling an auxiliary console.

**ADDRESS** *E018 5000 (NEXMI chip)*



#### bits<31:7>

Name: Reserved  
Mnemonic: None  
Type: -  
Reserved; must be zero.

#### bit<6>

Name: Error Strobe  
Mnemonic: None  
Type: R/W, 0

Error Strobe is used by diagnostics to provide a trigger signal to the backplane whenever an error is detected.

#### bit<5>

Name: Test Strobe  
Mnemonic: None  
Type: R/W, 0

Test Strobe is used by diagnostics to provide a trigger signal to the backplane at the beginning of each test.

#### bit<4>

Name: Reserved  
Mnemonic: None  
Type: -  
Reserved; must be zero.

# KA65A CPU Module XMI Private Space Registers

## NEXMI Output Port0 Register (OPORT0)

---

### bit<3>

Name: LED On  
Mnemonic: None  
Type: R/W, 0

This bit drives the self-test-passed LED and when set indicates that self-test was successful.

---

### bit<2>

Name: Terminal Enable  
Mnemonic: TERM ENA  
Type: R/W, 0

This bit enables the KA66A CPU module to drive the XMI console line on the backplane. If TERM ENA is set, console output will be transmitted both to the auxiliary console line and to the XMI console line on the backplane. If TERM ENA is clear, console output will be transmitted only to the auxiliary console line.

---

### bit<1:0>

Name: Terminal Select  
Mnemonic: TERM SEL  
Type: R/W, 0

This field selects the console terminal mode and is encoded as follows:

---

<1:0>	Console Terminal Mode
00	Auxiliary Console Mode. The auxiliary console line is connected to the NEXMI console terminal input.
01	System Console Mode. The XMI backplane console line is connected to the NEXMI console terminal input.
10	Auxiliary Console Loopback Mode. The auxiliary console output is connected back to the NEXMI console terminal input.
11	System Console Loopback Mode. The XMI console output is connected back to the NEXMI console terminal input. Note, however, that if TERM ENA is clear, the transmitted character is not transmitted on the XMI backplane console line.

---



# KA66A CPU Module XMI Nodespace Registers

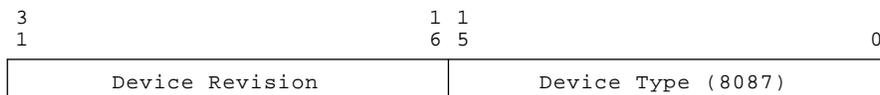
## Device Register (XDEV)

### Device Register (XDEV)

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

#### ADDRESS

*Nodespace base address + 0000 0000 (NEXMI chip)*



msb-p553-91

#### bits<31:16>

Name: Device Revision

Mnemonic: DREV

Type: R/W, 0

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

KA66A CPU Module Revision	DREV (decimal)	DREV (hex)
A0	1	0001
A1	1	0001
B0	2	0002
B1	2	0002
.		
.		
.		
Z0	26	001A

#### bits<15:0>

Name: Device Type

Mnemonic: DTYPE

Type: R/W, 0

Identifies the type of node. The Device Type field is broken into two subfields: Class and ID. The Class field indicates the major category of the node. The ID field uniquely identifies a particular device within a specified class. DTYPE contains 8087 (hex) for the KA66A CPU module.



# KA66A CPU Module XMI Nodespace Registers

## Bus Error Register (XBER)

---

### bit<31>

Name: Error Summary  
Mnemonic: ES  
Type: RO, 1

The state of ES represents the logical OR of the error bits STF, ETF, NSES, TTO, CNAK, RER, RSE, NRR, CRD, WDNAK, RIDNAK, WSE, PE, IPE, WEI, and CC in this register. Therefore, ES is asserted if any error bit is asserted. ES clears when all error bits are cleared.

---

### bit<30>

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

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

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

---

### bit<29>

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

Writing a one to NHALT while halts are enabled, forces the node to go into a "quiet" state while retaining as much state as possible. The KA66A CPU module will force the CPU to halt at the next instruction boundary and go into console mode waiting for console commands. The console code clears NHALT before exit to prevent an immediate reentry.

# KA66A CPU Module XMI Nodespace Registers

## Bus Error Register (XBER)

---

### bit<28>

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

XBAD indicates the state of the XMI BAD L signal. A one indicates that XMI BAD L is asserted. XMI BAD L is asserted when any one (or more) nodes assert the line and deasserts only when no node asserts it.

Writes to XBAD cause the state to be driven on the wired-OR XMI BAD L line by this node; writing a one asserts XMI BAD L, while writing a zero releases this node's contribution to XMI BAD L.

XBAD asserts on reset, causing XMI BAD L to assert. XMI BAD L remains asserted until all nodes stop asserting it.

---

### bit<27>

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

CC sets when the KA66A CPU module detects a single-bit CNF error. Single-bit CNF errors are automatically corrected by the XCLOCK chip in the XMI Corner. When CC sets, Error Summary (ES) also sets.

---

### bit<26>

Name: Reserved  
Mnemonic: None  
Type: –

Reserved; must be zero.

---

### bit<25>

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

When set, WEI indicates that the KA66A CPU module received a write error interrupt IVINTR transaction. When WEI sets, a hard error interrupt is sent to the CPU and Error Summary (ES) sets.

# KA66A CPU Module XMI Nodspace Registers

## Bus Error Register (XBER)

---

### bit<24>

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

When set, IPE indicates that the KA66A CPU module detected a parity error on an XMI cycle and the confirmation for the erroneous cycle was ACK. This indicates that at least one node (the responder) detected good parity during the cycle time that this KA66A CPU module detected a parity error. If the write to memory was successful, it could leave the cache incoherent. The KA66A CPU module checks all XMI transactions, not just those it generates. When IPE sets, both Parity Error (PE) and Error Summary (ES) set.

---

### bit<23>

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

When set, PE indicates that the NEXMI detected a parity error on an XMI cycle. When PE sets, Error Summary (ES) also sets, and Inconsistent Parity Error (IPE) may also set, if appropriate.

---

### bit<22>

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

When set, indicates that a write transaction directed to one of the NEXMI's CSRs was aborted due to missing data cycles. If WSE is set, Error Summary (ES) also sets.

---

### bit<21>

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

If the NEXMI transmits data in response to a CSR read, and the read data cycle is not acknowledged (receives a NO ACK), RIDNAK and Error Summary (ES) are set.

# KA66A CPU Module XMI Nodespace Registers

## Bus Error Register (XBER)

---

### bit<20>

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

When set, WDNAK indicates that a write data cycle transmitted by the KA66A CPU module received a NO ACK confirmation. WDNAK sets only if the reattempt fails or is disabled. When WDNAK sets, Error Summary sets. If error retry is enabled, Transaction Timeout (TTO) also sets.

---

### bit<19>

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

When set, CRD indicates that this KA66A CPU module received a CRD read response, meaning that memory saw a parity error when reading data out of memory and corrected it. When CRD sets, Error Summary (ES) also sets.

---

### bit<18>

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

When set, NRR indicates that a transaction initiated by this KA66A CPU module failed due to a read response timeout (no LOC, RER, CRD, or GRD). When NRR sets, Error Summary (ES) and Transaction Timeout (TTO) also set.

---

### bit<17>

Name: Read Sequence Error  
Mnemonic: RSE  
Type: R/W1C, 0

When set, RSE indicates that data has been returned out of sequence. When data is returned in response to a read operation, the NEXMI checks the received sequence number against the one that it expects. If the two do not match, the returned data cycles are out of sequence and RSE is set along with Error Summary (ES).

# KA66A CPU Module XMI Nodspace Registers

## Bus Error Register (XBER)

---

### bit<16>

Name: Read Error Response  
Mnemonic: RER  
Type: R/W1C, 0

When set, RER indicates that a node on the XMI received a Read Error Response, meaning that the result of a read transaction or an interrupt vector returned in an IDENT transaction is in error. When RER sets, Error Summary (ES) also sets.

---

### bit<15>

Name: Command NO ACK  
Mnemonic: CNAK  
Type: R/W1C, 0

When set, CNAK indicates that a command cycle transmitted by the KA66A CPU module received a NO ACK confirmation, usually caused by a reference either to a nonexistent memory location or to an I/O space location. This bit is set only if the error recovery reattempts fail or are disabled. When CNAK sets, Error Summary (ES) also sets.

---

### bit<14>

Name: Reserved  
Mnemonic: None  
Type: –

Reserved; must be zero.

---

### bit<13>

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

When set, TTO indicates that a transaction initiated by this KA66A CPU module failed due to a transaction timeout. The timeout counter is started when the NEXMI requests the XMI for a transaction. This bit is set only if retries fail. Write Data NO ACK (WDNAK), No Read Response (NRR), Command NO ACK (CNAK), and XBEER<OLR> indicate the cause of the timeout. If none of the bits are set, the NEXMI was never granted the XMI bus for the transaction. When TTO sets, Error Summary (ES) also sets.

# KA66A CPU Module XMI Nodespace Registers

## Bus Error Register (XBER)

---

### bit<12>

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

When set, NSES indicates that a node-specific error condition was detected. NSES is the logical OR of the implemented bits in XBEER. When NSES sets, Error Summary (ES) also sets. NSES clears when all error bits clear.

---

### bit<11>

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

When set, ETF indicates that the KA66A CPU module has not yet passed its extended test. This bit is cleared by console code when the KA66A CPU module passes its extended test. When ETF sets, Error Summary (ES) also sets.

---

### bit<10>

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

When set, STF indicates that the KA66A CPU module has not yet passed its self-test. This bit is cleared by console code when the KA66A CPU module passes its self-test. When STF sets, Error Summary (ES) also sets.

---

### bits<9:4>

Name: Failing Commander ID  
Mnemonic: FCID  
Type: RO

FCID holds the commander ID of a failing transaction and is equivalent to the XMI node number of the CPU logging the error. These bits are hard wired to the XMI backplane and are not effected by the lock.

---

### bits<3:0>

Name: Reserved  
Mnemonic: None  
Type: -

Reserved; must be zero.

## Failing Address Register (XFADR)

XFADR logs address and length information associated with a failing transaction. An associated register, XFAER, logs address extension bits, mask and command information associated with a failing transaction. XFADR and XFAER are latched at the start of every XMI transaction unless the registers are locked because one or more of XBER<20> (WDNAK), XBER<18> (NRR), XBER<17> (RSE), XBER<16> (RER), XBER<15> (CNAK), XBER<13> (TTO), or XBEER<1> (SEO) are set at the beginning of the transaction.

There are three interpretations of XFADR, depending on the XMI command. XFAER<31:28> determine the hex value of the XMI command.

**ADDRESS**                      *Nodespace base address + 0000 0008 (NEXMI chip)*

**XFADR, when the XMI command is neither an IDENT transaction nor an IVINTR transaction:**



msb-p345-90

### **bits<31:30>**

Name:        Failing Length  
Mnemonic:   FLN  
Type:         RO

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

## KA66A CPU Module XMI Nodspace Registers

### Failing Address Register (XFADR)

---

bits<29:0>

Name: Failing Address

Mnemonic: None

Type: RO

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

**NOTE: When XFADR contains a read or write address, the bit map for a 32-bit NDAL address is as follows:**

**If XFADR<29> = 0 (memory space), then 32-bit NDAL address is:**

**XFADR<29> + XFAER<17:16> + XFADR<28:0>**

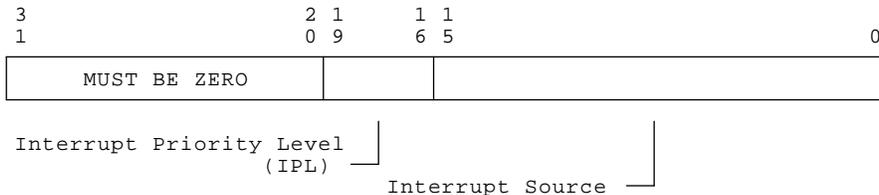
**If XFADR<29> = 1 (I/O space), then 32-bit NDAL address is:**

**111 + XFADR<28:0>**

# KA66A CPU Module XMI Nodespace Registers

## Failing Address Register (XFADR)

**XFADR, when the XMI command is 9 (hex), an IDENT transaction:**



msb-p346-90

### bits<31:20>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

### bits<19:16>

Name: Interrupt Priority Level  
Mnemonic: IPL  
Type: RO

IPL is a bit mask that specifies the interrupt priority level for the IDENT command as shown below:

Bit	IPL (hex)
19	17
18	16
17	15
16	14

### bits<15:0>

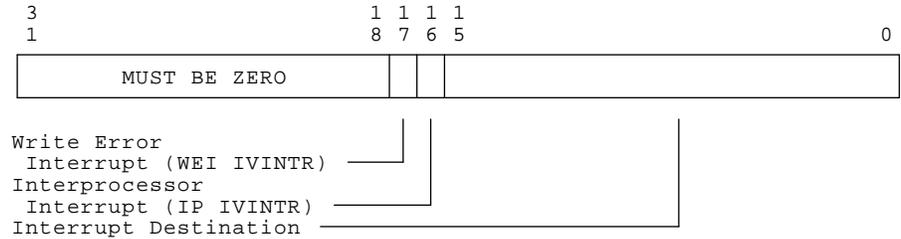
Name: Interrupt Source  
Mnemonic: None  
Type: RO

The Interrupt Source field is a bit mask that specifies the IDENT command target-node ID. Bit<14> corresponds to node E, bit<13> corresponds to node D, . . . bit<1> corresponds to node 1.

# KA66A CPU Module XMI Nodespace Registers

## Failing Address Register (XFADR)

**XFADR, when the XMI command is F (hex), an IVINTR transaction:**



msb-p347-90

### bits<31:18>

Name: Reserved  
Mnemonic: None  
Type: -  
Reserved; must be zero.

### bit<17>

Name: Write Error Interrupt  
Mnemonic: WEI IVINTR  
Type: RO

WEI IVINTR sets if the implied vector interrupt request was for a write error interrupt.

### bit<16>

Name: Interprocessor Interrupt  
Mnemonic: IP IVINTR  
Type: RO

IP IVINTR sets if the implied vector interrupt request was for an interprocessor interrupt.

### bits<15:0>

Name: Interrupt Destination  
Mnemonic: None  
Type: RO

The Interrupt Destination field is a bit mask that specifies the IVINTR command target-node ID. Bit<14> corresponds to node E, bit<13> corresponds to node D, . . . bit<1> corresponds to node 1.

---

## XMI General Purpose Register (XGPR)

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

---

### ADDRESS

*Nodespace base address + 0000 000C (NEXMI chip)*

3  
1

0



msb-p201-89

---

### bits<31:0>

Name: XMI General Purpose Register

Mnemonic: XGPR

Type: R/W, 0

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



# KA66A CPU Module XMI Nodespace Registers

## Node-Specific Control and Status Register (NSCSR)

---

### bit<5>

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

BP is set by console code to indicate that this node is the boot processor.

---

### bit<4>

Name: Warm Start  
Mnemonic: WS  
Type: RO, 0

WS sets to indicate that battery-backed-up power was maintained during a power failure and that the console code should attempt a "warm start." WS is loaded with the state of the XMI RESET L signal when the XMI DC LO L signal is deasserted. WS is not used after a node reset.

---

### bits<3:0>

Name: NEXMI Revision  
Mnemonic: NREV  
Type: RO, 0

NREV contains the revision of the NEXMI chip.



# KA66A CPU Module XMI Nodspace Registers

## XMI Control Register (XCR)

---

### bit<29>

Name: Data Out  
Mnemonic: DO  
Type: RO, 0

The DO bit is the value shifted out of the MSB of the counter under test.

---

### bit<28>

Name: Count  
Mnemonic: CNT  
Type: WO, 0

Writing a one to this bit causes the counter under test to increment by one.

---

### bit<27>

Name: Serial Shift  
Mnemonic: SHF  
Type: WO, 0

Writing a one to this bit causes the counter under test to serial shift by one from LSB toward MSB. The XCR<DI> is shifted into the LSB of the counter, and the MSB is shifted into the XCR<DO>.

---

### bits<26:24>

Name: Counter Select  
Mnemonic: CNTSEL  
Type: R/W, 0

CNTSEL is used to select a particular counter to test. It is encoded as follows:

---

<26:24>	Counter
000	XCC TTO counter (20 bits)
001	XWC0 TTO counter (20 bits)
010	XWC1 TTO counter (20 bits)
011	Lockout assertion timer (16 bits)
100	Lockout deassertion timer (16 bits)

---

---

### bits<23:22>

Name: Reserved  
Mnemonic: None  
Type: –

Reserved; must be zero.

# KA66A CPU Module XMI Nodespace Registers

## XMI Control Register (XCR)

---

### bits<21:19>

Name: XMI Force Bad Parity<2:1>  
Mnemonic: XMIFP  
Type: R/W, 0

Each bit of XMIFP corresponds to the XMI P<2:0> parity bits. When an XMIFP bit is set, it forces a bad parity on the XMI. Bad parity can be transmitted on XMI P<2:0>, during command/address cycles, and on XMI P<2> only, during data cycles.

---

### bit<18>

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

ESIO, when clear, causes the NEXMI to process invalidates from other nodes. ESIO, when set, causes the NEXMI to process invalidates for memory reads and writes generated by this node only. This allows a single CPU node to verify the operation of the invalidate logic. ESIO is set for diagnostic purposes only and must be clear during normal operation.

---

### bits<17:16>

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

TOS selects one of four timeout values used to detect both response and reattempt timeout conditions for commands and writebacks. TOS is for debug and diagnostic purposes only and must be cleared during normal operation. The timeout values are as follows:

---

TOS<1:0>	Timeout (Time/XMI Cycles)
00	16 ms/250K
01	16.38 $\mu$ sec/256
10	4.01 $\mu$ sec/64
11	1.92 $\mu$ sec/32

---

---

### bits<15:13>

Name: Reserved  
Mnemonic: None  
Type: –

Reserved; must be zero.

# KA66A CPU Module XMI Nodespace Registers

## XMI Control Register (XCR)

---

### bit<12>

Name: Lockout Debug Timeout Enable  
Mnemonic: LDTE  
Type: R/W, 0

When LDTE is set and lockout is enabled (XCR<LOCMODE> = 00, 01, or 10 (binary)), both the lockout assertion timer (LAT) and the lockout deassertion timer (LDT) are forced to one microsecond. When lockout is disabled (XCR<LOCMODE> = 11 (binary)), LDTE has no effect.

---

### bits<11:7>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

---

### bit<6>

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

CCID controls the generation of interrupts caused by corrected confirmations. A zero enables interrupts; a one disables interrupts. XBER<CC> is still set if a corrected confirmation error is detected regardless of the state of CCID. If interrupts are disabled, the soft error interrupt request to the NVAX chip is inhibited. If CC interrupts are disabled, software should clear XBER<CC> before reenabling interrupts to ensure that only new CC errors cause interrupts.

---

### bit<5>

Name: Corrected Read Data Interrupt Disable  
Mnemonic: CRDID  
Type: R/W, 0

CRDID controls the generation of interrupts caused by corrected read data. A zero enables interrupts; a one disables interrupts. XBER<CRD> is still set if a corrected confirmation error is detected regardless of the state of CRDID. If interrupts are disabled, the soft error interrupt request to the NVAX chip is inhibited. If CRD interrupts are disabled, software should clear XBER<CRD> before reenabling interrupts to ensure that only new CRD errors cause interrupts.

# KA66A CPU Module XMI Nodespace Registers

## XMI Control Register (XCR)

### bits<4:3>

Name: Trigger Control  
Mnemonic: TRIGC  
Type: R/W, 0

These bits control four XMI trigger conditions.

---

<4:3>	Trigger Conditions
00	Trigger disable: The trigger line will not be driven.
01	XMI trigger will be driven when soft error is asserted by NEXMI.
10	XMI trigger will be driven when hard error is asserted by NEXMI.
11	XMI trigger will be driven when error summary bit (XBER<ES>) is set.

---

### bit<2>

Name: XMI BAD Drive  
Mnemonic: XBADD  
Type: R/W, 1

When XBADD is written with a one, the XMI BAD line is asserted on the backplane. If this bit is written with a zero, this node's contribution to the assertion of XMI BAD is removed, and XBER<XBAD>, XBER<25>, is deasserted if no other node is asserting it. Reads show the current state of the driver from this node. To determine the state of the XMI BAD signal, the proper bit to read is XBER<XBAD>.

### bits<1:0>

Name: Lockout Mode  
Mnemonic: LOCMOD  
Type: R/W, 0

LOCMOD determines the lockout assertion timer (LAT) and lockout deassertion timer (LDT) values. These values are as follows:

---

<1:0>	Name	LAT	LDT
00	Default	256 us	2 ms
01		256 us	512 us
10		64 us	2 ms
11			Lockout Disable

---

Recommended use of the XMI counter test features:

Test Mode Setup:

- 1 Select counter under test by writing to the CNTSEL<2:0> field.

## KA66A CPU Module XMI Nodespace Registers

### XMI Control Register (XCR)

**2** Write a one to the <TM> bit (forces counter under test into test mode).

1 and 2 can be done in the same Write operation.

To shift a value into the counter:

**3** Write the DI bit with the value to be shifted into the LSB.

**4** Write a one to the <SHF> bit to activate the shift.

3 and 4 can be done in the same Write operation. The MSB is shifted into the <DO> bit.

To increment the counter:

**5** Write a one to the <CNT> bit to increment by one.



## KA66A CPU Module XMI Nodespace Registers

### Failing Address Extension Register (XFAER)

Hex	Command
0	Reserved
1	READ
2	IREAD
3	OREAD
4	DWMASK
5	Reserved
6	UWMASK
7	WMASK
8	INTR
9	IDENT
A	Reserved
B	TBDATA
C	Reserved
D	Reserved
E	Reserved
F	IVINTR

**NOTE:** When the command field <31:28> indicates an IDENT or an IVINTR, all other XFAER fields are zero and the format of XFADR is unique. See the XFADR register description.

---

#### bits<27:26>

Name: Reserved  
Mnemonic: None  
Type: –  
Reserved; must be zero.

---

#### bits<25:16>

Name: Address Extension  
Mnemonic: None  
Type: RO

The Address Extension field logs the value of XMI D<57:48> during the command cycle of a failing transaction. Address Extension contains address bits<38:29> of the specified address in read and write transactions.

## KA66A CPU Module XMI Nodspace Registers

### Failing Address Extension Register (XFAER)

---

**bits<15:0>**

Name: Mask

Mnemonic: None

Type: RO

The Mask field logs the value of XMI D<47:32> during the command cycle of a failing transaction. It contains the write mask for write transactions and is undefined for other transactions.

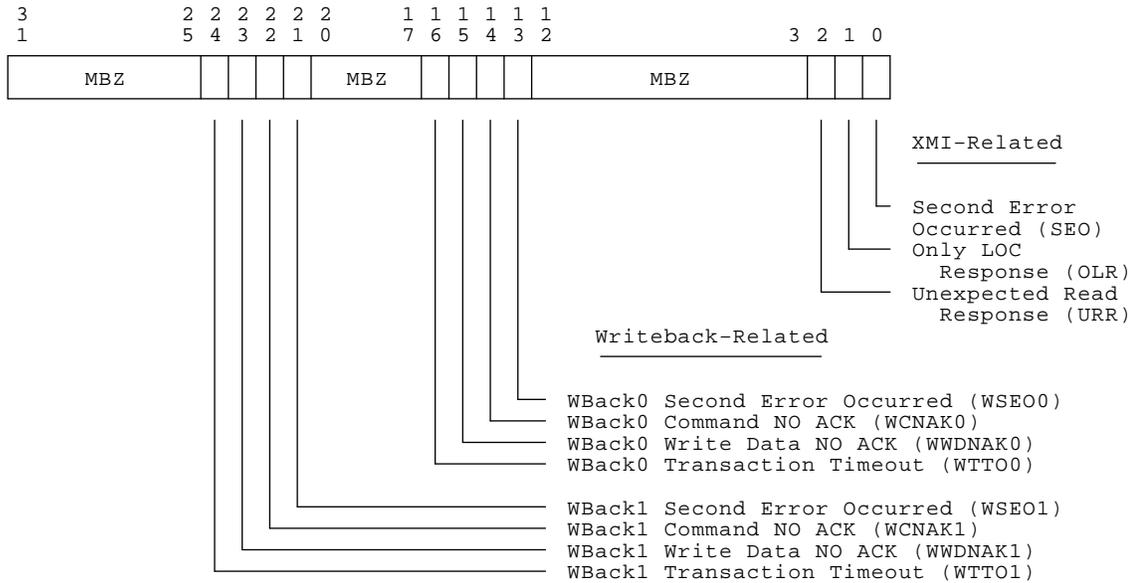
# KA66A CPU Module XMI Nodespace Registers

## Bus Error Extension Register (XBEER)

### Bus Error Extension Register (XBEER)

XBEER contains error status on XMI-related, writeback-related, and NDAL-related transactions. This status includes parity errors, NO ACKs, and timeouts.

**ADDRESS**      *Nodespace base address + 0000 0034 (NEXMI chip)*



msb-p557-91

**bits<31:25>**

Name:      Reserved  
Mnemonic:   None  
Type:        -  
Reserved; must be zero.

# KA66A CPU Module XMI Nodespace Registers

## Bus Error Extension Register (XBEER)

---

### bit<24>

Name: Writeback1 Transaction Timeout  
Mnemonic: WTTO1  
Type: R/W1C, 0

When a writeback transaction is initiated from writeback queue 1 by the NEXMI, a timeout counter is started. If the transaction does not complete before the timeout interval expires, this bit is set. WCNAK1 and WWDNAK1 indicate the cause of the timeout. If neither bit is set, the NEXMI was never granted the XMI bus for the transaction. If XBEER <WTTO1> is set, XBER <NSES> is also set.

---

### bit<23>

Name: Writeback1 Write Data NO ACK  
Mnemonic: WWDNAK1  
Type: R/W1C, 0

If the transaction timeout period (WTTO1) expires while the NEXMI is issuing the Disown Write command and is receiving a NO ACK in response to the transmission of any data cycles, this bit is set. XBEER <WTTO1> should also be set. If WWDNAK1 is set, XBER <NSES> is also set.

---

### bit<22>

Name: Writeback1 Command NO ACK  
Mnemonic: WCNAK1  
Type: R/W1C, 0

This bit is set if the transaction timeout period (WTTO1) expires while the NEXMI is receiving a NO ACK in response to transmission of the Disown Write command cycle. XBEER <WTTO1> should also be set. If WCNAK1 is set, XBER <NSES> is also set.

---

### bit<21>

Name: Writeback1 Second Error Occurred  
Mnemonic: WSEO1  
Type: R/W1C, 0

When set, this bit indicates that a second writeback hard error occurred while another one was being reported.

---

### bits<20:17>

Name: Reserved  
Mnemonic: None  
Type: -

Reserved; must be zero.

# KA66A CPU Module XMI Nodespace Registers

## Bus Error Extension Register (XBEER)

---

### bit<16>

Name: Writeback0 Transaction Timeout  
Mnemonic: WTTO0  
Type: R/W1C, 0

When a writeback transaction is initiated from writeback queue 0 by the NEXMI, a timeout counter is started. If the transaction does not complete before the timeout interval expires, this bit is set. WCNAK0 and WWDNAK0 indicate the cause of the timeout. If neither bit is set, the NEXMI was never granted the XMI bus for the transaction. If XBEER <WTTO0> is set, XBER <NSES> is also set.

---

### bit<15>

Name: Writeback0 Write Data NO ACK  
Mnemonic: WWDNAK0  
Type: R/W1C, 0

When set, this bit indicates that Disown Write data cycles were continually NO ACKed and the transaction eventually timed out (causing WTTO <16> to be set). If WWDNAK0 is set, XBER <NSES> is also set.

---

### bit<14>

Name: Writeback0 Command NO ACK  
Mnemonic: WCNAK0  
Type: R/W1C, 0

This bit is set if the transaction timeout period (WTTO0) expires while the NEXMI is receiving NO ACKs during transmission of the Disown Write command cycle. XBEER <WTTO0> should also be set. If WCNAK0 is set, XBER <NSES> is also set.

---

### bit<13>

Name: Writeback0 Second Error Occurred  
Mnemonic: WSEO0  
Type: R/W1C, 0

When set, this bit indicates that a second writeback hard error occurred while another one was being reported.

---

### bits<12:3>

Name: Reserved  
Mnemonic: None  
Type: -

Reserved; must be zero.

## KA66A CPU Module XMI Nodespace Registers

### Bus Error Extension Register (XBEER)

---

#### bit<2>

Name: Unexpected Read Response  
Mnemonic: URR  
Type: R/W1C, 0

When set, this bit indicates that the KA66A CPU module received a read response when it was not expecting one. A node is defined as not expecting a response when there are no outstanding reads or identifiers for that particular commander ID.

---

#### bit<1>

Name: Only LOC Response  
Mnemonic: OLR  
Type: R/W1C, 0

When set, this bit indicates that the CPU received only LOC responses while attempting a read-type transaction. This identifies the timeout case due to failure of the lockout mechanism to provide timely access to a resource such as interlock or ownership of a memory location. If OLR is set, XBER <TTO> is also set.

---

#### bit<0>

Name: Second Error Occurred  
Mnemonic: SEO  
Type: R/W1C, 0

When set, this bit indicates that a second non-writeback error occurred before the first one had been cleared. SEO is set whenever an error that normally locks the XFADR occurs a second time. Information of the second error is lost.

# KA66A CPU Module XMI Nodespace Registers

## Writeback 0 Failing Address Register (WFADR0)

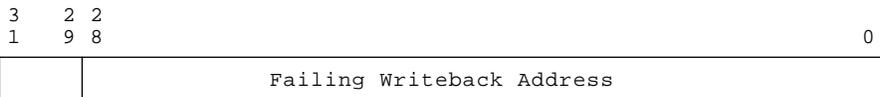
---

### Writeback 0 Failing Address Register (WFADR0)

WFADR0 is loaded at the start of every XMI writeback transaction when writeback queue 0 is used. WFADR0 re-creates the NDAL address. It is locked as a result of a writeback transaction timeout (XBEER<WTTO0>), and is unlocked once the error bits are cleared.

---

**ADDRESS**      *Nodespace base address + 0000 0040 (NEXMI chip)*



└─ Failing Writeback Address Extension

msb-p351-90

---

**bits<31:29>**

Name:      Failing Writeback Address Extension  
Mnemonic: None  
Type:      RO, 0

The Failing Writeback Address Extension field logs the value of XMI D <50:48>.

---

**bits<28:0>**

Name:      Failing Writeback Address  
Mnemonic: None  
Type:      RO, 0

The Failing Writeback Address field logs the writeback queue 0 XMI D <28:0> during the command cycle of a failing writeback transaction.

# KA66A CPU Module XMI Nodspace Registers

## Writeback 1 Failing Address Register (WFADR1)

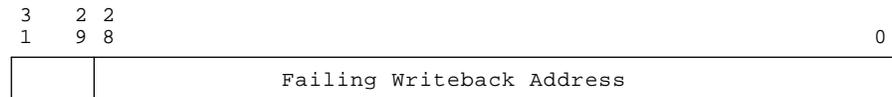
---

### Writeback 1 Failing Address Register (WFADR1)

WFADR1 is loaded at the start of every XMI writeback transaction when writeback queue 1 is used. WFADR1 re-creates the NDAL address. It is locked as a result of a writeback transaction timeout (XBEER<WTTO1>), and is unlocked once the error bits are cleared.

---

**ADDRESS**      *Nodspace base address + 0000 0044 (NEXMI chip)*



└─ Failing Writeback Address Extension

msb-p351-90

---

#### bits<31:29>

Name:      Failing Writeback Address Extension  
Mnemonic: None  
Type:      RO, 0

The Failing Writeback Address Extension field logs the value of XMI D <50:48>.

---

#### bits<28:0>

Name:      Failing Writeback Address  
Mnemonic: None  
Type:      RO, 0

The Failing Writeback Address field logs the writeback queue 1 XMI D <28:0> during the command cycle of a failing writeback transaction.

---

## 2.8 KA66A CPU Module Initialization, Self-Test, and Booting

**This section gives the KA66A CPU module initialization overview, describes the results of initialization, and discusses the bootstrapping or restarting of the operating system.**

---

### 2.8.1 Initialization Overview

The three ways to reset the KA66A CPU module are:

- **Power-Up Sequence**—When the VAX 6000 Model 600 is powered up, XMI AC LO L and XMI DC LO L are sequenced so that all XMI nodes are reset.
- **System Reset**—The XMI emulates a power-up sequence by asserting the XMI RESET L line, causing the power supply to sequence XMI AC LO L and XMI DC LO L as in a "real" power-up. The XMI does not differentiate between a "real" power-up and a system reset. A system reset is caused by:
  - Software that asserts XMI RESET L by writing to IPR55, IORESET, with an MTPR instruction. For example, the console INITIALIZE command generates a system reset if no argument is given by using this mechanism.
  - The XTC power sequencer asserts the XMI RESET L line when the control panel Restart button is pushed.
- **Node Reset**—Any CPU can be "node reset" by setting its XBER<30> (NRST) bit. The console INITIALIZE command generates a node reset if a node ID argument is provided. The difference between the node reset and a system reset is that XMI AC LO L is not sequenced during a node reset.

## KA66A CPU Module

Typing CTRL/P at the console terminal or certain errors also cause initialization. Refer to Section 2.10 for a discussion of error handling. The *VAX 6000 Series Owner's Manual* discusses the operation of the system console.

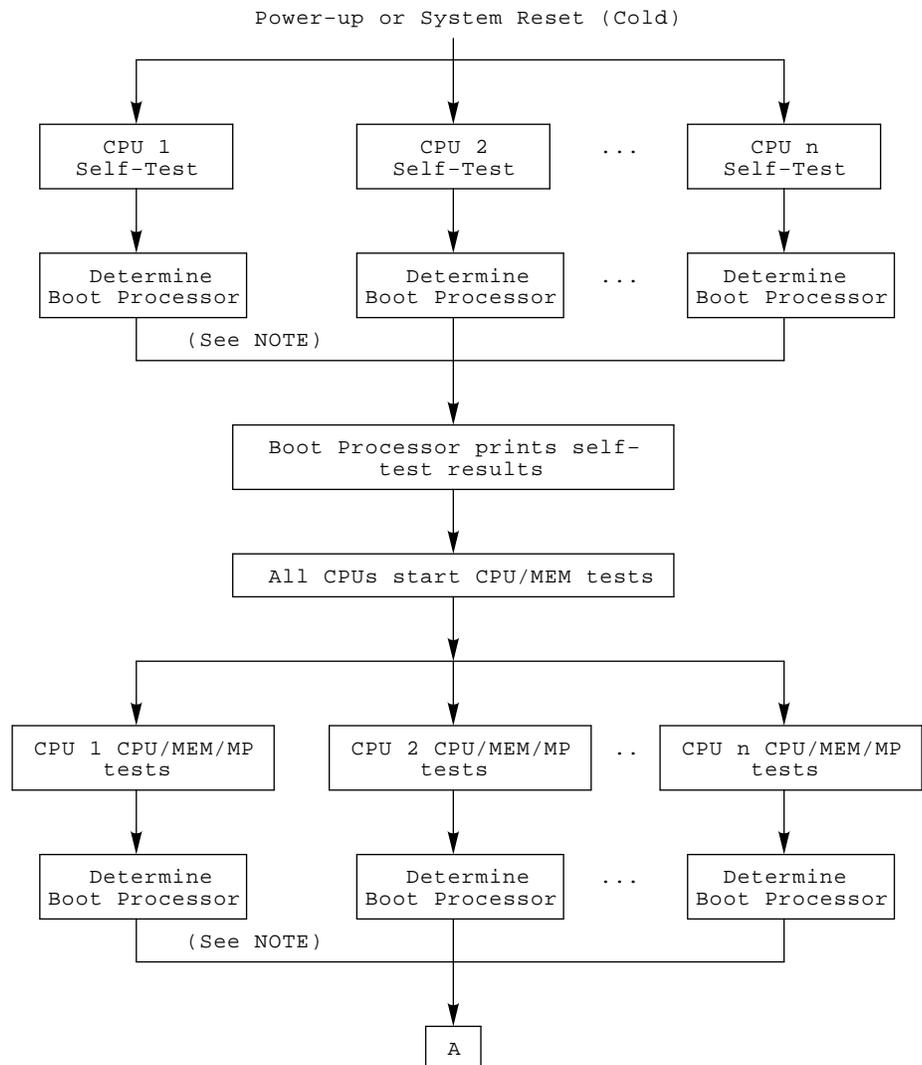
In response to a power-up or system reset, the KA66A CPU module(s) perform the following sequence:

- 1 Reset(s) to a known state. (Refer to the individual registers and their bits for their state on reset, after microcode self-test completes.)
- 2 All CPUs start executing the console program at E004 0000 in ROM. The console program initializes the registers and executes a complete ROM-based diagnostic (RBD) self-test and extended tests.
- 3 The operating system initialization code performs the final system initialization.

## 2.8.2 Detailed Initialization Description

The following is a flowchart and summary of the initialization process.

Figure 2–23 Initialization Flowchart

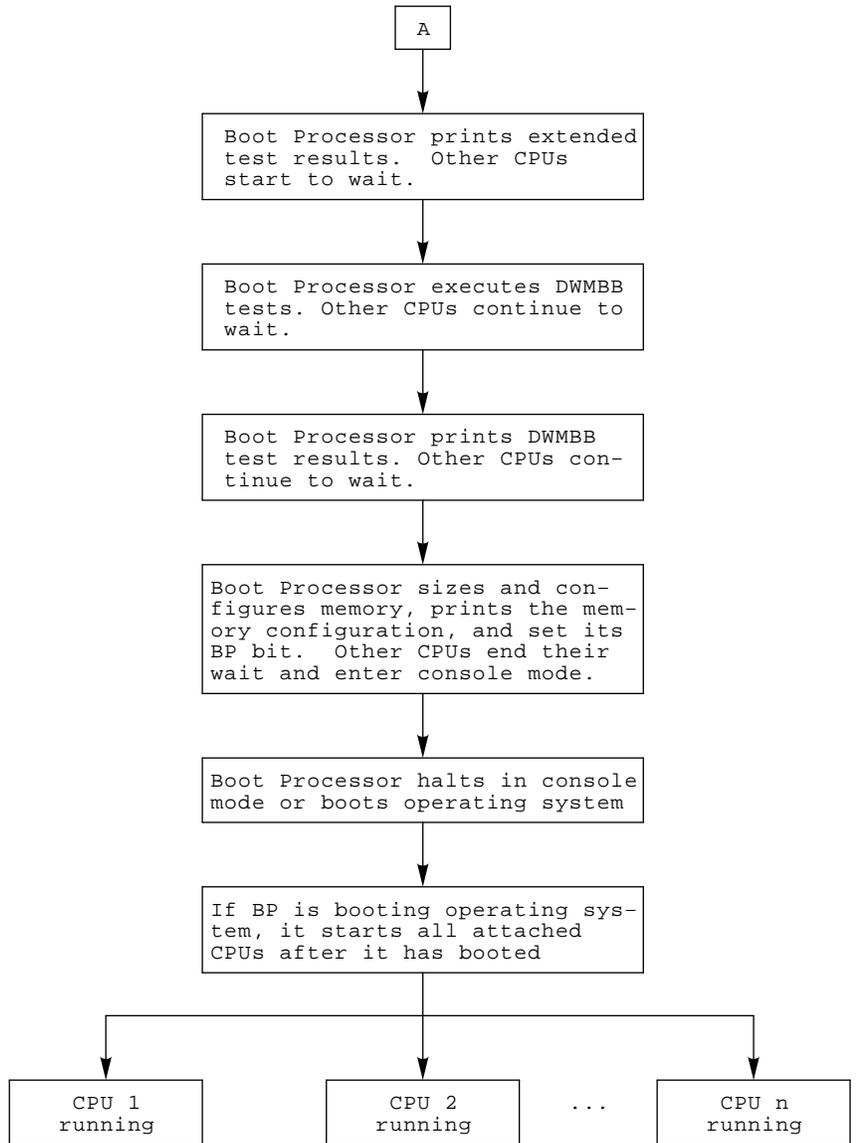


NOTE: The second determination of the Boot Processor occurs even if the original Boot Processor passes all memory tests.

msb-p352A-90

Figure 2–23 Cont'd on next page

Figure 2-23 (Cont.) Initialization Flowchart



msb-p353-90

The console program transfers control to the RBDs, which run a more extensive CPU self-test. After the RBDs complete, control is returned to the console program. Nodes do not access other nodes on the XMI during CPU self-test, limiting CPU self-test to intramodule operations and loopback transactions on the XMI to check their interface to the XMI.

If the CPU self-test is successful, the Self-Test Passed (STP) LED is lit, XBER<STF> is cleared, and the XMI BAD L signal is cleared.

After CPU self-test completes, a boot processor (BP) is selected from those CPU nodes that passed self-test. This is the first of two BP selections before the operating system starts. The BP prints the results of the self-test.

The BP then controls an additional test that requires all CPU nodes to access memory. This test is called the CPU/MEM/MP test and allows CPU nodes to check additional logic that could not be tested during the CPU self-test.

Each processor extinguishes its STP LED when the CPU/MEM tests are initiated. During the CPU/MEM tests, the CPU nodes continue testing themselves and access preallocated blocks in memory. These tests verify CPU logic that requires memory for testing.

The KA66A multiprocessing diagnostic is run automatically on power-up following the execution of CPU/memory interaction tests. The boot processor loads the multiprocessing test code from ROM into main memory at address 200(x) and then instructs all processors that passed CPU/MEM tests to jump to address 200(x) to begin testing. Testing verifies that CPUs can interrupt each other, perform appropriate invalidates, arbitrate for the bus, and handle memory locks correctly.

If these tests complete successfully, each CPU node lights its STP LED and clears its XBER<ETF> bit.

The second BP selection now occurs. If the original boot processor completes all of its CPU tests successfully, it remains the BP. Otherwise, another processor node is chosen to be the BP.

The BP then tests all the DWMBAs and DWMBBs (DWMBx) in the system. If the DWMBx tests are successful, the BP lights the DWMBx/A's yellow LED and the DWMBx/B's yellow LED.

Finally, the BP configures the memory nodes with correct interleave and address parameters, and initializes the console communications area (CCA).

During a warm restart ST0 and DWMBB tests are performed.

On node reset, the CPU initializes its state and runs self-test. The console program performs an additional CPU initialization before the operating system starts the processor running.

When the console initialization is completed, the boot processor either restarts the operating system (if a warm restart), boots the operating system, or halts in console mode. The secondary processors enter console mode and wait for the operating system to issue console commands that start them running.

---

### 2.8.2.1 NVAX CPU Hardware/Microcode Initialization

The NVAX chip initializes to the following state on power-up or the assertion of reset:

- 1 The VIC, P-cache, and B-cache are disabled.
- 2 The register log (RLOG) used in scoreboarding is cleared.
- 3 The Fbox is disabled.
- 4 The microstack is cleared.
- 5 The Mbox and Cbox are reset, and all previous operations are flushed.
- 6 The Fbox is reset.
- 7 The Ibox is stopped; it waits for a LOAD PC.
- 8 All instruction and operand queues are flushed.
- 9 The Ebox registers are cleared.
- 10 A power-up microtrap that starts the Ebox is initiated.

The NVAX chip microcode then does the following:

- 1 Hardware interrupt requests are cleared.
- 2 ICCS<6> is set to 0.
- 3 SISR<15:1> is set to 0.
- 4 ASTLVL is set to 4.
- 5 The Mbox PAMODE IPR is set to 30-bit physical address mode.
- 6 CPUID is set to 0.
- 7 The BPCR branch history algorithm is reset to the default value.
- 8 The backup PC is retrieved from the Ibox and saved in SAVPC.
- 9 PME is cleared.
- 10 The current PSL, halt code, and value of MAPEN are saved in SAVPSL.
- 11 MAPEN is cleared (memory management is disabled).
- 12 All state flags are cleared.
- 13 PSL is loaded with 041F 0000.
- 14 PC is loaded with E004 0000 (the address of the start of the console code).

---

### 2.8.2.2 Console Initialization

The console macrocode has the job of filling the gap between the initialized state described above and the initial state needed for the operating system. Entry to the initialization sequence starts at physical address E004 0000 (hex), the entry point of the console program. Housekeeping chores, such as establishing a scratch area and stack happen first.

The console program next turns on the Self-Test Passed (STP) LED, the first visible indication that the console program has begun executing. The console program then determines the type of reset as more initialization is performed for power-up starts than for warm starts.

Significant events of a power-up initialization include the following:

- 1** CPUID to the correct value from the system environment.
- 2** ECR (Ebox Control Register) as follows:
  - a.** Fbox Enable is set to enable the Fbox.
  - b.** Timeout External is cleared.
  - c.** FBOX ST4 Bypass Enable is set to enable Fbox stage 4 bypass.
  - d.** Write one to Timeout Occurred to clear any error.
  - e.** Timeout Test is cleared.
  - f.** ICCS EXT is set.
- 3** ICSR (Ibox Control Status Register) as follows:
  - a.** Set VIC Enable to leave the VIC enabled.
  - b.** Write one to Lock to clear any error.
- 4** Clear the PAMODE register MODE bit putting the system into 30-bit mode.
- 5** Write one to clear the Lock bit in TBSTS (TB Parity Status Register).
- 6** Initialize PCSTS (P-Cache Status) Register:
  - a.** Write one to clear the Lock bit.
  - b.** Write one to clear PTE ER WR.
  - c.** Write one to clear PTE ER.
- 7** Set CCTL (Cbox Control) as follows:
  - a.** Clear B-Cache Enable to disable the B-cache.
  - b.** Set B-Cache Size to 2 Mbytes (11 binary).
  - c.** Set <3:2> to a default setting of 01 (binary).
  - d.** Set <1> to a default setting of 1 (binary).
  - e.** Clear Force Hit.
  - f.** Clear Disable ECC Errors.
  - g.** Clear SW ECC.
  - h.** Clear Timeout Test.
  - i.** Clear Disable Pack to allow the write packing feature.
  - j.** Clear SW ETM.
  - k.** Write one to clear HW ETM.

- 8 Clear the various Cbox error registers:
  - a. BCETSTS (Backup Cache Error Tag Status): Write one to Lock, CORR, UNCORR, BAD ADDR, and LOST ERR to clear any errors.
  - b. BCEDSTS (Backup Cache Error Data Status): Write one to Lock, CORR, UNCORR, BAD ADDR, and LOST ERR to clear any errors.
  - c. CEFSTS (Cbox Error Fill Status): Write one to RDLK, Lock, Timeout, RDE, and LOST ERR to clear any errors.
  - d. NESTS (NDAL Error Status): Write one to NO ACK, BADWDATA, LOST OERR, PERR, INCON PERR, and LOST PERR to clear any errors.

---

### 2.8.2.3 Unnecessary Explicit Initialization

There is no need to explicitly initialize the translation buffer since the NVAX microcode performs an internal TBIA on any MTPR to the MAPEN IPR.

There is no need to explicitly initialize the data portions of the VIC, P-cache, or B-cache as long as the tags are initialized with all valid bits clear.

---

### 2.8.2.4 Warm Start Initialization

Some of the cold start and initialization steps are inappropriate with a warm start because the contents of memory are preserved. The sequence is modified by:

- Running a minimal set of tests for the memory interface and DWMBB that do not disturb the memory configuration or contents.
- No memory configuration is performed. Instead, each CPU searches memory for the CCA. If a nonboot processor fails to find the CCA, it displays an error code in its LEDs and hangs. If the BP fails to find the CCA, it reconfigures memory as if a cold start had occurred.

---

### 2.8.2.5 Node Reset

A node reset is a modified full system reset since some testing and initialization is inappropriate when the remainder of the system continues to function. The modifications are:

- Self-test results are not displayed.
- Additional tests are not run. Instead, the XBER<ETF> and XBER<XBAD> bits are cleared. Additional test results are not printed.
- DWMBB self-test is not run. No initialization of the DWMBB is performed and, therefore, no test results are printed.
- No memory configuration is performed and no revision banner is printed.
- The XBER<NRST> has its initial value stored in the console scratch memory and is then cleared by self-test. This stored value is used by the console program to determine if the entry resulted from a node reset or a system reset.

---

**2.8.2.6 Boot Processor Determination**

Each processor examines all other CPU XBER<STF> (Self-Test Fail), XBER<ETF> (Extended Test Fail), and NSCSR<BPD> (Boot Processor Disable) bits. Any processor with these bits clear is a candidate for BP. The candidate with the lowest XMI node ID is expected to become the BP and set its NSCSR<BP>. All nonboot processors wait for the designated BP to set its NSCSR<BP> bit. If there is a failure, failure codes are displayed on each processor's LEDs.

If no processor is eligible to become BP (any combination of the NRSCR<BPD> bit being set or all CPUs failing self-test), the system appears totally unresponsive. The system operator can then intervene to designate one of the processors as BP by typing ">>n" on the console terminal. Processor *n* then becomes the BP. This method of selecting the BP does not change the state of the BP's STF or BPD bits.

---

**2.8.2.7 Memory Configuration**

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

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

If the boot processor (BP) is unable to find the minimum required memory, it displays an error code on the LEDs and hangs. The BP also sets its NSCSR<Boot Processor Disable> bit, causing the nonboot processors, if any, to determine a new BP. The new BP repeats the memory configuration attempt.

---

**2.8.2.7.1 Selection of Interleave**

The interleave is specified by parameters stored in the EEPROM. These parameters are set with the SET MEMORY command. There are three types of interleave:

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

The VAX 6000 Model 600 supports interleave sets of mixed densities. Interleave sets are built from like-sized memory modules or memory modules whose cumulative value is equal to the largest memory module in the set. For example, a 64-Mbyte memory module can be combined with another 64-Mbyte memory module to form a two-way interleave set. If another 64-Mbyte memory module is not present, two 32-Mbyte memory modules can be used to complete the interleave set.

Any array containing unrecoverable errors is not included in a default interleave set. Instead, it is configured as uninterleaved. The remaining arrays that would have formed the set are freed up for inclusion in another interleave set, if one can be formed.

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

If the EEPROM specifies NO interleave, the console configures arrays in order by node ID, with the lowest numbered array at the lowest physical address.

---

### 2.8.2.7.2 Memory Testing and the Bitmap

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

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

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

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

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

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

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

---

### 2.8.2.8 DWMBB Configuration

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

- 1 The BI Starting Address Register (bb+20) and the BI Ending Address Register (bb+24) are initialized to the starting and ending limits of XMI memory (under 512 Mbytes).
- 2 The BICSR (bb+04) has its BI Broke bit cleared.
- 3 The DMA-B Transmit Buffer is disabled under these conditions:
  - The DWMBB/A module Device Register shows less than 10 (decimal).
  - The system contains five or more XMI commanders and the DWMBB/B module Device Register shows a revision of less than 0A (hex).

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

---

### 2.8.2.9 DWMVA Configuration

The console program performs minimal initialization of the DWMVAs following self-test.

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

---

## 2.8.3 Bootstrapping or Restarting the Operating System

A VAX processor can be in one of these five major states:

- Powered off.
- Bootstrapping (which is attempting to load and start) the operating system. If the memory has lost power before bootstrapping starts, it is a "cold start." If the memory's contents are valid because of the battery backup option, it is a "warm start."
- Halted.
- Restarting a halted operating system.
- Running.

It is the console program that bootstraps a copy of the operating system from a tape, disk device, or CD server, and can attempt to restart an existing memory-resident copy of the halted operating system.

Only the boot processor (BP, also called the primary processor) can execute a BOOT command or perform the automatic bootstrap sequence. Nonboot processors (also called secondary processors) remain in console mode awaiting further commands.

Only the BP attempts a bootstrap following a power-up. If the control panel lower key switch is set to Auto Start, the BP restarts and attempts a bootstrap. Any nonboot processors are in a halt until the operating system starts the nonboot processors by passing START commands through the console communications area (CCA). The nonboot processors do not restart system software unless the control panel lower key switch is set to Auto Start.

---

### 2.8.3.1 Operating System Restart

The boot processor console program attempts to start/restart the operating system whenever one of the following events occurs:

- Power is restored to the processor. If the memory was kept valid by the optional battery backup, it is a warm start; otherwise, it is a cold start.
- A system reset occurs. This is treated as a cold start.
- The running processor halts due to an error halt. This is a restart. A CTRL/P from the console terminal is not considered an error halt.

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

A nonboot processor console program attempts a restart following an error halt only if the control panel lower key switch is in the Auto Start position. For all other halt conditions, the BP is responsible for restarting the nonboot processor.

Restart of the operating system is controlled by a memory data structure called the restart parameter block (RPB), constructed by the operating system. The RPB is a page-aligned structure. The console program's warm start code searches memory for an RPB and, if a valid RPB is found, restarts the operating system at an address stored in the RPB. Figure 2-24 shows the RPB format.

**Figure 2-24 Restart Parameter Block Format**

---

PHYSICAL ADDRESS OF RPB
PHYSICAL ADDRESS OF RESTART ROUTINE
CHECKSUM OF THE FIRST 31 LONGWORDS OF RESTART ROUTINE
SOFTWARE RESTART IN PROGRESS FLAG BIT<0>

msb-p354-90

---

The algorithm used to locate the RPB is:

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

If all the above tests pass, a valid RPB has been found.

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

---

### 2.8.3.2 Failing Restart

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

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

The CCA\$Q\_SECSTART bits are set by the operating system when it attempts to start a nonboot processor. The operating system clears these bits when it is satisfied that the nonboot processor has successfully started executing.

The following scenario, which is peculiar to multiprocessors, is prevented by the CCA\$Q\_SECSTART bits:

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

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

---

### 2.8.3.3 Restart Parameters

The console program transfers control to the restart address when a valid RPB is found. The console program then passes these restart parameters in the GPRs, as specified by the *VAX Architecture Reference Manual*:

GPR10 – Halt PC

GPR11 – Halt PSL

GPR12, Argument Pointer – Halt code

GPR14, Stack Pointer – Address of the RPB + 512

---

### 2.8.3.4 Operating System Bootstrap

The console program causes the BP to attempt to bootstrap the operating system whenever one of the following events occurs:

- The control panel is Enabled and the BOOT command is typed on the console terminal.
- A restart is attempted and fails.
- A power-up occurs when the lower key switch is in the Auto Start position.

Bootstrap attempts to load the primary system bootstrap program, VMB, into memory and begin its execution. VMB is loaded from the device specified by the BOOT command or from a default device recorded in the EEPROM. The VAX 6000 Model 600 uses a set of minimal device handler

routines, called boot primitives, to read VMB from the boot device, a technique called "bootblock" booting.

The console program saves the target device information in system control RAM as the first phase of bootstrap. The target device information is propagated to all CPUs in the system since the location of the BP can change after a system reset. This causes all processors, memories, and I/O adapters to perform self-test, with the memories being tested as fast as possible. When the console program is reentered, following the reset, it determines that there was an "expected entry," and continues with the bootstrap.

The second phase of bootstrap begins as the console program searches tables in the EEPROM and then the ROM to locate a boot primitive that matches the specified device as the first phase of bootstrap. If a suitable primitive is found, the target device information is saved in GPRs and control transfers to the primitive.

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

Once VMB is loaded, the console program or the boot block passes control to it. The boot primitive preserves the boot parameters stored in the GPRs.

A bootstrap can also be triggered by the operating system, via the CCA\$V\_REBOOT flag in the CCA. This bit is only recognized by the BP.

---

### 2.8.3.5

#### **Boot Algorithm**

The console maintains a "bootstrap in progress" flag, stored at CCA\$V\_BOOTIP. This "cold start" flag is used to prevent repeated attempts to automatically bootstrap a failed system.

This algorithm is used to perform the system bootstrap:

- 1 If this boot attempt is a result of a console BOOT command, skip to Step 3.
- 2 If the CCA\$V\_BOOTIP flag is set, the boot fails.
- 3 Set the CCA\$V\_BOOTIP flag.
- 4 Store the boot device and parameters in system control RAM on all processors in the system and force a system reset.
- 5 When the console program is reentered on the BP, resume the bootstrap at this point.
- 6 Starting at location zero, search for the first page-aligned block of 256 Kbytes of good memory. If such a block cannot be found, the boot fails. This search is performed by scanning the bitmap built during memory configuration.

- 7 Search the boot primitive tables in the EEPROM and ROM, in that order, for a primitive that matches the specified boot device. If none is found, the bootstrap fails.
- 8 Load the GPRs with the boot parameters in the primitive.
- 9 The console initializes the registers, as described in Section 2.8.2.2.
- 10 Transfer control to the boot primitive.
- 11 Transfer control to the memory image of VMB or the boot block at the address loaded in the SP.

If the bootstrap fails, the console program displays a message on the console terminal and then displays the console prompt. If the bootstrap succeeds, the operating system clears CCA\$V\_BOOTIP.

### 2.8.3.6

#### Boot Parameters

The console program loads parameters into the GPRs before passing control to VMB. These parameters describe the boot device and any bootstrap options that are to be used. Table 2–34 shows how the registers are used.

**Table 2–34 Boot Parameters Loaded into GPRs**

Register	Bits	Description
GPR0	<7:0>	VMB device type code, supplied by the boot primitive
GPR1	<7:4>	XMI node number of the boot adapter
	<3:0>	VAXBI node number (if necessary)
GPR2	<15:0>	Remote (HSC) node numbers, if the Boot/Node qualifier was specified
GPR3		Boot device unit number
GPR4		Reserved
GPR5		Software boot control flags
GPR6		Used by the boot primitive to pass information to the bootblock program
GRP7		Physical address of the CCA
GPR8		Reserved
GPR9		Reserved
GPR10		The halt PC
GPR11		The halt PSL
AP		The halt code
FP		Reserved
SP		Address of the 256-Kbyte block of good memory + 512

---

**2.8.3.7 Bootstrap Software Sequence**

To determine whether the system time is accurate, the following sequence needs to be followed. See Section 2.5.3 for details on the watch chip.

- 1** Read the Valid bit in CSR D to determine whether the contents of the watch chip are correct.
- 2** If Valid is read as one, read the Busy bit in CSR A. If Valid is read as zero, go to Step 5.
- 3** If Busy is zero, read the watch chip date and time registers and convert the time to the 32-bit format.
- 4** If Busy is one, wait until it is zero and then go to Step 3.
- 5** If Valid is read as zero, prompt the operator for the date and time. Then convert the date and time to the watch chip register formats and load the watch chip as follows:
  - Write one to CSR B, bit <7> (Off). Write bits <6:0> as shown in Figure 2–18 at the same time.
  - Initialize CSR A as shown in Figure 2–17.
  - Write the date and time in the appropriate registers.
  - Start the watch chip by writing zero to CSR B, bit <7> Off. Write bits <6:0> as shown in Figure 2–18 at the same time.

---

## 2.9 Interprocessor Communication Through the Console Program

**Each CPU of a multiprocessor system must communicate with the other CPUs and the operating system. This section describes the interprocessor communication for a VAX 6000 Model 600 system.**

The console program runs on each processor of a multiprocessor VAX 6000 Model 600 system. These copies of the console program must be able to communicate with each other and with the operating system.

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

The boot processor (BP) controls the console terminal and, therefore, most of the communication in the VAX 6000 Model 600. There is no communication between secondary (nonboot) processors.

---

### 2.9.1 Required Communications Paths

A processor can be in one of four communication states: a running BP, a BP in console mode, a running nonboot processor, or a nonboot processor in console mode. The following are the communication paths:

- 1 Running processor to running processor, independent of boot or nonboot.

The console program is not involved. The processors are supported by the communications mechanisms within the operating system. These paths are used even when the communication is related to the console program. For example, when the system time is modified, the new time must be stored in the time-of-year clock on each processor. The operating system uses its own method to examine or propagate this information.

A special case of communications on these paths involves the XDELTA system debugger when it is entered on a nonboot processor. The operating system is responsible for passing characters to and from the boot processor and, thus, to the console terminal.

- 2 Running boot processor console program to/from nonboot processor console program.

The operating system on the BP must send complete console commands to the nonboot console, such as to start or stop the nonboot processor. The nonboot console program must be able to send responses (human readable messages) to the operating system on the boot processor, such as when the nonboot processor encounters

an error halt. The nonboot processor can send these responses at any time.

The nonboot processor does not send commands to the boot processor, and the BP does not send responses to the nonboot processor.

### 3 Console mode BP to/from running nonboot processor.

Whenever the boot processor halts, the nonboot processors eventually wait for resources locked by the BP. The boot processor console supports receiving complete responses from the running nonboot processor. If the halted BP attempts to send a command other than a STOP to a running processor, that command will time out.

### 4 Boot processor console to/from nonboot processor console—two different types of communication.

In the first type, the boot console sends complete commands to the nonboot processor, allowing the BP console to update the copy of a parameter stored on each processor. An example of this type of communication is to synchronize the console terminal baud rate whenever it is changed on the BP. The nonboot consoles send complete responses to the BP console to report, for example, a processor halt. Since responses arrive complete, there are no interleaving messages on the console terminal.

The nonboot processor does not send commands, and the boot processor does not send responses.

In the second type, the "Z" command allows the boot processor to communicate with VAXBI devices and with other XMI nodes. The consoles support character-at-a-time communications to implement the "Z" command, which transfers characters to and from another node so that the other node appears to be directly connected to the console terminal. The boot processor sends single characters of a command to the nonboot processor. The receiving nonboot processor performs all the processing of the input characters, including echoing and line editing. The nonboot processor sends single characters of a response to the BP for immediate display on the console terminal.

## 2.9.2 Console Communications Area

The console communications area (CCA) is the shared data structure in high physical memory used for communications between console programs. It consists of a one-page header followed by a variable number of pages containing buffers for each node. The header contains status information that must be visible systemwide. The buffers, one pair for each XMI node, are used for passing messages between processors.

The CCA is initialized by the boot (primary) processor at system reset. It is allocated beginning on a page boundary from the highest addressed page of system memory that can be located by the boot processor. The header lies in the lowest addressed page of the CCA, followed by buffers.

The CCA is not initialized under any other console entry conditions (node reset or halts). The address of the CCA is obtained from the console state remaining in system support RAM.

Diagnostic tests that must test or reconfigure memory could overwrite the CCA. If this should happen, the diagnostic tests must observe the following conventions:

- The diagnostic tests can only be run from the BP.
- The diagnostic tests must force the nonboot processors to stop polling the CCA.
- The diagnostic tests must rebuild the CCA after completing testing.
- The nonboot processors must wait for a signal passed through the XGPR register before locating the new CCA.

The location of the CCA is passed to the operating system at bootstrap time through GPR7. During system initialization, each processor is triggered to search for the CCA. This search starts at the highest addressed memory that can be located by each processor and then works backward. If a processor cannot locate the CCA, it enters an endless loop and cannot participate in the system. The algorithm used by the console program to the existing CCA is as follows:

- 1 Next = highest memory address in the system + 1 - 512.
- 2 If next <> 0, then "Failed to find CCA," and Exit.
- 3 If the contents of (next + CCA\$L\_BASE) <> next, then goto Step 7.
- 4 If the contents of (next + CCA\$W\_IDENT) <> "CC," then goto Step 7.
- 5 Compute sum of bytes at (next) through the contents of (next + CCA\$B\_CHKSUM - 1) ignoring overflow.
- 6 If sum = the contents of (next + CCA\$B\_CHKSUM), then "Exit with CCA found at next."
- 7 Next = next - 512.
- 8 Goto Step 2.

The overall layout of the CCA is shown in Figure 2-25. The contents of the fields are described in Table 2-35.

Figure 2–25 CCA Layout

				Offset (hex)
CCA\$L_BASE				00
CCA\$W_IDENT		CCA\$W_SIZE		04
CCA\$B_REV	CCA\$B_HFLAG	CCA\$B_CHKSUM	CCA\$B_NPROC	08
CCA\$Q_READY				0C
CCA\$Q_CONSOLE				14
CCA\$Q_ENABLED				1C
CCA\$L_BITMAP_SZ				24
CCA\$L_BITMAP				28
CCA\$L_BITMAP_CKSUM				2C
CCA\$R_RESERVED0			CCA\$B_TK_NODE	30
CCA\$Q_SECSTART				34
CCA\$Q_RESTARTIP				3C
CCA\$W_SSN_EXTENSION		CCA\$B_POWER	CCA\$B_PRIMARY	44
CCA\$L_RESERVED1				48
CCA\$L_RESERVED2				4C
CCA\$Q_USER_HALTED				50
CCA\$Q_SERIALNUM				58
CCA\$Q_HW_REVISION				60
CCA\$Q_RESERVED3				E0
CCA\$Q_RESERVED4				E8
CCA\$L_RESERVED5				F0
CCA\$L_CONSOLE_XGPR				130
CCA\$L_ENTRY_XGPR				170

msb-p568-91

# KA66A CPU Module

**Table 2-35 CCA Fields**

Field	Description
CCA\$L_BASE	Physical address of the base of the CCA.
CCA\$W_IDENT	The ASCII characters "CC".
CCA\$W_SIZE	The size, in bytes, of the CCA.
CCA\$B_REVISION	The revision number for the CCA.
CCA\$B_HFLAG	Systemwide status flags: <div style="text-align: center;"> <p style="text-align: center;">msb-p567-91</p> </div>
CCA\$V_DISABLE_MSG_2NDARY	This bit is used by the console when the primary is broadcasting the BOOT command to the secondaries and cannot process messages from the secondaries. The bit signals to the secondaries not to issue any console messages.
CCA\$V_REPROMPT	This bit is used by the console program to support the SET CPU command.
CCA\$V_REBOOT	This bit is tested whenever the console is entered as a result of a CTRL/P or a node halt. If the bit is set, the operating system is requesting a reboot. The system is rebooted from the default boot device. The front panel lower key switch does not inhibit such a reboot. This bit is ignored if the key switch is in the Secure position.
CCA\$V_BOOTIP	When set, a bootstrap is being attempted. This prevents repeated attempts to bootstrap after a failure.
CCA\$B_CHKSUM	Checksum of the first CCA\$B_CHKSUM-1 bytes of the CCA. Computed by doing signed, byte addition, ignoring any overflow.
CCA\$B_NPROC	The number of processors supported by the CCA. The normal value is 16. The maximum value is 64, limited by the bitmasks in the other fields.
CCA\$Q_READY	A bitmask of the processors that have data posted in their transmit buffer for processing by the boot processor. This field allows the operating system to use a Find First Set (FFS) instruction to locate any pending messages. The bits and nodes are numbered, starting with zero.
CCA\$Q_CONSOLE	A bitmask indicating the processors known to be in console mode. The appropriate bit is set and cleared by each processor as it enters and leaves console mode.

Table 2–35 (Cont.) CCA Fields

Field	Description
CCA\$Q_ENABLED	A bitmask indicating which processors are enabled to leave console mode. A processor sets or clears its bit during console initialization, based on a bit stored in the EEPROM. The EEPROM bit is set with the SET CPU command.
CCA\$L_BITMAP_SZ	The size, in bytes, of the physical memory bitmap. The bitmap is always an even number of longwords in length.
CCA\$L_BITMAP	The physical address of the physical memory bitmap. The bitmap contains one bit for each page of physical memory present on the system. The bit is clear if the page contains a hard error or if the page is in use by the bitmap or CCA. The bitmap is always page aligned.
CCA\$L_BITMAP_CKSUM	Reserved; not used.
CCA\$R_RESERVED0	Reserved; not used.
CCA\$B_TK_NODE	This field is used to pass to the operating system the XMI (in bits <7:4>) and VAXBI (in bits <3:0>) node numbers of the adapter that controls the TK tape drive. This field is set initially from a value stored in the EEPROM. If the initially specified node does not contain a TK tape drive adapter, the console program searches each VAXBI for a suitable adapter. The search starts with the highest XMI and from the highest VAXBI node ID on each VAXBI. The field sets to the location of the adapter, or zero if no adapter is found.
CCA\$Q_SECSTART	A bitmask indicating which processors are currently being started by the boot processor. The console program uses this information to avoid repeatedly forcing a bootstrap. This field is set and cleared by the operating system.
CCA\$Q_RESTARTIP	A bitmask indicating which processors are currently attempting restarts. Multiple flags are maintained to allow simultaneous error restarts to be performed. The operating system clears these fields if restart or boot succeeds.
CCA\$W_SSN_EXTENSION	The highest two bytes of the serial number.
CCA\$B_POWER	An ASCII character representing the type of power system.
CCA\$B_PRIMARY	The XMI node number of the primary processor. The console selects a primary or boot processor. If a reboot occurs, the console references these bits to determine which processor is the primary.
CCA\$L_RESERVED1	Reserved; not used.
CCA\$L_RESERVED2	Reserved; not used.
CCA\$Q_USER_HALTED	A bitmask indicating which processors entered console mode as a result of user intervention (CTRL/P or STOP command). This information allows the operating system to make decisions about timeouts in a symmetric multiprocessing configuration.
CCA\$Q_SERIALNUM	The system serial number. This field contains the least significant eight characters of the serial number string stored in the EEPROM.
CCA\$Q_HW_REVISION	Consists of a 16-quadword array containing compatibility and module revision information for the processors. Module revisions are an ASCII string. The quadword is zero for nonprocessor nodes. The layout of each quadword is:

Compat	MUST BE ZERO
Module Revision	

**Table 2–35 (Cont.) CCA Fields**

Field	Description								
	<p>The layout of the Compat field is:</p> <div style="text-align: center;"> <table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">7</td> <td style="text-align: center;">4</td> <td style="text-align: center;">3</td> <td style="text-align: center;">0</td> </tr> <tr> <td colspan="2" style="border: 1px solid black; text-align: center;">MBZ</td> <td colspan="2" style="border: 1px solid black;"></td> </tr> </table> <p style="margin-left: 100px;">└── COM_GRP</p> </div>	7	4	3	0	MBZ			
7	4	3	0						
MBZ									
COM_GRP	Compatibility Group. This binary field is used by the operating system to determine if all processors in the system are hardware compatible. Any processors not in the same group as the boot processor are not started.								
Module Revision	A four-character ASCII representation of the module revision. If the revision is only a single alphabetic character, the string begins with a leading blank. The alphabetic part of the revision is used to set the revision field of the CPU's XDEV<DREV> field. This field is set based on values stored in the EEPROM. If the EEPROM is unusable, the Module Revision is zero and the chip revision is set to 0 (hex).								
CCA\$Q_RESERVED3	Reserved; not used.								
CCA\$Q_RESERVED4	Reserved; not used.								
CCA\$L_RESERVED5	Reserved; not used.								
CCA\$L_CONSOLE_XGPR	An array of 16 longwords containing the XGPR value used while the console is operating.								
CCA\$L_ENTRY_XGPR	The XGPR value used by the system software before console entry.								

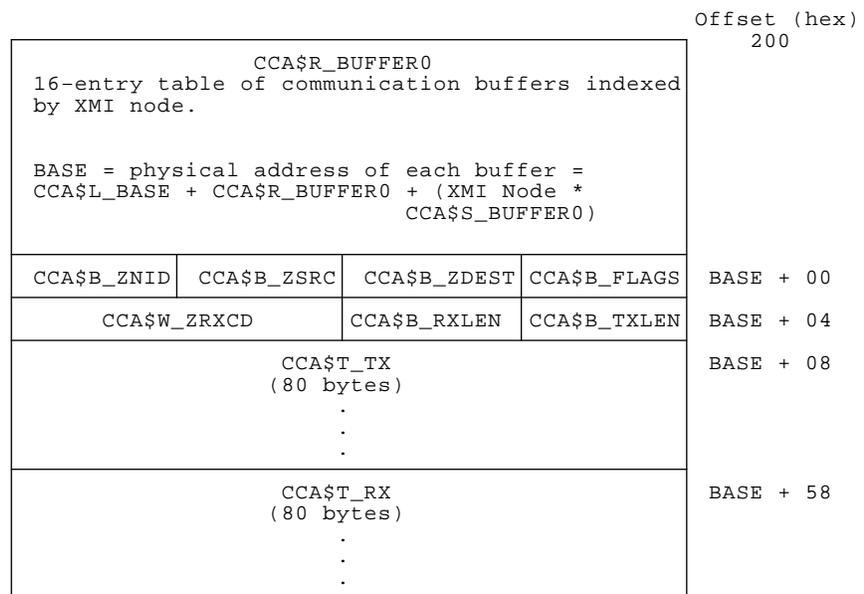
The CCA contains a buffer area for each possible XMI node. Each buffer area contains fields to support both message-oriented and character-at-a-time communications.

The address of the buffer area for XMI node *n* is given by:

$$\text{Buffer}_n = \text{Base address of CCA} + 512 + (n * 168)$$

The layout of the buffer area is shown in Figure 2–26, and the buffer fields are described in Table 2–36.

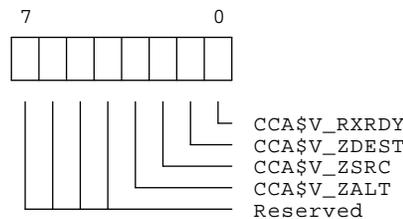
**Figure 2–26 Layout of XMI Node Buffers**



msb-p356-90

**Table 2–36 Buffer Fields**

Field	Description
CCA\$R_BUFFER0	A 16-entry table of communication buffers indexed by the XMI node. See Figure 2–26 for more information.
CCA\$B_ZNID	If CCA\$B_ZNID is non-zero, this field contains the XMI node number of the originator of the Z connection.
CCA\$B_ZSRC	If CCA\$B_ZSRC is non-zero, this field contains the XMI node number of the node transmitting "Z" command data to this node.
CCA\$B_ZDEST	When CCA\$V_ZDEST is non-zero, this field contains the XMI node number of the node receiving the "Z" command data that this node is sending. If the low four bits of this field identify a node that is a DWMBB, the high order four bits contain the destination VAXBI node number.
CCA\$B_FLAGS	Status flags:



msb-p388-91

**Table 2–36 (Cont.) Buffer Fields**

Field	Description
CCA\$V_ZALT	When set, the target of the current "Z" command cannot communicate through the CCA. The target is either a non-processor XMI node or a VAXBI node and must be accessed using alternate RXCD protocol, as described in the <i>VAXBI System Reference Manual</i> .
CCA\$V_ZSRC	When set, this node is receiving "Z" command data from the node listed in CCA\$B_ZSRC. This bit is always set or cleared by the node originating the "Z" command.
CCA\$V_ZDEST	When set, this node is sending "Z" command data to the node listed in CCA\$B_ZDEST.
CCA\$V_RXRDY	When set, there is a complete message in the CCA\$T_RX buffer. The equivalent bit for CCA\$T_TX is in CCA\$Q_READY of the CCA header.
CCA\$W_ZRXCD	This field is used for character-at-a-time communication in the same manner as a VAXBI RXCD Register. The layout is: <div style="text-align: center; margin: 10px 0;"> </div>
CCA\$V_ZRDY	When this bit is set, there is valid data in the other CCA\$W_ZRXCD fields.
CCA\$V_ZNODE	When CCA\$V_ZRDY is set, this four-bit field contains the XMI node number of the node that transmitted the data in CCA\$B_ZDATA.
CCA\$B_ZDATA	When CCA\$V_ZRDY is set, this field contains one byte of "Z" command data being sent to this node.
CCA\$B_RXLEN	If CCA\$V_RXRDY is set in CCA\$B_FLAGS, then this field contains the length, in bytes, of the message in CCA\$T_RX.
CCA\$B_TXLEN	If the bit corresponding to this node is set in CCA\$Q_READY, then this field contains the length, in bytes, of the message in CCA\$T_TX.
CCA\$T_TX	This buffer is used by the node to transmit a response to the BP. Only response data is passed through this buffer since a nonboot processor does not send commands to the boot processor.
CCA\$T_RX	This buffer is used by the node to receive a command from the boot processor. Only command data is passed through this buffer since a nonboot processor does not receive responses from the BP. Commands must end with a carriage return.

### 2.9.3 Sending a Message to Another Processor

The following two examples show how the CCA is manipulated when a complete message is sent between two processors.

For the first example, the boot processor, located at XMI node 1, sends a START command to the nonboot processor, located at XMI node 4.

- 1 Node 1 examines the CCA\$V\_RXRDY bit in the CCA buffer area for node 4. If the bit is clear, then go to Step 3.
- 2 Node 1 polls the bit until it clears or until a timeout of 12 seconds is reached. If a timeout occurs, an error is reported.
- 3 Node 1 moves the text of the START command into the CCA\$T\_RX buffer for node 4.
- 4 Node 1 sets the length of the command into the CCA\$B\_RXLEN field for node 4.
- 5 Node 1 sets the CCA\$V\_RXRDY bit for node 4 to indicate that a command is waiting.
- 6 Whenever node 4 enters its main console loop, it will eventually check for commands to execute. It will examine its local command buffer and then check its CCA\$V\_RXRDY bit for a command from another node.
- 7 Node 4 will now process the command contained in its CCA\$T\_RX buffer.
- 8 After reading the command, node 4 then clears its CCA\$V\_RXRDY bit, indicating that the buffer is again available.

For the second example, the nonboot processor, which is located at XMI node 4, halts, enters console mode, and sends a "halted" message to the boot processor, located at XMI node 1.

- 1 Node 4 examines bit 4 of the CCA\$Q\_READY field. If the bit is clear, then go to Step 3.
- 2 Node 4 polls this bit until it clears.
- 3 Node 4 moves the text of its response into its CCA\$T\_TX buffer.
- 4 Node 4 sets the length of the response in its CCA\$B\_TXLEN field.
- 5 Node 4 sets bit 4 in CCA\$Q\_READY to indicate that a response is waiting.
- 6 Node 4 issues an IVINTR interrupt to node 1. If node 1 is running, this alerts the operating system that a response is waiting. Node 4 polls CCA\$Q\_READY until bit 4 clears, preventing the nonboot processor from performing any action that might cause the response to be lost before the BP can display it.
- 7 If node 1 is running, it responds to the IVINTR and eventually checks for console responses, using an FFS instruction to check CCA\$Q\_READY. If node 1 was in console mode, it would be polling CCA\$Q\_READY and discover bit 4 set.
- 8 Node 1 (either the operating system or the console program) processes the response from the CCA\$T\_TX buffer for node 4. If the console program is running, it displays the response on the console terminal.
- 9 Node 1 clears bit 4 in CCA\$Q\_READY, indicating that the buffer is again available.

## 2.10 Error Handling

**This section describes the system-specific error exceptions and interrupts. It is organized with respect to the SCB vectors through which the event is dispatched. The SCB layout and SCB vector formats, as well as the exceptions and interrupts that result from normal system operation, are described in Section 2.2.5 and Section 2.2.6.**

Table 2–37 describes the levels of hardware-detected errors by level of severity. Table 2–38 lists the internally generated system control block (SCB) entry points. The complete list of supported SCB vectors is in Section 2.2.6. Table 2–39 lists the categories of errors, organized by entry point.

Refer to these sections:

- Section 2.2.6 for an explanation of the SCB.
- Section 2.2.5 for an explanation of exceptions and interrupts.

**Table 2–37 Hardware-Detected Errors**

Error	Description
Console halt	A halt to console mode is caused by one of the errors listed in Section 2.10.5. For some halt conditions, the console prompts for a command and waits for operator input. For other halt conditions, the console attempts a system restart or a system bootstrap.
Machine check	A hardware error occurred synchronously with the execution of instructions. The error can only occur for the instruction currently executing in the Ebox. Instruction-level recovery and retry may be possible.
Kernel stack not valid	During exception processing, a memory management exception occurred while trying to push information on the kernel stack.
Power fail	The power supply asserted the power fail signal XMI AC LO L.
Soft error interrupt	A hardware error occurred that was not fatal to the process or system. System error software should be able to recover and continue.
Hard error interrupt	A hardware error occurred asynchronously with respect to the execution of instructions. In most cases, a hard error interrupt signifies a serious system error, where data is lost or state is corrupted, and instruction-level recovery may not be possible. However, there are some recoverable system errors that are flagged by a hard error interrupt.

**Table 2–38 NVAX Chip Internally Generated SCB Entry Points**

<b>Mnemonic</b>	<b>SCB Index (hex)</b>	<b>Description</b>
SCB_MACHCHK <sup>1</sup>	04	Machine check
SCB_KSNV <sup>1</sup>	08	Kernel stack not valid
SCB_PWRFL <sup>1</sup>	0C	Power fail
SCB_RESPRIV	10	Reserved/privileged instruction
SCB_XFC	14	Extended function call (XFC) instruction
SCB_RESOP	18	Reserved operand
SCB_RESADD	1C	Reserved addressing mode
SCB_ACV	20	Access control violation
SCB_TNV	24	Translation not valid
SCB_TP	28	Trace pending
SCB_BPT	2C	Breakpoint trace fault
SCB_ARITH	34	Arithmetic fault
SCB_CHMK	40	Change mode to kernel
SCB_CHME	44	Change mode to executive
SCB_CHMS	48	Change mode to supervisor
SCB_CHMU	4C	Change mode to user
SCB_SMERR <sup>1</sup>	54	Soft error interrupt
SCB_HMERR <sup>1</sup>	60	Hard error interrupt
SCB_IPLSOFT	80 – BC	Software interrupt levels
SCB_INTTIM	C0	Interval timer interrupt
SCB_EMULATE	C8	Emulated instruction trap (PSL<FPD>=0)
SCB_EMULFPD	CC	Emulated instruction trap (PSL<FPD>=1)

<sup>1</sup>This section describes the entry-point vector in detail.

**Table 2–39 Error Summary Notification by Entry Point**

<b>SCB Index</b>	<b>Entry Point</b>	<b>Error Categories</b>
N/A	Console halt	Interrupt stack not valid, kernel-mode halt, double error halt, illegal SCB vector, node halt, system reset, initial power-up, HALT L assertion
04	Machine check	Memory management interrupt Microcode/CPU errors CPU stall timeout TB parity errors VIC tag or data parity errors B-cache uncorrectable data read errors Memory/NDAL read errors (NO ACK, timeout, or RDE from system)
08	Kernel stack not valid	

Table 2–39 (Cont.) Error Summary Notification by Entry Point

SCB Index	Entry Point	Error Categories
0C	Power fail	Power fail notification by XMI AC LO L
54	Soft error interrupt	VIC tag or data parity errors P-cache tag or data parity errors B-cache uncorrectable tag errors B-cache uncorrectable data read errors B-cache uncorrectable data errors in writebacks B-cache correctable tag and data errors Memory/NDAL read errors (NO ACK, timeout, or RDE on reads) NDAL parity errors NEXMI and XMI "soft" error notification by S ERR L
60	Hard error interrupt	B-cache uncorrectable data errors on write operations NDAL NO ACK on writes B-cache fill errors in NDAL ownership reads after merging write data in the cache data RAMs NEXMI and XMI "hard" error notification by H ERR L

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

System error handling can be logically divided into these steps:

- 1 State collection
- 2 Analysis
- 3 Recovery
- 4 Retry

### 2.10.1 Error State Collection

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

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

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

#### **Ibox**

ICSR: Ibox Control and Status Register

VMAR: VIC Memory Address Register

### **Ebox**

ECR: Ebox Control Register

### **Mbox**

TBSTS: TB Parity Status Register

TBADR: TB Parity Address Register

PCSTS: P-Cache Status Register

PCADR: P-Cache Parity Address Register

### **Cbox**

CCTL: Cbox Control Register

BCEDSTS: Backup Cache Error Data Status Register

BCEDIDX: Backup Cache Error Data Index Register

BCEDECC: Backup Cache Error Data ECC Register

BCETSTS: Backup Cache Error Tag Status Register

BCETIDX: Backup Cache Error Tag Index Register

BCETAG: Backup Cache Error Tag Register

CEFSTS: Cbox Error Fill Status Register

CEFADR: Cbox Error Fill Address Register

NESTS: NDAL Error Status Register

NEOADR: NDAL Error Output Address Register

NEOCMD: NDAL Error Output Command Register

NEICMD: NDAL Error Input Command Register

NEDATHI: NDAL Error Data High Register

NEDATLO: NDAL Error Data Low Register

### **NEXMI**

XBER: Bus Error Register

XBEER: Bus Error Extension Register

XFADR: Failing Address Register

XFAER: Failing Address Extension Register

NSCSR: Node-Specific Control and Status Register

WFADR0: Writeback 0 Failing Address Register

WFADR1: Writeback 1 Failing Address Register

NCSR: NDAL Control and Status Register

For the purposes of the rest of this section, it is assumed that each of these states is saved in a variable whose name is constructed by prepending "S\_" to the register name. For example, the ICSR would be saved in the variable S\_ICSR.

Memory allocation for each saved register should be .LONG, and in some cases registers may need to be saved twice since the state may change due to a more severe error.

Example 2-1 shows the collection of error state. Note the handling of error registers that might be overwritten in the event of a more severe error. For example, after a correctable B-cache data RAM error, BCEDIDX would hold the index of the correctable error. If an uncorrectable B-cache data RAM error occurs, BCEDIDX would be reloaded with the index of the more severe uncorrectable error. To ensure the data in BCEDIDX and BCEDECC matches the report in BCEDSTS, a conditional test is performed and these two registers are recaptured if both an uncorrectable

and correctable error are reported in BCEDSTS. Otherwise, BCEDIDX and BCEDECC could reflect a previous correctable error even though BCEDSTS reports a more severe error.

### Example 2-1 Error State Collection

---

```

;Save all error state upon entry to error handling routine
SAVE_STATE:
    MFPR    #PR19$_ICSR,S_ICSR           ; IBOX
    MFPR    #PR19$_VMAR,S_VMAR

    MFPR    #PR19$_ECR,S_ECR           ; EBOX

    MFPR    #PR19$_TBSTS,S_TBSTS       ; MBOX
    MFPR    #PR19$_TBADR,S_TBADR
    MFPR    #PR19$_PCSTS,S_PCSTS
    MFPR    #PR19$_PCADR,S_PCADR

    MFPR    #PR19$_CCTL,S_CCTL         ; CBOX
    MFPR    #PR19$_BCEDIDX,S_BCEDIDX
    MFPR    #PR19$_BCEDECC,S_BCEDECC
    MFPR    #PR19$_BCEDSTS,S_BCEDSTS
    BICL3   #^C<BCEDSTS$M_CORR ! BCEDSTS$M_LOCK>,S_BCEDSTS,R0
    CMLP    R0,#BCEDSTS$M_CORR ! BCEDSTS$M_LOCK
    BNEQ    10$
    MFPR    #PR19$_BCEDIDX,S_BCEDIDX
    MFPR    #PR19$_BCEDECC,S_BCEDECC

10$:
    MFPR    #PR19$_BCETIDX,S_BCETIDX
    MFPR    #PR19$_BCETAG,S_BCETAG
    MFPR    #PR19$_BCETSTS,S_BCETSTS
    BICL3   #^C<BCETSTS$M_CORR ! BCETSTS$M_LOCK>,S_BCETSTS,R0
    CMLP    R0,#BCETSTS$M_CORR ! BCETSTS$M_LOCK
    BNEQ    20$
    MFPR    #PR19$_BCETIDX,S_BCETIDX
    MFPR    #PR19$_BCETAG,S_BCETAG

20$:
    MFPR    #PR19$_CEFSTS,S_CEFSTS
    MFPR    #PR19$_CEFADR,S_CEFADR
    MFPR    #PR19$_NESTS,S_NESTS
    MFPR    #PR19$_NEOADR,S_NEOADR
    MFPR    #PR19$_NEOCMD,S_NEOCMD
    MFPR    #PR19$_NEICMD,S_NEICMD
    MFPR    #PR19$_NEDATHI,S_NEDATHI
    MFPR    #PR19$_NEDATLO,S_NEDATLO

; Collection of system environment error registers goes here

```

---

Flushing the B-cache can cause certain errors to occur. Therefore, it is recommended that the following state be collected:

#### From the Cbox

CCTL: Cbox Control Register

BCEDSTS: Backup Cache Error Data Status Register

BCEDIDX: Backup Cache Error Data Index Register

BCEDECC: Backup Cache Error Data ECC Register

BCETSTS: Backup Cache Error Tag Status Register

BCETIDX: Backup Cache Error Tag Index Register

BCETAG: Backup Cache Error Tag Register

NESTS: NDAL Error Status Register

NEOADR: NDAL Error Output Address Register

NEOCMD: NDAL Error Output Command Register

**From the System Environment**

All events that report an NVAX event sending a BADWDATA cycle on the NDAL are translated to an XMI TBDATA cycle, and causes the hexword block in the memory to be tagged as bad. Any subsequent read of that block will result in an XMI RER (which becomes an NDAL RDE).

For the purposes of the rest of this section, it is assumed that each of these states is saved in a variable whose name is constructed by prepending "SS\_" to the register name. For example, the BCEDSTS register would be saved in the variable SS\_BCEDSTS. (Note that some registers are saved immediately after the error occurred and again after the flush. These two saved states are distinguished by the prepended "S\_" in the first case and the prepended "SS\_" for the second.)

Memory allocation for each saved register should be .LONG.

Example 2–2 shows the collection of error state that would normally be performed during, and just after, flushing the B-cache.

**Example 2–2 Backup Cache Flushing and Error State Collection**


---

```

AFTER_BCFLUSH:
                                        ; CBOX
MFPR    #PR19$_CCTL,SS_CCTL
MFPR    #PR19$_BCEDIDX,SS_BCEDIDX
MFPR    #PR19$_BCEDECC,SS_BCEDECC
MFPR    #PR19$_BCEDSTS,SS_BCEDSTS
BICL3   #^C<BCEDSTS$M_CORR ! BCEDSTS$M_LOCK>,SS_BCEDSTS,R0
CMLPL   R0,#BCEDSTS$M_CORR ! BCEDSTS$M_LOCK
BNEQ    30$
MFPR    #PR19$_BCEDIDX,SS_BCEDIDX
MFPR    #PR19$_BCEDECC,SS_BCEDECC
;
30$:    MFPR    #PR19$_BCETIDX,SS_BCETIDX
MFPR    #PR19$_BCETAG,SS_BCETAG
MFPR    #PR19$_BCETSTS,SS_BCETSTS
BICL3   #^C<BCETSTS$M_CORR ! BCETSTS$M_LOCK>,SS_BCETSTS,R0
CMLPL   R0,#BCETSTS$M_CORR ! BCETSTS$M_LOCK
BNEQ    40$
MFPR    #PR19$_BCETIDX,SS_BCETIDX
MFPR    #PR19$_BCETAG,SS_BCETAG
;
40$:    MFPR    #PR19$_NESTS,SS_NESTS
MFPR    #PR19$_NEOADR,SS_NEOADR
MFPR    #PR19$_NEOCMD,SS_NEOCMD

; System environment:
; collection of system environment error registers affected by a BADWDATA
; cycle from NVAX goes here

```

---

**2.10.2 Error Analysis**

The error condition is analyzed with the error state obtained during the collection process. The purpose is to determine, if possible, what error event caused the error notification, and what other errors may also have occurred. See Section 2.10.6, Section 2.10.8, and Section 2.10.9 for guides

to analyze machine checks, hard error interrupts, and soft error interrupts, respectively.

**NOTE: Errors detected in or by one of the caches usually result in the cache being automatically disabled. To minimize the possibility of nested errors, error analysis and recovery for memory or cache-related errors should be performed with the P-cache disabled and the B-cache in ETM.**

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

In other cases one error event results in two related reports. For example, a B-cache uncorrectable data error during a writeback will be reported in NESTS as a BADWDATA event and as an uncorrectable data error in BCEDSTS. In this case, the BADWDATA event captures the full address of the data in error. Cases like this are handled as single error events.

In general, an error reporting register can report events that lead to machine checks, soft errors, or hard errors. A given error can result in either a machine check or a soft error interrupt, or both. Events that lead to hard error interrupts generally cannot also cause a machine check or soft error interrupt. Sometimes an error event that leads to a machine check or a soft error interrupt is closely related to an event that leads to a hard error interrupt (for example, a B-cache fill error on the first quadword of a fill for an OREAD done for a write causes a soft error interrupt, but the same error on a later quadword causes a hard error interrupt).

Analysis of simultaneous errors may be impossible. However, in cases where no single error register is used to report two errors, analysis of multiple errors is possible. Recovery from the set of errors is accomplished by recovering from all of them. For example, recovery from a P-cache tag parity error and a B-cache correctable data error which are reported together, is possible by following the recovery procedures for each error in sequence.

---

### 2.10.3 Error Recovery

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

Recovery from multiple errors is possible when analysis is possible (the errors are independantly reported), and when the recovery procedures are not in conflict. All recovery procedures in this section assume only one error is present. None of the procedures are valid in multiple error cases without further analysis. However, in cases where no conflict exists in the reporting of multiple errors (that is, no single error register is used to report two errors) and recovery from each error is possible, then recovery from the set of errors is accomplished by recovering from all of

them. For example, recovery from a P-cache tag parity error and a B-cache correctable data error being reported together is possible by following the recovery procedures for each error in sequence.

In some instances, it may be desirable to stop using the hardware that is the source of a large number of errors. Software should maintain error counts that can be compared against error thresholds on every error report. If the count (per unit time) exceeds the threshold, the hardware should be disabled.

**NOTE: Hard failures of one bit in the tag store can lead to unrecoverable errors requiring a full system crash. It would be appropriate to have an extremely low threshold for tag store correctable errors, especially if they recur in the same location or bit position.**

**NVAX use of the NDAL and XMI memory fetches is extremely high if the B-cache is disabled. In multiprocessor systems a CPU probably should be removed from the system rather than proceed with the B-cache off. In a single-processor system where the NDAL and memory are used a great deal, the performance of the I/O subsystem may decrease when the B-cache is off.**

### 2.10.3.1

#### Special Considerations when Memory Management Is Off

When memory management is turned off, a mispredicted Ibox prefetch can cause an erroneous interrupt. If the Ibox branch prediction unit chooses the wrong direction for a branch, and if the address of the mispredicted fetch operand points to a nonexistent physical memory location, several error bits will remain set in the NEXMI and Cbox error registers. There will be a pending hard error interrupt and soft error interrupt as well.

The XBER<TTO> and XBER<CNAK> bits will show that the address was nonexistent, and will force a hard error interrupt. The NEXMI will have sent back an RDE to the NVAX, and thus CEFSTS<RDE> and CEFSTS<Lock> will also be set. The Cbox will signal a soft error interrupt for this error. The following code example shows how the error can happen.

```

                                MOVL    #^X7FFF0000, R2
                                ...
                                CLRRL  R0
1$:    TSTL    R0
2$:    BEQL   4$
3$:    MOVL   (R2), R1
                                ...
4$:    MOVL   #^X1, R0
                                BRB    1$

```

In this example, R2 is first loaded with some value, then the program continues. This could be stale data from a previous code section, or it could be an incremented value that was legal until some final increment. Some time later, R0 is cleared and tested. The programmer might know that until R0 is non-zero, R2 will not be used. If R0 is equal to zero, the Ebox will execute the code starting at the label 4\$. In this example, R0 is obviously equal to zero, and the programmer might feel confident that the code at label 3\$ will not be executed at this time.

The Ibox, however, might guess that the branch will *not* be taken, and would then start to fetch the operand at label 3\$. If we assume that the address 7FFF 0000 is not in physical memory, the problem can be seen. Even though the Ebox would signify a branch mispredict, the Cbox would have sent out the nonexistent memory onto the XMI.

This error can only occur if the prefetched operand points to memory space, and only if the instruction causes a memory read. I/O space references are prevented from going ahead until the Ebox is actually executing the instruction that references them, and memory writes will not progress until the Ebox has the data to send.

When memory management is turned on, this error should not occur at all. The memory management software should prevent any virtual address from being mapped to a nonexistent physical address. A prefetch might cause an erroneous access violation, but that would be cleared out when the branch mispredict is sensed by the Ebox.

Anyone writing code with memory management off should check for this condition, and service the erroneous interrupts (clearing the error bits) whenever appropriate. For example, if the code is running at an elevated IPL with memory management off, the IPL should be dropped just before returning to the main line code. The hard error interrupt can then be taken, the footprint analyzed, and the error bits cleared. The soft error interrupt will be taken after returning from the hard error interrupt, and a flag can be used to signal that there should not be any error bits left.

In those cases where it is impossible to recognize and service the error, some methods are available to prevent the error from happening at all.

- There will be no erroneous prefetches in code that has no conditional branches. Therefore, to prevent the error, use only in-line code or unconditional branches during critical sequences.
- A prefetched address that is legal (not a nonexistent memory location) and is error-free (no uncorrectable errors) will not cause a problem. Therefore, a default value can be loaded in any register that might be taken as an erroneous branch. It is not important what data is in the register, only that it is an error-free memory location (or a safe I/O location).

---

### 2.10.3.2 Cache Coherence in Error Handling

To maintain cache coherence while enabling NVAX caches, certain procedures must be followed. Since many errors cause caches to be disabled, and since cache and memory error recovery is normally done with the P-cache and VIC (virtual instruction cache) off and the B-cache in ETM, the complete cache enable procedure is done as part of recovery from all cache and memory errors.

Once the B-cache is in ETM mode, it may not be coherent with memory if it is reenabled before being flushed. Therefore, a B-cache flush must be done before reenabling the B-cache after it has been in ETM.

While the B-cache is in ETM (or off), the P-cache will stay coherent with memory. However, before the B-cache is reenabled, the P-cache must be disabled. After the B-cache is reenabled, the P-cache must be flushed before it is reenabled.

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

The translation buffer is not automatically kept coherent with memory. Software uses the TBIS and TBIA functions to maintain coherence, and the LDPCTX instruction clears the process PTEs in the TB. Normally, in error recovery there is no need to flush the TB. For consistency and for the sake of beginning error retry in a known state, flushing the TB during error recovery is recommended. When a TB parity error occurs, Mbox hardware flushes the TB by itself (via an internally generated TBIA), but it would be appropriate for software to test the TB after a parity error.

The caches are flushed and enabled in a specific order. The ordering is necessary for coherence between the B-cache, P-cache, and memory. Disabling the caches should be done in the following order: first the B-cache, then the VIC, and finally the P-cache.

In error handling, the VIC and P-cache are disabled while the B-cache is placed in ETM. The B-cache flush from ETM procedure is done to turn off the B-cache altogether.

---

### 2.10.3.2.1 **Disabling and Flushing the Caches (Leaving the B-Cache in ETM)**

The order for disabling the NVAX caches (placing the B-cache in ETM) is:

- Disable the VIC (MTPR to ICSR)
- Disable the P-cache (MTPR to PCCTL)
- Put the B-cache in ETM (MTPR to CCTL)

The order for flushing the B-cache and disabling it is:

- Flush the B-cache (loop on MTPR to BCFLUSH IPRs)
- Clear the tags (loop on MTPR to BCTAG IPRs)
- Disable the B-cache (MTPR to clear ETM bits in CCTL)

Errors can occur as a result of flushing the B-cache. Before carrying out the procedure, BCEDSTS and BCETSTS should be clear of unrecoverable errors, and NESTS should be clear of unrecoverable outgoing errors. The MTPRs to BCFLUSH IPRs should be done one block at a time, checking the BCEDSTS and BCETSTS error registers after each block. If checking is not done, any unrecoverable error that occurs during the flush may become a lost unrecoverable error and the system will fail.

Errors that occur while flushing the B-cache are separate errors and should be handled independently of the initial error. However, certain errors may be expected during the flush procedure, based on the initial error. Also, the successful outcome of the B-cache flush procedure is important in determining whether to retry or restart the interrupted or machine checked instruction stream.

---

### 2.10.3.2.2 Enabling the Caches

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

- Disable the VIC, the P-cache, and the B-cache.
- Clear the B-cache (loop on MTPR to BCTAG IPRs).
- Enable the B-cache (MTPR to CCTL).
- Clear the P-cache (loop on MTPR to PCTAG IPRs).
- Enable the P-cache (MTPR to PCCTL).
- Flush the TB (TBIA).
- Clear the VIC (loop on MTPRs to VMAR and VTAG, writing an initial value).
- Enable the VIC (MTPR to ICSR).

---

### 2.10.3.3 Special Writeback Cache Recovery

Writeback caching can lead to special error cases. Some of them can be recovered.

---

#### 2.10.3.3.1 B-Cache Uncorrectable Error During Writeback

When a B-cache uncorrectable data RAM error occurs in a writeback, the status, cache index, and error syndrome are captured in BCEDSTS, BCEDIDX, and BCEDECC. As it is written back, the data is tagged-bad by the BADWDATA NDAL command. However, the address of the lost data is not captured in the B-cache error registers. But sending BADWDATA on the NDAL is treated as if it were an error, and the full address is captured in NEOADR while the status is captured in NESTS. The writeback can be held in the writeback queue for an indefinite amount of time. If a B-cache uncorrectable error on a writeback is detected, but NESTS does not show any outgoing error status, the writeback queue must be drained to continue the analysis and recovery. This is most easily accomplished by the following IPR write:

```
MFPR #PR19$_CWB,R0
```

S\_NESTS should be reloaded from NESTS after this operation. If S\_NESTS does not show the BADWDATA error status after draining the writeback queue, and it shows no other outgoing error, then there is a serious inconsistency and the system should be crashed.

---

#### 2.10.3.3.2 Memory State

Memory supports the writeback cache by maintaining some amount of state for each hexword (each cacheable block) in memory. An ownership bit, an interlock bit, and an owner ID is stored for each hexword.

It is always assumed that an Ownership Read command NO ACKed on the NDAL does *not* affect the ownership bit in memory. If the command was ACKed, and at least one data word was returned (even if it was an RDE), then the ownership bit is assumed to be *set* in memory. If the command

was ACKed but no data was returned, then a serious system error has occurred and the ownership bit is indeterminate.

---

### 2.10.3.3.2.1 Accessing Memory State

In recovering from certain errors it is necessary to read the state memory has stored with each hexword. This section assumes there is a routine, called Memory State, that returns this state, given a block address.

Memory State can have side effects. This routine could cause a read timeout error in the memory module and a corresponding machine check. Software must be prepared to handle this possibility. Before calling Memory State, software should confirm that all registers that may report expected errors are clear of errors. In an XMI system, CEFSTS is the register to check because a memory read timeout is the only error that is expected as a side effect of Memory State.

---

### 2.10.3.3.2.2 Repairing Memory State (Fill Errors)

In recovering from various B-cache fill errors, it is necessary to reset the ownership state in memory. This can be accomplished by sending a Write Disown (WDISOWN) command to that memory.

In cases where the fill error resulted from "lost"<sup>1</sup> data that cannot be recovered, the ownership bit may still be set in memory while no cache owns the block. If the data is private to one process, then the system may be able to continue operating after stopping that one job. The system-dependent procedure is then used to reset the ownership bit.

For certain B-cache fill errors, an attempt is made to reset the ownership bit in memory, while maintaining or restoring the correct data to the memory block.

All the data is in memory. One or more quadwords of data are also in the cache, and one quadword has been altered by the Cbox in processing a write to that block from the Mbox. Memory's ownership bit is set (meaning it "thinks" a cache owns the block). The owner ID stored with the block in memory indicates this CPU. The cache tag for the block does not indicate that the block is owned. In general, if no writes to this block time out, and the block is private to one process, then the repair can be done.

To recover from the first situation listed above, one of the correct quadwords in the B-cache is accessed and used in the XMI procedure for resetting memory's ownership bit. The side effect of this procedure is that the data extracted from the B-cache is written to memory. Given that the block is private to one process and no writes have timed out in memory, this data is still correct. (Note that software must somehow ensure that no writes to this block are pending in the memory before beginning the repair. This can be done by waiting an amount of time equal to an MS65A write timeout time.)

---

<sup>1</sup> In this case the more general sense of "lost" is implied. That is, memory's ownership bit is set, but no cache writes the data back when a read is done to that location. It may be possible to identify which CPU memory "thinks" it owns the data, but it is often not possible to determine which error caused this situation to arise.

To recover from the second situation listed above in an XMI system, the same procedure is followed, but the data written back is part of the altered quadword. The remainder of the altered quadword is written to the block after the repair.

---

#### 2.10.3.3.2.3 Repairing Memory State (Tagged-Bad Locations)

In recovering from B-cache uncorrectable data RAM errors on writebacks, it is necessary to reset the tagged-bad data state for a block in memory. In general, before clearing the tagged-bad data state of memory, software must first ensure that no more accesses to the block can occur. Otherwise there is a danger that some process on some other processor or a DMA I/O device will see incorrect data and not detect an error.

On the XMI, a sequence of operations involving writes to registers in a memory module followed by a write to the memory block in question is required. To do this, the B-cache should be off, because a KA66A CPU module will not issue a write to memory when the B-cache enable bit is set.

---

#### 2.10.3.3.3 Extracting Data from the B-Cache

Prior to extracting data from the B-cache, it must be flushed and disabled. The B-cache is then placed in force hit mode and the data is extracted.

**NOTE: The code that executes this procedure and its local data must be in I/O space. The TB entries (PTEs) that map this code and local data must be fixed in the TB. (This is most easily done by flushing the TB by an MTPR to TBIA and then accessing all the relevant pages in sequence.) Otherwise, B-cache Force Hit will interfere with instruction fetch, operand access, and PTE fetches in TB miss sequences.**

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

**NOTE: In force hit mode, B-cache data RAM ECC errors are detected (unless CCTL<Disable Errors> is set). Software should prepare for an ECC error (BCEDSTS unrecoverable error bits should be clear).**

---

#### 2.10.3.3.4 Address Determination Procedure for Recovery from Uncorrectable B-Cache Data RAM Errors

After an uncorrectable data RAM error in the B-cache, only the index of the block is stored, not the complete physical address. The procedure for constructing the physical address of the error is given here. It is assumed that the block has not been replaced.

To construct a physical address from the contents of S\_BCEDIDX and the tag indicated by that register, first read the tag with an MFPR from BCTAG IPRs. Check that the tag data and check bits are correct or correctable. Extract the address tag portion of the corrected result and combine with S\_BCEDIDX.

**NOTE: The above procedure is used in the event of a B-cache data RAM error. If it fails because the tag also has an uncorrectable error,**

then the error should be considered unrecoverable. However, the search procedure described in the next section could be used to obtain useful information for the error log (specifically, which blocks this CPU has marked owned in memory for this cache index).

---

#### 2.10.3.3.5 Special Address Determination Procedure for Recovery from Uncorrectable B-Cache Tag Store Errors

An uncorrectable tag store error in the B-cache can cause certain interesting error cases. In some of these cases data may be lost (the copy in the B-cache was overwritten). In other cases, the data is still good in the cache. In all cases, the address of the lost data is not directly known. A special procedure must be used to determine this address.

This section describes the generic address determination procedure for use in recovering from uncorrectable tag store errors. Specific error event descriptions in Section 2.10.6, Section 2.10.8, and Section 2.10.9 refer to this procedure for address determination. The possible outcomes of this procedure are as follows:

- The single address of a lost data block is found. Retry and recovery information for the error is found in the specific error event description that referred to this address determination procedure.
- No address is found. It can be assumed that no block was owned by the B-cache (or the error was transient). Retry and recovery information for the error is found in the specific error event description that referred to this address determination procedure.
- Multiple addresses are found. This is a multiple unrecoverable error situation, and the system should be crashed.

The procedure for determining the address of a lost data block follows. Note that this procedure assumes the relevant tag in the B-cache is not valid/owned. This procedure is for analyzing the result of errors in that tag.

This procedure assumes that Memory State will return the ownership state and the physical ID of the CPU which memory "thinks" owns the block. The B-cache should be in ETM. Search all memory block addresses whose index portion matches the index of the B-cache tag with the error. CHECK memory state for the block. If this CPU is the owner of that block, then the block is lost. Continue the search even if one lost block is found. Zero, one, or multiple lost blocks could be present.

**NOTE: This procedure is specific to recovering from tag store errors in one CPU. So when the memory state for a block indicates another cache in the system owns a particular block, that block is not counted as lost. That block may be "lost" in the more general sense (if the cache indicated as the owner no longer "knows" that it owns the block or is somehow unable to write it back). The purpose here is only to find blocks that are definitely lost as a result of errors involving this CPU.**

---

**2.10.3.4 Cache and TB Test Procedures**

Testing is generally done using the force hit mode of a cache. The code and data of the test procedure must reside in I/O space. Assuming memory management is enabled during this procedure, the needed PTEs must be in the TB before entering force hit mode in the P-cache or B-cache. For the B-cache, testing should be done with errors disabled. The ECC logic should be tested thoroughly on one location by forcing various check bit patterns and examining the syndrome latched on the read (BCEDECC is loaded on every read in B-cache disable-errors mode). P-cache and VIC parity checking should be tested by writing bad parity into the arrays. TB testing may be accomplished by writing to MTBTAG and MTBPTE (with care not to change any TB entry necessary for the test code and data and not to cause two TB entries to exist for one address). PROBER and PROBEW (setting PSL<PRV MOD>) are then used to verify the protection bits. Testing the modify bit would be difficult, although approaches exist.

---

**2.10.3.5 NEXMI Error Handling**

The NEXMI has the following error handling attributes:

- All XMI read transactions are reattempted until either:
  - 1 The command is acknowledged and completed successfully, or
  - 2 A transaction timeout (XBER<TTO>) or error condition (for example, RER) occurs
- When a CPU requests data from memory, it waits for the data to be returned. If the data is not delivered from the XMI, the NEXMI will eventually time out and return an RDE to the NDAL. XBER<TTO> will also be signaled in this case.
- All XMI write transactions are reattempted until successfully acknowledged or until a transaction timeout occurs, whichever happens first.
- The XMI interface maintains complete error status on a failed XMI transaction that was initiated by this node. The status includes the failed command, commander ID, address, and an error bit that indicates the type of error that occurred. The status remains locked until software resets the error bit(s).
- NEXMI errors are signaled by posting an interrupt with either the S ERR L (soft error) or H ERR L (hard error) lines on the NDAL. In general, an error that is accompanied by some positive action (such as an RDE being returned to the NVAX) will be reported as a soft error, while a serious error that has no other reporting mechanism will be reported as a hard error. Errors that are generally recoverable (such as a bus parity error) are always reported as a soft error.

Some errors (such as certain TTO errors) can be caused by either a read or a write, depending upon the nature of the command. Such errors are reported as hard errors. Software needs to determine how serious the error really is. In these cases, a "second error occurred" bit is available to help in this determination.

- The NEXMI provides parity generation and checking on the XMI and NDAL buses. All transactions except XMI invalidate on the XMI or NDAL containing parity errors are ignored by the NEXMI.

---

### 2.10.4 Error Retry

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

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

Strictly speaking, an REI from a hard or soft error interrupt handler is not a retry since these interrupts are recognized between macroinstructions. A machine check exception is an instruction abort, and an REI from the handler will cause the failing instruction to be retried.

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

It is expected that software handles machine checks, soft error interrupts, and hard error interrupts independently. For example, after handling a machine check from which retry is to occur, software does not check for errors that might cause a pending hard or soft error interrupt. The machine check handler is exited by REI (after trimming the machine check information off the stack). If the IPL of the machine checked instruction stream is low enough, any pending hard or soft error interrupt is taken before the retry occurs. However, if the interrupted instruction stream was running at a high IPL, the system will continue without dealing with the remaining errors.

---

#### 2.10.4.1 General Multiple Error Handling Philosophy

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

Multiple error scenarios can be grouped into the following classes:

- Class 1 errors are multiple distinct errors for which no error report interferes with the analysis of any other (for example, no lost error bits set).
- Class 2 errors are multiple errors, which could have been caused by the NVAX CPU pipeline issuing more than one reference to a given block before the error interrupt or machine check forced a pipeline flush.

- Class 3 errors are multiple errors for which analysis is complicated because the reports interfere with each other.

Class 1 errors should be treated as separate errors, each with its own recovery. Retry or restart evaluation is based on the cumulative result of the recovery and repair procedures for each error.

Specific cases of class 2 errors are identified in which lost errors are tolerated. These cases are selected because the NVAX pipeline can easily cause them (given one error), and because sufficient safeguards exist to ensure that correct operation is maintained. Section 2.10.4.2 lists these cases.

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

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

---

#### 2.10.4.2

#### Retry Special Cases

Some multiple error scenarios of class 2 are listed below. They are likely made by the NVAX pipeline's tendency to prefetch operands. The safeguard that exists in all cases is that errors inconsistent with correct operation after the error (such as lost data) will invariably cause a hard error interrupt or be detectable by the analysis accompanying the machine check or soft error interrupt.

- Lost B-cache data RAM uncorrectable ECC errors and addressing errors (BCEDSTS<LOST ERR>)
- Lost B-cache fill errors (timeouts and RDEs) (CEFSTS<LOST ERR>)
- Lost NDAL output errors (NO ACKs) (NESTS<LOST OERR>)

**NOTE: Retry from a machine check is done even when a hard error interrupt might be pending. If the machine checked I-stream were running at a high enough IPL, it would not be interrupted immediately. Typical hard errors are write errors that cannot cause a machine check. So the fact that a serious error is ignored in the machine check retry equation is not considered a problem. The other error would probably have occurred anyway, and it would not have interrupted the I-stream until the IPL was lowered.**

---

## 2.10.5 Console Halt and Halt Interrupt

A console halt is not an exception but is a transfer of control by the NVAX CPU microcode directly into the console program in the boot ROM at address E004 0000 (hex). Console halts are initiated at power-up, by certain microcode-detected double-error conditions, and by assertion of the external halt interrupt, HALT L.

A halt interrupt is generated by either of two conditions:

- CTRL/P is typed on the (unsecured) console terminal

- XBER<NHALT> (Node Halt) is asserted

No exception stack frame is associated with a console halt, but the SAVPC (IPR42) and SAVPSL (IPR43) provide the necessary information for continuation.

The PSL, halt code, MAPEN<0>, and a validity bit are saved in SAVPSL.

Table 2–11 lists and describes the console halt codes.

**NOTE: In certain error conditions detected during the execution of a string instruction, the state packup sequence leaves the FPD bit set in the SAVPSL register, but the SAVPC register pointing at the instruction following the string instruction, rather than at the string instruction itself. If the FPD bit is not set in the SAVPSL register, SAVPC is correct. As error halts are not normally restartable, this is not a problem. For a console halt due to the assertion of HALT L, which is the only normally restartable console halt, SAVPC is always correct, even if the halt interrupt was detected during the execution of a string instruction.**

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

To complete the hardware restart sequence and thereby pass control to the console macrocode, the CPU is initialized.

## 2.10.6 Machine Check Exception

A machine check exception indicates a serious system error that is sometimes recoverable by restarting the instruction.

The recoverability is a function of the:

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

A machine check results from an internally detected consistency error, such as the microcode reaches an "impossible" state, or from an externally detected hardware error, such as a memory parity error.

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

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

```
MTPR    #0, #PR$_MCESR      ; PR$_MCESR=38
```

The machine check stack frame is shown in Figure 2–11, and its parameters are described in Table 2–8. These parameters are parsed by the error handling macrocode to determine what caused the machine check.

Figure 2–27 contains the machine check parse tree, which indicates the causes of each machine check. For those machine checks that have multiple causes, the registers and bits that isolate the cause are listed. The sections following the parse tree provide a description of the machine check, the procedure to recover, and the conditions for restarting the operation.

Figure 2-27 Machine Check Exception Parse Tree

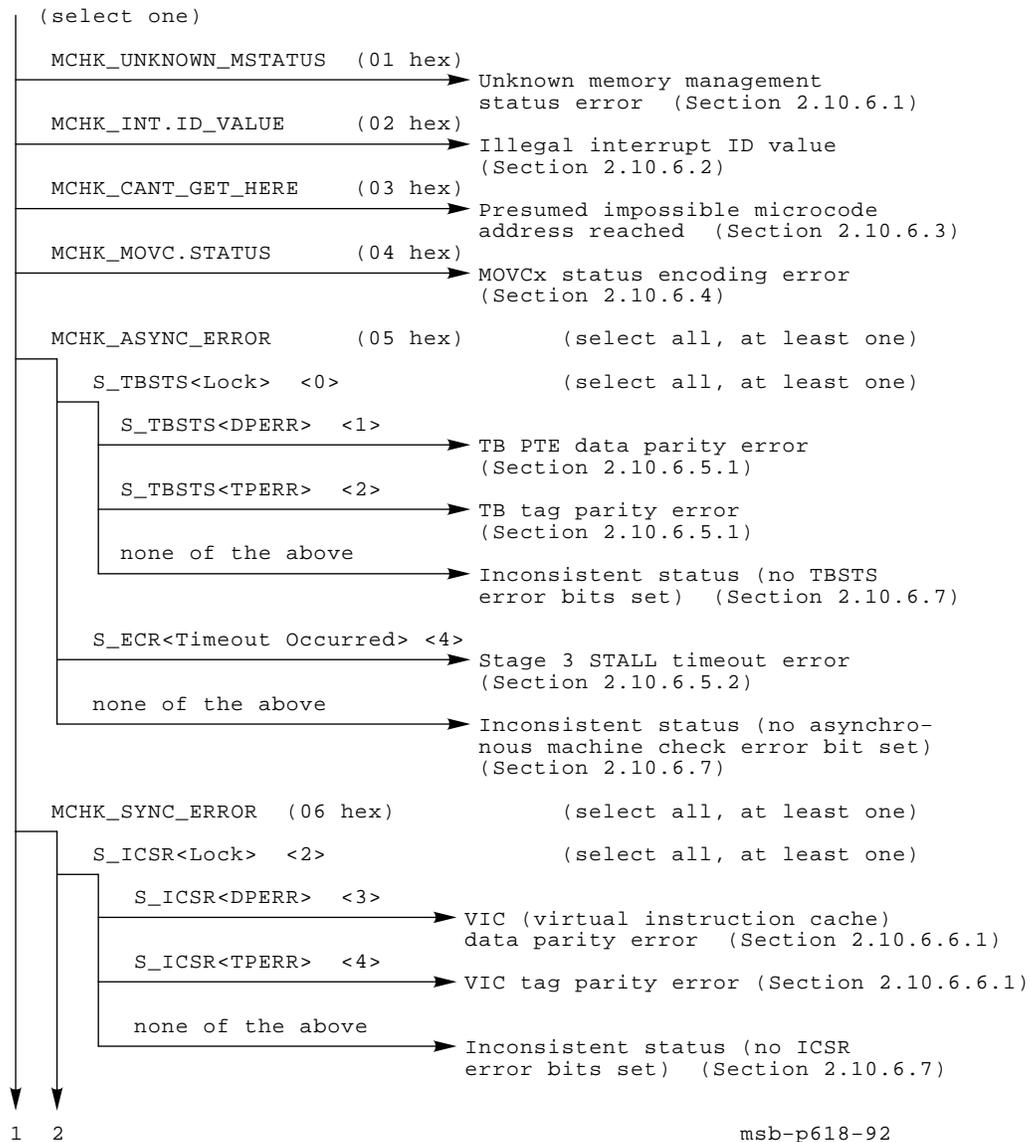
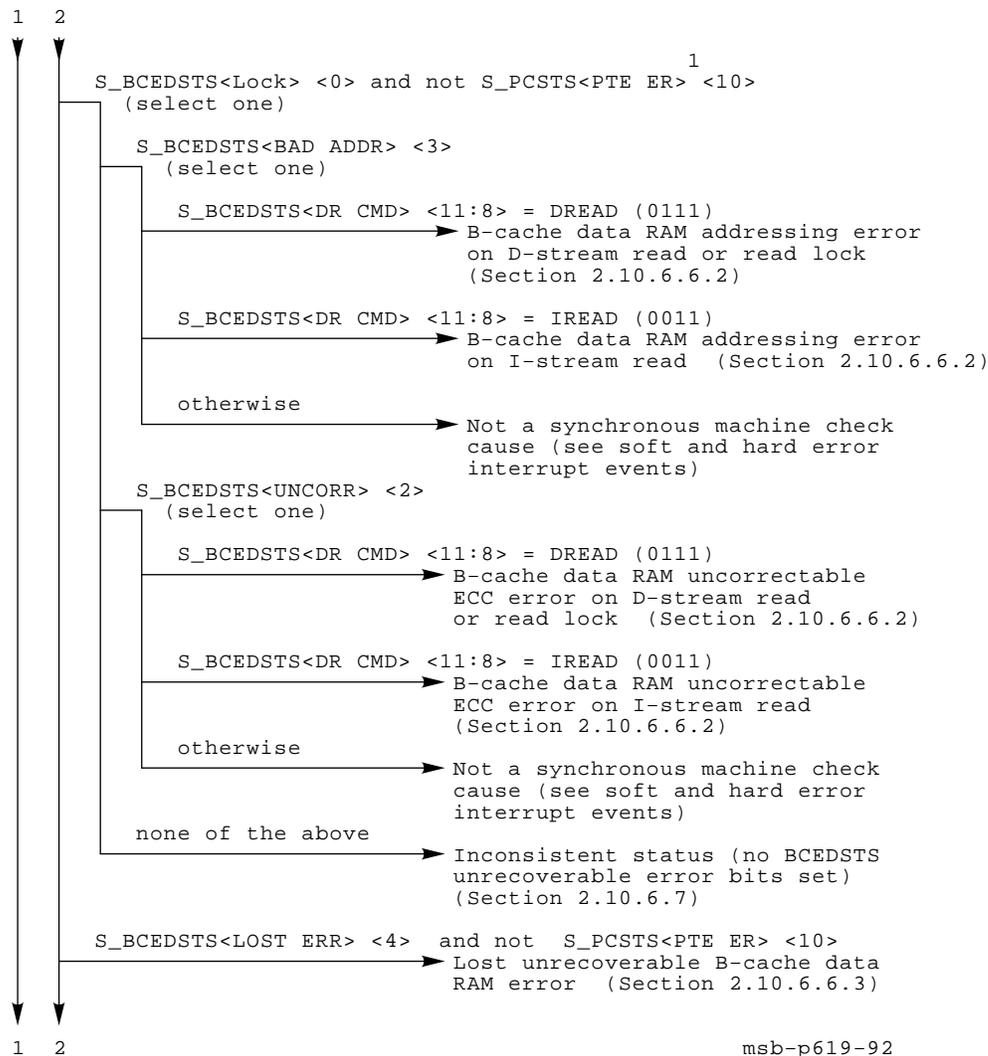


Figure 2-27 Cont'd on next page

Figure 2-27 (Cont.) Machine Check Exception Parse Tree



1  
At least one potential PTE cause must be found or the status is inconsistent (see Section 2.10.6.6). Some outcomes indicate a potential synchronous machine check cause, not a potential PTE read error cause. These errors should be treated separately.

Figure 2-27 Cont'd on next page

Figure 2-27 (Cont.) Machine Check Exception Parse Tree

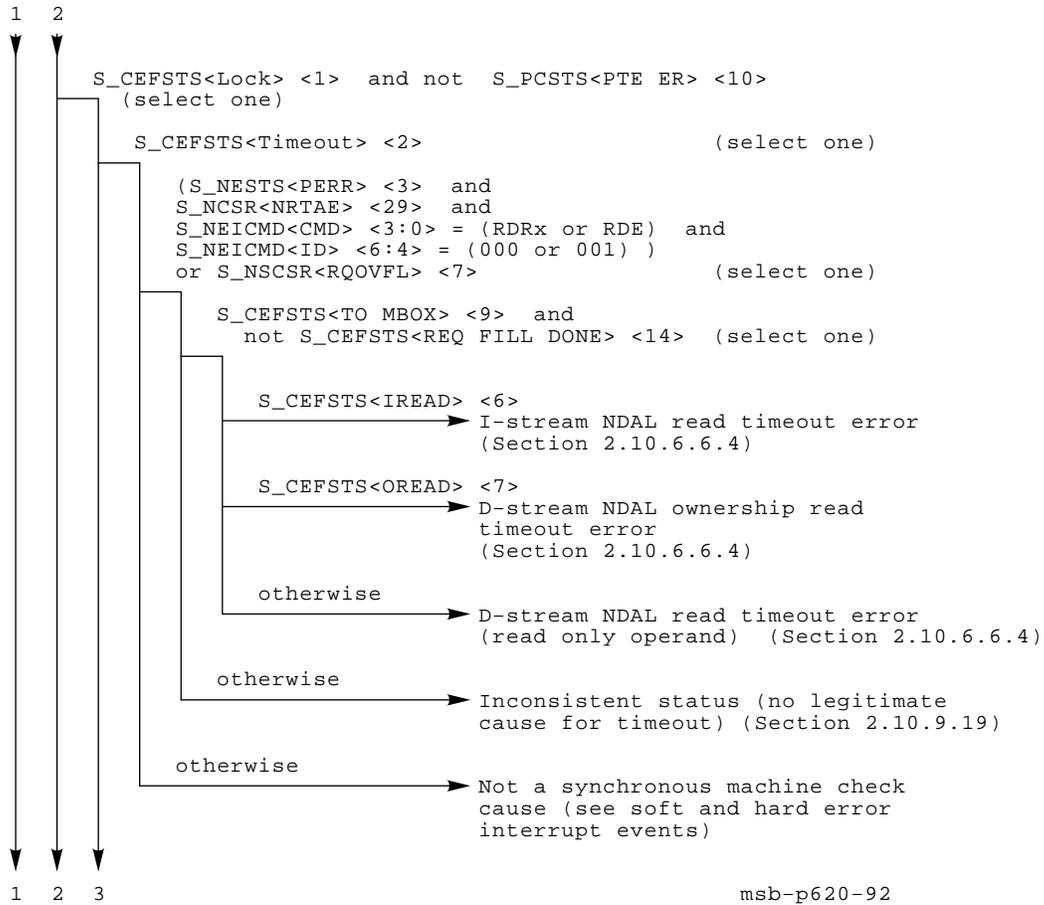


Figure 2-27 Cont'd on next page

Figure 2-27 (Cont.) Machine Check Exception Parse Tree

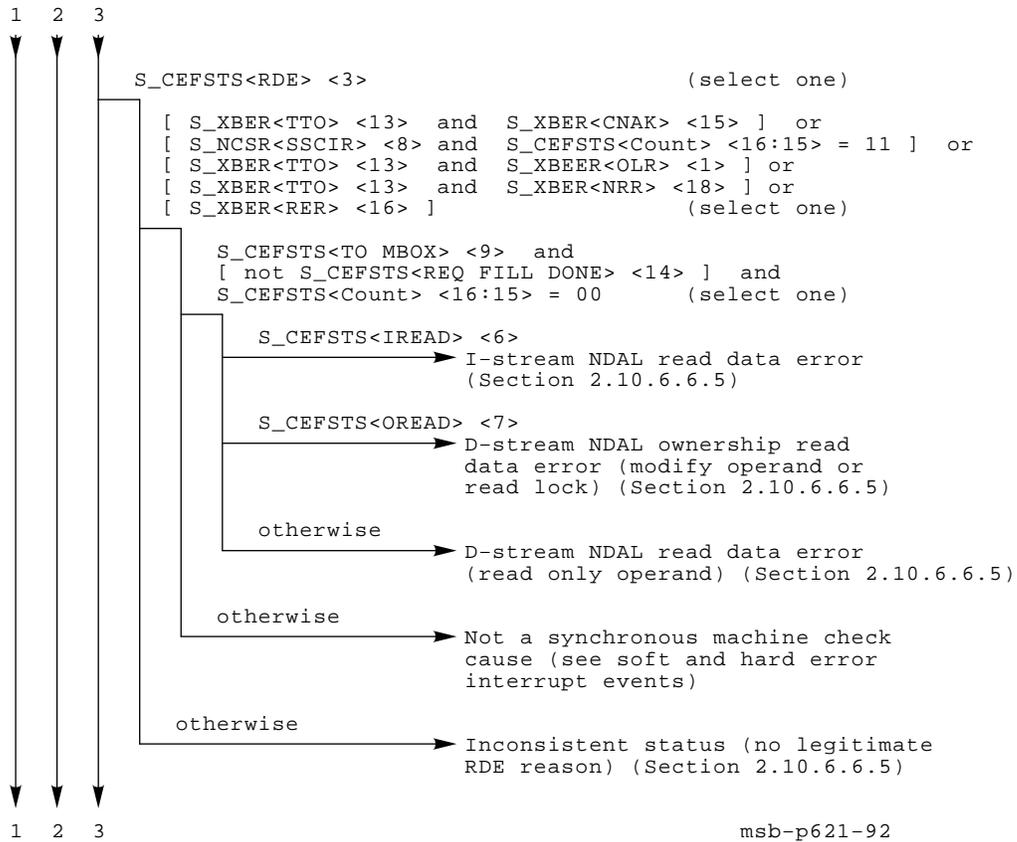


Figure 2-27 Cont'd on next page

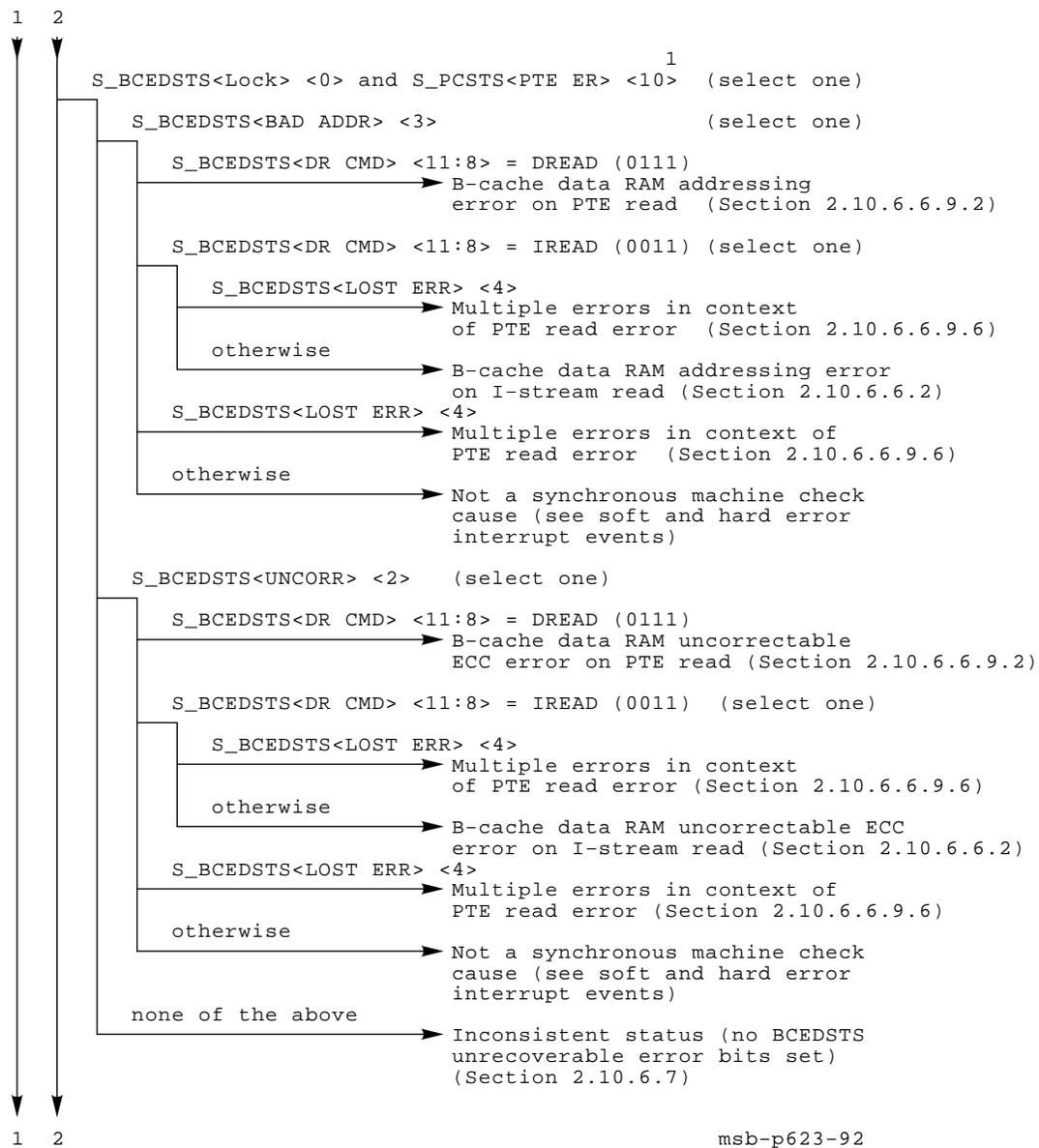
Figure 2-27 (Cont.) Machine Check Exception Parse Tree



msb-p622-92

Figure 2-27 Cont'd on next page

Figure 2-27 (Cont.) Machine Check Exception Parse Tree

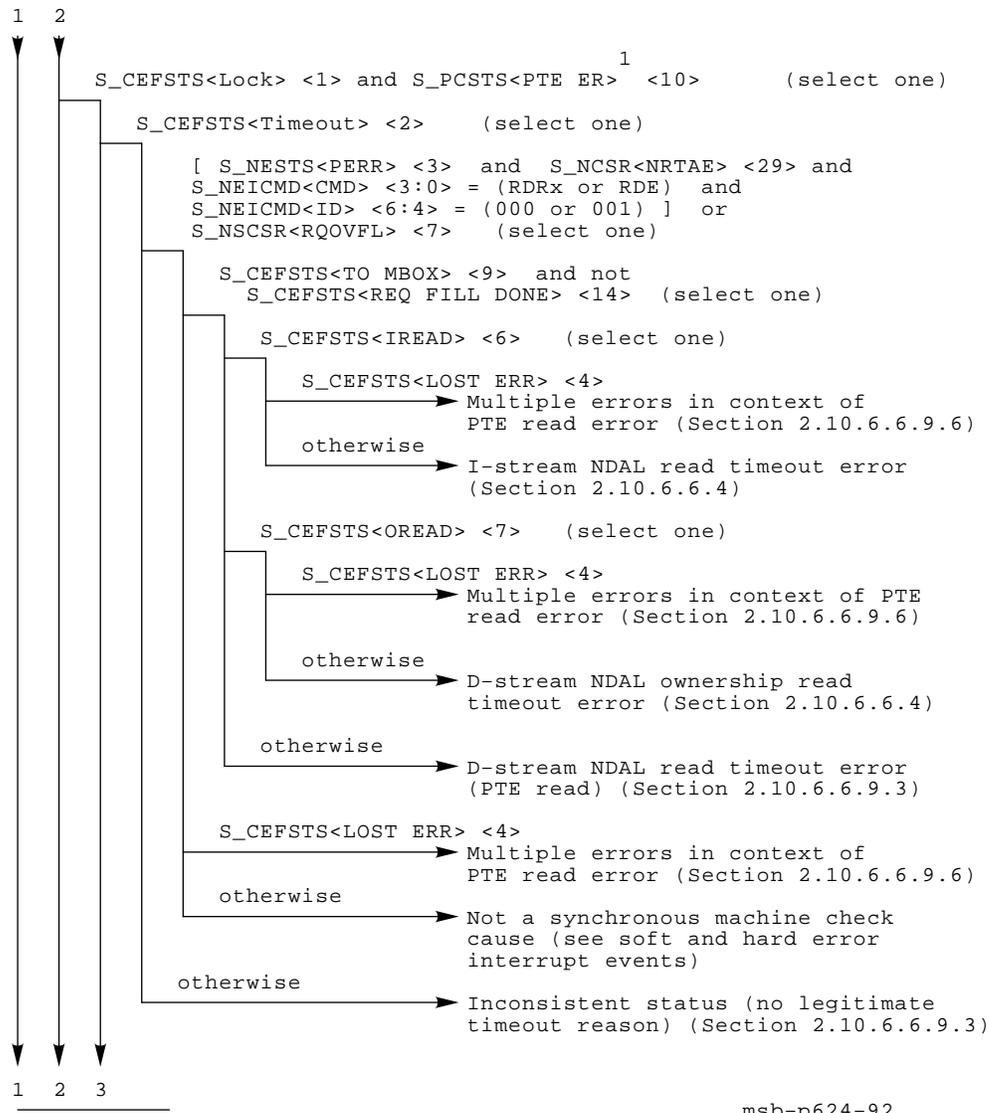


1

At least one potential PTE cause must be found or the status is inconsistent (see Section 2.10.6.7). Some outcomes indicate a potential synchronous machine check cause, not a potential PTE read error cause. These errors should be treated separately.

Figure 2-27 Cont'd on next page

Figure 2-27 (Cont.) Machine Check Exception Parse Tree



At least one potential PTE cause must be found or the status is inconsistent. (see Section 2.10.6.7). Some outcomes indicate a potential synchronous machine check cause, not a potential PTE read error cause. These errors should be treated separately.

Figure 2-27 Cont'd on next page

Figure 2-27 (Cont.) Machine Check Exception Parse Tree

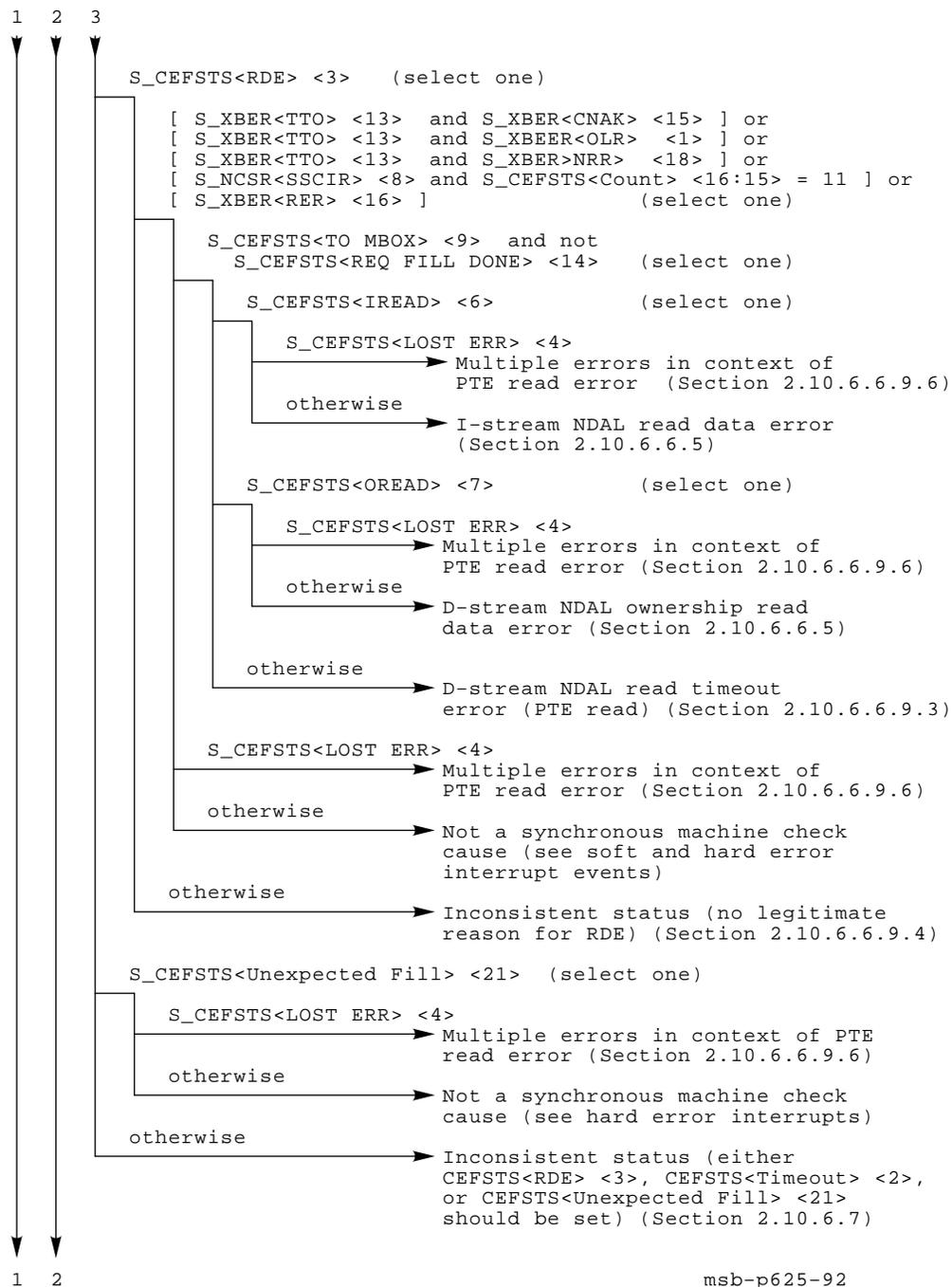
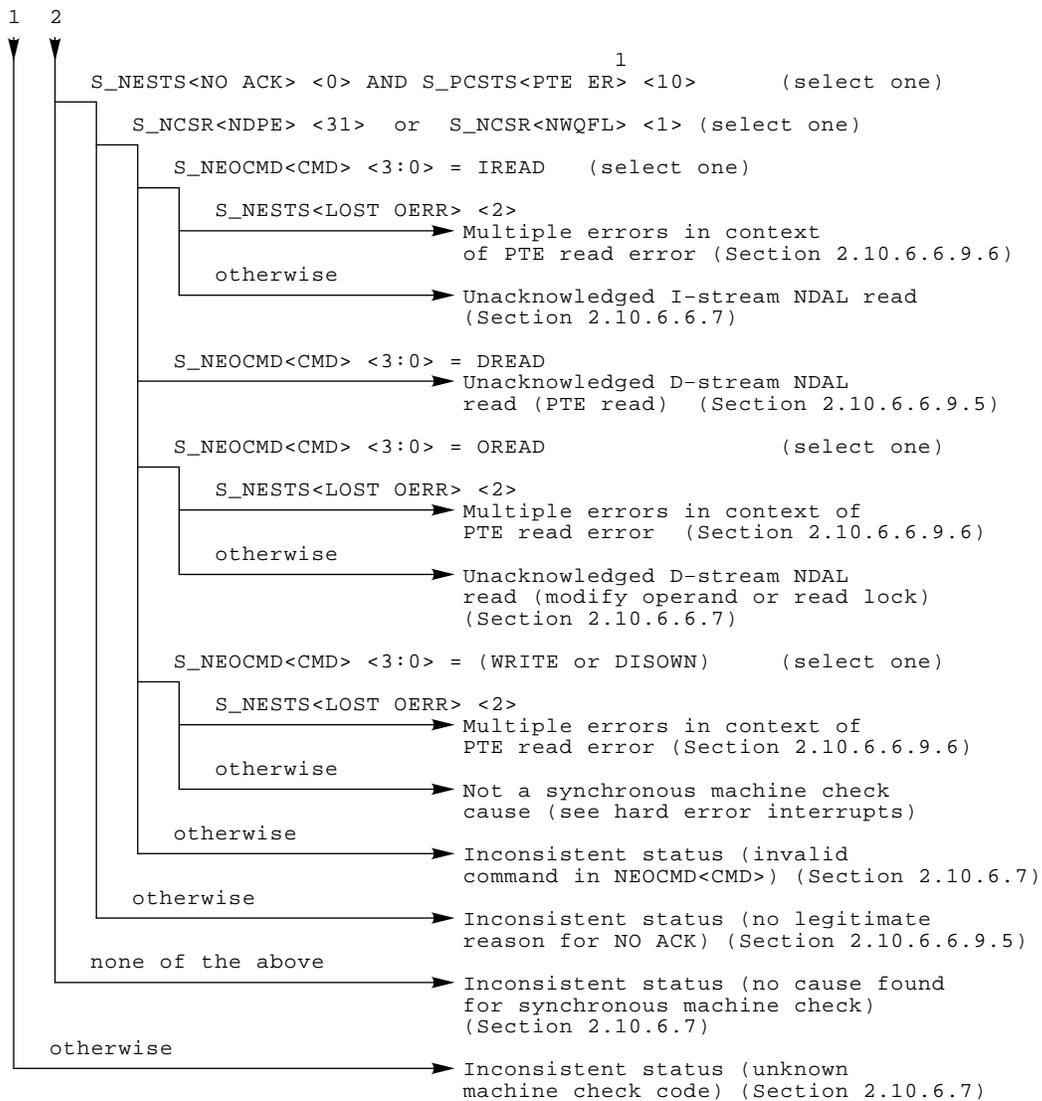


Figure 2-27 Cont'd on next page

Figure 2-27 (Cont.) Machine Check Exception Parse Tree



msb-p628-92

1  
At least one potential PTE cause must be found or the status is inconsistent (see Section 2.10.6.7). Some outcomes indicate a potential synchronous machine check cause, not a potential PTE read error cause. These errors should be treated separately.

---

**2.10.6.1 MCHK\_UNKNOWN\_MSTATUS**

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

*Recovery procedure:* No explicit error recovery is required.

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

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

---

**2.10.6.2 MCHK\_INT.ID\_VALUE**

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

*Recovery procedure:* No explicit error recovery is required.

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

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

---

**2.10.6.3 MCHK\_CANT\_GET\_HERE**

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

*Recovery procedure:* No explicit error recovery is required.

*Retry condition:* Retry if:

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

---

**2.10.6.4 MCHK\_MOVC.STATUS**

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

*Recovery procedure:* No explicit error recovery is required.

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

(PSL<FPD> = 1)

---

**2.10.6.5 MCHK\_ASYNC\_ERROR**

**MCHK\_ASYNC\_ERROR** reports serious errors that interrupt the microcode at an arbitrary point. Many internal machine states are questionable and recovery is typically not possible.

---

## 2.10.6.5.1 TB Parity Errors

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

- TB PTE data parity error  
A parity error in the PTE data portion of a TB entry had a parity error.
- TB tag parity error  
A parity error in the tag portion of a TB entry had a parity error.

*Recovery procedure:* Clear TBSTS<Lock>.

*Retry condition:* Since the Ibox is nearly always able to issue instruction prefetches, TB parity errors could occur at any time, making it impossible to determine what machine state is incorrect. There is no guarantee that all writes with a different PSL<CUR MOD> completed successfully. Therefore, even the stack frame PSL<CUR MOD> cannot be used to determine whether system data is uncorrupted.

Retry is not possible. Crash the system.

---

## 2.10.6.5.2 Ebox Stage 3 STALL Timeout Error

*Description:* Stage 3 STALL timeout errors occur when the Ebox microcode is stalled waiting for some result or action that will probably never occur. The timeout can occur for any number of reasons that are impossible to determine. This error should never happen, but it would indicate a serious failure in the NVAX CPU chip.

*Recovery procedure:* Clear ECR<Timeout Occurred>.

*Retry condition:* Should this error occur, it is not possible to determine what machine state is incorrect. Retry is not possible. Crash the system.

---

## 2.10.6.6 MCHK\_SYNC\_ERROR

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

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

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

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

For example, consider the following two errors:

- A B-cache data RAM uncorrectable error on a writeback gets reported in BCEDSTS and in NESTS.
- An NDAL command NO ACK gets reported in NESTS.

The NO ACK error makes recovery from the writeback error much more difficult. It is unlikely that these two errors would occur together, since they are unrelated events. This case is considered unrecoverable.

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

In several cases, lost errors are tolerated. See Section 2.10.4.2 for a list of these special cases. In each case, the strong tendency to prefetch data exhibited by the NVAX pipeline makes the particular lost error likely, given that one error of that kind occurred. Also, in each case, if data is lost in the lost error, a hard error interrupt is posted. So these errors are tolerated as long as they do not cause a hard error interrupt.

Errors in opcode or operand specifier fetching are always detected before architecturally visible state within the CPU is modified. This means the VR bit from the machine check stack frame should be one. This error handling analysis attempts to recover from multiple errors, so the retry condition for each error is made as general as possible. If the machine check handler finds only errors of the kind listed here, then VR should be one and it is an inconsistent report if it is not (see Section 2.10.6.7).

- VIC parity errors
- B-cache data RAM uncorrectable ECC and addressing errors in I-stream reads
- B-cache timeout errors and fill read data errors in I-stream reads
- Unacknowledged NDAL I-stream reads

---

#### 2.10.6.6.1 VIC Parity Errors

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

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

The quadword virtual address of the error is in VMAR.

*Pending interrupts:* A soft error interrupt should be pending.

*Recovery procedure:* Disable and flush the VIC by rewriting all the tags. See Section 2.10.3.2. Clear ICSR<Lock>.

*Retry condition:* Retry if:

$$(VR = 1) \text{ OR } (PSL\langle FPD \rangle = 1)$$

---

#### 2.10.6.6.2 B-Cache Data RAM Uncorrectable ECC Errors and Addressing Errors

*Description (addressing errors):* A B-cache addressing error was detected by the Cbox in an I-stream or D-stream read during a B-cache hit. Addressing errors are the result of a mismatch between the address the Cbox drives to the RAMs for a read access and the address used to write that location. A multiple-bit data error can appear to be an addressing error, although it is extremely unlikely.

*Description (uncorrectable ECC errors):* A B-cache uncorrectable data error was detected by the Cbox in an I-stream or D-stream read during a B-cache hit. Uncorrectable data errors are the result of a multiple-bit error in the data read from the B-cache. An addressing error with a single-bit data error will appear as an uncorrectable data error.

*Description (all cases):* The B-cache is in ETM. S\_BCEDIDX contains the cache index of the error, and S\_BCEDECC contains the syndrome calculated by the ECC logic.

The physical address of the reference can be found by reading the tag for the data block (using the procedure in Section 2.10.3.3.4). If the block's tag is found to contain an uncorrectable ECC error, then the address cannot be determined.

The address should not be in I/O space. If it is, it is an inconsistent status (see Section 2.10.6.7).

*Pending interrupts:* A soft error interrupt should be pending.

*Recovery procedure (addressing errors):* Clear BCEDSTS<Lock, BAD ADDR>.

*Recovery procedure (uncorrectable ECC errors):* Clear BCEDSTS<Lock, UNCORR>.

*Recovery procedure (both cases):* Flush the B-cache and then clear CCTL<HW ETM>. If the data is owned by the B-cache and if the error repeats itself (is not transient), a writeback error will result from the flush procedure. Software should prepare for this possibility by clearing NESTS and BCEDSTS errors.

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

$$(VR = 1) \text{ OR } (PSL\langle FPD \rangle = 1)$$

If a writeback error occurs in the B-cache flush, then the data is presumed to be unrecoverable. See Section 2.10.9.10 for a description of handling an error in a writeback. Given that the address is available (no error in the tag store), software should determine if the error is fatal to one process or the whole system and take appropriate action.

---

**2.10.6.6.3 B-Cache Lost Data RAM Access Error**

*Description:* A lost B-cache data RAM error may or may not have been a machine check cause. Lost B-cache data RAM errors that cause machine checks are always read errors and can be retried unless the aborted instruction has altered essential state. Whether or not it is a machine check cause, the error causes either a soft or hard error interrupt.

Lost B-cache data RAM errors may be caused by more than one operand prefetch to the same cache block.

Recovery from lost B-cache data RAM errors depends on whether the pending interrupt is hard or soft. The machine check error handling software should defer recovery until the expected hard or soft error interrupt occurs. Once the interrupt is taken, the error recovery and restart instructions found in the hard error interrupt and soft error interrupt sections should be referenced. See Section 2.10.8.4.2 and Section 2.10.9.15.

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

The B-cache is in ETM.

*Pending interrupts:* It is possible that both or either hard or soft error interrupts are pending.

*Recovery procedure:* No specific recovery action is required. The BCEDSTS<LOST ERR> should be cleared by the hard or soft error interrupt handler, and the B-cache must remain in ETM until the error interrupt occurs.

*Retry condition:* Retry only if:

$$(VR = 1) \text{ OR } (PSL\langle FPD \rangle = 1)$$

---

**2.10.6.6.4 NDAL I-Stream or D-Stream Read or D-Stream Ownership Read Timeout Errors**

*Description:* An I-stream or D-stream read or D-stream Ownership Read timed out in the Cbox before all the fill quadwords were received. This is *not* the method by which the NEXMI will notify the NVAX CPU that a location is inaccessible. All outstanding NDAL read-type cycles (IREAD, DREAD, OREAD) are normally terminated by at least one return data cycle, either RDRx or RDE.

The most likely reasons for a Cbox read timeout are an NDAL parity error (S\_NESTS<PERR> and S\_NCSR<NRTAE> set) and an XMI responder queue overflow (S\_NSCSR<RQOVFL> set). S\_CEFSTS<Count> indicates the number of quadwords received before the error, and should always be 11 (binary). The physical address is in S\_CEFADR.

Table 2–40 shows the cycle type that timed out, based upon the error bits in S\_CEFSTS during error analysis. See Section 2.10.8.4 and Section 2.10.9.13 for more details about the possible underlying system environment errors, and Section 2.10.9.19 for details on NDAL parity errors.

CEFSTS<Write> should not be set. If it is, it is an inconsistent status (see Section 2.10.6.7).

- I-stream read  
The B-cache may be in ETM, depending upon the related error bits.
- D-stream read  
The B-cache may be in ETM, depending upon the related error bits.
- D-stream ownership read  
The B-cache is in ETM. No write data has been merged with the returning fills.

The address should not be in I/O space. If it is, it is an inconsistent status (see Section 2.10.6.7).

*Pending interrupts (all cases):* A soft error interrupt will always be pending. If S\_NSCSR<RQOVFL> is set, a hard error interrupt is also pending.

*Recovery procedure (all cases):* Clear CEFSTS<Lock, Timeout>, and either S\_NESTS<PERR> and S\_NCSR<NRTAE>, or S\_NSCSR<RQOVFL>.

See Section 2.10.9.13 for more details on recovery.

*Additional recovery procedures for D-stream ownership read:* Flush the B-cache and then clear CCTL<HW ETM>.

*Retry condition (I-stream or D-stream read):* Retry if the address is not in I/O space and:

$$(VR = 1) \text{ OR } (PSL\langle FPD \rangle = 1)$$

*Retry condition (D-stream ownership read):* Given that no data is lost, retry if the memory state repair procedure is successful or not called for and if:

$$(VR = 1) \text{ OR } (PSL\langle FPD \rangle = 1)$$

If the hexword block could not be repaired or data is lost, software must determine if the error is fatal to one process or the whole system and take appropriate action.

*Post retry recovery:* If the same fill error recurs on retry, then the block is probably "lost."<sup>1</sup> Software must determine if the error is fatal to one process or the whole system and take appropriate action.

**NOTE: It may be appropriate to cause each CPU in the system to flush its B-cache, and then retry.**

<sup>1</sup> In this case the more general sense of "lost" is implied. That is, memory's ownership bit is set, but no cache writes the data back when a read is done to that location. It is possible to identify which CPU actually owns the block (or rather, which CPU is thought to own it according to the memory), but it is often not possible to determine which error caused the situation to arise.

**It may be that another error (such as an uncorrectable tag store error on a coherence request) will be repaired by the soft error interrupt handler before the retry actually occurs, fortuitously repairing the cause of the fill error.**

#### 2.10.6.6.5 NDAL I-Stream or D-Stream Read or D-Stream Ownership Read Data Errors

*Description:* An I-stream or D-stream read or D-stream ownership read terminated with an RDE (read data error) NDAL cycle before all the fill quadwords were received. S\_CEFSTS<Count> indicates the number of quadwords received before the error. (S\_CEFSTS<Count> should always be 11 (binary) if the address is in I/O space.) If S\_CEFSTS<Count> is 0 or the address is an I/O space address (in which case S\_CEFSTS<Count> would equal 11), then the first data word returned was an RDE.

CEFSTS<Write> should not be set. If it is, it is an inconsistent status (see Section 2.10.6.7).

There are several reasons why the NEXMI might send back an RDE in response to a read command.

- The NVAX attempted a read command to a nonexistent memory location (NXM) in either XMI or system support space. In the case of the XMI, the read command will be retried until the NEXMI timeout is reached, at which point the command will be aborted, S\_XBER<CNAK> and S\_XBER<TTO> will be set, and an RDE will be returned to the NVAX. This will cause a hard error interrupt.

If a system support space NXM read is performed, an RDE will also be returned (though much faster, since no retry is performed), and S\_NCSR<SSCIR> will be set. A soft error interrupt is posted. In both cases (XMI and system support), the RDE will be the first (and only) data word returned. So S\_CEFSTS<Count> will equal 00 (or 11 if the read is to I/O space).

- A read command was ACKed by an XMI node, but for some reason the data was not able to be returned. An example of this is an Ownership Read to a memory location that is currently owned by another node. If the memory "thinks" that a CPU owns the block, but no CPU really does (due to a previous error in one of the caches), then no writeback (disown) will be performed, and the memory will continually return a locked response for every NEXMI retry. Eventually, the NEXMI will time out and send back an RDE with the S\_XBER<TTO> and S\_XBER<OLR> set. This will produce a hard error interrupt.

Another example of this is if the memory ACKed the transaction, but never sent back any read data at all. This could happen if the CPU node did a read to an owned memory location, and the disown never happened. The memory would eventually time out and purge the read from its input queue, and the NEXMI would time out and send back an RDE to the NVAX with S\_XBER<TTO> and S\_XBER<NRR> set. This will also produce a hard error interrupt. In both the above cases, the RDE would be the first return data word, and S\_CEFSTS<Count> would equal 00.

- A previous BADWDATA was written to that block in memory, tagging it bad for all future reads (until it is cleared by the system). This would return an XMI RER as the first (and only) data word, and this would be translated to an NDAL RDE. S\_XBER<RER> would be set for this type of error, and a soft error interrupt would be posted. The RDE would be the first data word returned in this case, too.
- A previous system error, such as a memory location being corrupted and causing an ECC syndrome miscompare, could return an RDE on any of the words. If the S\_CEFSTS<Count> shows that the RDE was *not* the first word returned, then only a memory error could be the cause. If the count shows that it *was* the first word, then a corrupted data word in the memory is only one possibility. S\_XBER<RER> would be set and a soft error interrupt would be posted.
- An RDE in response to an XMI IDENT command has some special properties, and is included here as a separate list item even though it is, in fact, a D-stream read. There are several different XMI adapters, and two error return possibilities depending upon the adapter. In each case, the adapter realizes after it has interrupted the CPU that there is no data to be returned.

The first IDENT error return scenario is a simple S\_XBER<RER>, where the adapter returns data within the NVAX timeout period, and tags the return data as wrong and uncorrectable. In the second IDENT error scenario, the adapter simply fails to return any data in response to the IDENT, and the NEXMI eventually times out and sets the S\_XBER<TTO> and S\_XBER<NRR> bits.

See Table 2–40 for a listing of the cycle types and their error bit decode meaning, based upon the bits set in S\_CEFSTS. For all the cases above, the physical address is in S\_CEFADR.

- I-stream read  
The B-cache may be in ETM, depending upon the related error bits.
- D-stream read  
The B-cache may be in ETM, depending upon the related error bits.
- D-stream ownership read  
The B-cache is in ETM. No write data has been merged with the returning fills.

The address should not be in I/O space. If it is, it is an inconsistent status (see Section 2.10.6.7).

*Pending interrupts (all cases):* A soft error interrupt will always be pending. A hard error interrupt may also be pending, depending upon the reason for the read error. The discussion above explains when a hard error interrupt should be expected.

*Recovery procedure (all cases):* Clear CEFSTS<Lock, RDE>, and whichever of the bits XBER<TTO>, XBER<CNAK>, XBER<OLR>, XBER<NRR>, NCSR<SSCIR>, or XBER<RER> is appropriate (based upon the reason for the error).

See Section 2.10.9.14 for more details on recovery.

*Retry condition (I-stream or D-stream read):* Retry if the address is not in I/O space and:

$$(VR = 1) \text{ OR } (PSL\langle FPD \rangle = 1)$$

*Retry condition (D-stream ownership read):* Given that no data is lost, retry if the memory state repair procedure is successful or not called for and if:

$$(VR = 1) \text{ OR } (PSL\langle FPD \rangle = 1)$$

If the hexword block could not be repaired or data is lost, software must determine if the error is fatal to one process or the whole system and take appropriate action.

*Post retry recovery:* If the same fill error recurs on retry, then the block is probably "lost."<sup>1</sup>

**NOTE: It may be appropriate to cause each CPU in the system to flush its B-cache, and then retry.**

**It may be that another error (such as an uncorrectable tag store error on a coherence request) will be repaired by the soft error interrupt handler before the retry actually occurs, fortuitously repairing the cause of the fill error.**

#### 2.10.6.6.6 Lost B-Cache Fill Error

*Description:* Some number of fill errors occurred and were not latched because CEFSTS and CEFADR already contained a report of an unrecoverable error. There is no guarantee this error could have caused a machine check, though it may be a cause. Lost B-cache fill errors that cause machine checks are always read errors, and can be retried unless the aborted instruction has altered essential state. If it is a machine check cause, the error will have caused a soft error interrupt. Lost B-cache fill errors that could have caused a machine check are dealt with in the sections on hard and soft error interrupts.

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

Recovery for lost B-cache fill errors depends on whether the pending interrupt is hard or soft. The machine check error handling software should defer recovery until the expected hard or soft error interrupt occurs. Once the interrupt is taken, the error recovery and restart instructions found in the hard error interrupt and soft error interrupt sections should be referenced. See Section 2.10.8.4.2 and Section 2.10.9.15.

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

The B-cache may be in ETM and, if it is, S\_CCTL<HW ETM> will be set.

<sup>1</sup> In this case the more general sense of "lost" is implied. That is, memory's ownership bit is set, but no cache writes the data back when a read is done to that location. It is possible to identify which CPU actually owns the block (or rather, which CPU is thought to own it according to the memory), but it is often not possible to determine which error caused the situation to arise.

*Pending interrupts:* Both or either a hard or soft error interrupt should be pending.

*Recovery procedure:* No specific recovery action is required. The BCEDSTS<LOST ERR> should be cleared by the hard or soft error interrupt handler, and the B-cache must remain in ETM until the error interrupt occurs.

*Retry condition:* Retry only if:

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

---

#### 2.10.6.6.7 Unacknowledged NDAL I-Stream or D-Stream Read or D-Stream Ownership Read

*Description:* An I-stream or D-stream read or D-stream ownership read was NO ACKed by the NEXMI. The physical address is in S\_CEFADR. The NEXMI will generally ACK any legal command on the NDAL, without regard to the address. If the address later turns out to be nonexistent, or if some other error prevents return read data, an RDE will be returned to the NVAX long before the Cbox times out. Potential reasons for an NDAL bus NO ACK are as follows:

- An NDAL parity error was sensed by the NEXMI during NVAX command transfer cycle. S\_NESTS<NO ACK> should be set, S\_NEOCMD will contain the command that was refused, and S\_NEOADR will contain the address. S\_NCSR<NDPE> should also be set, since the NEXMI is assumed to have NO ACKed the cycle due to a parity error. If the NVAX also sensed the parity error (S\_NESTS<PERR> is set), then more information is available in the S\_NEICMD and S\_NEDATLO registers. This is discussed in Section 2.10.9.19.
- The NEXMI refused the command because the non-writeback queue was full. This should be prevented by the NEXMI's control of CPU GRANT L, but if it does happen, S\_NCSR<NWQFL> will be set.

For I-stream or D-stream reads the B-cache may be in ETM. For D-stream ownership reads the B-cache is in ETM.

The address should not be in I/O space. If it is, it is an inconsistent status (see Section 2.10.6.7).

*Pending interrupts (all cases):* A soft error interrupt should be pending.

*Recovery procedure (all cases):* Clear NESTS<NO ACK> and either NCSR<NDPE> or NCSR<NWQFL>.

*Additional recovery procedure for D-stream ownership read:* Flush the B-cache and then clear CCTL<HW ETM>.

*Retry condition:* Retry if:

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

---

### 2.10.6.6.8 Lost NDAL Output Error

*Description:* Some number of NDAL output errors occurred and were not latched because NESTS, NEOADR, NEDATHI, and NEDATLO already contained a report of an unrecoverable error. Although this error could be the cause of the machine check, it might not be. Lost NDAL output errors that cause machine checks are always read errors, and can be retried unless the aborted instruction has altered essential state. If it is a machine check cause, the error will have caused a soft error interrupt. Lost NDAL output errors that do not cause a machine check are dealt with in the sections on hard and soft error interrupts.

Recovery for lost NDAL output errors depends on whether the pending interrupt is a hard or soft. The machine check error handling software should defer recovery until the expected hard or soft error interrupt occurs. Once the interrupt is taken, the error recovery and restart instructions found in the hard and soft error interrupt sections should be referenced. See Section 2.10.8.6 and Section 2.10.9.17.

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

The B-cache may be in ETM and, if it is, S\_CCTL<HW ETM> will be set.

*Pending interrupts:* Both or either a hard or soft error interrupt should be pending.

*Recovery procedure:* No specific recovery action is required. The BCEDSTS<LOST ERR> should be cleared by the hard or soft error interrupt handler, and the B-cache must remain in ETM until the error interrupt occurs.

*Retry condition:* Retry only if:

$$(VR = 1) \text{ OR } (PSL\langle FPD \rangle = 1)$$

---

### 2.10.6.6.9 PTE Read Errors

PTE read errors happen in reads issued by the Mbox in handling a TB miss. Handling of these errors differs from handling the same underlying error (B-cache data RAM error, B-cache fill error, or NDAL NO ACK error) when PTE read is not the cause.

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

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

multiple errors that interfere with the analysis of the PTE error are not considered recoverable.

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

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

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

---

### 2.10.6.6.9.1 PTE Read Errors in Interruptable Instructions

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

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

- One of the interruptable instructions listed above incurred the PTE read error. In this case, SAVEPC from the machine check stack frame points to the interrupted instruction, and PC in the stack frame points to the next instruction.
- An REI instruction loaded a PSL with FPD set and a certain PC. The Ibox incurred the PTE read error in fetching the opcode pointed to by that PC. In this case, the PC in the stack frame points to the instruction that was the target of the REI, and SAVEPC from the stack frame is unpredictable.

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

### 2.10.6.6.9.2 B-Cache Data RAM Uncorrectable ECC Errors and Addressing Errors on PTE Reads

*Description (addressing errors):* A B-cache addressing error was detected by the Cbox in a PTE read during a B-cache hit. Addressing errors are the result of a mismatch between the address the Cbox drives to the RAMs for a read access and the address used to write that location. A multiple-bit data error can appear to be an addressing error, although it is extremely unlikely.

*Description (uncorrectable ECC errors):* A B-cache uncorrectable data error was detected by the Cbox in a PTE read during a B-cache hit. Uncorrectable data errors are the result of a multiple-bit error in the data read from the B-cache. An addressing error with a single-bit data error will appear as an uncorrectable data error.

*Description (all cases):* The B-cache is in ETM. S\_BCEDIDX contains the cache index of the error, and BCEDECC contains the syndrome calculated by the ECC logic. The physical address of the PTE read can be found by reading the tag for the data block (using the procedure in Section 2.10.3.3.4). If the physical address is in I/O space, it is an inconsistent status. See Section 2.10.6.7.

If the block's tag contains an ECC error, the address cannot be determined.

S\_BCEDSTS<LOST ERR> may be set. This error is probably due to the same PTE error occurring more than once. This is an acceptable assumption unless a hard error interrupt occurs after handling this error.

It should never be the case that both S\_BCEDSTS<BAD ADDR> and S\_BCEDSTS<UNCORR> are set. If they are, the state is inconsistent (see Section 2.10.6.7).

*Pending interrupts:* A soft error interrupt should be pending.

*Recovery procedure (addressing errors):* Clear BCEDSTS<Lock, BAD ADDR>.

*Recovery procedure (uncorrectable ECC errors):* Clear BCEDSTS<Lock, UNCORR>.

*Recovery procedure (both cases):* Flush the B-cache and then clear CCTL<HW ETM>. Clear PCSTS<PTE ER>. If the data is owned by the B-cache and if the error repeats itself (is not transient), then a writeback error will result from the flush procedure. Software should prepare for this by clearing NESTS and BCEDSTS errors.

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

$$(VR = 1) \text{ AND } (PSL\langle FPD \rangle = 0) \text{ AND } (S\_PCSTS\langle PTE ER WR \rangle = 0)$$

Crash the system if:

$$(PSL\langle FPD \rangle = 1) \text{ OR } (S\_PCSTS\langle PTE ER WR \rangle = 1)$$

If a writeback error occurs in the B-cache flush, then the data is presumed to be unrecoverable. See Section 2.10.9.10 for a description of handling an error in a writeback. Software must determine if the error is fatal to one process or the whole system and take appropriate action.

### 2.10.6.6.9.3 NDAL PTE Read Timeout Errors

*Description:* A PTE read timed out in the Cbox before any fill quadword was received. This is not the method by which the NEXMI will notify the NVAX CPU that a location is inaccessible. All outstanding NDAL read-type cycles are normally terminated by at least one return data cycle, either RDRx or RDE.

The only recoverable reasons for a Cbox timeout are an NDAL parity error that strikes the return data or an XMI responder queue overflow. S\_CEFSTS<Count> indicates the number of quadwords received before the error. The physical address of the PTE is in S\_CEFADR.

Section 2.10.6.6.4 discusses Cbox timeout errors in the context of non-PTE machine check errors, but some of that general discussion applies here as well. The system environment analysis for this kind of timeout is the same, since there is no distinction outside the NVAX between a fill read and a PTE read. Section 2.10.9.19 has a more complete discussion of NDAL parity errors and how they should be analyzed. Table 2-40 contains information about decoding the S\_CEFSTS bits and determining what was in progress during the error.

CEFSTS<Write> should not be set. If it is, it is an inconsistent status (see Section 2.10.6.7).

The B-cache is not in ETM. The read was not an ownership read, so this error could not have caused the ownership bits in memory to be left in the wrong state.

S\_CEFSTS<LOST ERR> may be set. This error is probably due to the same PTE error occurring more than once. This is an acceptable assumption unless a hard error interrupt occurs after handling this error.

*Pending interrupts:* A soft error interrupt should be pending. If S\_NSCSR<RQOVFL> is set, a hard error interrupt should also be pending.

*Recovery procedure:* Clear CEFSTS<Lock, Timeout>, PCSTS<PTE ER>, and either NCSR<NDPE, NTRAE> or NSCSR<RQOVFL>.

*Retry condition:* Retry if:

$$(VR = 1) \text{ AND } (PSL\langle FPD \rangle = 0) \text{ AND } (S\_PCSTS\langle PTE ER WR \rangle = 0)$$

Otherwise, crash the system.

*Post retry recovery:* If the same fill error recurs on retry, then the block is probably "lost." <sup>1</sup> Software must determine if the error is fatal to one process or the whole system and take appropriate action.

**NOTE: It may be appropriate to cause each CPU in the system to flush its B-cache, and then retry.**

<sup>1</sup> In this case the more general sense of "lost" is implied. That is, memory's ownership bit is set, but no cache writes the data back when a read is done to that location. It is possible to identify which CPU actually owns the block (or rather, which CPU is thought to own it according to the memory), but it is often not possible to determine which error caused the situation to arise.

**It may be that another error (such as an uncorrectable tag store error on a coherence request) will be repaired by the soft error interrupt handler before the retry actually occurs, fortuitously repairing the cause of the fill error.**

#### 2.10.6.6.9.4 NDAL PTE Read Data Errors

*Description:* A PTE read ended with an RDE (read data error) NDAL cycle before any of the fill quadwords were received. S\_CEFSTS<Count> indicates the number of quadwords received before the error. S\_CEFSTS<Count> should be 00 (binary) for a memory space address, or 11 for an I/O space address, since the first word returned was an RDE. The physical address of the PTE is in S\_CEFADR. Section 2.10.6.6.5 contains a more complete discussion about this type of error in a non-PTE context. The system environment analysis for an RDE is almost identical. Briefly, here are the reasons that the first return data word could be an RDE.

- The NVAX attempted a read command to a nonexistent memory location (NXM) in system support or XMI space. A system support NXM sets S\_NCSR<SSCIR>, and an XMI NXM sets S\_XBER<CNAK> and S\_XBER<TTO>.
- A read command was ACKed, but the data was not able to be returned. S\_XBER<TTO> is set along with either S\_XBEER<OLR> or S\_XBER<NRR>.
- A previous BADWDATA was written to that block in memory, tagging it bad for all future reads. S\_XBER<RER> is set.
- A previous system error, such as a memory location being corrupted and causing an ECC syndrome miscompare, was sensed on the first quadword requested. This could be any quadword within the hexword, since the requested quadword is always returned first. S\_XBER<RER> is set.

CEFSTS<Write> should not be set. If it is, it is an inconsistent status (see Section 2.10.6.7).

The B-cache is not in ETM. The read could not have been an ownership read, so this error could not have caused the ownership bits in memory to be left in the wrong state.

S\_CEFSTS<LOST ERR> may be set. This error is probably due to the same PTE error occurring more than once. This is an acceptable assumption unless a hard error interrupt occurs after handling this error.

*Pending interrupts:* A soft error and/or a hard error interrupt should be pending.

*Recovery procedure:* Clear CEFSTS<Lock, RDE>, PCSTS<PTE ER>, and whatever appropriate NEXMI bits are set (see Section 2.10.6.6.5).

*Retry condition:* Retry if:

$$(VR = 1) \text{ AND } (PSL\langle FPD \rangle = 0) \text{ AND } (S\_PCSTS\langle PTE ER WR \rangle = 0)$$

Otherwise, crash the system.

*Post retry recovery:* If the same fill error recurs on retry, then the block is probably "lost."<sup>1</sup> Software must determine if the error is fatal to one process or the whole system and take appropriate action.

**NOTE: It may be appropriate to cause each CPU in the system to flush its B-cache, and then retry.**

**It may be that another error (such as an uncorrectable tag store error on a coherence request) will be repaired by the soft error interrupt handler before the retry actually occurs, fortuitously repairing the cause of the fill error.**

---

#### 2.10.6.6.9.5 Unacknowledged NDAL PTE Read

*Description:* A PTE read was NO ACKed by the NEXMI. The NEXMI will generally ACK any legal command on the NDAL, without regard to the address. If the address later turns out to be nonexistent, or if some other error prevents return read data, an RDE will be returned to the NVAX long before the Cbox times out. So, the most likely reason for this error is an NDAL parity error.

The physical address of the PTE is in S\_NEOADR. The B-cache is not in ETM.

Refer to Section 2.10.6.6.7 for a more general discussion of an unacknowledged read, and Section 2.10.9.19 for a more complete analysis of an NDAL parity error. The system environment response to a PTE read is a subset of the response to the more general I-stream or D-stream read command.

S\_CEFSTS<LOST OERR> may be set. This error is probably due to the same PTE error occurring more than once. This is an acceptable assumption unless a hard error interrupt occurs after handling this error.

*Pending interrupts:* A soft error interrupt should be pending.

*Recovery procedure:* Clear NESTS<NO ACK>, PCSTS<PTE ER>, and the appropriate NEXMI error bits as per Section 2.10.6.6.7.

*Retry condition:* Retry if:

$$(VR = 1) \text{ AND } (PSL\langle FPD \rangle = 0) \text{ AND } (S\_PCSTS\langle PTE ER WR \rangle = 0)$$

Otherwise, crash the system.

---

#### 2.10.6.6.9.6 Multiple Errors That interfere with Analysis of PTE Read Errors

Because PTE read errors lead to several unusual cases, retry is not recommended.

*Pending interrupts:* Both or either a hard or soft error interrupt should be pending.

*Recovery procedure:* No specific recovery action is required.

*Retry condition:* No retry is possible. Crash the system.

---

<sup>1</sup> In this case the more general sense of "lost" is implied. That is, memory's ownership bit is set, but no cache writes the data back when a read is done to that location. It is possible to identify which CPU actually owns the block (or rather, which CPU is thought to own it according to the memory), but it is often not possible to determine which error caused the situation to arise.

---

**2.10.6.7 Inconsistent Status in Machine Checks**

*Description:* A presumed impossible error report was found in the error registers. This could be due to a hardware failure.

*Pending interrupts:* Both or either a hard or soft error interrupt should be pending.

*Recovery procedure:* No specific recovery action is called for.

*Retry condition:* No retry is possible. Crash the system.

---

## 2.10.7 Power Fail Interrupt

Power fail interrupts are requested by the XTC power sequencer to report an imminent loss of power to the CPU module.

Power fail interrupts are requested at IPL 1E (hex) and are dispatched through SCB vector 0C (hex). The stack frame for a power fail interrupt is shown in Figure 2-5.

The VAX 6000 Model 600 supports the standard XMI time of 4 milliseconds to execute the software necessary to save processor state for systems without a battery backup unit; software has 500 milliseconds to save processor state in systems with battery backup.

Software must flush the cache to memory in the power fail service routine since backup cache state is not saved across a power fail. This routine must do the following:

- 1 Save the current contents of the CCTL register.
- 2 Put the B-cache in software ETM mode by modifying the CCTL register.
- 3 Write to each BCFLUSH register from address 0140 0000 to 0200 0000 in increments of 20 (hex).
- 4 Restore the CCTL to its original state.

In a system without a battery backup unit, software may not have time to flush the entire cache to memory.

## 2.10.8 Hard Error Interrupt

A hard error interrupt reports an error that was detected asynchronously with instruction execution.

A hard error interrupt results in an interrupt at IPL 1D (hex) being dispatched through SCB vector 60 (hex). Typically, these errors indicate that machine state has been corrupted and that a retry is not possible. The stack frame for a hard error interrupt is shown in Figure 2–5.

Figure 2–28 contains the hard error interrupt parse tree, which indicates the causes of each hard error interrupt. For those hard error interrupts that have multiple causes, the registers and bits that isolate the cause are listed. The sections following the parse tree provide a description of the hard error, the procedure to recover, and the conditions for restarting the operation.

**Figure 2–28 Hard Error Interrupt Parse Tree**

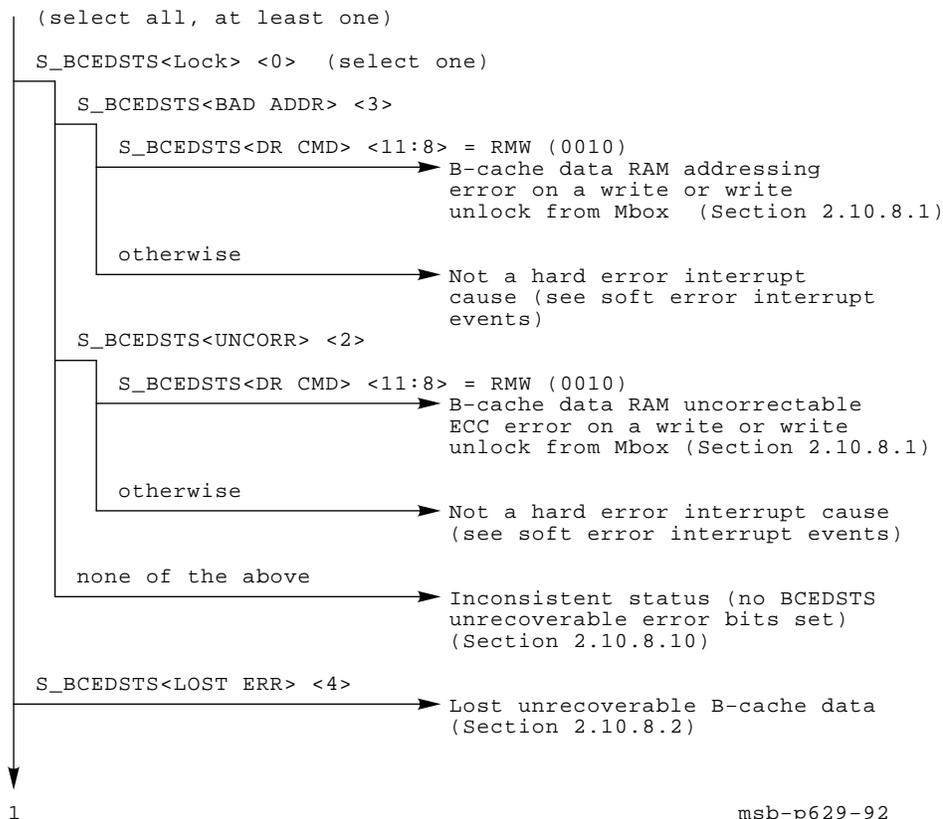


Figure 2–28 Cont'd on next page

Figure 2-28 (Cont.) Hard Error Interrupt Parse Tree

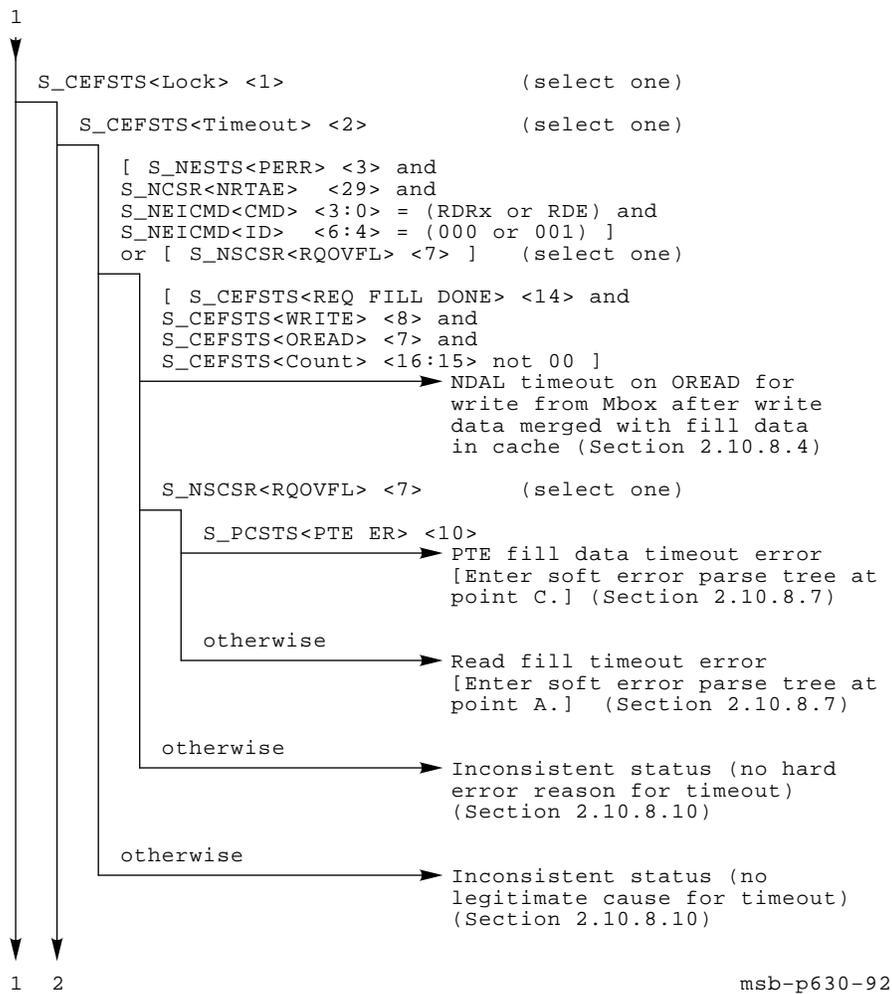


Figure 2-28 Cont'd on next page

Figure 2-28 (Cont.) Hard Error Interrupt Parse Tree

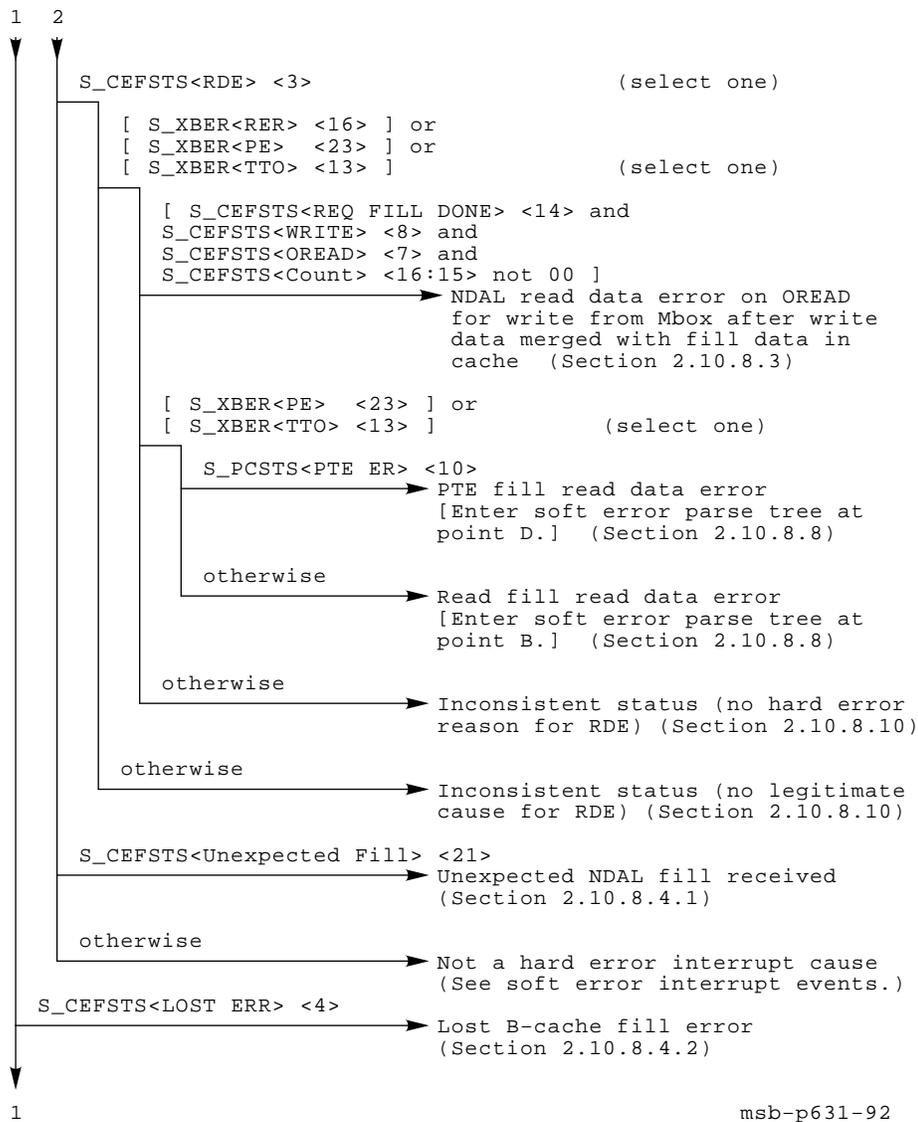


Figure 2-28 Cont'd on next page

Figure 2-28 (Cont.) Hard Error Interrupt Parse Tree

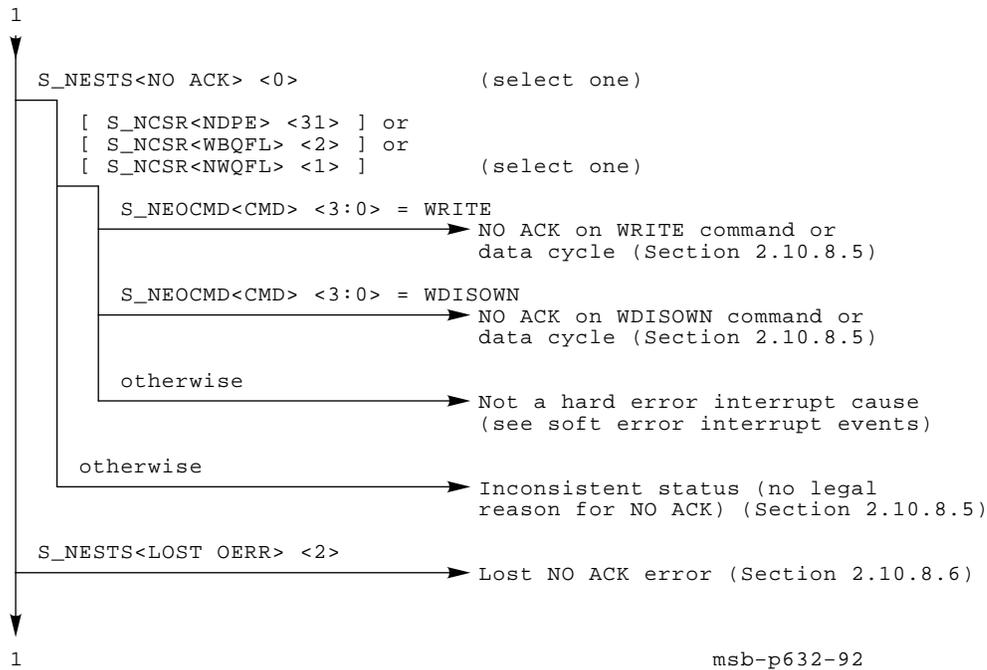
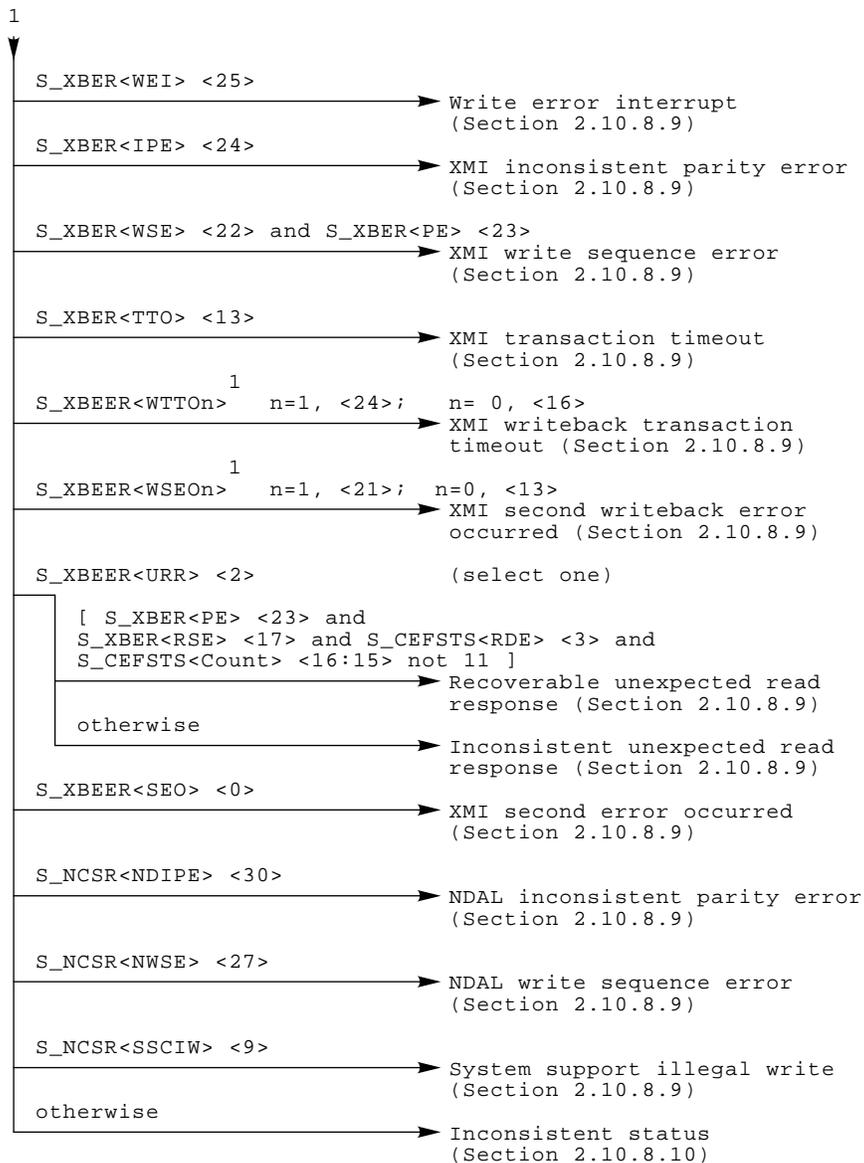


Figure 2-28 Cont'd on next page

Figure 2-28 (Cont.) Hard Error Interrupt Parse Tree



msb-p633-92

1  
Where "n" is either 0 or 1, depending upon which half of the writeback queue incurred the error.

### 2.10.8.1 **Uncorrectable Data Errors and Addressing Errors During Write or Write Unlock Processing**

*Description:* In processing a write or write unlock, the Cbox detected an addressing error or an uncorrectable ECC error on the data read from the B-cache data RAMs. The write data has already been merged with the corrupted B-cache data, and the write of the merged ("bad") data occurred. Data from the write is lost.

There are two types of uncorrectable B-cache data RAM errors: addressing errors and uncorrectable ECC errors. Both are detected through the ECC check logic. Uncorrectable ECC errors indicate that two or more bits of the stored data quadword have changed, and the error correcting code cannot correct the data. A multiple-bit data error can appear to be an addressing error, though it is extremely unlikely. A single-bit error combined with an addressing error appears as an uncorrectable error.

Addressing errors indicate that the location read from the data RAM was probably written using a different address than the one used to read it back. The actual hardware failure could have occurred in the previous data RAM write or the current read. Addressing errors are more serious than uncorrectable ECC errors since they indicate the integrity of the entire B-cache is questionable. Also, there is less than a 100% chance that a given addressing error will result in recognition of an addressing error. This is because addressing errors are recognized by encoding the parity of the address with the data and checking it on read back. All single-bit addressing errors are detectable. Note that addressing errors on writes are never detected if that data is never read out again.

The Cbox inverts three of the check bits being written back into the data RAMs to ensure that if the data is read again an uncorrectable error will be detected. If a subsequent read occurs, S\_BCEDSTS<LOST ERR> should be set, and the instruction which issued the read will machine check. However this mechanism is not fully reliable at ensuring that a subsequent read will detect the error (see Section 2.10.10, Note on Tagged-Bad Data Mechanisms).

For either case, the physical address is determined from the contents of S\_BCEDIDX using the procedure in Section 2.10.3.3.4. (If the physical address is found to be in I/O space, it is an inconsistent status. See Section 2.10.8.10.) S\_BCEDECC contains the syndrome calculated by the ECC logic. The B-cache is in ETM.

If the block's tag is found to contain an ECC error, then the address cannot be determined.

It should never be the case that both S\_BCEDSTS<BAD ADDR> and S\_BCEDSTS<UNCORR> are set. If they are, it is an inconsistent status (see Section 2.10.8.10).

*Recovery procedure (addressing error):* Clear BCEDSTS<BAD ADDR, Lock>.

*Recovery procedure (uncorrectable ECC error):* Clear BCEDSTS<UNCORR, Lock>.

*Recovery procedure (both cases):* The data in this block is lost. Flush the B-cache and then clear CCTL<HW ETM>. Flushing the B-cache should cause a writeback error (in which BADWDATA will be sent on the NDAL), so BCEDSTS and NESTS should be cleared beforehand. Then use the system-specific procedure to clear the tagged-bad state from this block in memory.

It is possible that no writeback error will occur, or that it will happen at the wrong address. This would occur if an error in the data RAMs caused the data to appear as correctable, or without error, even though it was written with three ECC bits inverted. Also, this could occur if the data was written to a different location than intended (addressing error). If this happens, then the block in memory will incorrectly appear to be good data.

**NOTE: When clearing the tagged-bad data state of memory, software must ensure that no more accesses to the block can occur. Otherwise, a process on another processor or a DMA I/O device could see incorrect data and not detect an error.**

*Restart condition (addressing error):* Addressing errors occur on data RAM reads and writes. Because the Cbox writes "bad" data back into the location, there is no way to distinguish transient read errors from transient write errors. Therefore, the worst case has to be assumed: some previous data was written to the wrong place in the B-cache or the failing data has been written to the wrong location in the B-cache. In other words, not only is the block whose address is known corrupted, but another block is as well. No restart is possible. Crash the system.

*Restart condition (uncorrectable ECC error):* If the address of the data is available and no unexpected writeback errors occurred during the B-cache flush, software must determine if the lost data is fatal to one process or the whole system and take the appropriate action.

If the address of the data could not be determined or unexpected errors occurred during the B-cache flush, crash the system.

---

### 2.10.8.2 Lost B-Cache Data RAM Hard Errors

*Description:* Some number of unrecoverable B-cache data RAM errors occurred and were not latched because BCEDSTS already contained a report of an unrecoverable error. There is no guarantee this error could have caused a hard error interrupt, though it may be a cause.

Lost B-cache data RAM errors may be caused by more than one operand prefetch to the same cache block.

B-cache data RAM errors that cause a hard error interrupt indicate that write data has been lost. Specifically, a read-modify-write operation for a write or write unlock had an uncorrectable ECC error or an addressing error. The data was written back into the RAMs with three check bits inverted.

The B-cache is in ETM.

*Pending interrupts:* A soft error interrupt may be pending.

*Recovery procedure:* Clear BCEDSTS<LOST ERR>. Flush the B-cache and then clear CCTL<HW ETM>.

*Restart condition:* No restart is possible since the errors which were not recorded could potentially have caused lost write data and no indication of what data is lost exists (based on the fact that this error was reported by a hard error interrupt). Also, the possibility exists that a subsequent read to any location that had this error could receive incorrect data with no error indication. Crash the system.

### 2.10.8.3 Read Data Error in Quadword OREAD Fill After Write Data Merged

*Description:* A D-stream ownership read for a write or write unlock terminated by receiving an RDE fill response after the requested quadword was received. In the XMI environment, the requested quadword is always returned first. Thus, this error implies that at least one data word was returned by the NEXMI. There are several reasons why a read cycle might be terminated with an RDE some time after the first data word has been returned.

- An uncorrectable memory error occurred within the MS65A memory adapter. This could happen on any word, but within the context of this analysis the error must have happened on the second, third, or fourth return data word. S\_XBER<RER> is set, and S\_CEFSTS<Count> is a value other than 00.
- An XMI parity error caused one of the read return data words to be missed by the NEXMI. S\_CEFSTS<RDE> and S\_XBER<PE> are set. The exact footprint for this type of error will be different, depending upon which data word was affected by the parity error. The error could not have happened on the first data word, since this analysis assumes that the initial data word was returned correctly and merged with the write data. Here are the other possibilities, in addition to the bits described above. See Section 2.10.8.9 for more details.
  - A parity error strikes the second return data word. S\_XBER<RSE> and S\_XBEER<URR> are set. S\_CEFSTS<Count> = 01 (binary).
  - A parity error strikes the third return data word. S\_XBER<RSE> is set. S\_CEFSTS<Count> = 10.
  - A parity error strikes the last return data word. S\_XBER<TTO> and S\_XBER<NRR> are set. S\_CEFSTS<Count> = 11.

The quadword physical address is in S\_CEFADR. The address should not be in I/O space. If it is, it is an inconsistent status (see Section 2.10.8.10). The merged data is in the B-cache in the quadword indicated in S\_CEFADR. The ownership and valid bits in the B-cache are not set.

*Pending interrupts:* A soft error interrupt will be pending.

*Recovery procedure:* Clear CEFSTS<Lock>, CEFSTS<RDE>, and XBER<RER> or XBER<PE> (whichever is appropriate). Flush the B-cache, then clear CCTL<HW ETM>.

Since at least one quadword of data was received successfully, the MS65A memory will have set its ownership bit. Thus, subsequent reads and writes to the same location may fail while this error is being handled.

The data in memory should be unchanged. The quadword containing the merged data is in the B-cache.

In general, the memory block cannot be repaired. However, we know that the memory block is left owned. So, if no writes to the block have timed out in memory, and the block is private to the interrupted job, it can be repaired by the following procedure:

- Extract the addressed quadword from the B-cache (see Section 2.10.3.3.3).
- Reset memory's ownership state (see Section 2.10.3.3.2.2) and write the extracted quadword to memory.

**NOTE: Software must ensure that no writes to this block are pending in the memory before beginning the repair. This can be done by waiting an amount of time equal to a memory subsystem write timeout time.**

*Restart condition:* If memory state repair is successful, restart. Otherwise, software must determine if the lost data is fatal to one process or the whole system and take the appropriate action.

---

#### 2.10.8.4

#### Timeout in Quadword OREAD Fill After Write Data Merged

*Description:* A D-stream ownership read for a write or write unlock timed out after the requested quadword was received. Since the requested quadword is returned first, this error implies that at least one data word was returned by the NEXMI. There are several reasons why an OREAD can time out in the Cbox before all the return data words have been returned. In each case, S\_CEFSTS<Timeout> is set, and S\_CEFSTS<Count> is a value other than 00.

- An NDAL parity error caused one of the return data words to be missed. S\_NESTS<PERR> and S\_NCSR<NRTAE> are set. See Section 2.10.9.19 for more details. A soft error interrupt will be pending.
- The XMI responder queue is full when one of the data words is returned from the MS65A memory. S\_NSCSR<RQOVFL> is set. See Section 2.10.8.9 for more details.

The quadword physical address is in S\_CEFADR. The address should not be in I/O space. If it is, it is an inconsistent status (see Section 2.10.8.10). The merged data is in the B-cache in the quadword indicated in S\_CEFADR. The ownership and valid bits in the B-cache are not set.

*Pending interrupts:* A soft error interrupt may be pending.

*Recovery procedure:* Clear CEFSTS<Lock>, CEFSTS<Timeout>, and either S\_NESTS<PERR> and S\_NCSR<NRTAE>, or S\_NSCSR<RQOVFL>. Flush the B-cache and then clear CCTL<HW ETM>.

Since at least one quadword of data was received successfully, the MS65A memory will have set its ownership bit. Thus, subsequent reads and writes to the same location may fail while this error is being handled.

The data in memory should be unchanged. The quadword containing the merged data is in the B-cache.

In general, the memory block cannot be repaired. However, the memory block is left owned. So, if no writes to the block have timed out in memory, and the block is private to the interrupted job, it can be repaired by the following procedure:

- Extract the addressed quadword from the B-cache (see Section 2.10.3.3.3).
- Reset memory's ownership state (see Section 2.10.3.3.2.2) and write the extracted quadword to memory.

**NOTE: Software must ensure that no writes to this block are pending in the memory before beginning the repair. This can be done by waiting an amount of time equal to a memory subsystem write timeout time.**

*Restart condition:* If memory state repair is successful, restart. Otherwise, software must determine if the lost data is fatal to one process or the whole system and take the appropriate action.

---

#### 2.10.8.4.1 Unexpected Fill Error

*Description:* At least one fill was received when none for that transaction ID was expected by the NVAX CPU. This can only occur if a serious NDAL error has occurred. Reads previous to this event may have received incorrect data.

If S\_CEFSTS<RDE> is set, the unexpected fill was an RDE NDAL transaction.

The B-cache is in ETM. S\_CEFADR is UNPREDICTABLE.

*Recovery procedure:* Clear CEFSTS<Lock, Unexpected Fill>. Flush the B-cache and then clear CCTL<HW ETM>.

*Restart condition:* Data may have been corrupted in memory because of incorrect read data being processed. Crash the system.

---

#### 2.10.8.4.2 Lost B-Cache Fill Error

*Description:* At least one fill error occurred in an OREAD after write data was merged, or an unexpected fill was received. The error was not latched because CEFSTS and associated registers already contained a report of an unrecoverable error. There is no guarantee that this error caused a hard error interrupt, although it may be a cause.

The B-cache may be in ETM. Read S\_CCTL<HW ETM> to find out.

*Pending interrupts:* A soft error interrupt may be pending.

*Recovery procedure:* Clear CEFSTS<LOST ERR>. If the B-cache is in ETM, flush it and then clear CCTL<HW ETM>.

*Restart condition:* Data has been corrupted but the address is unknown. Crash the system.

---

### 2.10.8.5 NDAL NO ACK During WRITE or WDISOWN

*Description:* When the Cbox issues a WRITE or WDISOWN on the NDAL and it is not acknowledged, the Cbox requests a hard error interrupt. The transaction is not retried by hardware, so the data is lost. Typically, for writebacks, the B-cache location is overwritten soon after this error, so there is no way to recover the data from the B-cache.

Although the reason for the error does not help the analysis of whether the system can be continued safely, it is interesting from an error logging standpoint to determine exactly what went wrong. There are two major reasons why the NEXMI might NO ACK a cycle:

- An NDAL parity error might have caused the NEXMI to refuse the cycle. S\_NCSR<NDPE> is set. See Section 2.10.9.19 for more details on NDAL parity errors and their footprints.
- Either the writeback queue (for WDISOWN) or the non-writeback queue (for WRITE) is full when the command appears on the NDAL. The use of CPU GRANT L is supposed to make this impossible, and as such it is considered a serious error. If this is the reason, then either S\_NCSR<WBQFL> or S\_NCSR<NWQFL> is set (depending upon what kind of command was refused).

The B-cache is in ETM. S\_NEOADR contains the physical address. S\_NEOCMD contains the byte mask and NDAL command.

*Pending interrupts:* A soft error interrupt will be pending.

*Recovery procedure:* Clear NCSR<NDPE, WBQFL, NWQFL> (whichever is appropriate). Clear NESTS<NO ACK>. First flush the B-cache and then clear CCTL<HW ETM>.

*Retry condition:* Software must determine if the lost data is fatal to one process or the whole system and take the appropriate action.

---

### 2.10.8.6 Lost NDAL NO ACK Hard Errors

*Description:* Some number of outgoing NDAL WRITE or WDISOWN commands were not acknowledged and were not latched because NESTS, NEOCMD, and NEOADR already contained a report of an NDAL output error. There is no guarantee that this error caused the hard error interrupt, although it may be a cause.

*Pending interrupts:* A soft error interrupt may be pending.

*Recovery procedure:* Clear NESTS<LOST NO ACK>.

*Restart condition:* No restart is possible since the errors which were not recorded could potentially have caused lost write data. No indication of what data is lost exists. Crash the system.

---

**2.10.8.7 Read Data Timeout with Potential Soft Error Cause**

*Description:* The S\_NSCSR<RQOVFL> bit may signify that at least one return data word was missed when it came across the XMI. This is a serious error, but still potentially recoverable. The overflow bit itself will cause a hard error interrupt, even though the underlying analysis is exactly the same as if an NDAL parity error had caused S\_CEFSTS<Timeout> to be set (except, of course, for the parity error bit being set).

As such, the hard error parse tree can use the soft error tree at selected entry points. If the hard error interrupt analysis follows the S\_NSCSR<RQOVFL> to the point where this cause is likely, the soft error parse tree should be entered at the specified place. The analysis should still be done in the context of the hard error, and if the final analysis shows that the cause is indeed nothing more than a simple soft and recoverable error, then the recovery can take place. A flag should be set so that the pending soft error interrupt will be correctly handled when the hard error analysis is done.

*Pending interrupts:* A soft error interrupt will be pending.

*Recovery procedure:* If a legitimate soft error is found, then clear NSCSR<RQOVFL> along with the error bits specified in the soft error recovery procedure. Otherwise, crash the system.

---

**2.10.8.8 Read Data Error with Potential Soft Error Cause**

*Description:* A read data error was detected that caused a hard error interrupt, but the actual error cause may be more appropriate for analysis using the soft error interrupt parse tree. Enter the soft error parse tree at the specified point, and look for consistent status in that context. This analysis is still carried out in the hard error interrupt domain, so if a legitimate error is found, the pending soft error interrupt must be prepared for.

There are two major footprints which imply that a soft error analysis might be correct:

- S\_XBER<TTO> is set, which implies that a read was unable to complete within the NEXMI timeout period. An RDE was sent back to the NVAX to signify this fact. S\_XFADR contains the failing address and length; S\_XFAER contains the command, mask, and extended address.

Depending upon the reason for the timeout, different associated timeout error bits may also be set. See Section 2.10.8.9 for more details.

- If no other timeout-related error bits are set, then the node giving the command was never granted the XMI bus.
- If S\_XBER<CNAK> is set, the command was continually NO ACKed on the XMI bus after receiving grant.
- If S\_XBER<NRR> is set, the command was ACKed, but not enough data was returned. This might be caused by a parity error striking the last return data word, in which case it would be accompanied by S\_XBER<PE>.

- If S\_XBEER<OLR> is set, the command was ACKed, but an XMI LOC response was continually returned.
- S\_XBER<PE> and S\_XBER<RSE> are set, which means that a return data word was missed due to a parity error. The return data word after the one that had the parity error was sensed as being out of sequence. This might also be accompanied by S\_XBEER<URR>, depending upon which return data word was affected. See Section 2.10.8.9 for more details. The following combinations of error bits are possible:
  - A parity error strikes the first or second return data word. S\_XBER<RSE> and S\_XBEER<URR> are set. S\_CEFSTS<Count> = (00 or 01) (binary).
  - A parity error strikes the third return data word. S\_XBER<RSE> is set. S\_CEFSTS<Count> = 10.

*Pending interrupts:* A soft error interrupt will be pending.

*Recovery procedure:* If a legitimate soft error is found, then clear NCSR<TTO>, XBER<PE>, XBER<RSE> and/or XBEER<URR> along with the error bits specified in the soft error recovery procedure. Otherwise, crash the system.

---

#### 2.10.8.9 NEXMI Hard Error Interrupts

*Description:* Errors which occur in the system environment that result in loss of data or cannot notify the NVAX CPU by returning RDE notify the CPU of the error by asserting H ERR L. The following register bits are consistent with a NEXMI hard error interrupt:

- XBER<WEI>
- XBER<IPE>
- XBER<WSE>
- XBER<TTO>
- XBEER<WTTO0>
- XBEER<WTTO1>
- XBEER<WSEO0>
- XBEER<WSEO1>
- XBEER<URR>
- XBEER<SEO>
- NCSR<NDIPE>
- NCSR<NWSE>
- NCSR<SSCIW>
- NCSR<RQOVFL>

Some NEXMI error bits will be set as a byproduct of an error that also sets an NVAX error register bit (usually in the Cbox). Those error bits are described in the sections that deal with the NVAX-detected error. This section describes NEXMI error bits that are seen without an accompanying NVAX error, or which would normally cause a soft error interrupt in the NVAX, but are hard errors due to the NEXMI error bit.

- **S\_XBER<WEI>:** If the NEXMI receives a write error IVINTR transaction on the XMI bus, it will set this bit and post a hard error interrupt. The recovery procedure depends upon why the other node signaled the error.
- **S\_XBER<IPE>:** If the NEXMI senses a parity error on the XMI bus, but some other node ACKs the transaction, then an inconsistent parity error has occurred. This is very serious, since another node has probably used information that was in error. Crash the system.
- **S\_XBER<WSE>:** A write transaction that was directed to one of the XMI CSR registers had a command cycle that was followed by a non-WDAT cycle. This means that a CSR write was supposed to happen and did not. The most likely reason for this error is an XMI parity error.

The KA66A reads and writes its own nodespace (public) XMI CSRs by going out of the module, across the XMI, and back into the module. As such, the CSR write could have come from any occupied node, even the same node that has the WSE bit set. XMI nodes will generally retry a failed write transaction until it is either successful, or the XMI timeout counter expires. If the retry is successful, then there will be no indication in any commander error register that the cycle was ever refused.

So, if S\_XBER<WSE> is set, then S\_XBER<PE> should also be set. If no other node has an indication that a write command was NO ACKed, then it can be assumed that a transient parity error caused the failure, but the transaction was successfully retried. If S\_XBER<PE> is *not* set, then crash the system.

- **S\_XBER<TTO>:** A transaction was unable to be completed within the NEXMI timeout period and was aborted. S\_XFADR contains the failing address and length; S\_XFAER contains the command, mask, and extended address. There are many different types of XMI timeout errors. Since most command types are retried automatically on the XMI bus by the NEXMI controller, each one indicates a nontransient error on the bus.

If the TTO was due to a read-type command, the NEXMI would have returned an RDE to the NVAX when the timeout finally occurred. So S\_CEFSTS<RDE> should also be set. In that case, analysis of the TTO will be done through Cbox error registers.

If the TTO was due to a write-type command, no RDE would be sent, so the NEXMI registers will contain the only failing error indication and the NVAX will not detect the error.

In the case of a read timeout, it is likely that a hard error interrupt and a soft error interrupt will be pending. If the initial hard error analysis finds no matching error, it is still legitimate to follow the appropriate soft error interrupt tree until a cause is found. If all the evidence points to the soft error interrupt cause, and if the only hard error cause is the TTO bit itself, then recovery can proceed based upon the assumption that the error is, indeed, a normal soft error type.

The following timeout conditions are possible:

- The command is continually NO ACKed on the XMI bus. S\_XBER<CNAK> is set. This is likely caused by a read (or write) to a nonexistent address. If the command was a read, it is probably recoverable. The address information is latched in the S\_XFADR and S\_CEFADR registers.
- A read command is ACKed, but no data (or not enough data) is returned from the responder device. S\_XBER<NRR> is set. The error recovery procedure for this is device specific.
- A read command is ACKed, but each time it is retried an XMI LOC response is returned from the responder. S\_XBEER<OLR> is set. The XMI LOCKOUT signal is supposed to prevent this from occurring, so this is a serious error. Recovery might be possible, depending upon the context of the NVAX instruction being executed.
- The command is a write, and it is ACKed, but each retry finds the data word NO ACKed. S\_XBER<WDNAK> is set. There is no reason that this should occur, and it is unrecoverable.
- The node was never granted the XMI bus. S\_XBER<CNAK>, S\_XBER<NRR>, and S\_XBEER<OLR> are clear. There should be no occasion where the bus is so busy that a node never gets onto the bus.
- **S\_XBEER<WTTO0,WTTO1>**: A writeback transaction was unable to complete within the NEXMI timeout period. There are two write timeout bits, one for each half of the writeback buffer. Each bit independently shows whether that half had an error. The failing address is latched in S\_WFADR<sub>n</sub> (where "n" is either 0 or 1, depending upon which WTTO<sub>n</sub> bit is set). If both S\_WTTO0 and S\_WTTO1 are set, then the contents of S\_WFADR<sub>n</sub> are invalid. There are several reasons why a timeout might occur on a writeback:
  - The DWMASK command was NO ACKed on the XMI until the NEXMI timeout was reached. S\_XBEER<WCNAK<sub>n</sub>> is set (where *n* is either 0 or 1, depending upon which WTTO<sub>n</sub> was set). Since the NEXMI automatically retries WDISOWN instructions, a transient failure (such as a parity error or a filled memory queue) would not be likely to cause this error. The target address could be nonexistent.
  - The DWMASK was ACKed, but one of the data words was continually NO ACKed until timeout. S\_XBEER<WWDNAK<sub>n</sub>> is set. Since the command was ACKed, the address in memory must have been legitimate. The MS65A memory has separate

input queues for the command and data information, and if the data queue was full it would NO ACK a data cycle. However, the command retry would likely be successful long before timeout.

- The node was never granted the XMI bus. S\_XBEER<WCNAKn> and S\_XBEER<WWDNAKn> are both clear. There should be no occasion where the bus is so busy that a node never gets onto the bus.
- **S\_XBEER<WSEO0,WSEO1>**: A second writeback hard error occurred before the first one was analyzed and cleared. Information is definitely lost. Crash the system.
- **S\_XBEER<URR>**: The NEXMI received an XMI read response cycle directed at its node ID when it was not expecting any read data. The NEXMI will not ACK the transfer on the XMI, and it will not pass this unexpected data onto the NDAL. As such, this would *not* be a potential cause of the similar (but unrelated) error signaled by S\_CEFSTS<Unexpected Fill>.

There are two likely (and potentially recoverable) reasons for an unexpected read response (URR) to happen:

- A parity error occurred on the XMI bus just as a legitimate read return data word was being delivered to the KA66A CPU.
- A device on the XMI took too long to return the data word, and the KA66A CPU had timed out and was no longer expecting the response.

The following sequence of events shows how a parity error could cause a URR:

- The XMI read return data cycle with the parity error will be ignored by the NEXMI, and S\_XBER<PE> will be set.
- The next return data word will be out of sequence, aborting the NEXMI XMI read data transaction and setting S\_XBER<RSE>.
- The next legitimate return data words will then be unexpected, and S\_XBEER<URR> will be set.

A URR will only be sensed if the parity error happens on a return data word *other* than the last word. If the parity error causes the last return data word to be lost, then S\_XBER<NRR> will be set instead. The conditions that set the no read response (NRR) are discussed in the S\_XBER<TTO> discussion above.

The NEXMI will send an RDE back to the NVAX when it senses the read sequence error (RSE), and the error information should also be logged in the Cbox error registers. S\_CEFSTS<RDE> and S\_CEFSTS<Lock> should be set, and S\_CEFSTS<Count> should show that a data word other than the last one was expected when the RDE arrived.

A URR that is *not* accompanied by any other error bits does not abide by the above analysis. Since there is no obvious reason for the URR, it must be assumed that a system inconsistency exists, and the user

should take whatever steps are appropriate to correct the situation. Further guidance involves device-specific error analysis.

- **S\_XBEER<SEO>**: A second hard error occurred before the first one had been analyzed and cleared. Since there is no way to determine what happened, and since hard errors are by their nature serious, recovery is not possible. Crash the system.
- **S\_NCSR<NDIPE>**: The NEXMI sensed that an NDAL parity error occurred, yet the cycle was ACKed by a node. This means that one of the two NDAL nodes went ahead with bad information. Crash the system.
- **S\_NCSR<NWSE>**: A WRITE or WDISOWN command was sent by the NVAX without the proper number of WDATA cycles. This would be sensed by the NVAX as a NO ACK on a write cycle, and is unrecoverable. Crash the system.
- **S\_NCSR<SSCIW>**: A write directed at system support space was ACKed on the NDAL by the NEXMI, but the cycle turned out to be invalid once it was inspected more closely. This should never happen, and no register holds the address of the failing transaction. Crash the system.
- **S\_NSCSR<RQOVFL>**: The XMI responder queue in the NEXMI was full, yet at least one more word was available from the XMI bus with no place to put it. This should be prevented by the use of XMI SUPPRESS L. Although this is a serious error, it is recoverable.

It is most likely that the data word that arrived after the XMI input queue was full was a read return data word, since return data will not be stopped by the suppression signal. As such, an invalidate transaction was probably not missed, and the system is still coherent. S\_CEFSTS<Timeout> should be set.

This error would cause a hard error interrupt. However, if no other hard error is found in the parse tree that matches the saved error bits, it might still be a legitimate error for soft error interrupt analysis. In that case, the normal soft error analysis can be done, and NSCSR<RQOVFL> can be cleared at the end of the soft error recovery.

*Recovery procedure:* Clear the error status bits in the system registers and perform any necessary system-dependent recovery procedure.

*Restart condition:* Depends on the error. If the system environment reports the address of the lost data, software may be able to determine if the lost data is fatal to one process or to the whole system and take appropriate action.

---

#### 2.10.8.10

#### **Inconsistent Status in Hard Error Interrupts**

*Description:* A presumed impossible error report was found in the error registers. This could be due to a hardware failure.

*Recovery procedure:* No specific recovery action is called for.

*Restart condition:* No retry is possible. Crash the system.

---

## 2.10.9 Soft Error Interrupt

Soft error interrupts are requested to report errors which were detected, but did not affect instruction execution. This results in an interrupt at IPL 1A (hex) to be dispatched through SCB vector 54 (hex).

This section describes errors that can cause a soft error interrupt. The parse tree (Figure 2–29) shows how to determine the cause of a given soft error. Each error is then described and a recovery procedure is given. Where appropriate, the conditions for restart are given. See Section 2.10.3 and Section 2.10.4 for more on error recovery and error retry.

Many errors that cause a soft error interrupt can also lead to a machine check exception. For this reason, a soft error interrupt with no apparent cause is not an inconsistent state unless the CPU has executed an instruction while the IPL was lower than 1A (hex) since the most recent machine check exception.

When a soft error interrupt is the only notification for any memory read error that could cause a machine check, the error did not cause a machine check for one of the following reasons:

- The error, a P-cache fill error, did not occur on the quadword requested by the Ebox or Ibox.
- The Ebox took an interrupt before accessing an instruction or operand that was prefetched by the Ibox.
- A prefetched instruction or operand belonged to an instruction following a mispredicted branch, so the Ebox never executed the instruction and it was flushed from the pipeline when the branch mispredict was recognized.
- The Ebox took an exception for a different reason before attempting to use an instruction execution dispatch or access an operand prefetched by the Ibox. (The pipeline was flushed because of the exception.)

Figure 2–29 Soft Error Interrupt Parse Tree

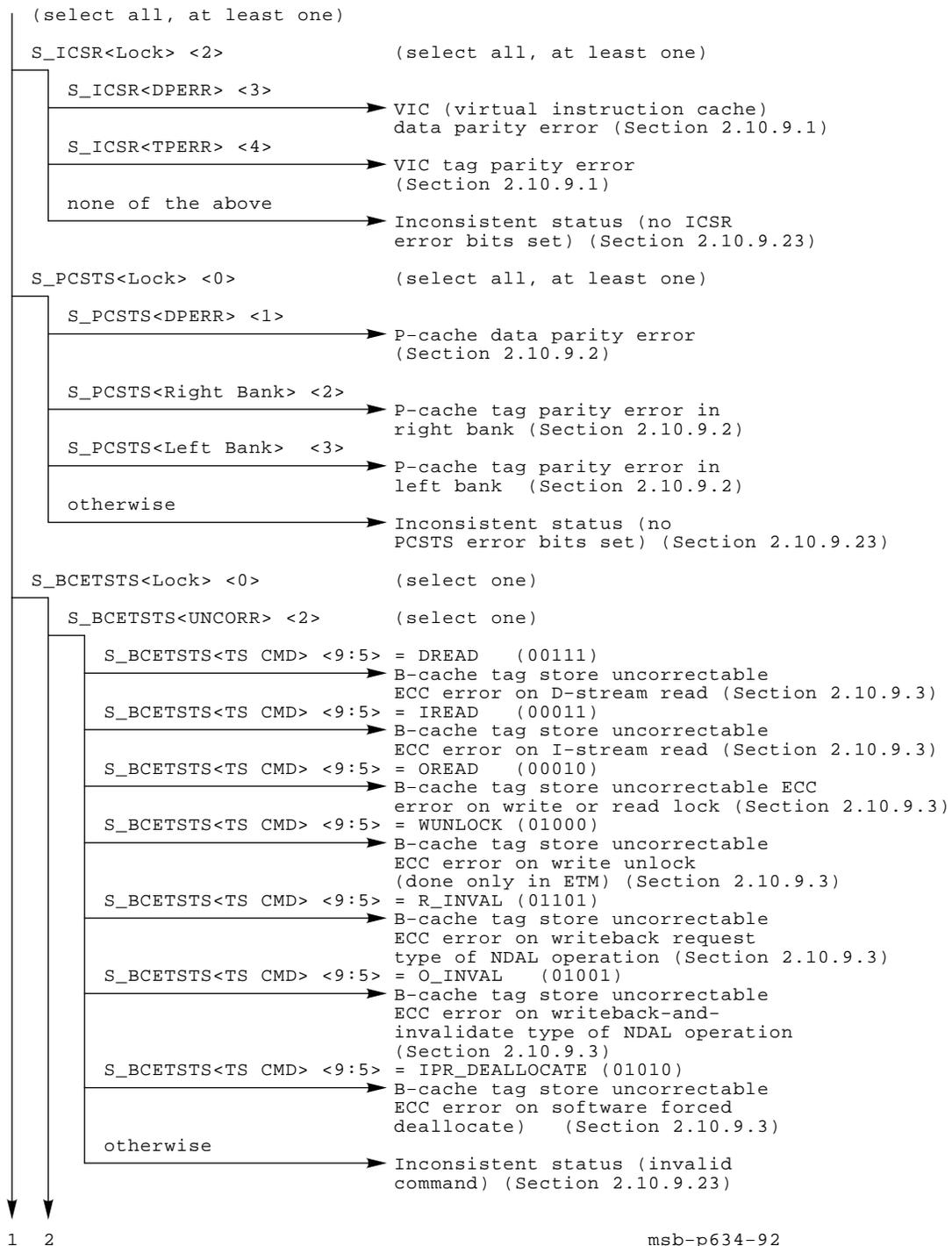


Figure 2–29 Cont'd on next page

Figure 2–29 (Cont.) Soft Error Interrupt Parse Tree

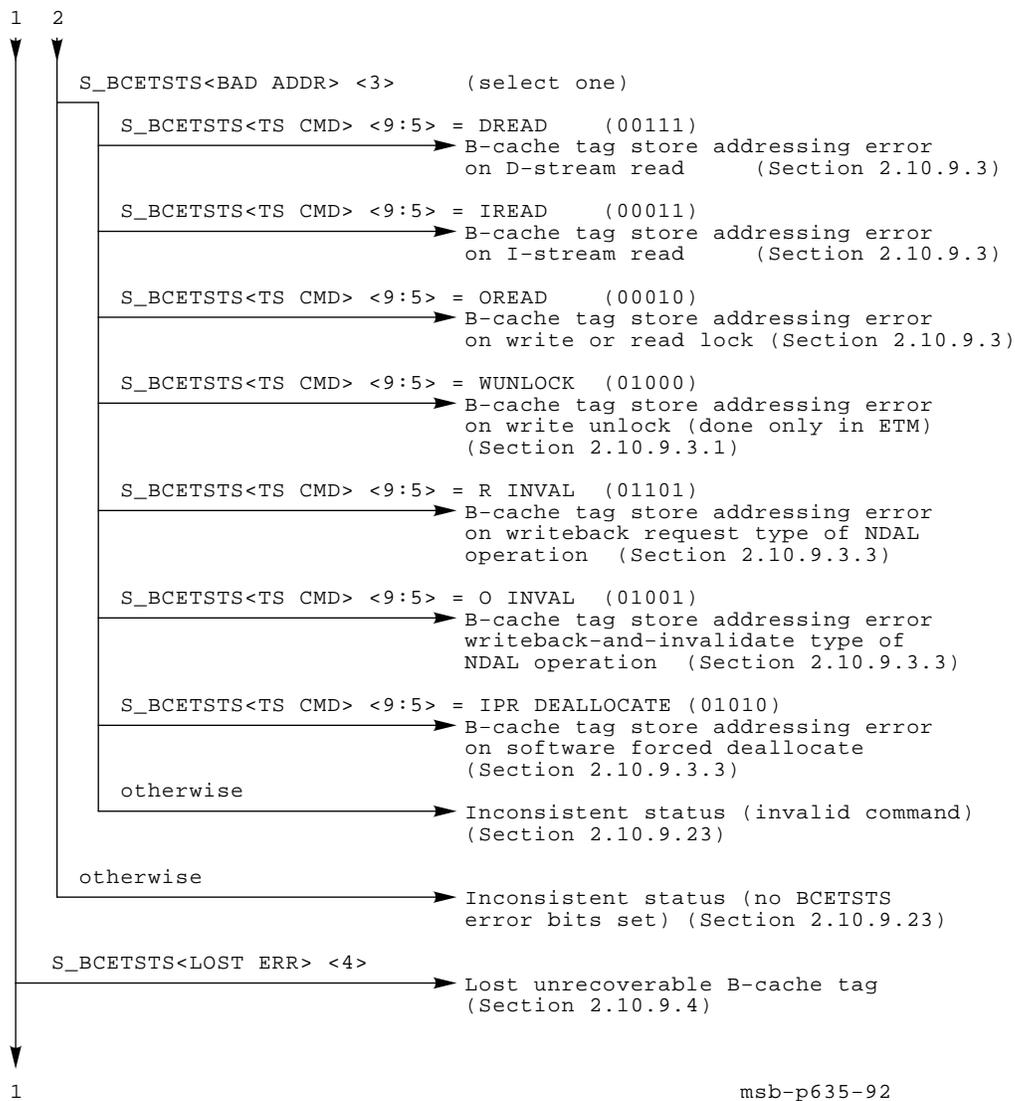


Figure 2–29 Cont'd on next page

Figure 2–29 (Cont.) Soft Error Interrupt Parse Tree

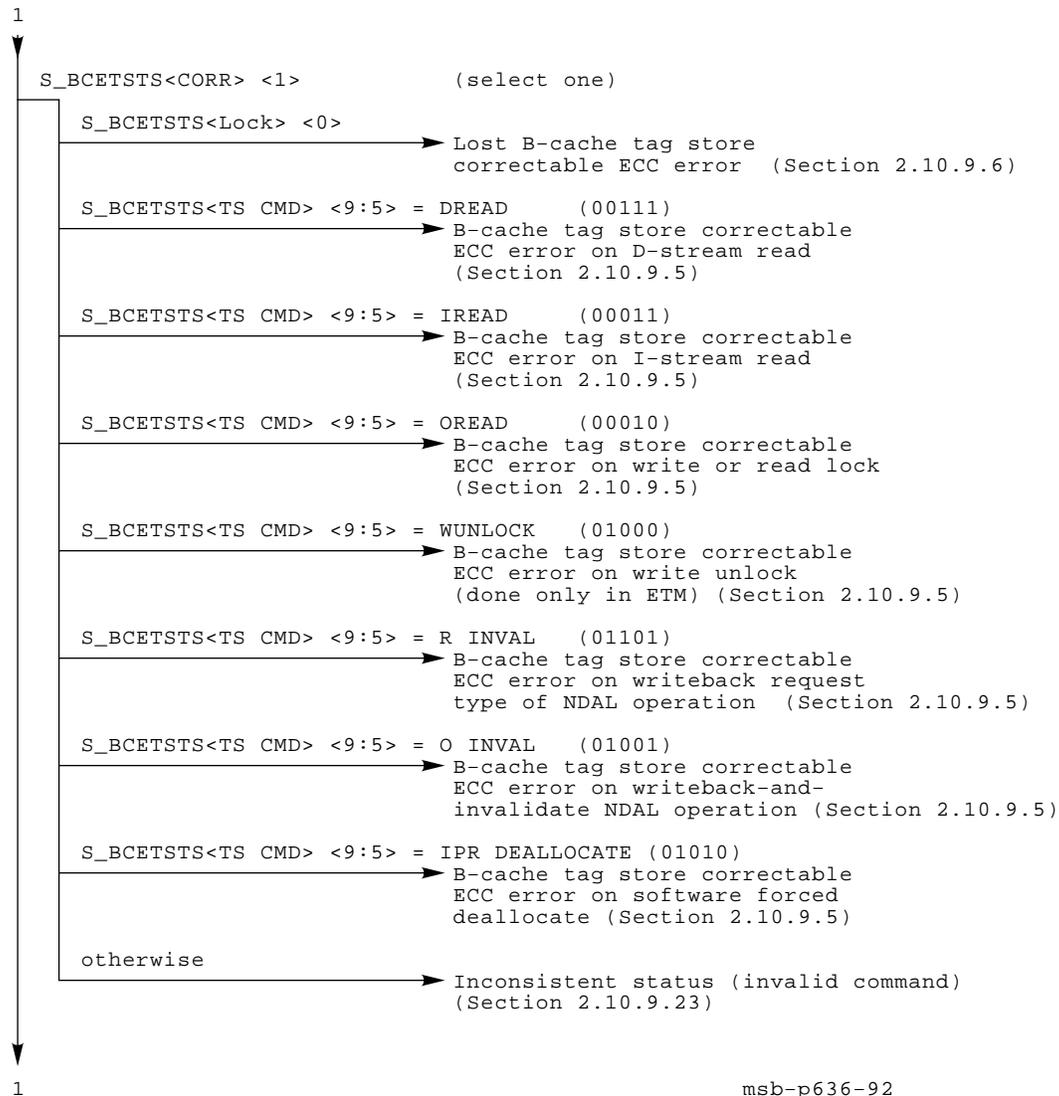
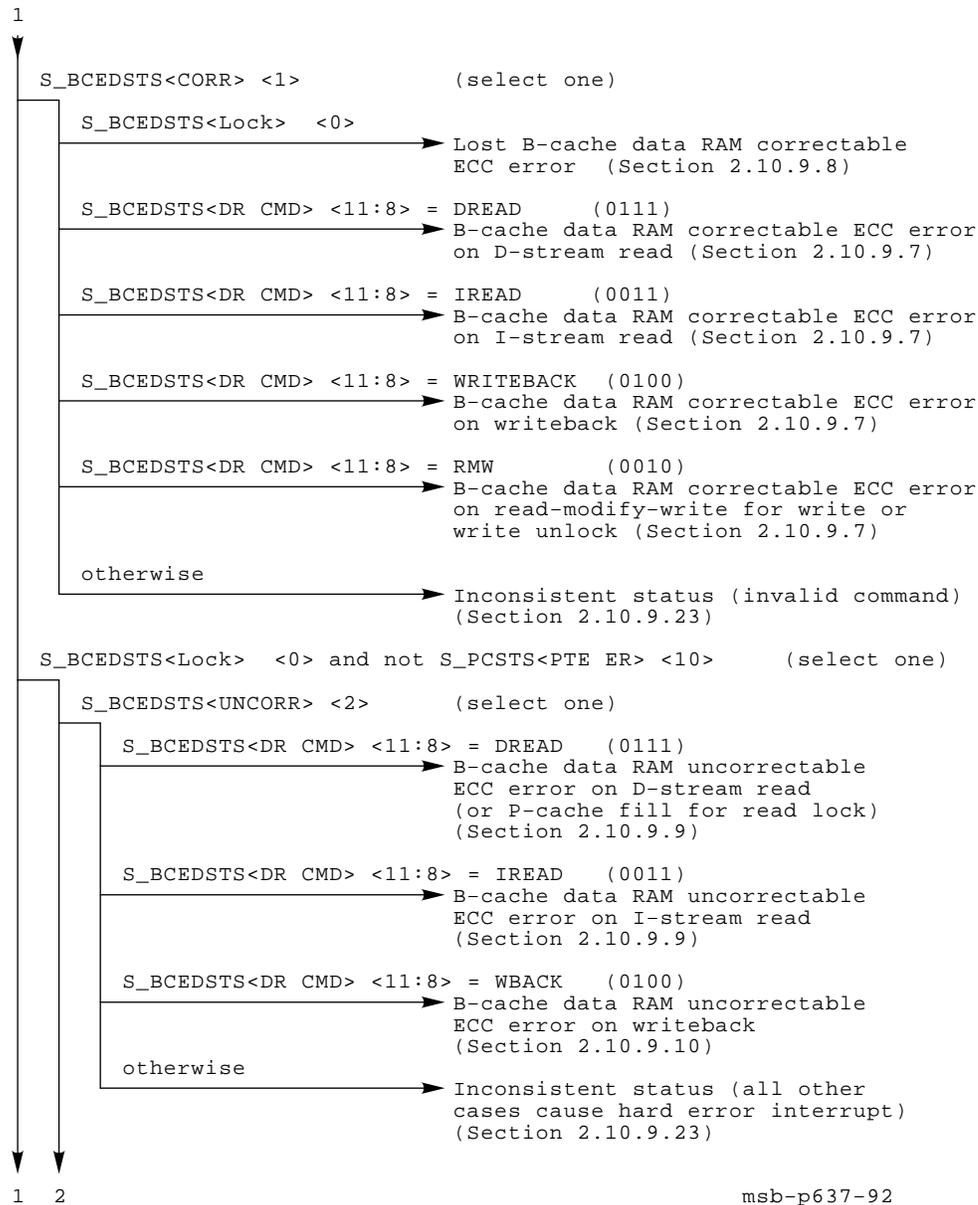


Figure 2–29 Cont'd on next page

Figure 2–29 (Cont.) Soft Error Interrupt Parse Tree



msb-p637-92

Figure 2–29 Cont'd on next page

Figure 2-29 (Cont.) Soft Error Interrupt Parse Tree

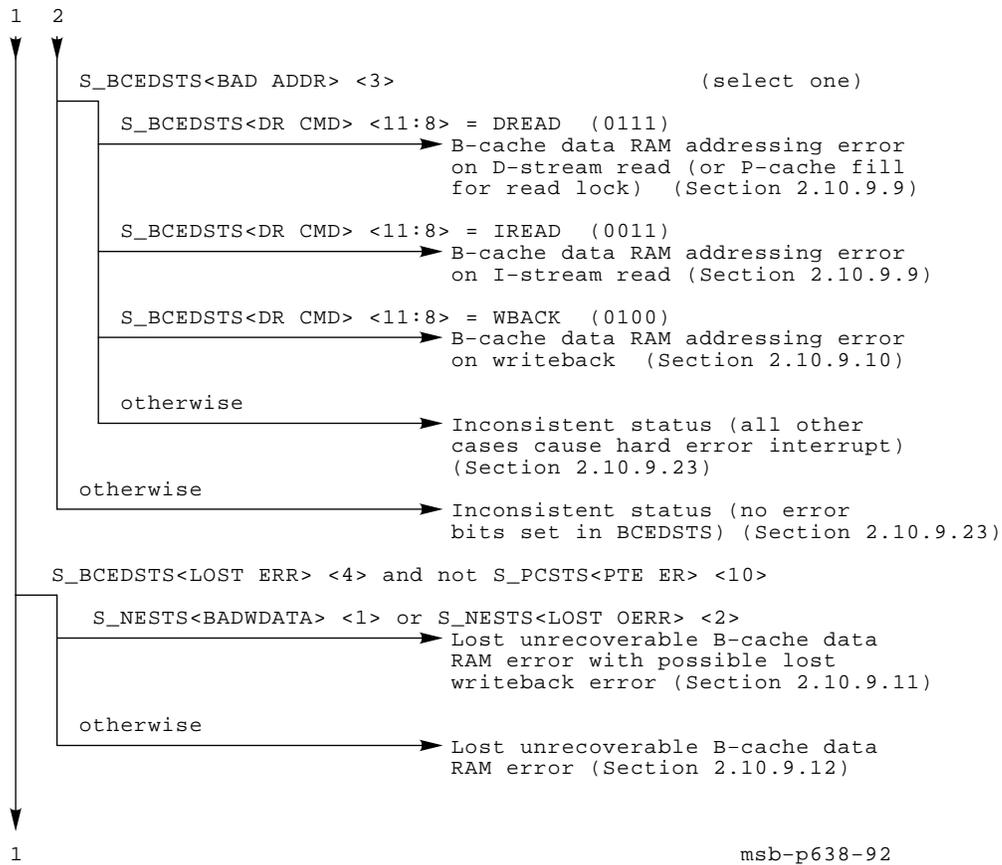


Figure 2-29 Cont'd on next page

Figure 2-29 (Cont.) Soft Error Interrupt Parse Tree

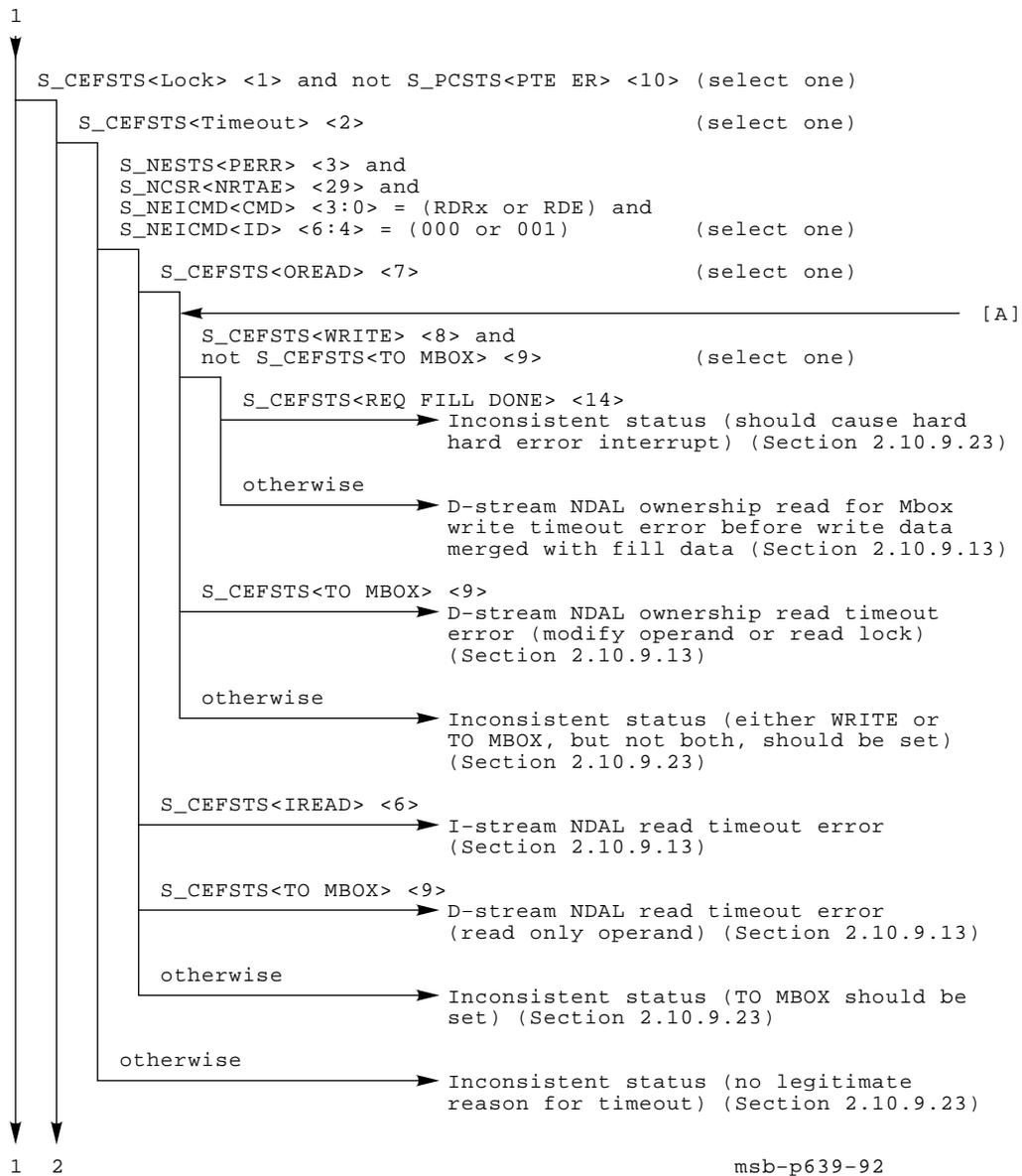


Figure 2-29 Cont'd on next page

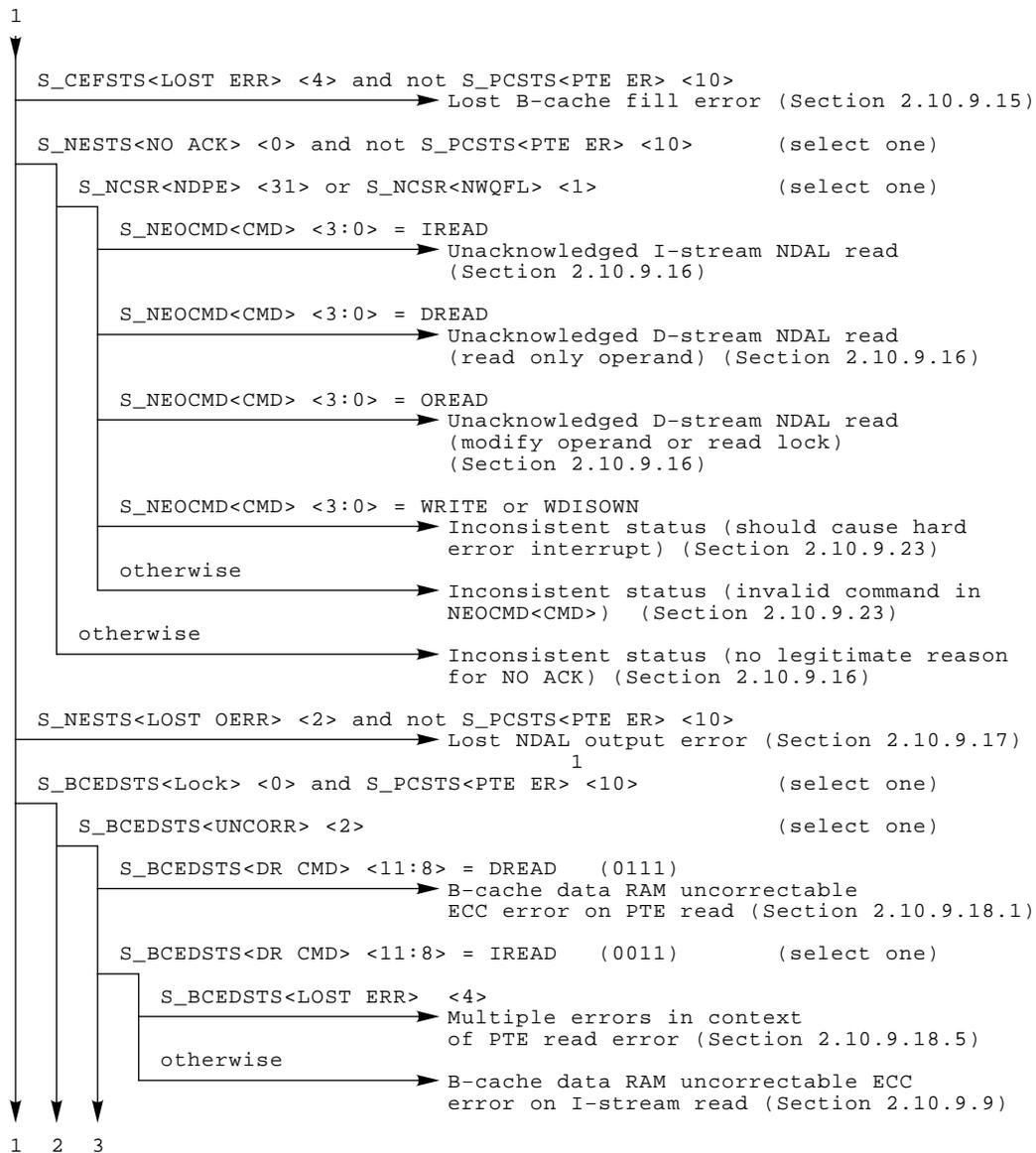
Figure 2-29 (Cont.) Soft Error Interrupt Parse Tree



1  
If this path is entered from the hard error parse tree, then add S\_XBER<PE> or S\_XBER<TTO> as other legal qualifiers.

Figure 2-29 Cont'd on next page

Figure 2–29 (Cont.) Soft Error Interrupt Parse Tree



msb-p641-92

1  
 At least one potential PTE cause must be found or the status is inconsistent (see Section 2.10.9.23). Some outcomes indicate a potential soft error interrupt cause, not a potential PTE read error cause. These errors should be treated separately.

Figure 2–29 Cont'd on next page

Figure 2–29 (Cont.) Soft Error Interrupt Parse Tree



1  
At least one potential PTE cause must be found or the status is inconsistent (see Section 2.10.9.23). Some outcomes indicate a potential soft error interrupt cause, not a potential PTE read error cause. These errors should be treated separately.

Figure 2–29 Cont'd on next page

Figure 2-29 (Cont.) Soft Error Interrupt Parse Tree

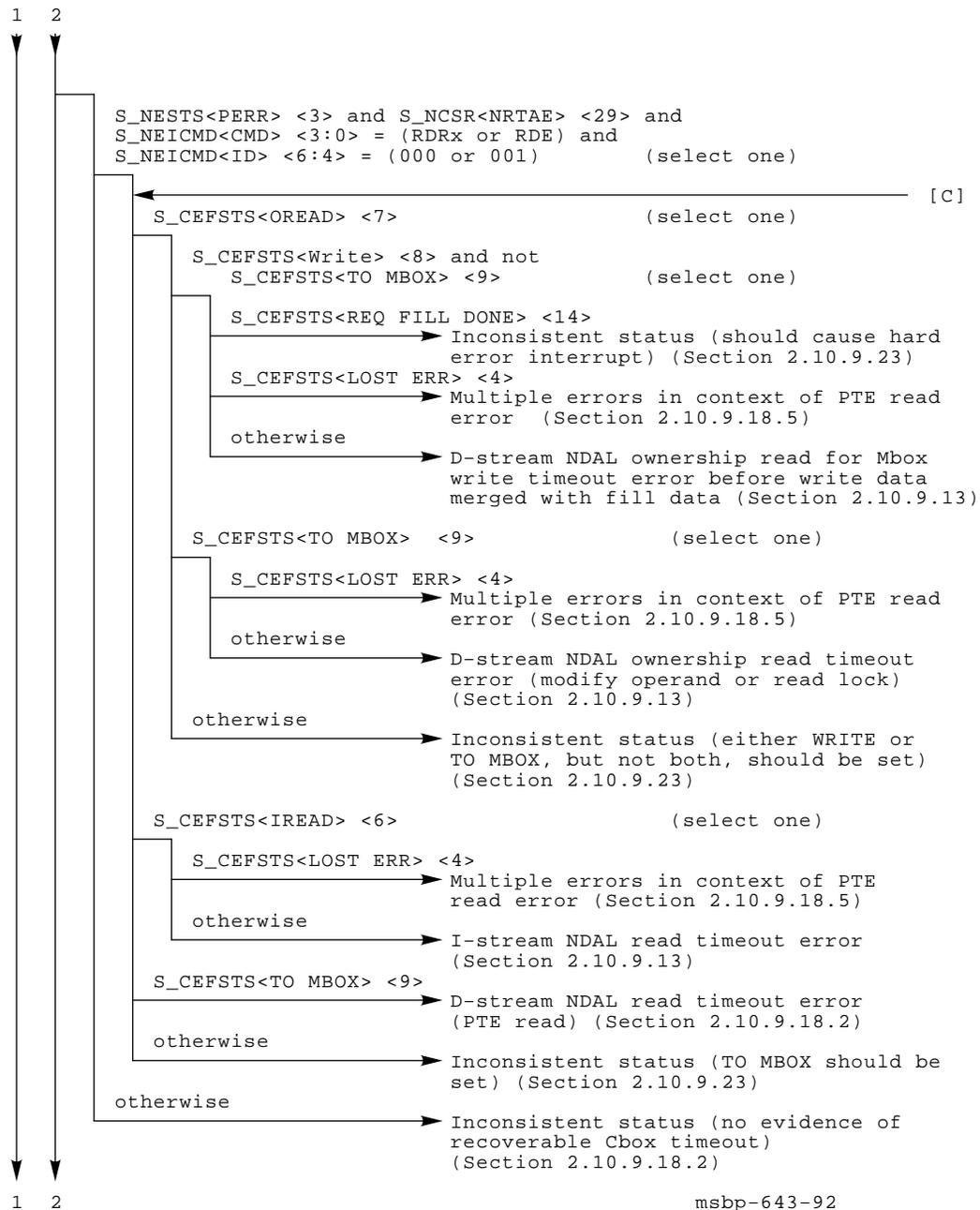


Figure 2-29 Cont'd on next page

Figure 2-29 (Cont.) Soft Error Interrupt Parse Tree

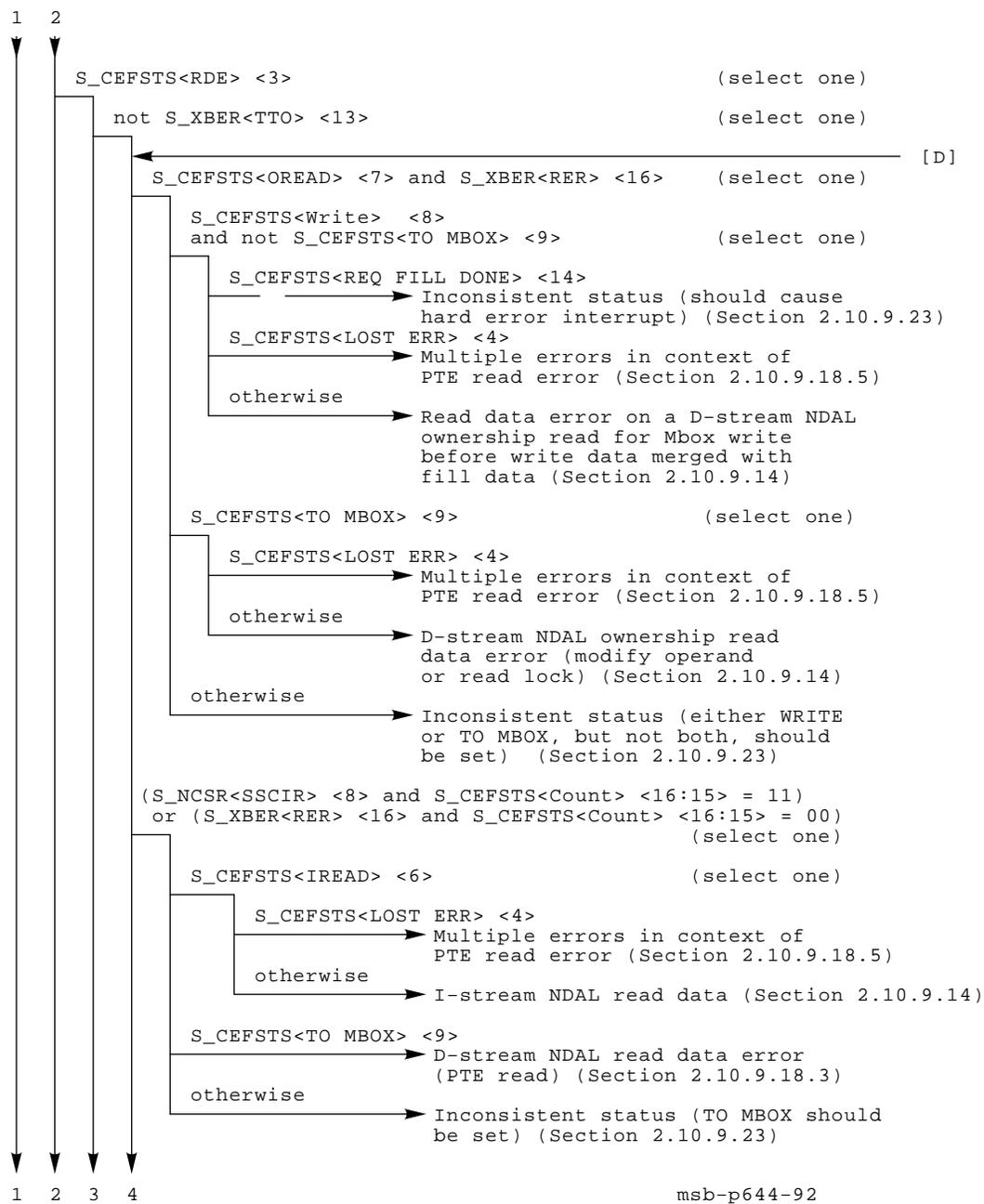
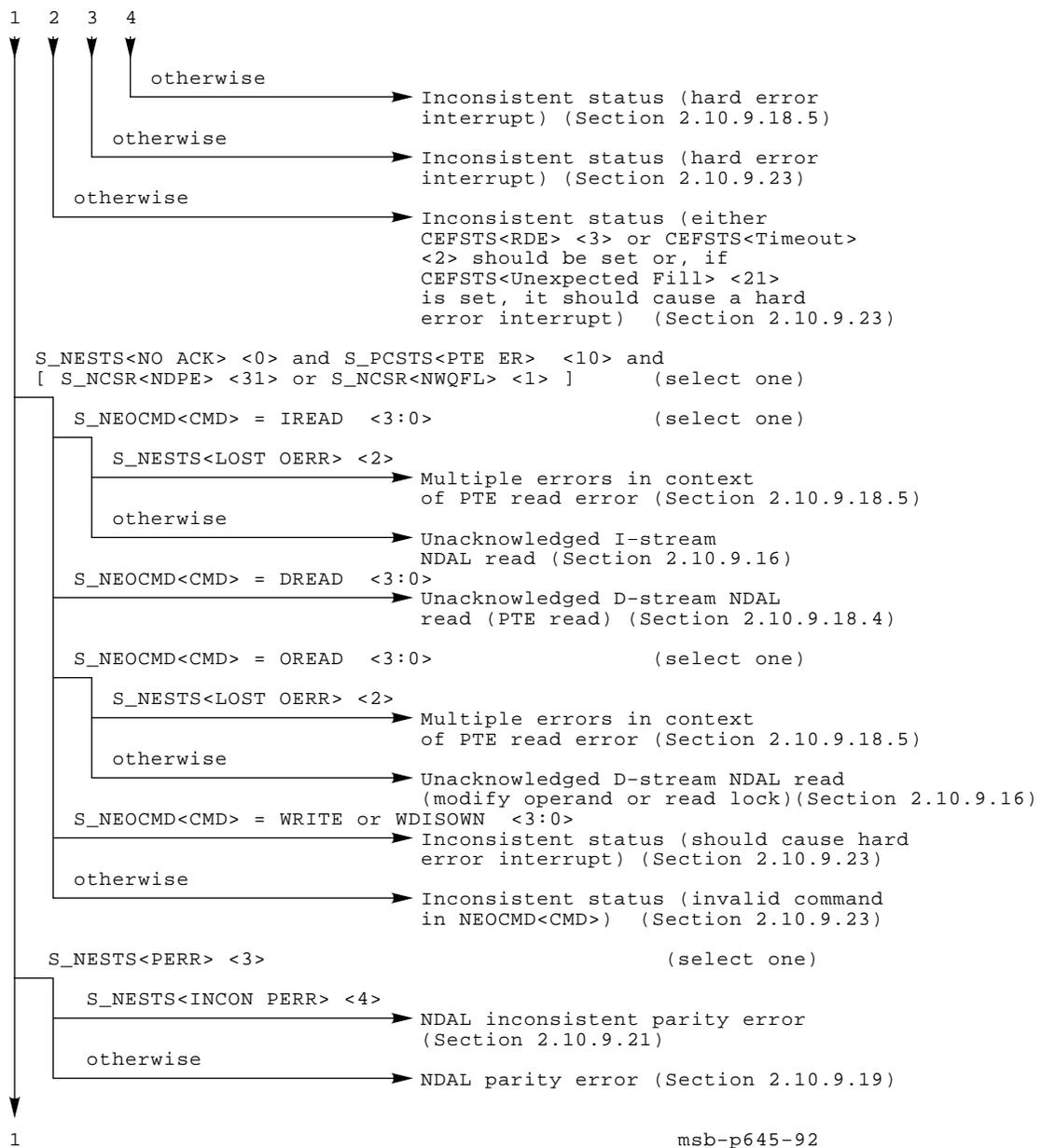


Figure 2-29 Cont'd on next page

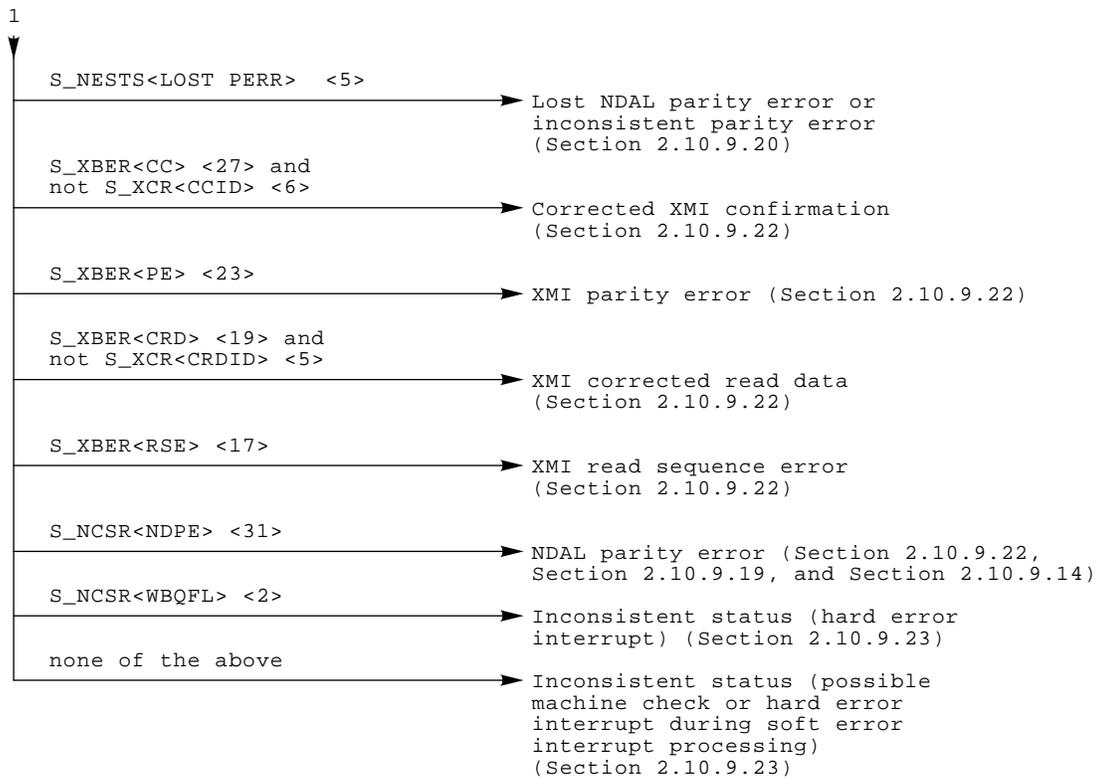
Figure 2–29 (Cont.) Soft Error Interrupt Parse Tree



At least one potential PTE cause must be found or the status is inconsistent (see Section 2.10.9.23). Some outcomes indicate a potential soft error interrupt cause, not a potential PTE read error cause. These errors should be treated separately.

Figure 2–29 Cont'd on next page

Figure 2–29 (Cont.) Soft Error Interrupt Parse Tree



msb-p646-92

---

**2.10.9.1 VIC Parity Errors**

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

The quadword virtual address of the error is in S\_VMAR.

*Recovery procedure:* Disable and flush the VIC by rewriting all the tags (see Section 2.10.3.2) and clear ICSR<Lock>.

---

**2.10.9.2 P-Cache Parity Errors**

*Description:* A parity error was detected in the P-cache. Either a tag parity error or a data parity error is reported, although tag parity errors in both the left and right banks may be reported simultaneously. The reference, whether it was a read or write, was passed to the Cbox as if the P-cache had missed. No data is lost. The P-cache is disabled because PCSTS<Lock> is set.

S\_PCADR contains the physical address of operation incurring the error. The address should not be in I/O space. If it is, it is an inconsistent status (see Section 2.10.9.23).

*Recovery procedure:* Clear PCSTS<Lock>. Flush the P-cache and initialize the P-cache tag store.

---

**2.10.9.3 B-Cache Tag Store Uncorrectable ECC Errors**

*Description:* An uncorrectable ECC error or an addressing error resulted from reading the B-cache tag store. The B-cache is in ETM. The hexword physical address of the transaction incurring the error is in S\_BCETIDX. (If the physical address is in I/O space, it is an inconsistent status. See Section 2.10.9.23.) S\_BCETAG contains the actual tag data and check bits read during the failing access. Check the tag data and determine the syndrome. The result of this check should give the result expected from S\_BCETSTS<UNCORR> and S\_BCETSTS<BAD ADDR>.

If both S\_BCETSTS<BAD ADDR> and S\_BCETSTS<UNCORR> are set, the status is inconsistent (see Section 2.10.9.23).

For any normal Mbox command (that is, not BCFLUSH), this error leads to a fill of the block whose tag had the error. This is because the Cbox converts uncorrectable tag store errors into misses and sends the associated reference to memory. For reads, the reference sent out is a Read or an Ownership Read, and when the data returns it is loaded in the B-cache. For writes, an Ownership Read is sent, and when the data returns the write is merged with it and is loaded in the B-cache. When the fill finishes successfully, the tag is updated (overwriting the bad tag). If the fill times out, the tag is not overwritten.

In some cases, this error leads to an NVAX CPU read timeout and/or a write timeout in memory. This occurs when the block was valid-owned in the B-cache and is the same block that is being accessed by the failing operation. Errors resulting from these lost blocks are handled separately.

Write unlocks are a special case. No tag lookup is done for write unlocks unless the B-cache is in ETM. If the B-cache is in ETM, and the tag store error occurs for that transaction, the write unlock is sent to memory.

*Recovery procedure (all cases):* Clear BCETSTS<Lock>. If it is an addressing error, clear BCETSTS<BAD ADDR>. Otherwise, clear BCETSTS<UNCORR>.

---

**2.10.9.3.1 Case: BCETSTS<TS CMD>=W UNLOCK**

*Recovery procedure:* Write an invalid tag with good ECC to the tag with the error (using the BCTAG access path). Then flush the B-cache and clear CCTL<HW ETM>. Software should prepare for another tag error during the B-cache flush by clearing BCETSTS of unrecoverable errors.

*Restart condition:* The B-cache was in ETM at the time the write unlock arrived. The data in memory may be corrupt and memory's ownership bit was cleared. Memory is corrupted at the location indicated by S\_BCETIDX. Software must determine if the error is fatal to one process or the whole system and take appropriate action.

---

**2.10.9.3.2 Case: BCETSTS<TS CMD>=DREAD, IREAD, OREAD**

*Recovery procedure:* First flush the B-cache and then clear CCTL<HW ETM>. Software should prepare for another tag error during the B-cache flush by clearing BCETSTS of unrecoverable errors. After flushing the B-cache, it is necessary to determine if any block is "lost." If a block's memory ownership bit is set and no writeback cache in the system has it owned, then the block is said to be lost. Use the procedure in Section 2.10.3.3.5. This procedure can result in finding no lost blocks, one lost block, or multiple lost blocks.

*Restart condition:* One lost block is not recoverable. Software must determine if the lost data is fatal to one process or the whole system and take appropriate action.

If multiple blocks are lost, crash the system.

---

**2.10.9.3.3 Case: BCETSTS<TS CMD>=R INVAL, O INVAL, IPR DEALLOCATE**

*Recovery procedure:* First flush the B-cache and then clear CCTL<HW ETM>. Software should prepare for another tag error during the B-cache flush by clearing BCETSTS of unrecoverable errors. After flushing the B-cache, it is necessary to determine if any block is "lost." If a block's memory ownership bit is set and no writeback cache in the system has it owned, then the block is said to be lost. Use the procedure in Section 2.10.3.3.5. This procedure can result in finding no lost blocks, one lost block, or multiple lost blocks.

If only one block is lost, memory's owner ID information indicates this CPU, write a valid-owned tag with the address of the lost block into the tag that had the error (using the BCTAG access means). Then flush this location to memory. An error could occur with this flush, in which case the data is not recoverable.

*Restart conditions:* If one block is lost, and the repair procedure did not incur an error, restart.

If the repair procedure was not successful, the data is not recoverable. Software must determine if the lost data was fatal to one process or the whole system and take appropriate action.

If multiple blocks are lost, crash the system.

---

## 2.10.9.4 Lost B-Cache Tag Store Errors

Some number of unrecoverable B-cache tag store errors occurred and were not latched because BCETSTS already contained a report of an unrecoverable error. All unrecoverable tag store errors cause soft error interrupts, so this is definitely a cause of the soft error interrupt.

Lost B-cache tag store errors may be caused by more than one operand prefetch to the same cache block.

The B-cache is in ETM.

Unrecoverable tag store errors can cause lost data by overwriting blocks in the B-cache.

Unrecoverable tag store errors in ETM on write unlocks can cause corrupted memory data.

*Recovery procedure:* Clear BCETSTS<LOST ERR>. Flush the B-cache and then clear CCTL<HW ETM>. Software should prepare for another tag error during the B-cache flush by clearing BCETSTS of unrecoverable errors.

*Restart condition:* Lost write unlock errors may have corrupted memory. Crash the system.

---

## 2.10.9.5 B-Cache Tag Store Correctable ECC Errors

*Description:* A correctable error occurred in accessing the B-cache tag store. The B-cache is not in ETM. S\_BCETIDX contains the physical address of the error. (If the physical address is in I/O space, it is an inconsistent status. See Section 2.10.9.23.) The index portion of S\_BCETIDX indicates which tag store entry had the error. S\_BCETAG contains the actual tag data and check bits read during the failing access. Check the tag data and determine the syndrome. The result of this check should be a correctable single-bit error.

*Recovery procedure:* Clear BCETSTS<CORR>.

If the operation was anything but a tag lookup for an explicit IPR deallocate operation (that is, BCFLUSH), software should flush that one location by writing the BCFLUSH IPR. Flushing the location invalidates it and forces the data to be written back to memory if it is owned. This can be done without putting the B-cache into ETM mode.

---

## 2.10.9.6 Lost B-Cache Tag Store Correctable ECC Errors

*Description:* A correctable error occurred in accessing the B-cache tag store, but it is lost because of an uncorrectable tag store error.

*Recovery procedure:* Clear BCETSTS<CORR>.

The B-cache should be flushed (and it would be because of the uncorrectable error in any case).

---

**2.10.9.7 B-Cache Data RAM Correctable ECC Errors**

*Description:* A correctable error occurred in accessing the B-cache data RAM. The B-cache is not in ETM. S\_BCEDIDX contains the cache index of the error, and S\_BCEDECC contains the syndrome calculated by the ECC logic. It is not possible to reliably determine the physical address of the error, since the B-cache is not in ETM and the block can be overwritten at any time after the error.

*Recovery procedure:* Clear BCEDSTS<CORR>.

If the operation was a read (S\_BCEDSTS<DR CMD>=DREAD or IREAD), software should flush that one location using the BCFLUSH IPR. Flushing the location invalidates it and forces the data to be written back to memory if it is owned. This can be done without putting the B-cache into ETM mode.

---

**2.10.9.8 Lost B-Cache Data RAM Correctable ECC Errors**

*Description:* A correctable error occurred in accessing the B-cache data RAM, but it is lost because of an uncorrectable data RAM error. The address and syndrome of the error are not known.

*Recovery procedure:* Clear BCEDSTS<CORR>.

The B-cache should be flushed. This effectively scrubs the B-cache data RAM location by invalidating it and forcing it to be written back if it is owned.

---

**2.10.9.9 B-Cache Data RAM Uncorrectable ECC Errors and Addressing Errors on I-Stream or D-Stream Reads**

*Description (addressing error):* A B-cache addressing error was detected by the Cbox in an I-stream or D-stream read during a B-cache hit. Addressing errors are the result of a mismatch between the address the Cbox drives to the RAMs for a read access and the address used to write that location. A multiple-bit data error can appear to be an addressing error, although it is extremely unlikely.

*Description (uncorrectable ECC error):* A B-cache uncorrectable ECC error was detected by the Cbox in an I-stream or D-stream read during a B-cache hit. Uncorrectable data errors are the result of a multiple-bit error in the data read from the B-cache. An addressing error with a single-bit data error will appear as an uncorrectable data error.

*Description (both cases):* The B-cache is in ETM. S\_BCEDIDX contains the cache index of the error, and S\_BCEDECC contains the syndrome calculated by the ECC logic. The physical address of the reference can be found by reading the tag for the data block. See Section 2.10.3.3.4 for a procedure to read the tag. (If the physical address is in I/O space, it is an inconsistent status. See Section 2.10.9.23.)

If the block's tag is found to contain an ECC error, then the address cannot be determined.

If both S\_BCEDSTS<BAD ADDR> and S\_BCEDSTS<UNCORR> are set, the status is inconsistent (see Section 2.10.9.23).

*Recovery procedure:* Clear BCEDSTS<Lock> and BCEDSTS<BAD ADDR> or BCEDSTS<UNCORR>.

Flush the B-cache and then clear CCTL<HW ETM>. If the data is owned by the B-cache and if the error repeats itself (is not transient), then a writeback error will result from the flush procedure. Software should prepare for this by clearing NESTS and BCEDSTS errors.

*Restart condition:* If a writeback error occurs in the B-cache flush, then the data is presumed to be unrecoverable. See the next section for a description of handling an error in a writeback. Software must determine if the error is fatal to one process or the whole system and take appropriate action.

If the address of the error in the flush is not the same as that of the original error, a multiple error exists in the data RAMs and is a serious failure. Crash the system.

---

#### 2.10.9.10 B-Cache Data RAM Uncorrectable ECC Errors and Addressing Errors on Writebacks

*Description (addressing error):* A B-cache addressing error was detected by the Cbox in a writeback. Addressing errors are the result of a mismatch between the address the Cbox drives to the RAMs for a read access and the address used to write that location. A multiple-bit data error can appear to be an addressing error, although it is extremely unlikely. The NDAL WDATA cycle was converted to a BADWDATA cycle. Memory should have tagged the location as bad and unreadable.

*Description (uncorrectable ECC error):* A B-cache uncorrectable ECC error was detected by the Cbox in a writeback. Uncorrectable data errors are the result of a multiple-bit error in the data read from the B-cache. An addressing error with a single-bit data error will appear as an uncorrectable data error. The NDAL WDATA cycle was converted to a BADWDATA cycle. Memory should have tagged the location as bad and unreadable.

*Description (both cases):* The B-cache is in ETM. S\_NESTS<BADWDATA> should be set. If it is not, and S\_NESTS<LOST OERR> and S\_NESTS<NO ACK> are not set, then the writeback that incurred the error is still in the writeback queue. Software should force the writeback queue to be drained (causing the second error event to occur) by reading from the Clear Write Buffer Register.

MFPR #PR19\$\_CWB,R0

After this, NESTS, NEOADR, and NEOCMD should be captured again.

If S\_NESTS<BADWDATA> is set, then S\_NEOADR contains the physical address of the lost writeback data. (If the physical address is found to be in I/O space, status is inconsistent. See Section 2.10.9.23.)

If S\_NESTS<BADWDATA> is not set but S\_NESTS<LOST OERR> is, then the address of the lost writeback data is not available.

If after draining the writeback queue, S\_NESTS<BADWDATA> is not set, then an inconsistency exists (see Section 2.10.9.23).

If both S\_BCEDSTS<BAD ADDR> and S\_BCEDSTS<UNCORR> are set, status is inconsistent (see Section 2.10.9.23).

*Recovery procedure:* Clear BCEDSTS<Lock> and NESTS<BADWDATA>, if it is set. If it is an addressing error, clear BCEDSTS<BAD ADDR>; otherwise clear BCEDSTS<UNCORR>. Flush the B-cache, then clear CCTL<HW ETM>. Then use the memory repair procedure to undo the tagged-bad data in memory (see Section 2.10.3.3.2.2).

**NOTE: When clearing the tagged-bad data state of memory, software must ensure that no more accesses to the block can occur. Otherwise, a process on another processor or a DMA I/O device could see incorrect data and not detect an error.**

*Restart conditions:* If the data is lost, software must determine if the error is fatal to one process or the whole system and take appropriate action. If the address of the lost data cannot be determined, crash the system.

---

#### 2.10.9.11 Lost B-Cache Data RAM Errors with Possible Lost Writebacks

*Description:* Lost B-cache data RAM errors that cause a soft error interrupt (when S\_NESTS indicates the possibility of a lost writeback error) indicate that data errors occurred on reads or writebacks, but no new write data was lost. S\_NESTS reports the writeback error, unless multiple NDAL output errors have occurred.

The B-cache is in ETM.

Lost B-cache data RAM errors of this kind can be caused by an operand prefetch from a B-cache block followed by a write to the same block.

If S\_NESTS<BADWDATA> is set, then S\_NEOADR contains the physical address of a writeback. (If the physical address is in I/O space, it is an inconsistent status. See Section 2.10.9.23.)

*Recovery procedure:* Clear BCEDSTS<LOST OERR>. Flush the B-cache and then clear CCTL<HW ETM>. Writeback errors can occur during the flush. Software should prepare for this by clearing NESTS and BCEDSTS errors.

If S\_NESTS<BADWDATA> is set, clear NESTS<BADWDATA>. Use the memory repair procedure to undo the tagged-bad data in memory (see Section 2.10.3.3.2.2). The B-cache must be flushed before this repair procedure.

**NOTE: When clearing the tagged-bad data state of memory, software must ensure that no more accesses to the block can occur. Otherwise, a process on another processor or a DMA I/O device could see incorrect data and not detect an error.**

*Restart condition (S\_NESTS<LOST OERR> set):* There is no way to determine how many writebacks failed. They all should have gone to memory with BADWDATA cycles, where memory would have them marked as tagged-bad data. So an unknown block may be tagged-bad in memory. If so, the next access to that block could come from the system itself, even if it "belonged" only to one process. This will cause the system to crash. But there is a chance that the next access will come from a user process. This would allow the system to stay up, although that process would have to be stopped.

If the system's implementation of tagged-bad data is not reliable (see Section 2.10.10, Note on Tagged-Bad Data Mechanisms), software should crash the system. If it is reliable, restart.

*Restart condition (S\_NESTS<LOST OERR> not set):* The writeback data is lost but the address is known. Software must determine if the error is fatal to one process or the whole system and take appropriate action.

---

#### 2.10.9.12 Lost B-Cache Data RAM Errors Without Lost Writebacks

*Description:* Lost B-cache data RAM errors that cause only a soft error interrupt (when S\_NESTS indicates no possibility of writeback error) indicate that data errors occurred on reads. No write data was lost.

Lost B-cache data RAM errors may be caused by more than one operand prefetch to the same cache block.

The B-cache is in ETM.

*Recovery procedure:* Clear BCEDSTS<LOST OERR>. Flush the B-cache and then clear CCTL<HW ETM>. Writeback errors may occur during the flush. Software should prepare for this by clearing NESTS and BCEDSTS errors.

*Restart condition:* Only reads from the B-cache failed. Restart is possible unless any error encountered during a B-cache flush is fatal.

---

#### 2.10.9.13 NDAL I-Stream or D-Stream Read or D-Stream Ownership Read Timeout Errors

*Description:* An I-stream or D-stream read or D-stream ownership read timed out in the Cbox before all the fill quadwords were received. This is *not* the method by which the system interface (NEXMI) will notify the NVAX CPU that a location is inaccessible. All outstanding NDAL read-type cycles (IREAD, DREAD, OREAD) are normally terminated by at least one return data cycle, either RDRx or RDE.

Thus, the Cbox timeout error can only be caused by a serious system error, or an NDAL parity error on the returned data. S\_CEFSTS<Count> indicates the number of quadwords received before the error and should always be 11 (binary) if the address is in I/O space. The physical address is in S\_CEFADR. Table 2-40 shows the cycle type that timed out, based upon the error bits in S\_CEFSTS during error analysis.

**Table 2-40 S\_CEFSTS Cycle Type Decode**

Command Type		<IREAD>	<OREAD>	<WRITE>	<TO MBOX>
IREAD		1	0	0	1
IREAD	(aborted)	1	0	0	0
DREAD		0	0	0	1
OREAD	(for read-modify or read lock)	0	1	0	1
OREAD	(for write)	0	1	1	0

- I-stream or D-stream read  
The B-cache is in ETM, since the parity error is assumed to have been sensed by the NVAX.
- D-stream ownership read  
The B-cache is in ETM. No write data has been merged with the returning fills.

The address should not be in I/O space. If it is, it is an inconsistent status (see Section 2.10.9.23).

If the ownership read was for an Mbox write, the write was sent on the NDAL after the OREAD timed out.

If the ownership read was for a read lock, the corresponding write unlock should have been received from the Ebox. The write unlock is sent as a quadword WDISOWN by the Cbox, so no memory location is left owned. (If the error was on the requested quadword, a machine check would have resulted. If a separate error prevents the write unlock, that will be reported in other error registers.)

*Recovery procedure (all cases):* Clear CEFSTS<Lock, Timeout>, NESTS<PERR>, and NCSR<NRTAE>.

Since the NEXMI always returns the requested data word first, S\_CEFSTS<REQ FILL DONE> should be set if and only if S\_CEFSTS<Count> shows that at least one data word has been returned. If the status of the two is inconsistent, no recovery is possible. Crash the system.

In terms of a soft error interrupt, an NDAL parity error is the only potentially recoverable reason for the Cbox timeout to occur (there is one hard error interrupt scenario that would lead to a timeout of this kind, but this analysis assumes that only a soft error has occurred).

Other error registers must be inspected to ensure that they are consistent with the assumption that a parity error has occurred during one of the return data words. See Section 2.10.9.19 for more information on what to look for when attempting diagnosis of a parity error that could cause this timeout. The following error indications show that the status is inconsistent, and they imply that the parity error might not be the cause of the problem. If any of these are true, crash the system.

- The command in progress was *not* a read.
- The parity error was not the type that would cause a timeout.
- No parity error was logged in S\_NESTS<PERR>.
- S\_NESTS<INCON PERR>, S\_NESTS<LOST PERR>, or S\_NCSR<NDIPE> is set.
- S\_NCSR<NRTAE> is *not* set.
- The returned data count does not match S\_CEFSTS<REQ FILL DONE>.

The ownership bit in the XMI memory will be set if the transaction was ACKed on the XMI. A continued NO ACK response from the memory causes the NEXMI to time out and send back an RDE to the NVAX long before the Cbox timeout has occurred. But if the RDE was sent back as a nonexistent memory response, a hard error interrupt would be generated, and a soft interrupt analysis would then imply inconsistent status. In that case, crash the system.

Assuming that the analysis passes this consistency check, the response that was lost due to the parity error must have been a legitimate return data word: RDRx or RDE. An RDE would represent a word in memory that was uncorrected, rather than a NEXMI timeout. This further implies that the memory, having ACKed the transaction, set the ownership bit for an NDAL/XMI OREAD command.

*Additional recovery procedures for D-stream ownership read (S\_CEFSTS<Write> set):* First flush the B-cache and then clear CCTL<HW ETM>.

The memory is assumed to have set the ownership bit for this read, since it originally ACKed the read command. This means that the write data must have been lost, and a hard error interrupt is expected. Use the system procedure for resetting the ownership bit in memory.

*Additional recovery procedures for D-stream ownership read (S\_CEFSTS<Write> not set):* First flush the B-cache and then clear CCTL<HW ETM>.

Memory will have set the ownership bit, but the data is presumably still good. The B-cache block is marked invalid in the B-cache tag store. However, if the error occurred on a read lock, the corresponding write unlock should have occurred and it will have cleared the ownership bit for this block in memory.

If S\_CEFSTS<Count> is greater than 0, then part of the data is in the B-cache. If S\_CEFSTS<REQ FILL DONE> is set, then the quadword in the B-cache block pointed to by S\_CEFADR is valid (except in the case of a read lock, but the data should not be needed for memory repair in that case).

If S\_CEFSTS<Count> is greater than 0, and if the address in S\_CEFADR is not in I/O space, then the block was not owned before the operation began. In that case, use the system procedures (see Section 2.10.3.3.2.1) to determine if memory's ownership bit is set, and this CPU owns the block. If so, use the procedure in Section 2.10.3.3.2.2 to reset it.

*Restart condition:* Restart if the memory state repair procedure is successful (or no repair is necessary), no data is lost, and the address is not in I/O space. If the hexword block could not be repaired or data is lost, software must determine if the error is fatal to one process or the whole system and take appropriate action.

*Post restart recovery:* If the same fill error recurs on restart, then the block is probably "lost." <sup>1</sup> Software must determine if the error is fatal to one process or the whole system and take appropriate action. If it is fatal to only one process, use the system procedure for resetting the ownership bit in memory.

**NOTE: It may be appropriate to cause each CPU in the system to flush its B-cache, and then restart. It may also be that another error (such as an uncorrectable tag store error on a coherence request) will be repaired by the soft error interrupt handler before the restart actually occurs, thus repairing the cause of the fill error.**

#### 2.10.9.14 NDAL I-Stream or D-Stream Read or D-Stream Ownership Read Data Errors

*Description:* An I-stream or D-stream read or D-stream ownership read terminated with an RDE (read data error) NDAL cycle before all the fill quadwords were received. S\_CEFSTS<Count> indicates the number of quadwords received before the error. (S\_CEFSTS<Count> should always be 11 (binary) if the address is in I/O space.) If S\_CEFSTS<Count> is 0 or the address is an I/O space address, then the first data word returned was an RDE.

There are several reasons why the NEXMI might send back an RDE in response to an NVAX read command, and for this to be consistent state in the analysis of a soft error interrupt:

- The NVAX attempted a read command to a nonexistent memory location (NXM) in system support space. The RDE will be returned as the first data word. S\_NCSR<SSCIR> is set. S\_CEFSTS<Count> equals 11, and the S\_CEFADR shows an address in I/O space.
- A previous BADWDATA was written to that block in memory, tagging it bad for all future reads (until it is cleared by the system). This would return an XMI RER as the first (and only) data word, and this would be translated to an NDAL RDE. S\_XBER<RER> is set for this type of error. S\_CEFSTS<Count> equals 00, since this is in memory space.
- A previous system error, such as a memory location being corrupted and causing an ECC syndrome miscompare, could return an RDE on any of the words. If the S\_CEFSTS<Count> shows that the RDE was *not* the first word returned, then only a memory error could be the cause. If the count shows that it *was* the first word, then a corrupted data word in the memory is only one possibility.
- The data returned by a responder (such as an MS65A memory) was not in the correct sequence. An XMI parity error could cause this bit to be set, since a data word would then be missed. In that case, S\_XBER<RSE> and S\_XBER<PE> are set.

<sup>1</sup> In this case the more general sense of "lost" is implied. That is, memory's ownership bit is set, but no cache writes the data back when a read is done to that location. It is possible to identify which CPU owns the block (or rather, which CPU is thought to own it according to the memory), but it is often not possible to determine which error caused the situation to arise.

For the RSE error to be consistent with a soft error interrupt, the parity error must have struck the third return data word. Any other missed return data word would cause a hard error interrupt due to other error bits. So, for the RSE to be analyzed here, S\_CEFSTS<Count> must equal 11 (waiting for the last word).

See Table 2–40 for a listing of the cycle types and their error bit decode meaning, based upon the bits set in S\_CEFSTS. For all the cases above, the physical address is in S\_CEFADR.

- I-stream or D-stream read  
The B-cache is not in ETM.
- D-stream ownership read  
The B-cache is in ETM. No write data has been merged with the returning fills.

If the address is in I/O space, status is inconsistent (see Section 2.10.9.23).

If the Ownership Read was for an Mbox write, the write was sent on the NDAL after the OREAD was aborted.

If the Ownership Read was for a read lock, the corresponding write unlock should have been received from the Ebox. The write unlock is sent as a quadword WDISOWN by the Cbox, so no memory location is left owned. (If the error was on the requested quadword, a machine check would have resulted. If a separate error prevents the write unlock, that will be reported in other error registers.)

*Recovery procedure (all cases):* Clear CEFSTS<Lock, RDE>, and either XBER<RER>, NCSR<SSCIR>, XBER<RSE> and/or XBER<PE> (whichever is appropriate).

*Additional recovery procedures for D-stream ownership read (S\_CEFSTS<Write> set):* Flush the B-cache and then clear CCTL<HW ETM>.

It is assumed that the memory ACKed the transaction, since a NO ACK command would result in a hard error interrupt, not a soft error interrupt. Thus, the ownership bit for this block will remain set in memory. In that case, the write data must have been lost, and a hard error interrupt is expected anyway. This analysis is thus inconsistent. Crash the system.

*Additional recovery procedures for D-stream ownership read (S\_CEFSTS<Write> not set):* Flush the B-cache and then clear CCTL<HW ETM>.

Again, the memory must have ACKed the transaction for this analysis to be consistent. As such, the ownership bit for this block will remain set in memory. The data in memory could still be good. The B-cache block is marked invalid in the B-cache tag store. However, if the error occurred on a read lock, the corresponding write unlock should have occurred and it will have cleared the ownership bit for this block.

If S\_CEFSTS<Count> is greater than 0 (and S\_CEFADR is not in I/O space), then part of the data is in the B-cache. The XMI memory *always* returns the requested word first. So, if S\_CEFSTS<REQ FILL DONE> is also set, then the quadword in the B-cache block pointed to by S\_CEFADR is valid (except in the case of a read lock, but the data should not be

needed for memory repair in that case). If S\_CEFSTS<REQ FILL DONE> is not set, then the status is inconsistent. Crash the system.

If S\_CEFSTS<Count> is greater than 0 (and S\_CEFADR is not in I/O space), then it is also known that the block was not owned before the operation began. In this case, use the procedures in Section 2.10.3.3.2 to determine if memory's ownership bit is set, and if it is, use the system procedure (see Section 2.10.3.3.2.2) to reset it. The easiest way to do this is to write a quadword of correct data back to the memory in the process of resetting the ownership bit. Section 2.10.3.3.3 describes procedures for extracting data from the B-cache data RAMs in this case.

If memory's ownership bit was left set as a result of this error and no nondestructive procedure exists for restoring it, then the hexword block is lost.

*Restart condition:* Restart if the memory state is not in error or if the repair procedure is successful, no data is lost, and the address is not in I/O space. If the hexword block could not be repaired or data is lost, software must determine if the error is fatal to one process or the whole system and take appropriate action.

*Post restart recovery:* If the same fill error recurs on restart, then the block is probably "lost."<sup>1</sup> Software must determine if the error is fatal to one process or the whole system and take appropriate action. (If it is fatal to only one process, use the system procedure for resetting the ownership bit in memory.)

**NOTE: It may be appropriate to cause each CPU in the system to flush its B-cache, and then restart. It may also be that another error (such as an uncorrectable tag store error on a coherence request) will be repaired by the soft error interrupt handler before the restart actually occurs, thus repairing the cause of the fill error.**

---

#### 2.10.9.15 Lost B-Cache Fill Error

*Description:* Some number of fill errors occurred and were not latched because CEFSTS and CEFADR already contained a report of an unrecoverable error. Lost B-cache fill errors that do not cause hard error interrupts are always read errors.

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

Lost B-cache fill errors may leave blocks marked owned by this CPU in memory without the B-cache actually owning the block.

The B-cache may be in ETM. Read S\_CCTL<HW ETM> to find out.

*Recovery procedure:* Clear CEFSTS<LOST ERR>. If the B-cache is in ETM, flush the B-cache and then clear CCTL<HW ETM>.

---

<sup>1</sup> In this case the more general sense of "lost" is implied. That is, memory's ownership bit is set, but no cache writes the data back when a read is done to that location. It is possible to identify which CPU owns the block (or rather, which CPU is thought to own it according to the memory), but it is often not possible to determine which error caused the situation to arise.

*Restart condition:* Although lost B-cache fill errors may leave blocks marked owned by this CPU in memory without the B-cache actually owning the block, XMI systems maintain reliable ownership bits and it is safe to restart.

In the absence of additional errors, the memory/cache ownership mechanism ensures that no other process can access the block whose ownership bit is set in memory and is not owned by any cache. Cache coherence in the system depends on this mechanism.

---

#### 2.10.9.16 Unacknowledged NDAL I-Stream or D-Stream Read or D-Stream Ownership Read

*Description:* An I-stream or D-stream read or D-stream ownership read was NO ACKed by the NEXMI. The physical address is in S\_CEFADR. The NEXMI will generally ACK any legal command on the NDAL, without regard to the address. If the address turns out to be nonexistent, or if some other error prevents return read data, an RDE will be returned to the NVAX long before the Cbox times out. Potential reasons for an NDAL bus NO ACK are:

- An NDAL parity error was sensed by the NEXMI during an NVAX command transfer cycle. S\_NESTS<NO ACK> should be set, S\_NEOCMD will contain the command that was refused, and S\_NEOADR will contain the address. S\_NCSR<NDPE> should also be set, since the NEXMI is assumed to have NO ACKed the cycle due to a parity error. If the NVAX also sensed the parity error (S\_NESTS<PERR> is set), then more information is available in the S\_NEICMD, S\_NEDATLO registers. This is discussed in Section 2.10.9.19.
- The NEXMI refused the command because the non-writeback queue was full. This should be prevented by the NEXMI's control of CPU GRANT L, but if it does happen, S\_NCSR<NWQFL> will be set. If the command is a read, recovery is possible.

For I-stream and D-stream reads the B-cache is not in ETM. For D-stream ownership reads the B-cache is in ETM.

If the address is in I/O space, status is inconsistent (see Section 2.10.9.23).

If the ownership read was for an Mbox write, the write was sent on the NDAL after the OREAD timed out. If the write was also NO ACKed, a hard error interrupt would have been posted. That is handled as a separate error.

*Recovery procedure (all cases):* Clear NESTS<NO ACK> and either S\_NCSR<NDPE> or S\_NCSR<NWQFL>.

*Additional recovery procedure for D-stream ownership read:* Flush the B-cache and then clear CCTL<HW ETM>. No error is expected during the B-cache flush.

---

**2.10.9.17 Lost NDAL Output Error**

*Description:* Some number of NDAL output errors occurred. Some number of read NO ACKs and/or BADWDATAs were missed. A hard error interrupt would have occurred if a write or writeback was NO ACKed.

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

The B-cache may be in ETM. Read S\_CCTL<HW ETM> to find out.

*Recovery procedure:* Clear NESTS<LOST OERR>. If CCTL<HW ETM> is set, flush the B-cache and then clear CCTL<HW ETM>.

*Restart conditions:* Lost NDAL output errors may leave tagged bad locations in memory. Restart (see Section 2.10.10, Note on Tagged-Bad Data Mechanisms).

---

**2.10.9.18 PTE Read Errors**

PTE read errors are read errors that happen in reads issued by the Mbox in handling a TB miss. Handling of these errors differs from handling the same underlying error (B-cache data RAM error, B-cache fill error, or NDAL NO ACK error) when PTE read is not the cause.

If S\_PCSTS<PTE ER> is set, then a PTE read issued by the Mbox in processing a TB miss had an unrecoverable error. The TB miss sequence was aborted because of the error. The original reference can be any I-stream or D-stream read or write.

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

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

---

**2.10.9.18.1 B-Cache Data RAM Uncorrectable ECC Errors and Addressing Errors on PTE Reads**

*Description (addressing errors):* A B-cache addressing error was detected by the Cbox in a PTE read during a B-cache hit. Addressing errors are the result of a mismatch between the address the Cbox drives to the RAMs for a read access and the address used to write that location. A multiple-bit data error can appear to be an addressing error, although it is extremely unlikely.

*Description (uncorrectable ECC errors):* A B-cache uncorrectable data error was detected by the Cbox in a PTE read during a B-cache hit. Uncorrectable data errors are the result of a multiple-bit error in the data read from the B-cache. An addressing error with a single-bit data error will appear as an uncorrectable data error.

*Description (all cases):* The B-cache is in ETM. S\_BCEDIDX contains the cache index of the error, and S\_BCEDECC contains the syndrome calculated by the ECC logic. The physical address of the PTE read can be found by reading the tag for the data block (using the procedure in Section 2.10.3.3.4). (If the physical address is in I/O space, status is inconsistent. See Section 2.10.9.23.)

If the block's tag contains an ECC error, the address cannot be determined.

S\_BCEDSTS<LOST ERR> may be set. This lost error is probably due to the same PTE error occurring more than once. This is an acceptable assumption unless a hard error interrupt occurs after handling this error.

If both S\_BCEDSTS<BAD ADDR> and S\_BCEDSTS<UNCORR> are set, the status is inconsistent (Section 2.10.9.23).

*Recovery procedure (addressing errors):* Clear BCEDSTS<Lock, BAD ADDR>.

*Recovery procedure (uncorrectable ECC errors):* Clear BCEDSTS<Lock, UNCORR>.

*Recovery procedure (both cases):* Flush the B-cache and then clear CCTL<HW ETM>. Clear PCSTS<PTE ER>. If the data is owned by the B-cache and if the error repeats itself (is not transient), a writeback error will result from the flush procedure. Software should prepare for this by clearing NESTS and BCEDSTS errors.

*Restart condition:* If no writeback error occurs in the B-cache flush, restart if:

$$(S\_PCSTS<PTE ER WR> = 0)$$

The system should be reset if:

$$(S\_PCSTS<PTE ER WR> = 1)$$

If a writeback error occurs in the B-cache flush, the data is presumed to be unrecoverable. See Section 2.10.9.10 for a description of handling an error in a writeback. Software must determine if the error is fatal to one process or the whole system and take appropriate action.

### 2.10.9.18.2 NDAL PTE Read Timeout Errors

*Description:* A PTE read timed out in the Cbox before *any* fill quadword was received. This is not the method by which the NEXMI will notify the NVAX CPU that a location is inaccessible. All outstanding NDAL read-type cycles are normally terminated by at least one return data cycle, either RDRx or RDE. The only cause of a Cbox timeout that is both consistent with a soft error interrupt analysis and recoverable is an NDAL parity error on the returned data. S\_CEFSTS<Count> indicates the number of quadwords received before the error (and should always be 11 (binary) if the address is in I/O space).

Section 2.10.9.13 discusses Cbox timeout errors in the context of non-PTE errors, but some of that general discussion applies here as well. The system environment analysis for this kind of timeout is the same, since there is no distinction outside the NVAX between a fill read and a PTE read. Section 2.10.9.19 has a more complete discussion of NDAL parity errors and how they should be analyzed. Table 2-40 contains information

about decoding the S\_CEFSTS bits and determining what was in progress during the error.

If CEFSTS<Write> is set, status is inconsistent (see Section 2.10.6.7).

The physical address of the PTE is in S\_CEFADR. The B-cache is not in ETM. The read could not have been an Ownership Read, so this error did not cause the ownership bits in memory to be left in the wrong state.

S\_CEFSTS<LOST ERR> may be set. This error is probably due to the same PTE error occurring more than once. This is an acceptable assumption unless a hard error interrupt occurs after handling this error.

*Recovery procedure:* Clear CEFSTS<Lock, Timeout>, PCSTS<PTE ER>, NESTS<PERR>, and NCSR<NRTAE>.

*Restart condition:* Restart if:

$$(S\_PCSTS<PTE ER WR> = 0).$$

Otherwise, reset the system.

*Post restart recovery:* If the same fill error recurs on restart, then the block is probably "lost." <sup>1</sup> Software must determine if the error is fatal to one process or the whole system and take appropriate action. If it is fatal to only one process, use the system procedure for resetting memory's ownership bit.

**NOTE: It may be appropriate to cause each CPU in the system to flush its B-cache, and then restart. It may also be that another error (such as an uncorrectable tag store error on a coherence request) will be repaired by the soft error interrupt handler before the restart actually occurs, thus repairing the cause of the fill error.**

### 2.10.9.18.3 NDAL PTE Read Data Errors

*Description:* A PTE read ended with an RDE (read data error) NDAL cycle before any of the fill quadwords were received. S\_CEFSTS<Count> indicates the number of quadwords received before the error. S\_CEFSTS<Count> should be 0 (binary) for a memory space address, or 11 for an I/O space address, since the first word returned was an RDE. The physical address is in S\_CEFADR. Section 2.10.9.14 contains a more complete discussion about this type of error in a non-PTE context. The system environment analysis for an RDE is almost identical, except only those cases that return an RDE as the first word apply here. The reasons that the first return data word could be an RDE, and still be a soft error interrupt are:

- The NVAX attempted a read command to a nonexistent memory location in system support space. S\_NCSR<SSCIR> will be set.
- A previous BADWDATA was written to that block in memory, tagging it bad for all future reads.

<sup>1</sup> In this case the more general sense of "lost" is implied. That is, memory's ownership bit is set, but no cache writes the data back when a read is done to that location. It is possible to identify which CPU actually owns the block (or rather, which CPU is thought to own it according to the memory), but it is often not possible to determine which error caused the situation to arise.

- A previous system error, such as a memory location being corrupted and causing an ECC syndrome mismatch, was sensed on the first quadword requested. This could be any quadword within the hexword, since the requested quadword is always returned first.

If CEFSTS<Write> is set, status is inconsistent (see Section 2.10.6.7).

The physical address of the PTE is in S\_CEFADR. The B-cache is not in ETM. The read could not have been an Ownership Read, so this error did not cause the ownership bits in memory to be left in the wrong state.

S\_CEFSTS<LOST ERR> may be set. This error is probably due to the same PTE error occurring more than once. This is an acceptable assumption unless a hard error interrupt occurs after handling this error.

*Recovery procedure:* Clear CEFSTS<Lock, RDE>, PCSTS<PTE ER>, and either XBER<RER> or NCSR<SSCIR> (whichever is appropriate).

*Restart condition:* Restart if:

$$(S\_PCSTS<PTE ER WR> = 0).$$

Otherwise, reset the system.

*Post restart recovery:* If the same fill error recurs on restart, then the block is probably "lost." <sup>1</sup> Software must determine if the error is fatal to one process or the whole system and take appropriate action. (If it is fatal to only one process, use the system procedure for resetting memory's ownership bit.)

**NOTE: It may be appropriate to cause each CPU in the system to flush its B-cache, and then restart. It may also be that another error (such as an uncorrectable tag store error on a coherence request) will be repaired by the soft error interrupt handler before the restart actually occurs, thus repairing the cause of the fill error.**

#### 2.10.9.18.4 Unacknowledged NDAL PTE Read

*Description:* A PTE read was NO ACKed by the NEXMI. The NEXMI will generally ACK any legal command on the NDAL, without regard to the address. If the address turns out to be nonexistent, or if some other error prevents return read data, an RDE will be returned to the NVAX long before the Cbox times out. So, the most likely reason for this error is an NDAL parity error.

The physical address of the PTE is in S\_NEOADR. The B-cache is not in ETM.

Refer to Section 2.10.9.16 for a more general discussion of an unacknowledged read, and Section 2.10.9.19 for a more complete analysis of an NDAL parity error. The system environment response to a PTE read is a subset of the response to the more general I-stream or D-stream read command.

<sup>1</sup> In this case the more general sense of "lost" is implied. That is, memory's ownership bit is set, but no cache writes the data back when a read is done to that location. It is possible to identify which CPU actually owns the block (or rather, which CPU is thought to own it according to the memory), but it is often not possible to determine which error caused the situation to arise.

S\_CEFSTS<LOST OERR> may be set. This error is probably due to the same PTE error occurring more than once. This is an acceptable assumption unless a hard error interrupt occurs after handling this error.

*Recovery procedure:* Clear NESTS<NO ACK>, PCSTS<PTE ER>, and either NCSR<NDPE> or NCSR<NWQFL> (whichever is appropriate).

*Restart condition:* Restart if:

(PCSTS<PTE ER WR> = 0).

Otherwise, reset the system.

---

#### 2.10.9.18.5 Multiple Errors That Interfere with Analysis of PTE Read Errors

Because PTE read errors lead to several unusual cases, restart is not recommended in the event that other errors cloud the analysis of the PTE read error.

*Recovery procedure:* No specific recovery action is called for.

*Restart condition:* Restart is not possible. Reset the system.

---

#### 2.10.9.19 NDAL Parity Errors

*Description:* An NDAL cycle with a parity error was detected by the NVAX CPU or the NEXMI. This discussion applies whenever S\_NESTS<PERR> or S\_NCSR<NDPE> is set. If S\_NESTS<PERR> is set, then S\_NEICMD, S\_NEDATHI, and S\_NEDATLO will contain the information captured from the cycle. The B-cache will be in ETM if the NVAX detected the parity error. It might be in ETM if the parity error was not detected and another error bit set (either due to the parity error or otherwise).

There are several possible cycle "types" that could have been on the bus at the time the parity error occurred:

- A NULL cycle could have been driven onto the bus by the NEXMI. This happens when no other node requests the bus, and prevents floating bus levels. Since state is not changed, recovery from such an error is possible.
- The NEXMI could have been returning data to the NVAX. This is also potentially recoverable, although it is more difficult than a NULL.
- The NEXMI could have been sending an invalidate to the NVAX. This is recoverable only if the invalidate was of a DREAD type. WRITE and OREAD invalidates cannot be recovered, since the P-cache might be incoherent long enough to cause a problem.
- The NVAX could have been sending a command or write data to the NEXMI. Recovery is possible, since the NEXMI refused the command and no action was taken.

*General recovery procedure:* Before analysis is begun, ensure that all the error information has reached the appropriate register. For example, the error recovery code should first clear out all the outstanding commands, and make sure that all pertinent timeouts have been registered. This can be accomplished by the following steps:

- 1 Place the B-cache into ETM if it is not already in that state.

- 2 Clear out the Write Buffer (MFPR from CWB).
- 3 Flush the VIC and P-cache.

At the end of the analysis, if recovery is possible, all the significant and related error bits should be cleared, the B-cache flushed, and then enabled.

*Recovery guidelines:* Parity provides minimum protection. Parity detects only single-bit failures, double-bit failures will not be detected. This makes any recovery attempt somewhat risky, and recovery should only be attempted if all indications show exactly the same error scenario. Several error bits make this decision easier.

- If S\_NESTS<INCON PERR> or S\_NCSR<NDIPE> is set, then one NDAL node saw the parity error and the other did not. Furthermore, the node that did not see the parity error ACKed the transaction and might have taken the wrong action. Recovery is difficult and risky. Reset the system.
- If S\_NESTS<LOST PERR> is set, then a further parity error was detected, but no information was captured in the command and data registers. Analysis is impossible. Reset the system.

In general, it is better if both nodes see the parity error. So if S\_NESTS<PERR> and S\_NCSR<NDPE> are both set, the failure is solid and the chances of recovery are favorable. There are, however, certain situations that allow for only one node to see the error, and in those cases recovery is still possible.

If the NVAX detected the parity error, and S\_NESTS<PERR> is set, the error analysis is made easier. The S\_NEICMD, S\_NEDATHI, and S\_NEDATLO registers should first be analyzed to determine where the parity error occurred. If correct parity is shown for all three parity-protected fields after the initial analysis, then the status is inconsistent and no recovery is possible. Reset the system.

If at least one of the parity-protected fields shows a parity error, then determine which field(s) are likely to be accurate. Based upon that data, other evidence of this cycle type should be searched for. If the parity error is indicated in the CMD/ID field, then no information is available about the likely error. Further analysis without this crucial information is risky, and a parity error in either the CMD or ID field is so damaging that the system should be reset.

If the parity is calculated to be good for the CMD and ID fields, then error analysis uses that information for further recovery decisions. The four cycle types of interest are:

- The S\_NEICMD<CMD> field shows a NULL, and the S\_NEICMD<ID> field shows the NEXMI node (010). None of the error bits S\_CEFSTS<Timeout>, S\_NESTS<NO ACK>, or S\_NCSR<NRTAE> should be set. This all points to a benign NULL cycle on the bus at the time of the parity error, and no special recovery is necessary.

- The S\_NEICMD<CMD> field shows a return data cycle (RDRx or RDE), and the S\_NEICMD<ID> field shows one of the NVAX nodes (000,001). S\_NCSR<NRTAE> should be set to indicate that a return data word was NO ACKed by the NVAX, and S\_CEFSTS<Timeout> should be set to indicate that the NVAX waited for the Cbox timeout and did not receive the data. S\_NESTS<NO ACK> should be clear.

In this case, it is not a problem that the NVAX saw the parity error (S\_NESTS<PERR> is set), but the NEXMI did not (S\_NCSR<NDPE> is clear). Since a signal is more likely to be degraded at the far end of the transmission line, the NEXMI may not have sensed the problem. If the rest of the evidence points to a missed return data word, then it is still reasonable to assume that the NVAX missed the return word and can recover.

This is potentially recoverable. The failing address is stored in S\_CEFADR. For more information about recovering from this error, see Section 2.10.9.13 or Section 2.10.9.18.2, depending upon what further analysis shows.

- The S\_NEICMD<CMD> field shows an invalidate cycle (WRITE, DREAD, or OREAD), and the S\_NEICMD<ID> field shows the NEXMI node. None of the error bits S\_CEFSTS<Timeout>, S\_NESTS<NO ACK>, or S\_NCSR<NRTAE> should be set. This is recoverable only if the invalidate type can be identified to be a DREAD, since that would not have required the cached word to be invalidated. Either the original XMI node would have retried the read (in which case the second invalidate would be sensed and a writeback would be performed), or the error procedure would flush the B-cache and write the data back. If the invalidate is of type WRITE or OREAD, then the P-cache could become incoherent, and the system should be reset.

This is another case where the S\_NCSR<NDPE> can be clear (NEXMI did not sense the parity error), yet recovery still be possible, if the invalidate type is a DREAD.

- The S\_NEICMD<CMD> field shows an NVAX command cycle (WRITE, WDISOWN, IREAD, DREAD, OREAD), and the S\_NEICMD<ID> shows one of the NVAX nodes. If the S\_NEDATHI register is shown to have good parity, then the length can also be checked to ensure consistency. S\_NESTS<NO ACK> should be set, which means that S\_NEOCMD and S\_NEOADR contain the information about the outbound cycle. This should be compared to the S\_NEICMD and S\_NEDATLO registers to ensure consistency. In this case, even a parity error in the S\_NEDATLO register should be visible, since we are assuming single-bit errors, and the S\_NEOADR register provides the correct data. S\_NCSR<NRTAE> should be clear.

If the NVAX did not see the parity error (S\_NESTS<PERR> is clear), but the NEXMI did (S\_NCSR<NDPE> is set), then recovery is possible. Since the NVAX is the transmission device, it is possible that only the receiving device (NEXMI) senses a problem. In that case, S\_NESTS<NO ACK> would be set, and the pertinent command information would be stored in the S\_NEOCMD and S\_NEOADR registers. If the rest of the evidence points to a parity error on an NVAX generated command, then recovery can be attempted.

For more information regarding recovery from this type of error, see Section 2.10.9.16 or Section 2.10.9.18.4, depending upon what further analysis shows.

*Restart condition:* If the cycle can be identified specifically, then restart. Otherwise, reset the system.

---

### 2.10.9.20 Lost Parity Errors

*Description:* S\_NESTS<LOST PERR> indicates that several NDAL cycles with a parity error were detected by the NVAX, before the first parity error had been serviced, the latched information saved, and the latching registers cleared for a new value. This means that at least one cycle with a parity error has no saved information about it. The B-cache is in ETM. If an invalidate cycle was missed that would have hit in the B-cache, the P-cache may now be incoherent. Since there is no way to even guess what the offending cycle was, recovery is not possible.

*Recovery procedure:* It is impossible to determine whether the interrupted instruction stream may give the effect of out of order writes (because the P-cache may have missed an invalidate). Reset the system.

---

### 2.10.9.21 Inconsistent Parity Errors

*Description:* S\_NESTS<INCON PERR> or S\_NCSR<NDIPE> indicates that an inconsistent parity error was detected. This describes a condition where one node on the NDAL detected a parity error, yet that same cycle was ACKed by another. Since a node will not ACK a cycle with a parity error, there is an inconsistency between what the nodes sensed.

*Recovery procedure:* Since the error information is inconsistent, no error recovery is possible. Reset the system.

---

### 2.10.9.22 NEXMI Soft Error Interrupts

*Description:* Errors that do not result in loss of data, or can notify the CPU by returning RDE, are corrected automatically by the hardware.

These errors normally notify the CPU of the error by asserting ERR L. The following saved error register bits are consistent with a NEXMI soft error interrupt.

- XBER<CC>
- XBER<PE>
- XBER<CRD>
- XBER<RSE>
- XBER<RER>
- NCSR<NDPE>
- NCSR<NRTAE>
- NCSR<SSCIR>
- NCSR<WBQFL>
- NCSR<NWQFL>

Most NEXMI soft errors are caused by some problem that also signals the NVAX CPU of the event through some Cbox error. Those bits are discussed in the sections that pertain to the associated Cbox errors. The NEXMI soft error bits that can occur alone are discussed here.

- **S\_XBER<CC>:** This causes a soft error if S\_XCR<CCID> is clear. It means that a correctable error has been identified by the XMI interface on the XMI ACK lines. If only a single bit has been corrupted, the ACK indication is still correct, and system recovery is possible with no special actions (other than logging the error).
- **S\_XBER<PE>:** An XMI parity error was sensed by the NEXMI. This error, when unaccompanied by any other error, is generally recoverable. Either the cycle was a NULL cycle, or the command was retried successfully and no special recovery is necessary.

Other errors can be caused by an NDAL parity error such as RSE (read sequence error), NRR (no read return), and URR (unexpected read response). These cases are discussed in the sections that pertain to those other errors.

- **S\_XBER<CRD>:** This error causes a soft error interrupt if S\_XCR<CRDID> is clear. It indicates that a device has returned good data, but that correction was necessary to provide the data accurately. The most common reason for this error is a single-bit failure in the MS65A memory. No special recovery is necessary, although the operating system might want to see if the error was hard (reproducible) or soft (transient). A soft single-bit error can be repaired by performing a read-modify-write operation on that data word.
- **S\_XBER<RSE>:** This error bit signifies that a read sequence error occurred on the XMI. A normal return data sequence from a memory or I/O adapter has the words come back in a particular order. If one data word is missed for some reason, the next data word that is returned (after the one that was missed) will be out of sequence, and the XMI commander will know that something went wrong. This error bit can get set if an XMI parity error causes the returned data word to be missed, in which case S\_XBER<PE> will also be set. This is discussed in those sections where a recoverable RSE is likely.

An unexpected RSE is a different matter. It means that an adapter sent back an out-of-order return data packet for no obvious reason. This is a very serious system error. The error source is elsewhere in the system. Unless the error source can be determined and the error repaired, reset the system.

- **S\_NCSR<NDPE>:** An NDAL parity error was sensed by the NEXMI. This indication is discussed in Section 2.10.9.19.
- **S\_NCSR<WBQFL>:** An NDAL DISOWN command was refused (and NO ACKed) because the internal NEXMI writeback queue was full. However, the NDAL NO ACK will be sensed immediately by the NVAX and analyzed on that basis. Because of this, it is considered a soft error from the NEXMI perspective. Recovery is unlikely, since the writeback will not be retried and the data will be overwritten soon. The situation that would cause this bit to be set will flag a hard error from within the NVAX, so this bit would be analyzed and cleared

before ever seeing it as part of a soft error interrupt. If this bit is set without a hard error also being posted, then the system status is inconsistent and the system should be reset. See Section 2.10.8.5 for a discussion of possible recovery from this error.

*Recovery procedure:* Clear the error status bits in the NEXMI registers and perform any necessary system recovery procedure. This might include clearing memory or I/O registers.

*Restart condition:* Typically, restart is possible, although in cases where data is lost software may have to kill one process or crash the system.

---

### 2.10.9.23 Inconsistent Status in Soft Error Interrupts

*Description:* A presumed impossible error report was found in the error registers. This could be due to a hardware failure.

*Recovery procedure:* No specific recovery action is called for.

*Restart condition:* No restart is possible. Reset the system.

---

## 2.10.10 Note on Tagged-Bad Data Mechanisms

Writebacks sent as BADWDATA are supposed to be tagged-bad data in memory, and further reads to that block should fail. In VAX 6000 Model 600 systems, the "tagged-bad data" storage mechanism identifies bad data as reliably as it does good data. Thus, system operation can continue after such an error because any process that accesses that data will see an RDE error for that block and will machine check before it uses the bad data.

The B-cache data RAMs use a relatively unreliable mechanism for tagged-bad data. Three ECC check bits are flipped in the stored value. This mechanism would often prevent a subsequent read from succeeding, but it is not sufficiently reliable to allow missing tagged-bad blocks in the B-cache to be tolerated. As a result, all errors that may have left a tagged-bad block in the B-cache without some error address register pointing it out are cause to reset the system.

---

### 2.10.11 Kernel Stack Not Valid Exception

A kernel stack not valid exception occurs when a memory management exception is detected during an attempt to push information on the kernel stack during microcode processing of another exception. A console halt with an error code of ERR\_INTSTK is taken if a memory management exception is encountered while attempting to push information on the interrupt stack.

The kernel stack not valid exception is dispatched through SCB vector 08 (hex).

# 3

---

## MS65A Memory Module

The MS65A memory module is a metal-oxide semiconductor (MOS), dynamic random access memory (DRAM), that provides up to 128 Mbytes of data storage. The memory array can be used in any VAX 6000 system and communicates over the XMI bus.

This chapter contains the following sections:

- Module Description
- Self-Test and Initialization
- Control and Status Registers
- Error Handling

### 3.1 Module Description

---

**The MS65A memory module is a dynamic random access memory (DRAM) that communicates through the XMI bus to provide system memory.**

The MS65A memory module consists of the following major components:

- XMI Corner
- Memory control gate array
- Memory storage array

The MS65A memory module has the following features:

- The memory module contains MOS dynamic RAM (DRAM) arrays, a CMOS memory control gate array (that contains error correction code (ECC) logic and control logic), an EEPROM storage element, and an XMI interface (the XMI Corner).
- Storage arrays are made up of two or four banks with 155 or 299 DRAMs.
- ECC logic detects single-bit and double-bit errors and corrects single-bit errors.
- Memory self-test checks all DRAMs, the data path, and control logic on power-up.
- Quadwords, octawords, and hexwords can be read from memory.
- Quadwords, octawords, and hexwords can be written to memory.
- The memory can be configured by the system for 2-, 4-, 8-way or no interleaving.

The **XMI Corner** is the module's interface to the XMI bus and contains CMOS memory control gate arrays and interface logic. Its primary purpose is to transfer data between the MS65A memory module and the XMI bus.

The **memory control gate array** controls the MS65A memory module and transfers data between the XMI Corner and the DRAMs. The memory control gate array also controls address multiplexing, command decoding, arbitration, CSR logic functions, and the EEPROM.

The **memory storage array** within the memory control gate array includes address and control logic to modify address bits received from the XMI Corner. These modified address bits are used to control the selection of the DRAMs during reading and writing.

A block is defined as a contiguous 32-byte quantity of data, also known as a hexword. The MS65A memory module uses an indicator to represent the state of each block. This indicator is known as the *block state*. The block state is used to determine the proper read or write response on commands.

All power for the XMI memory array is supplied from the +5V rail. If the optional battery backup unit (BBU) is not installed and system power is lost, memory is lost as well.

If the optional BBU is installed, it supplies power to the system regulators in the event of a power failure, ensuring that no data is lost. The BBU supplies power to the XMI for up to 10 minutes.

### 3.2 Self-Test and Initialization

---

**The MS65A memory module performs an initialization and self-test sequence on a cold power-up or when the sequence is requested by a console command. On power-up the console firmware loads the Starting Address Register (STADR) with the starting address, the Segment/Interleave Register (INTLV) with the interleave mode, and the Ending Address Register (ENADR) with the ending address.**

During a cold power-up the memory control gate array is initialized, all memory locations are tested, and the control and status registers are initialized.

A warm power-up occurs when the system (excluding memory if a battery backup unit (BBU) is present) loses power. During a warm power-up, self-test is not run and memory contents are not modified.

Memory self-test takes about 5 seconds for a 32-Mbyte memory module, 10 seconds for a 64-Mbyte memory module, and 20 seconds for a 128-Mbyte module. While self-test runs, the fault light on the system front panel is on. When self-test completes, the fault light goes off and the console printout of self-test begins. For details on the self-test console printout, refer to Chapter 6 in the *VAX 6000 Series Owner's Manual*.

---

### 3.2.1 Starting and Ending Addresses

The memory responds to starting addresses on any 16-Mbyte boundary. The ending address is also on any 16-Mbyte boundary. The ending address must be greater than the starting address, otherwise commands are not acknowledged. The ending address minus the starting address must be equal to or less than the memory size multiplied by the number of ways interleaved.

$$EA - SA = \text{Memory Size} * (\# \text{ of ways interleaved})$$

Starting addresses for memory can be in the range from 0 to 7F FF00 0000 (16 Mbytes to 512 Gbytes in 16-Mbyte multiples). Ending addresses can range from 0 to 80 0000 0000 (0 to 512 Gbytes in 16-Mbyte multiples). Ending addresses greater than 80 0000 0000 are not permitted. The area above 80 0180 0000 is reserved for CSR addresses.

---

### 3.2.2 Interleaving

Interleaving achieves greater throughput to memory by optimizing memory access time and increasing the effective memory transfer rate. This is done by operating memory modules in parallel.

The memory array supports 2-way, 4-way, 8-way or no interleaving at the system level. Up to eight memory array modules can be interleaved. Interleaving is done on hexword boundaries.

### 3.3 Control and Status Registers

The CSR names and their relative addresses are shown in Table 3–1. Detailed descriptions of the CSRs are also included in this section.

**Table 3–1 MS65A Memory Module Control and Status Registers**

CSR Name	Mnemonic	Address
Device Register	XDEV	BB <sup>1</sup> + 0000 0000
Bus Error Register	XBER	BB + 0000 0004
Memory Control Register 1	MCTL1	BB + 0000 0014
Memory ECC Error Register	MECER	BB + 0000 0018
Memory ECC Error Address Register	MECEA	BB + 0000 001C
Memory Control Register 2	MCTL2	BB + 0000 0030
TCY Tester Register	TCY	BB + 0000 0034
Block State ECC Error Register	BECER	BB + 0000 0038
Block State ECC Address Register	BECEA	BB + 0000 003C
Starting Address Register	STADR	BB + 0000 0050
Ending Address Register	ENADR	BB + 0000 0054
Segment/Interleave Control Register	INTLV	BB + 0000 0058
Memory Control Register 3	MCTL3	BB + 0000 005C
Memory Control Register 4	MCTL4	BB + 0000 0060
Block State Control Register	BSCTL	BB + 0000 0068
Block State Address Register	BSADR	BB + 0000 006C
EEPROM Control Register	EECTL	BB + 0000 0070
Timeout Control/Status Register	TMOER	BB + 0000 0074

<sup>1</sup>"BB" refers to the base address of an XMI node (E180 0000 + (node ID x 8000)).

The memory contains 19 control and status registers (CSRs) to control the memory and log errors. All CSRs are 32 bits long and respond only to longword read and write transactions. Only full writes are performed. If a parity error occurs during a write, the operation is aborted and the contents of the CSRs are unchanged.

Some bits in the registers are cleared on power-up, while others need a one written to them to clear.

The CSRs start at an address dependent upon the node ID. All CSR addresses are designated as BB + *n*, where *n* is the relative offset of the register.

The following definitions apply to the descriptions of the control and status registers.

**CRD error** – A correctable single-bit error.

**RER error** – A general uncorrectable multiple-bit error indicator that includes an RDS error (a hard unrecoverable error), a row parity error, a column parity error, or a byte write error.

**RO** – Indicates a read-only register.

**RO, 0** – Indicates a read-only register, cleared on power-up.

**R/CW, 0** – Indicates a read conditional write register, cleared on power-up.

**R/W** – Indicates a read and write register.

**R/W, 0** – Indicates a read and write register, cleared on power-up.

**R/W1C** – Indicates a read and write register, write a one to clear.

**R/W1C, 0** – Indicates a read and write register, write a one to clear, and cleared on power-up.

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

**WO, 0** – Indicates a write-only register, cleared on power-up.

# MS65A Memory Module Registers

## Device Register (XDEV)

---

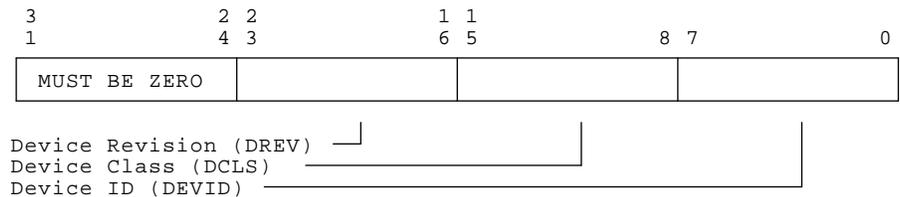
### Device Register (XDEV)

The Device Register contains information to identify the MS65A memory module. The fields are loaded during node initialization. A zero value indicates an uninitialized node. The device class and device ID fields match those of the device type code of the MS62A memory module.

---

#### ADDRESS

*Nodespace base address + 0000 0000*



msb-p245-90

---

#### bits<31:24>

Name: Reserved  
Mnemonic: None  
Type: RO, 0  
Reserved; must be zero.

---

#### bits<23:16>

Name: Device Revision  
Mnemonic: DREV  
Type: R/W

Identifies the revision level of the MS65A memory module. The use of the Device Revision field is implementation dependent. The field does not indicate the hardware revision level, only the functional level.

---

#### bits<15:8>

Name: Device Class  
Mnemonic: DCLS  
Type: RO

Identifies the type of node. The device type for an MS65A memory module is 40 (hex) and 4001 (hex) for an MS62A memory module. This value is set permanently in the Device Register.

# MS65A Memory Module Registers

## Device Register (XDEV)

---

**bits<7:0>**

Name: Device ID

Mnemonic: DEVID

Type: R/W

Identifies the ID of node. The ID for an MS65A memory module is 0000 0001 (binary) or 01 (hex).



# MS65A Memory Module Registers

## Bus Error Register (XBER)

---

### bit<27>

Name: Corrected Confirmation Received  
Mnemonic: CCR  
Type: R/W1C, 0

When set, the XMI Corner Interface (XCI) bus detected a single-bit error on any of the three XMI confirmation lines. This error is always logged, whether the MS65A memory module is responsible for the faulty bus cycle or not.

---

### bits<26:24>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

---

### bit<23>

Name: Bus Parity Error  
Mnemonic: BPE  
Type: R/W1C, 0

When set, the memory detected a parity error on the XMI data, ID, fault, or parity lines. The MS65A does not acknowledge any cycle that contains a bus parity error. Once the cycle is not acknowledged, all other bus cycles associated with the transaction are flushed from memory. No log is kept of which parity fields had the errors. This error is always logged, whether the MS65A memory module is responsible for the faulty bus cycle or not.

---

### bit<22>

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

When set, indicates that the memory aborted a write transaction, due to one or more missing data cycles. All command and write data entries associated with this error are flushed from queues.

# MS65A Memory Module Registers

## Bus Error Register (XBER)

---

### bit<21>

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

When set, indicates that the memory failed to receive a response for a LOC, RER, CRD, or GRD data response cycle. Any remaining quadwords of the data packet being transmitted are sent normally.

---

### bits<20:13>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

---

### bit<12>

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

When set, this bit indicates that a node-specific error condition has been detected. The exact nature of the error is located in the memory error status registers. A hierarchy of these registers is shown in Figure 3–1.

---

### bit<11>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

---

### bit<10>

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

This bit is set when self-test starts and is cleared when self-test successfully completes.

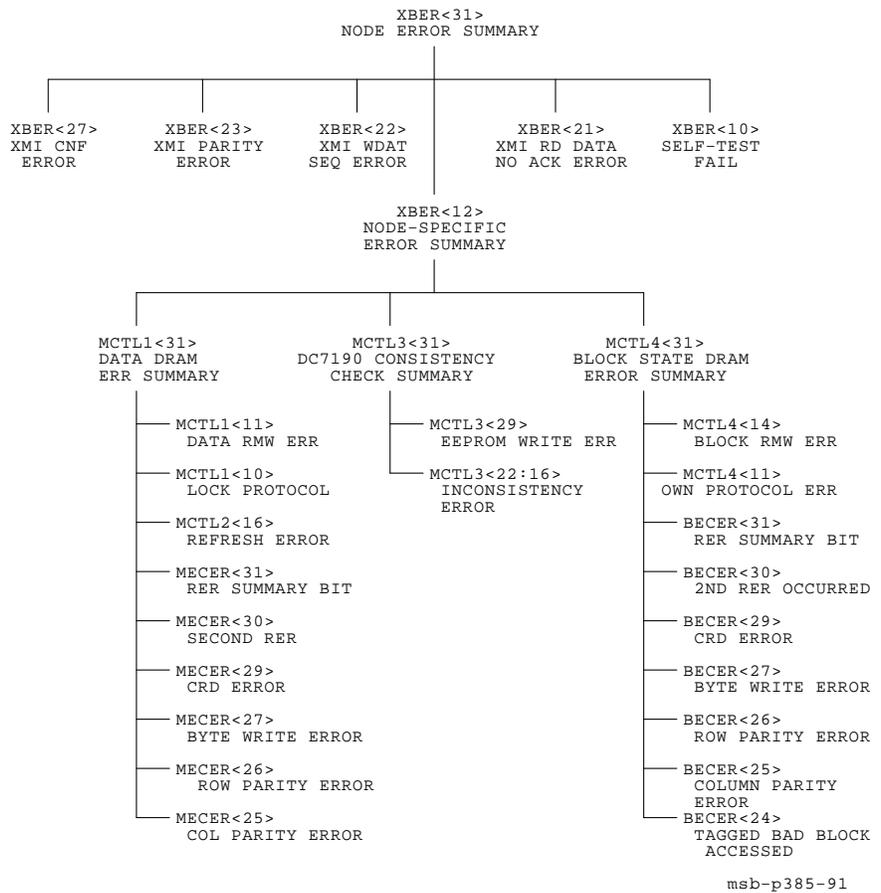
# MS65A Memory Module Registers

## Bus Error Register (XBER)

bits<9:0>

Name: Reserved  
 Mnemonic: None  
 Type: RO  
 Reserved; must be zero.

Figure 3–1 Error Bit Hierarchy





# MS65A Memory Module Registers

## Memory Control Register 1 (MCTL1)

---

### bit<29>

Name: Data ECC Disable  
Mnemonic: DECCD  
Type: R/W, 0

This bit is used for diagnostic purposes.

---

### bits<28:18>

Name: Memory Size  
Mnemonic: MEMSIZ  
Type: R/W

These bits specify the memory module size in 256-Kbyte increments. An upper extension of these bits is located in the MCTL4 Memory Size <28:18> field.

---

### bits<17:16>

Name: RAM Type  
Mnemonic: RAMTYP  
Type: R/W

These bits show the DRAM type used. They are used in conjunction with <28:18> and Memory Control Register 4 (MCTL4) <17:16>.

---

### bit<15>

Name: Inhibit CRD Status Generation  
Mnemonic: ICRD  
Type: R/W, 0

This bit inhibits the reporting of CRD status to the commander on read cycles. When this bit is set, any CRD response is changed to a GRD response. CRD errors are still logged, and RER errors are logged and reported.

---

### bit<14>

Name: On-Board Memory Valid  
Mnemonic: MEMVAL  
Type: RO, 0

This bit indicates that valid data is stored in memory. The bit is set on the first write to the module memory space.

# MS65A Memory Module Registers

## Memory Control Register 1 (MCTL1)

---

### bit<13>

Name: Enable 2-Mbyte Protection Mode  
Mnemonic: EPM  
Type: R/W, 0

When set, the operation of the MCTL1 ECC Diagnostic<30> MCTL1 ECC Disable <29>, MCTL4 Block State Diagnostic Mode <30>, and MCTL4 Block State ECC Disable <29> bits are inhibited in the first 2 Mbytes of memory space.

---

### bit<12>

Name: Reserved  
Mnemonic: None  
Type: RO

Reserved; must be zero.

---

### bit<11>

Name: Interlock Sequence Error  
Mnemonic: INSEQ  
Type: R/W1C, 0

When set, this bit indicates that a system protocol violation has been observed around an executed UWMASK to the on-board memory.

---

### bit<10>

Name: Data Read Modify Write Error  
Mnemonic: DRMWER  
Type: R/W1C, 0

When set, an uncorrectable error was detected in the data DRAM field during a Read Modify Write cycle.

---

### bits<9:8>

Name: Reserved  
Mnemonic: None  
Type: RO

Reserved; must be zero.

---

### bits<7:0>

Name: Diagnostic Check  
Mnemonic: DCK  
Type: R/W, 0

These bits are used in ECC diagnostic mode as substitute check bits.

## Memory ECC Error Register (MECER)

The Memory ECC Error Register logs ECC error status and syndrome information for the data DRAMs. The MECER logs ECC error information during read cycles only. If an RER error occurs during a Write Mask cycle, DRMWER bit<10> in the MCTL1 Register is set. Errors associated with block state DRAMs are logged in MCTL4.

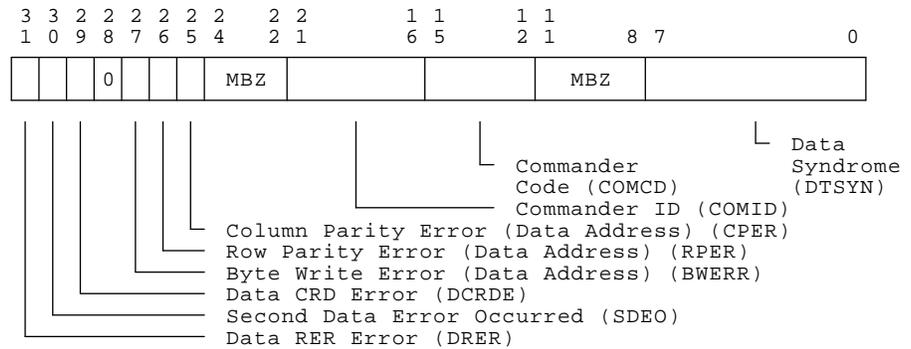
This register logs ECC error type and error syndrome information when correctable and uncorrectable errors occur during Read transactions. During a Write Mask transaction, only DRMWER bit<10> in the MCTL1 Register logs the fact that the ECC error occurred.

For read accesses, the register logs the first correctable error and holds it until either an uncorrectable error occurs or the error is cleared. Additional correctable errors are only reported and are not logged. An uncorrectable error will overwrite a logged correctable error. A correctable error will not overwrite a logged uncorrectable error or a previously logged correctable error until the error has been cleared.

This register logs errors during module self-test.

### ADDRESS

*Nodespace base address + 0000 0018*



msb-p236-90

### bit<31>

Name: Data RER Error

Mnemonic: DRER

Type: R/W1C, 0

When set, an uncorrectable error occurred during a read transaction.

# MS65A Memory Module Registers

## Memory ECC Error Register (MECER)

---

### bit<30>

Name: Second Data Error Occurred  
Mnemonic: SDEO  
Type: R/W1C, 0

When set, an RER or CRD error occurred before the previous one was cleared from the register. The error information is logged only if the second error is an RER type error and the first error was a CRD. A second CRD error after the initial CRD error will set this bit.

---

### bit<29>

Name: Data CRD Error  
Mnemonic: DCRDE  
Type: R/W1C, 0

When set, a CRD error occurred during a read transaction. This includes a single-bit error in the check bits, even though no correction is done on the data bits. The error address and error syndrome are valid if no RER error log exists.

---

### bit<28>

Name: Reserved  
Mnemonic: None  
Type: RO

Reserved; must be zero.

---

### bit<27>

Name: Byte Write Error (Data Address)  
Mnemonic: BWERR  
Type: RO, 0

When set, an RER error was due to reading a location that was marked bad during a partial write cycle that had previously detected an RER error. Cleared when MECER<31> is cleared. The setting of this bit also sets MECER<31>.

---

### bit<26>

Name: Row Parity Error (Data Address)  
Mnemonic: RPER  
Type: RO, 0

When set, an RER error occurs due to a row address parity error. Cleared when MECER<31> is cleared. The setting of this bit also sets MECER<31>.

# MS65A Memory Module Registers

## Memory ECC Error Register (MECER)

---

### bit<25>

Name: Column Parity Error (Data Address)  
Mnemonic: CPER  
Type: RO, 0

When set, an RER error occurs due to a column address parity error. Cleared when MECER<31> is cleared. The setting of this bit also sets MECER<31>.

---

### bits<24:21>

Name: Reserved  
Mnemonic: None  
Type: RO

Reserved; must be zero.

---

### bits<21:16>

Name: Commander ID  
Mnemonic: COMID  
Type: RO, 0

This field logs the 6-bit commander ID (ID <5:0>) that was involved in the transaction that caused the logging of MECER errors. A CRD response is sent back to the commander and the CRD occurrence is logged in the MECER. The MECER error log shows which CPU or I/O module encountered the error. These bits are not cleared when the error status bits are reset by a system commander. Therefore, they hold the ID of the last MECER error logged (or a zero if no errors occurred since the last cold start).

---

### bits<15:12>

Name: Commander Code  
Mnemonic: COMCD  
Type: RO, 0

This field logs the 4-bit command code associated with the logged failure. The commander code identifies the command type associated with the error that occurred. Knowing the type of command helps the user to determine why the memory error occurred. These bits are not cleared when the error status bits are reset. Therefore, they hold the command code of the last MECER error logged (or a zero if no errors occurred since the last cold start).

## MS65A Memory Module Registers

### Memory ECC Error Register (MECER)

---

**bits<7:0>**

Name: Data Syndrome

Mnemonic: DTSYN

Type: RO, 0

These bits are the syndrome bits of the location in an RER or CRD error, when the memory module is not in diagnostic mode.

## Memory ECC Error Address Register (MECEA)

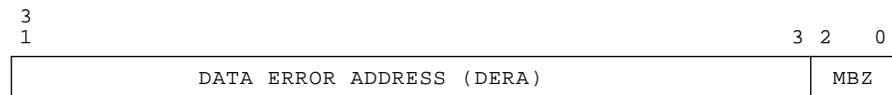
The Memory ECC Error Address Register logs the address of correctable and uncorrectable errors logged in the Memory ECC Error Register.

For read accesses, this register logs the address of the first corrected read data (CRD) error and holds it until a double-bit uncorrectable error (RER) occurs or the error is cleared. An RER error causes a logged CRD error address to be overwritten. A CRD will not overwrite a logged RER error address. If multiple RER errors occur, only the first error address is logged.

This register logs errors during self-test.

### ADDRESS

*Nodespace base address + 0000 001C*



msb-p235-90

### bits<31:3>

Name: Data Error Address  
Mnemonic: DERA  
Type: RO, 0

The error address of the RER or CRD error logged in the Memory ECC Error Register. This register is valid only if the RER or CRD error log bits are set in the Memory ECC Error Register. This address is the bus address of the cycle that was being performed at the time of the error.

### bits<2:0>

Name: Reserved  
Mnemonic: None  
Type: RO

Reserved; must be zero.

# MS65A Memory Module Registers

## Memory Control Register 2 (MCTL2)

---

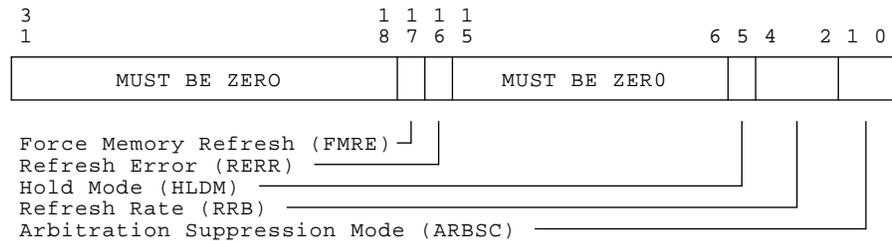
## Memory Control Register 2 (MCTL2)

The second memory control register contains additional control and error status information.

---

### ADDRESS

*Nodespace base address + 0000 0030*



msb-p232-90

---

### bits<31:18>

Name: Reserved  
 Mnemonic: None  
 Type: RO  
 Reserved; must be zero.

---

### bit<17>

Name: Force Memory Refresh  
 Mnemonic: FMRE  
 Type: WO, 0

When set by a commander, the gate array initiates a single memory refresh operation. This causes a refresh cycle to be generated for all on-board DRAMs.

---

### bit<16>

Name: Refresh Error  
 Mnemonic: RERR  
 Type: R/W1C, 0

When set, a second refresh request was asserted before the first one was implemented, meaning that a refresh was missed.

## MS65A Memory Module Registers

### Memory Control Register 2 (MCTL2)

---

#### bits<15:6>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

---

#### bit<5>

Name: Hold Mode  
Mnemonic: HLDM  
Type: RO, 0  
This bit informs the gate array which hold mode is needed for read-data response transfers.

---

#### bits<4:2>

Name: Refresh Rate  
Mnemonic: RRB  
Type: R/W  
This field controls the module's DRAM refresh rate.

---

#### bits<1:0>

Name: Arbitration Supression Mode  
Mnemonic: ARBSC  
Type: R/W, 0  
These field controls the Arbitration Supression mode.

# MS65A Memory Module Registers

## TCY Tester Register (TCY)

---

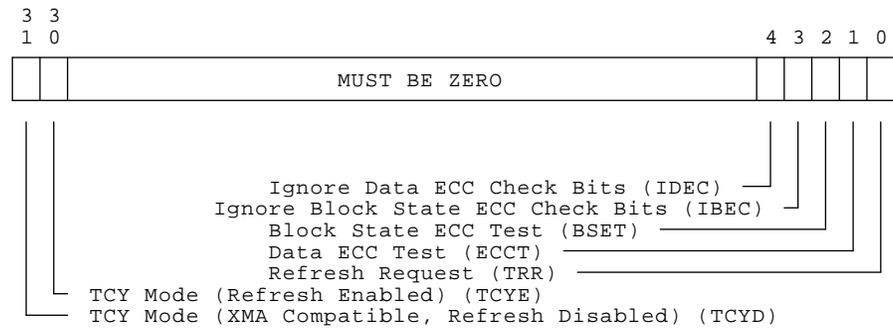
### TCY Tester Register (TCY)

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

---

#### ADDRESS

*Nodespace base address + 0000 0034*



msb-p242-90



# MS65A Memory Module Registers

## Block State ECC Address Register (BECEA)

---

### Block State ECC Address Register (BECEA)

The Block State ECC Address Register is used to log a block state ECC error-related address. This register logs any relevant error addresses during self-test as well as during normal operations. It provides diagnostic bit information if self-test fails and Bus Error Register (XBER) bit <10> is set. The contents of this register are valid only if Block State ECC Error Register (BECER) <31>, <29>, or <24> is set.

Bits in this register are used for logging error addresses during self-test and normal memory operations. This register provides diagnostic error information if self-test fails.

---

**ADDRESS**      *Nodespace base address + 0000 003C*



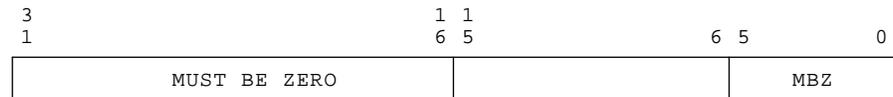
msb-p223-90

## Starting Address Register (STADR)

The Starting Address Register configures the starting memory address of the MS65A memory module with a total memory address space less than or equal to 512 Mbytes. It is used to place an MS65A memory module above the 512-Mbyte limit in the address space. It can also be used for configuring systems with less than 512 Mbytes of total memory.

### ADDRESS

*Nodespace base address + 0000 0050*



Starting Address (STADD) ↙

msb-p241-90

### bits<31:16>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

### bits<15:6>

Name: Starting Address  
Mnemonic: STADD  
Type: R/W, 0  
This field contains the 10-bit memory starting address.

### bits<5:0>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

# MS65A Memory Module Registers

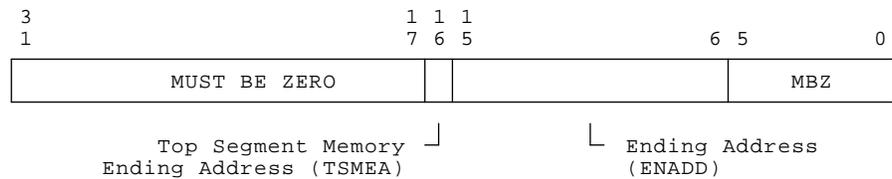
## Ending Address Register (ENADR)

### Ending Address Register (ENADR)

The Ending Address Register configures the ending memory address of an MS65A memory module with a total memory address space less than or equal to 512 Mbytes. It is also used to place an MS65A memory module above the 512-Mbyte limit in the address space. MS65A memory modules use this register along with the Starting Address Register (STADR) and the Segment/Interleave Register (INTLV) to configure memory addresses.

#### ADDRESS

*Nodespace base address + 0000 0054*



msb-p228-90

#### bits<31:17>

Name: Reserved  
 Mnemonic: None  
 Type: RO  
 Reserved; must be zero.

#### bit<16>

Name: Top Segment Memory Ending Address  
 Mnemonic: TSMEA  
 Type: R/W, 0

This bit configures the ending address as the top byte of the 16-Gbyte memory address segment.

## MS65A Memory Module Registers

### Ending Address Register (ENADR)

**bits<15:6>**

---

Name: Ending Address

Mnemonic: ENADD

Type: R/W, 0

This field contains the 10-bit memory ending address.

---

**bits<5:0>**

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

# MS65A Memory Module Registers

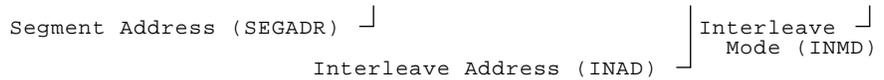
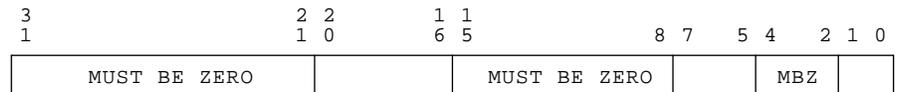
## Segment/Interleave Register (INTLV)

### Segment/Interleave Register (INTLV)

The Segment/Interleave Register gives memory interleave information. This register is used to establish the interleave mode and allows the system to program in a non-zero segment value. It sets up an optional segment comparison address for the upper bits of the memory bus address.

#### ADDRESS

*Nodespace base address + 0000 0058*



msb-p230-91

#### bits<31:21>

Name: Reserved  
 Mnemonic: None  
 Type: RO  
 Reserved; must be zero.

#### bits<20:16>

Name: Segment Address  
 Mnemonic: SEGADR  
 Type: R/W, 0

This field is optionally configured to place memory address space at a 16-Gbyte offset in the memory bus address space. Reading these bits tells the user the address of the failure. A zero in the field leaves the memory segment based in the lowest 16-Gbyte memory address space.

#### bits<15:8>

Name: Reserved  
 Mnemonic: None  
 Type: RO  
 Reserved; must be zero.

## MS65A Memory Module Registers

### Segment/Interleave Register (INTLV)

---

#### bits<7:5>

Name: Interleave Address

Mnemonic: INAD

Type: R/W, 0

This field contains the address used for interleaving. The interleaving address determines the address in which the memory module will respond.

---

#### bits<4:2>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

---

#### bits<1:0>

Name: Interleave Mode

Mnemonic: INMD

Type: R/W, 0

This field shows how many ways the module is interleaved (none, 1-way, 2-way, 4-way, or 8-way interleaving). The Interleave Mode bits are also used to determine the address in which the memory module will respond.

# MS65A Memory Module Registers

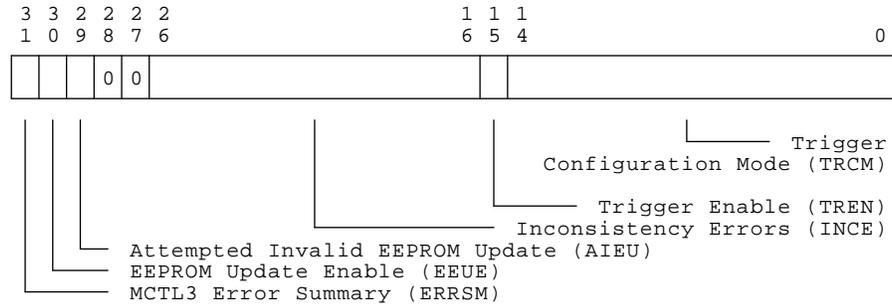
## Memory Control Register 3 (MCTL3)

### Memory Control Register 3 (MCTL3)

The third memory control register contains control and error status information.

#### ADDRESS

*Nodespace base address + 0000 005C*



msb-p233-90

#### bit<31>

Name: MCTL3 Error Summary  
 Mnemonic: ERRSM  
 Type: RO, 0

This bit represents the error summary status for Memory Control Register 3. When clear, all contributing error bits have been reset.

#### bit<30>

Name: EEPROM Update Enable  
 Mnemonic: EEUE  
 Type: R/W, 0

This bit enables EEPROM updates. When it is set, selected locations of the EEPROM can be updated.

#### bit<29>

Name: Attempted Invalid EEPROM Update  
 Mnemonic: AIEU  
 Type: R/W1C, 0

This status bit is set if an invalid EEPROM update was attempted.

## MS65A Memory Module Registers

### Memory Control Register 3 (MCTL3)

---

#### bits<28:23>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

---

#### bits<22:16>

Name: Inconsistency Errors  
Mnemonic: INCE  
Type: R/W1C, 0  
This field is used to report internal consistency errors. Set bits usually indicate that the gate array has a fault.

---

#### bit<15>

Name: Trigger Enable  
Mnemonic: TREN  
Type: R/W, 0  
This bit controls trigger enabling. It is dependent on the state of TRCM bit <14>.

---

#### bits<14:0>

Name: Trigger Configuration Mode  
Mnemonic: TRCM  
Type: R/W  
This field is used to enable or disable trigger generation.



# MS65A Memory Module Registers

## Memory Control Register 4 (MCTL4)

---

### bit<29>

Name: Block State ECC Disable  
Mnemonic: BSED  
Type: R/W, 0

This bit is used for diagnostic purposes.

---

### bits<28:23>

Name: Reserved  
Mnemonic: None  
Type: RO

Reserved; must be zero.

---

### bits<22:18>

Name: Memory Size  
Mnemonic: MEMSIZ  
Type: R/W, 0

This field provides an upper extension to the MCTL1 Memory Size <28:18> field.

---

### bits<17:16>

Name: RAM Type  
Mnemonic: RAMTYP  
Type: R/W, 0

This field shows the type of DRAM used.

---

### bit<15>

Name: Module Population  
Mnemonic: MODP  
Type: R/W, 0

This bit shows the DRAM population status of the MS65A memory module.

---

### bit<14>

Name: Block Read Modify Write Error  
Mnemonic: BRME  
Type: R/W1C, 0

This bit is set if an error is found on the block state DRAM field.

# MS65A Memory Module Registers

## Memory Control Register 4 (MCTL4)

---

### bits<13:12>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

---

### bit<11>

Name: Ownership Sequence Error  
Mnemonic: OSQE  
Type: R/W 1C  
This bit is set when a bus protocol violation has occurred.

---

### bits<10:9>

Name: Block State Code  
Mnemonic: BSCD  
Type: R/W, 0  
This bit is used for diagnostic purposes.

---

### bits<8:5>

Name: Block State ID  
Mnemonic: BSID  
Type: R/W, 0  
This field is used for diagnostic purposes.

---

### bits<4:0>

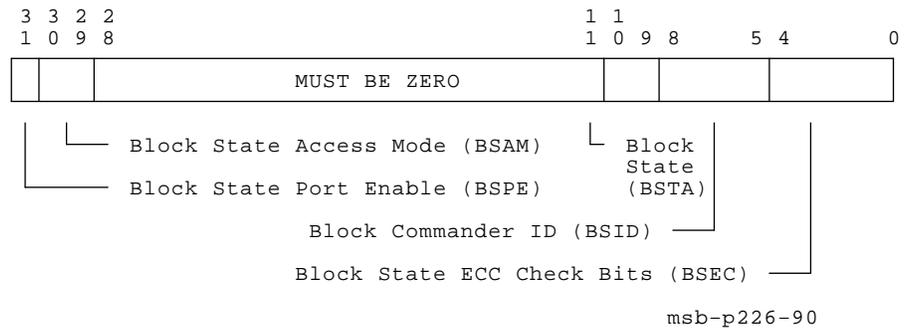
Name: Block ECC Check Bits  
Mnemonic: BSEC  
Type: R/W, 0  
This field is used for diagnostic purposes.

## Block State Control Register (BSCTL)

The Block State Control Register is used to access a block state DRAM field directly in diagnostic or error recovery modes of operation. This register is used with the Block State Address Register (BSADR).

### ADDRESS

*Nodespace base address + 0000 0068*



### bit<31>

Name: Block State Port Enable  
Mnemonic: BSPE  
Type: R/CW, 0

### bits<30:29>

Name: Block State Access Mode  
Mnemonic: BSAM  
Type: R/CW, 0

### bits<28:11>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

# MS65A Memory Module Registers

## Block State Control Register (BSCTL)

---

### bits<10:9>

Name: Block State  
Mnemonic: BSTA  
Type: RO, 0

---

### bits<8:5>

Name: Block Commander ID  
Mnemonic: BCID  
Type: RO, 0

---

### bits<4:0>

Name: Block State ECC Check Bits  
Mnemonic: BSECB  
Type: RO, 0

---

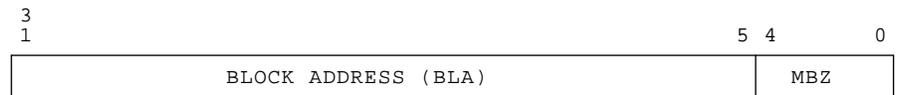
## Block State Address Register (BSADR)

The Block State Address Register is used to set up a block address for commander access to the block state DRAM field. This register is used with the Block State Control Register (BSCTL).

---

### ADDRESS

*Nodespace base address + 0000 006C*



msb-p225-90

---

### bits<31:5>

Name: Block Address

Mnemonic: BLA

Type: R/W, 0

These bits contain the block address.

---

### bits<4:0>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.



## MS65A Memory Module Registers

### EEPROM Control Register (EECTL)

---

#### bits<29:27>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

---

#### bits<26:16>

Name: EEPROM Address  
Mnemonic: EEADD  
Type: R/W1C, 0  
This field contains the EEPROM address.

---

#### bits<15:8>

Name: Reserved  
Mnemonic: None  
Type: RO  
Reserved; must be zero.

---

#### bits<7:0>

Name: EEPROM Data  
Mnemonic: EEDAT  
Type: R/CW, 0  
This field contains the EEPROM data.



## 3.4 Error Handling

---

**The memory module performs single-bit correction and double-bit detection on the data stored. Two error schemes are used: one for the data DRAMs and another for the block state DRAMs.**

Memory logic detects and corrects single-bit DRAM errors. Double-bit errors are detected, but not corrected. All ones or all zeros are considered to be uncorrectable errors. For more information, refer to the Bus Error Register (XBER) bit descriptions.

Block state DRAM ECC errors are handled in the following way. Each 6-bit block state/commander ID location of the block state field has a 5-bit ECC check field. Each block state access operation uses an 11-bit pattern that is transferred between the memory control gate array and the block state DRAM array. The ECC check bit pattern is a function of the 6-bit block state commander ID pattern and the hexword block address.

## MS65A Memory Module

---

# Index

---

## A

---

ABORT instruction • 2–249  
Aborts • 2–17  
AC LO L  
    See XMI AC LO L signal  
Address Extension field • 2–195  
AIEU (Attempted Invalid EEPROM Update) bit • 3–32  
Arbitration Supression Mode (ARBSM) bit • 3–23  
ARBSM (Arbitration Supression Mode) • 3–23  
Architecture • 1–2  
Arithmetic exceptions • 2–18  
Attempted Invalid EEPROM Update (AIEU) bit • 3–32

---

## B

---

Backup Cache box (Cbox) • 2–47  
Backup Cache Data ECC Register (BCDECC) • 2–103  
Backup Cache Error Data ECC Register (BCEDECC) • 2–115  
Backup Cache Error Data Index Register (BCEDIDX) • 2–114  
Backup Cache Error Data Status Register (BCEDSTS) • 2–111  
Backup Cache Error Tag Index Register (BCETIDX) • 2–108  
Backup Cache Error Tag Register (BCETAG) • 2–109  
Backup Cache Error Tag Status Register (BCETSTS) • 2–105  
Battery backup unit  
    location • 1–6, 1–8  
B-cache data extraction • 2–243  
BCDECC (Backup Cache Data ECC) register • 2–103  
BCEDECC (B-Cache Error Data ECC) register • 2–115  
BCEDIDX (Backup Cache Error Data Index) register • 2–114  
BCEDSTS (Backup Cache Error Data Status) register • 2–111  
BCETAG (Backup Cache Error Tag) register • 2–109  
BCETIDX (Backup Cache Error Tag Index) register • 2–108  
BCETSTS (Backup Cache Error Tag Status) register • 2–105  
BCID (Block Commander ID) bits • 3–38  
BECEA (Block State ECC Address Register) • 3–26  
BECER (Block State ECC Error Register) • 3–25  
BLA (Block Address ) bits • 3–39  
Block Address (BLA) bits • 3–39  
Block Commander ID (BCID) bits • 3–38  
Block ECC Check (BSEC) bits • 3–36  
Block Read Modify Write Error (BRME) bit • 3–35  
Block state • 3–3  
Block State (BSTA) bits • 3–38  
Block State Access Mode (BSAM) bits • 3–37  
Block State Address Register (BSADR) • 3–39  
Block State Code (BSCD) bit • 3–36  
Block State Control Register (BSCTL) • 3–37  
Block State Diagnostic Mode (BSDM) bit • 3–34  
Block State ECC Address Register (BECEA) • 3–26  
Block State ECC Check (BSECB) bits • 3–38  
Block State ECC Disable (BSED) bit • 3–35  
Block State ECC Error Register (BECER) • 3–25  
Block State ID (BSID) bits • 3–36  
Block State Port Enable (BSPE) bit • 3–37  
Bootblock booting • 2–217  
Boot processor  
    See BP  
Boot Processor (BP) bit • 2–187  
Boot Processor Disable (BPD) bit • 2–186  
Bootstrap in progress flag • 2–217  
Bootstrapping • 2–214  
BP • 2–207, 2–211, 2–214  
BP (Boot Processor) bit • 2–187  
BPD (Boot Processor Disable) bit • 2–186  
BPD bit • 2–211  
Branch prediction • 2–43  
BRME (Block Read Modify Write Error) bit • 3–35  
BSADR (Block State Address Register) • 3–39  
BSAM (Block State Access Mode) bits • 3–37  
BSCD (Block State Code) bit • 3–36  
BSCTL (Block State Control Register) • 3–37  
BSDM (Block State Diagnostic Mode) bit • 3–34  
BSEC (Block ECC Check) bits • 3–36  
BSECB (Block State ECC Check) bits • 3–38  
BSED (Block State ECC Disable) bit • 3–35  
BSID (Block State ID) bits • 3–36  
BSPE (Block State Port Enable) bit • 3–37  
BSTA (Block State) bits • 3–38  
Bus Error Extension Register (XBEER) • 2–197  
Bus Error Register (XBER) • 2–174, 3–10

## Index

Bus Parity Error (PE) bit • 3–11  
BWERR (Byte Write Error (Data Address) bit) • 3–18  
Byte Write Error (Data Address) (BWERR) bit • 3–18

---

## C

---

Cache coherence in error handling • 2–239  
Cache test procedures • 2–245  
Cbox • 2–6, 2–47  
Cbox (Backup Cache box) • 2–47  
Cbox Control Register (CCTL) • 2–99  
Cbox Error Fill Address Register (CEFADR) • 2–117  
Cbox Error Fill Status Register (CEFSTS) • 2–118  
CC (Corrected Confirmation) bit • 2–176  
CCA • 2–207, 2–210, 2–211, 2–214, 2–220, 2–221 to 2–228  
CCASB\_CHKSUM • 2–224  
CCASB\_FLAGS • 2–227  
CCASB\_HFLAGS • 2–224  
CCASB\_NPROC • 2–224  
CCASB\_POWER • 2–225  
CCASB\_PRIMARY • 2–225  
CCASB\_REVISION • 2–224  
CCASB\_RXLEN • 2–228  
CCASB\_TK\_NODE • 2–225  
CCASB\_TXLEN • 2–228  
CCASB\_ZDATA • 2–228  
CCASB\_ZDEST • 2–227, 2–228  
CCASB\_ZIND • 2–227  
CCASB\_ZSRC • 2–227  
CCASL\_BASE • 2–224  
CCASL\_BITMAP • 2–225  
CCASL\_BITMAP\_CKSUM • 2–225  
CCASL\_BITMAP\_SZ • 2–225  
CCASL\_CONSOLE\_XGPR • 2–226  
CCASL\_ENTRY\_XGPR • 2–226  
CCASL\_RESERVED1 • 2–225  
CCASL\_RESERVED2 • 2–225  
CCASQ\_CONSOLE • 2–224  
CCASQ\_ENABLED • 2–225  
CCASQ\_HW\_REVISION • 2–225  
CCASQ\_READY • 2–224  
CCASQ\_RESERVED3 • 2–226  
CCASQ\_RESERVED4 • 2–226  
CCASQ\_RESERVED5 • 2–226  
CCASQ\_RESTARTIP • 2–215, 2–225  
CCASQ\_SECSTART • 2–225  
CCASQ\_SERIALNUM • 2–225  
CCASQ\_USER\_HALTED • 2–225  
CCASR\_RESERVED0 • 2–225  
CCAST\_RX • 2–228  
CCAST\_TX • 2–228  
CCASV\_BOOTIP • 2–217, 2–224  
CCASV\_DISABLE\_MSG\_2NDARY • 2–224  
CCASV\_REBOOT • 2–224  
CCASV\_REBOOT flag • 2–217  
CCASV\_REPROMPT • 2–224  
CCASV\_RXRDY • 2–228  
CCASV\_ZALT • 2–228  
CCASV\_ZNODE • 2–228  
CCASV\_ZSRC • 2–228  
CCASW\_IDENT • 2–224  
CCASW\_SIZE • 2–224  
CCASW\_SSN\_EXTENSION • 2–225  
CCASW\_ZRXCD • 2–228  
CCAS\_SECSTART • 2–215  
CCID bit  
    See Corrected Confirmation Interrupt Disable bit  
CCR (Corrected Confirmation Received) bit • 3–11  
CCTL (Cbox Control) register • 2–99  
CEFADR (Cbox Error Fill Address) register • 2–117  
CEFSTS (Cbox Error Fill Status) register • 2–118  
CMD (Command) field • 2–194  
CNAK (Command NO ACK) bit • 2–179  
CNT (Count) bit • 2–189  
CNT (TODR increment) bit • 2–164  
CNTSEL (Counter Select) field • 2–189  
Column Parity Error (Data Address) (CPER) bit • 3–19  
COMCD (Commander Code) bit • 3–19  
COMID (Commander ID) bits • 3–19  
Command (CMD) field • 2–194  
Commander Code (COMCD) bit • 3–19  
Commander ID (COMID) bits • 3–19  
Command NO ACK (CNAK) bit • 2–179  
Console communications area  
    See CCA  
Console halt • 2–23 to 2–25  
Console program • 2–220  
Console Receiver Control and Status (RXCS) Register • 2–76  
Console Receiver Data Buffer (RXDB) Register • 2–78  
Console Saved Processor Status Longword (SAVPSL) • 2–85  
Console Saved Program Counter Register (SAVPC) • 2–84  
Console Transmitter Control and Status (TXCS) Register • 2–80  
Console Transmitter Data Buffer (TXDB) Register • 2–82

Control panel  
 location • 1–6  
 Cooling system  
 location • 1–6, 1–8  
 Corrected Confirmation (CC) bit • 2–176  
 Corrected Confirmation Interrupt Disable bit • 2–191  
 Corrected Confirmation Received (CCR) bit • 3–11  
 Corrected Read Data (CRD) bit • 2–178  
 Corrected Read Data Interrupt Disable bit • 2–191  
 Count (CNT) bit • 2–189  
 Counter Select (CNTSEL) field • 2–189  
 CPER (Column Parity Error (Data Address)) bit • 3–19  
 CPUID (CPU Identification) register • 2–71  
 CPU Identification Register (CPUID) • 2–71  
 CPU Type field • 2–91  
 CRD (Corrected Read Data) bit • 2–178  
 CRD bit • 2–212  
 CRDID bit  
 See Corrected Read Data Interrupt Disable bit  
 CSR (Control and Status Registers) • 3–6  
 CTP (CTRL/P Enable) bit • 2–164  
 CTRL/P Enable (CTP) bit • 2–164

---

## D

---

Data CRD Error (DCRDE) bit • 3–18  
 Data ECC Diagnostic Mode (DECCM) bit • 3–14  
 Data ECC Disable (DECCD) bit • 3–15  
 Data Error Address (DERA) bit • 3–21  
 Data In (DI) bit • 2–188  
 Data Out (DO) bit • 2–189  
 Data Read Modify Write Error (DRMWER) bit • 3–16  
 Data RER Error (DRER) bit • 3–17  
 Data Syndrome (DTSYN) bits • 3–20  
 Data types supported by the KA66A CPU module • 2–8  
 DCK (Diagnostic Check) • 3–16  
 DC LO L  
 See XMI DC LO L signal  
 DCLS (Device Class) bits • 3–8  
 DCRDE (Data CRD Error) bit • 3–18  
 DECCD (Data ECC Disable) bit • 3–15  
 DECCM (Data ECC Diagnostic Mode) bit • 3–14  
 DERA (Data Error Address) bit • 3–21  
 Device Class (DCLS) bits • 3–8  
 Device ID (DEVID) bits • 3–9  
 Device Register (XDEV) • 2–173, 3–8  
 Device Revision (DREV) bits • 3–8

Device Revision (DREV) field • 2–173  
 Device Type (DTYPE) field • 2–173  
 DEVID (Device ID) bits • 3–9  
 DI (Data In) bit • 2–188  
 Diagnostic Check (DCK) • 3–16  
 Disabling caches • 2–240  
 DO (Data Out) bit • 2–189  
 DRER (Data RER Error) bit • 3–17  
 DREV (Device Revision) bits • 3–8  
 DREV (Device Revision) field • 2–173  
 DREV field • 2–226  
 DRMWER (Data Read Modify Write Error) bit • 3–16  
 DTSYN (Data Syndrome) bits • 3–20  
 DTYPE (Device Type) field • 2–173  
 DWMBA • 2–210  
 DWMVA adapter • 1–3

---

## E

---

Ebox • 2–5, 2–43  
 Ebox Control Register (ECR) • 2–96  
 ECR (Ebox Control) register • 2–96  
 EEADD (EEPROM Address) bits • 3–41  
 EECTL (EEPROM Control Register) • 3–40  
 EEDAT (EEPROM Data) bits • 3–41  
 EEOC (EEPROM Operation Command) bit • 3–40  
 EEPROM Address (EEADD) bits • 3–41  
 EEPROM Control Register (EECTL) • 3–40  
 EEPROM Data (EEDAT) bits • 3–41  
 EEPROM Operation Command (EEOC) bit • 3–40  
 EEPROM Update Enable (EEUE) bit • 3–32  
 EEUE (EEPROM Update Enable) bit • 3–32  
 EFNDALP (Enable Force NDAL Parity) bit • 2–164  
 EFSMIDP (Enable Force XMI WDAT Parity) bit • 2–166  
 EFSMIP (Enable Force XMI non-WDAT Parity) bit • 2–166  
 Emulated instruction exceptions • 2–20  
 Enable 2-Mbyte Protection Mode (EPM) bit • 3–16  
 Enable Force NDAL Parity (EFNDALP) bit • 2–164  
 Enable Force XMI non-WDAT Parity (EFSMIP) bit • 2–166  
 Enable Force XMI WDAT Parity (EFSMIDP) bit • 2–166  
 Enable Self-Invalidates Only (ESIO) bit • 2–190  
 Enabling caches • 2–241  
 ENADD (Ending Address) bits • 3–29  
 ENADR (Ending Address Register) • 3–28  
 Ending Address (ENADD) bits • 3–29  
 Ending Address Register (ENADR) • 3–28

## Index

EPM (Enable 2-Mbyte Protection Mode) bit • 3–16  
ERR (Error) bit • 2–78  
Error (ERR) bit • 2–78  
Error analysis (general) • 2–236  
Error handling • 2–231 to 2–330, 3–43  
Error recovery (general) • 2–237  
Error Retry • 2–246  
Error Summary (ES) bit • 2–175, 3–10  
ERRSM (MCTL3 Error Summary) bit • 3–32  
ERRSUM (Memory Register Error Summary) bits • 3–14  
ERSUM (MCTL4 Error Summary) bit • 3–34  
ES (Error Summary) bit • 2–175, 3–10  
ESIO (Enable Self-Invalidates Only) bit • 2–190  
ETF (Extended Test Fail) bit • 2–180, 2–207, 2–210  
Exceptions  
    arithmetic • 2–18  
    console halt • 2–23  
    emulated instruction • 2–20  
    machine check • 2–21  
    memory management • 2–19  
Extended Test Fail (ETF) bit • 2–180

---

## F

Failing Address Extension Register • 2–194  
Failing Address field • 2–182  
Failing Commander ID (FCID) field • 2–180  
Failing Length (FLN) field • 2–181  
Failing Writeback Address Extension field • 2–201, 2–202  
Failing Writeback Address field • 2–201, 2–202  
Faults • 2–17  
Fbox • 2–5  
FCID (Failing Commander ID) field • 2–180  
First Part Done (FPD) bit • 2–20, 2–21  
FLN (Failing Length) field • 2–181  
FMRE (Force Memory Refresh) bit • 3–22  
Force Full (FRCFL) bit • 2–166  
Force Memory Refresh (FMRE) bit • 3–22  
FP BOOT DISABLE (Front Panel Boot Disable) bit • 2–169  
FPD (First Part Done) bit • 2–20, 2–21  
FPD bit • 2–249  
FP EEPROM ENABLE (Front Panel EEPROM Enable) bit • 2–169  
Framing Error (FRM ERR) bit • 2–79  
FRCFL (Force Full) bit • 2–166  
FRM ERR (Framing Error) bit • 2–79

Front Panel Boot Disable (FP BOOT DISABLE) bit • 2–169  
Front Panel EEPROM Enable (FP EEPROM ENABLE) bit • 2–169

---

## H

Halts  
    console • 2–23  
Hard error interrupts • 2–277  
HLDM (Hold Mode Refresh Error) bit • 3–23  
Hold Mode Refresh Error (HLDM) bit • 3–23

---

## I

I/O Reset (IORESET) Register • 2–90  
Ibox (Instruction Box) • 2–4, 2–42 to 2–43  
Ibox Control and Status Register (ICSR) • 2–140  
ICCS (Interval Clock Control and Status) register • 2–72  
ICR (Interval Count) register • 2–75  
ICRD (Inhibit CRD Status Generation) bit • 3–15  
ICSR (Ibox Control and Status) register • 2–140  
IE (Interrupt Enable) bit • 2–73  
IEEO (Initiate EEPROM Operation) bit • 3–40  
Implied Vector Interrupts (IVINTR) • 2–60 to 2–61  
INAD (Interleave Address) bits • 3–31  
INCE (Inconsistency Error) bits • 3–33  
Inconsistency Error (INCE) bits • 3–33  
Inconsistent Parity Error (IPE) bit • 2–57, 2–177  
Inhibit CRD Status Generation (ICRD) bit • 3–15  
Initialization • 2–203 to 2–213, 3–4  
Initiate EEPROM Operation (IEEO) bit • 3–40  
INMD (Interleave Mode) bits • 3–31  
INSEQ (Interlock Sequence Error) bit • 3–16  
Instruction Box (Ibox) • 2–42 to 2–43  
Instruction set supported by the KA66A CPU module • 2–9  
Interleave Address (INAD) bits • 3–31  
Interleave Mode (INMD) bits • 3–31  
Interleaving • 3–5  
Interlock Sequence Error (INSEQ) bit • 3–16  
Interprocessor communication • 2–220 to 2–230  
Interprocessor Interrupt (IP IVINTR) bit • 2–184  
Interprocessor IVINTR (IP IVINTR) Response • 2–60  
Interrupt Destination field • 2–184  
Interrupt Enable (IE) bit • 2–73  
Interrupt Priority Level (IPL) field • 2–183

Interrupt Priority Level (IPL) register • 2–15  
 Interrupts  
     hard error • 2–277  
 Interrupt Source field • 2–183  
 Interval Clock • 2–49  
 Interval Clock Control and Status Register (ICCS) • 2–72  
 Interval Count Register (ICR) • 2–75  
 INTLV (Segment/Interleave Register) • 3–30  
 Invalid bit • 2–88  
 IORESET (I/O Reset) register • 2–90  
 IPE (Inconsistent Parity Error) bit • 2–57, 2–177  
 IP IVINTR (Interprocessor Interrupt) bit • 2–184  
 IPL (Interrupt Priority Level) field • 2–183  
 IPL (Interrupt Priority Level) register • 2–15  
 IPORT (NEXMI Input Port) register • 2–168  
 IVINTR mask generation • 2–60

---

## L

---

LDPCTX (Load Process Context) instruction • 2–14  
 LDTE (Lockout Debug Timeout Enable) bit • 2–191  
 Load Process Context (LDPCTX) instruction • 2–14  
 Lockout Debug Timeout Enable (LDTE) bit • 2–191  
 Lockout Mode field • 2–192  
 LOCMOD field  
     See Lockout Mode field

---

## M

---

Machine check codes • 2–23  
 Machine Check Error Summary Register (MCESR) • 2–83  
 Machine check exceptions • 2–21  
 Machine checks • 2–249  
 Machine check stack frame • 2–21 to 2–23  
 MAPEN (Memory Management Enable) register • 2–13, 2–14  
 MAPEN<0> bit • 2–87  
 Mask field • 2–196  
 Mbox (Memory Management box) • 2–5, 2–44 to 2–47  
 MCESR (Machine Check Error Summary) register • 2–83  
 MCTL1 (Memory Control Register 1) • 3–14  
 MCTL2 (Memory Control Register 2) • 3–22  
 MCTL3 (Memory Control Register 3) • 3–32  
 MCTL3 Error Summary (ERRSM) bit • 3–32

MCTL4 (Memory Control Register 4) • 3–34  
 MCTL4 Error Summary (ERSUM) bit • 3–34  
 MECEA (Memory ECC Error Address Register) • 3–21  
 MECER (Memory ECC Error Register) • 3–17  
 Memory • 1–3  
     Memory configuration • 2–211  
     Memory Control Register 1 (MCTL1) • 3–14  
     Memory Control Register 2 (MCTL2) • 3–22  
     Memory Control Register 3 (MCTL3) • 3–32  
     Memory Control Register 4 (MCTL4) • 3–34  
     Memory ECC Error Address Register (MECEA) • 3–21  
     Memory ECC Error Register (MECER) • 3–17  
     Memory interleave • 2–211  
     Memory management • 2–11 to 2–14  
     Memory Management box (Mbox) • 2–44 to 2–47  
     Memory Management Enable (MAPEN) register • 2–13, 2–14  
     Memory Management Exception Address Register (MMEADR) • 2–143  
     Memory Management Exception PTE Address (MMEPTE) Register • 2–144  
     Memory management exceptions • 2–19  
     Memory Management Exception Status Register (MMESTS) • 2–145  
     Memory module  
         description • 3–2  
     Memory Register Error Summary (ERRSUM) bits • 3–14  
     Memory registers • 3–8 to 3–43  
     Memory Size (MEMSIZ) bits • 3–15, 3–35  
     Memory state access • 2–242  
     MEMSIZ (Memory Size) bits • 3–15, 3–35  
     MFPR (Move From Processor Register) instruction • 2–13  
     Microcode Patch Revision field • 2–91  
     Microcode Revision field • 2–92  
     MMEADR (Memory Management Exception Address) register • 2–143  
     MMEPTE (Memory Management Exception PTE Address) register • 2–144  
     MMESTS (Memory Management Exception Status) register • 2–145  
     MODP (Module Population) bit • 3–35  
     Module Population (MODP) bit • 3–35  
     Module Revision field • 2–226  
     Move From Processor Register (MFPR) instruction • 2–13  
     Move To Processor Register (MTPR) instruction • 2–13

## Index

MTPR (Move To Processor Register) instruction • 2-13  
MVAL (On-Board Memory Valid) bit • 3-15

---

## N

---

NCSR (NDAL Control and Status) register • 2-162  
NDAL Control and Status Register (NCSR) • 2-162  
NDAL Error Data High Register (NEDATHI) • 2-130  
NDAL Error Data Low Register (NEDATLO) • 2-132  
NDAL Error Input Command Register (NEICMD) • 2-133  
NDAL Error Output Address Register (NEOADR) • 2-126  
NDAL Error Output Command Register (NEOCMD) • 2-127  
NDAL Error Status Register (NESTS) • 2-123  
NDALFP (NDAL Parity) field • 2-163  
NDAL Inconsistent Parity Error (NIDPE) bit • 2-162  
NDAL Parity (NDALFP) Field • 2-163  
NDAL Parity Error (NDPE) bit • 2-162  
NDAL Read Transmit ACK Error (NRTAE) bit • 2-163  
NDAL Write Sequence Error (NWSE) bit • 2-163  
NDPE (NDAL Parity Error) bit • 2-162  
NEDATHI (NDAL Error Data High) register • 2-130  
NEDATLO (NDAL Error Data) register • 2-132  
NEICMD (NDAL Error Input Command) register • 2-133  
NEOADR (NDAL Error Output Address) register • 2-126  
NEOCMD (NDAL Error Output Command) register • 2-127  
NESTS (NDAL Error Status) register • 2-123  
NEXMI error handling • 2-245  
NEXMI Input Port Register (IPORT) • 2-168  
NEXMI Output Port0 Register (OPORT0) • 2-170  
NEXMI Output Port1 Register (OPORT1) • 2-172  
NEXMI Revision (NREV) field • 2-187  
Next Interval Count Register (NICR) • 2-74  
NHALT • 2-214  
NHALT (Node Halt) bit • 2-175  
NICR (Next Interval Count) register • 2-74  
NIDPE (NDAL Inconsistent Parity Error) bit • 2-162  
NLU (Not last used) pointer • 2-12, 2-14  
Node halt  
    See NHALT  
Node halt (NHALT) bit • 2-175  
Node ID (Node Identification) field • 2-169  
Node Identification (NODE ID) field • 2-169  
Node Reset (NRST) bit • 2-175, 3-10  
Node-Specific Control and Status Register (NCSCSR) • 2-186  
Node-Specific Error Summary (NSES) bit • 2-180, 3-12  
Nonboot processor • 2-214  
Nonstandard Microcode bit • 2-92  
Non-Writeback Queue Full (NWQFL) bit • 2-167  
No Read Response (NRR) bit • 2-178  
Not last used (NLU) pointer • 2-12, 2-14  
NREV (NEXMI Revision) field • 2-187  
NRR (No Read Response) bit • 2-178  
NRST (Node Reset) bit • 2-175, 2-203, 2-210, 3-10  
NRTAE (NDAL Read Transmit ACK Error) bit • 2-163  
NCSCSR (Node-Specific Control and Status) register • 2-186  
NSES (Node-Specific Error Summary) bit • 2-180, 3-12  
NVAX chip overview • 2-3 to 2-6  
NWQFL (Non-Writeback Queue Full) bit • 2-167  
NWSE (NDAL Write Sequence Error) bit • 2-163

---

## O

---

OLR (Only Lockout Response) bit • 2-200  
On-Board Memory Valid (MVAL) bit • 3-15  
Only Lockout Response (OLR) bit • 2-200  
OPORT0 (NEXMI Output Port0) register • 2-170  
OPORT1 (NEXMI Output Port1) register • 2-172  
OSQE (Ownership Sequence Error) bit • 3-36  
Overrun Error (OVR ERR) bit • 2-78  
OVR ERR (Overrun Error) bit • 2-78  
Ownership Sequence Error (OSQE) bit • 3-36

---

## P

---

P0 Base Register (P0BR) • 2-13, 2-14  
P0BR (P0 Base Register) • 2-14  
P0BR register (P0 Base Register) • 2-13  
P0 Length Register (POLR) • 2-13, 2-14  
POLR (P0 Length Register) • 2-13, 2-14  
P1 Base Register (P1BR) • 2-13, 2-14  
P1BR (P1 Base Register) • 2-14  
P1BR register (P1 Base Register) • 2-13  
P1 Length Register (P1LR) • 2-13, 2-14  
P1LR (P1 Length Register) • 2-13, 2-14  
Page frame number (PFN) • 2-12  
Page frame numbers • 2-11  
Page table entries • 2-11

Page table entry (PTE) • 2–12  
 PAMODE (Physical Address Control) register • 2–11  
 PAMODE (Physical Address Mode) register • 2–11, 2–142  
 Parity Error (PE) bit • 2–57, 2–177  
 Patchable Control Store Control Register (PCSCR) • 2–93  
 P-Cache Control Register (PCCTL) • 2–157  
 P-Cache Parity Address Register (PCADR) • 2–153  
 P-Cache Parity Status Register (PCSTS) • 2–154  
 PCADR (P-Cache Parity Address) register • 2–153  
 PCB (process control block) • 2–28  
 PCBB (Process Control Block Base) register • 2–28  
 PCCTL (P-Cache Control) register • 2–157  
 PCSCR (Patchable Control Store Control) register • 2–93  
 PCSTS (P-Cache Status) register • 2–154  
 PE (Bus Parity Error) bit • 3–11  
 PE (Parity Error) bit • 2–57, 2–177  
 PFN (page frame number) • 2–12  
 Physical Address Control (PAMODE) register • 2–11  
 Physical Address Mode Register (PAMODE) • 2–11, 2–142  
 Physical address space • 2–10 to 2–11  
 Power fail interrupt • 2–276  
 Power regulators  
   location • 1–6, 1–8  
 Primary processor  
   See BP  
 Primary system bootstrap program  
   See VMB  
 Process context • 2–14  
 Process control block (PCB) • 2–28  
 Process Control Block Base (PCBB) register • 2–28  
 Processor • 1–3  
 Processor status longword (PSL) • 2–15  
 PROT (Protection) field • 2–12  
 Protection (PROT) field • 2–12  
 PSL (processor status longword) • 2–15  
 PTE (page table entry) • 2–12

---

## R

---

RAMTYP (RAM Type) bits • 3–15, 3–35  
 RAM Type (RAMTYP) bits • 3–15, 3–35  
 RBAT (ROM Bus Access Time) bit • 2–165  
 R bit • 2–249  
 RCV BRK (Received Break) bit • 2–79  
 RDNAK (Read Data NO ACK) bit • 3–12

RDS errors • 2–212  
 Read Data NO ACK (RDNAK) bit • 3–12  
 Read Error Response (RER) bit • 2–179  
 Read/IDENT Data NO ACK (RIDNAK) bit • 2–177  
 Read Sequence Error (RSE) bit • 2–178  
 Received Break (RCV BRK) bit • 2–79  
 Received Data bits • 2–79  
 Receiver Done (RX DONE) bit • 2–76  
 Receiver Interrupt Enable (RX IE) bit • 2–76  
 Refresh Error (RERR) bit • 3–22  
 Refresh Rate (RRB) bits • 3–23  
 Registers, KA66A CPU module • 2–63 to 2–159  
 Repairing Memory State • 2–242, 2–243  
 RER (Read Error Response) bit • 2–179  
 RERR (Refresh Error) bit • 3–22  
 Responder Queue Overflow bit • 2–186  
 Restarting • 2–214  
 Restart parameter block  
   See RPB  
 RIDNAK (Read/IDENT Data NO ACK) bit • 2–177  
 ROM Bus Access Time (RBAT) bit • 2–165  
 Row Parity Error (Data Address) (RPER) bit • 3–18  
 RPB • 2–215  
 RPER (Row Parity Error (Data Address)) bit • 3–18  
 RRB (Refresh Rate) bits • 3–23  
 RSE (Read Sequence Error) bit • 2–178  
 RXCS (Console Receiver Control and Status) register • 2–76  
 RXDB (Console Receiver Data Buffer) register • 2–78  
 RX DONE (Receiver Done) bit • 2–76  
 RX IE (Receiver Interrupt Enable) bit • 2–76

---

## S

---

SAVPC (Console Saved Program Counter) register • 2–84  
 SAVPSL (Console Saved Processor Status Longword) • 2–85  
 SBR (System Base Register) • 2–13, 2–14  
 SCB (System control block) • 2–25  
 SCBB (System Control Block Base) register • 2–25  
 SCB entry points • 2–231  
 SDEO (Second Data Error Occurred) bit • 3–18  
 SECCON (Secure Console) bit • 2–167  
 Secondary processor  
   See Nonboot processor  
 Second Data Error Occurred (SDEO) bit • 3–18  
 Second Error Occurred (SEO) bit • 2–200  
 Secure Console (SECCON) bit • 2–167

## Index

- SEGADR (Segment Address) bits • 3–30
  - Segment Address (SEGADR) bits • 3–30
  - Segment/Interleave Register (INTLV) • 3–30
  - Self-test • 3–4
  - Self-Test Fail (STF) bit • 2–180, 3–12
  - Self-Test Loop Disable (STL DISABLE) bit • 2–168
  - SEO (Second Error Occurred) bit • 2–200
  - SID (System Identification) register • 2–91
  - SIRR (Software Interrupt Request Register) • 2–15
  - SISR (Software Interrupt Summary Register) • 2–15
  - SLR (System Length Register) • 2–13, 2–14
  - Software Interrupt Request Register (SIRR) • 2–15
  - Software Interrupt Summary Register (SISR) • 2–15
  - SSC Illegal Read (SSCIR) bit • 2–165
  - SSC Illegal Write (SSCIW) bit • 2–165
  - SSCIR (SSC Illegal Read) bit • 2–165
  - SSCIW (SSC Illegal Write) bit • 2–165
  - STADD (Starting Address) bits • 3–27
  - STADR (Starting Address Register) • 3–27
  - Starting Address (STADD) bits • 3–27
  - Starting Address Register (STADR) • 3–27
  - STF (Self-Test Fail) bit • 2–180, 3–12
  - STF bit • 2–207, 2–211
  - STL DISABLE (Self-Test Loop Disable) bit • 2–168
  - STP LED (Self-Test Passed LED) bit • 2–207
  - System
    - architecture • 1–2
    - front view • 1–6
    - rear view • 1–8
    - typical • 1–5
  - System Base Register (SBR) • 2–13, 2–14
  - System control block • 2–25
  - System Control Block Base (SCBB) register • 2–25
  - System Identification (SID) Register • 2–91
  - System Length Register (SLR) • 2–13, 2–14
- 
- ## T
- 
- TB (translation buffer) • 2–12
  - TB.V (Translation Buffer Valid) bit • 2–12
  - TBADR (TB Parity Address) register • 2–148
  - TBCHK (Translation Buffer Check) register • 2–13, 2–14
  - TBIA (Translation Buffer Invalidate All) register • 2–13
  - TBIS (Translation Buffer Invalidate Single) register • 2–13
  - TB Parity Address Register (TBADR) • 2–148
  - TB Parity Status Register (TBSTS) • 2–149
  - TBSTS (TB Parity Status) register • 2–149
  - TB test procedures • 2–245
  - TCY (TCY Tester Register) • 3–24
  - TCY Tester Register (TCY) • 3–24
  - Test Mode (TM) bit • 2–164, 2–188
  - TF tape drive
    - location • 1–6
  - Time-of-Day Register (TODR) • 2–49
  - Time-of-Year Clock (TOY) • 2–50
  - Timeout Control/Status Register (TMOER) • 3–42
  - Timeout Select (TOS) bit • 2–190
  - TK tape drive
    - location • 1–6
  - TM (Test Mode) bit • 2–164, 2–188
  - TMOER (Timeout Control/Status Register) • 3–42
  - TODR (Time-of-Day Register) • 2–49
  - TODR increment (CNT) bit • 2–164
  - Top Segment Memory Ending Address (TSMEA) bit • 3–28
  - TOS (Timeout Select) bit • 2–190
  - TOY (Time-of-Year Clock) • 2–50
  - Transaction Timeout (TTO) bit • 2–179
  - Translation buffer (TB) • 2–12
  - Translation Buffer Check (TBCHK) register • 2–13, 2–14
  - Translation Buffer Invalidate All (TBIA) register • 2–13
  - Translation Buffer Invalidate Single (TBIS) register • 2–13
  - Translation Buffer Valid (TB.V) bit • 2–12, 2–14
  - Transmit Data field • 2–82
  - Transmitter Interrupt Enable (TX IE) bit • 2–80
  - Transmitter Ready (TX RDY) bit • 2–80
  - Trap2 bit • 2–249
  - Traps • 2–17
  - TREN (Trigger Enable) bit • 3–33
  - TRGM (Trigger Configuration Mode) bits • 3–33
  - TRIGC field
    - See Trigger Control field
  - Trigger Configuration Mode (TRGM) bits • 3–33
  - Trigger Control field • 2–192
  - Trigger Enable (TREN) bit • 3–33
  - TSMEA (Top Segment Memory Ending Address) bit • 3–28
  - TTO (Transaction Timeout) bit • 2–179
  - TXCS register • 2–80
  - TXDB (Console Transmitter Data Buffer) register • 2–82
  - TX IE (Transmitter Interrupt Enable) bit • 2–80
  - TX RDY (Transmitter Ready) bit • 2–80

---

## U

---

Unexpected Read Response (URR) bit • 2–200  
 URR (Unexpected Read Response) bit • 2–200  
 UWP bit • 2–249

---

## V

---

VAXBI bus • 1–3  
 VAXBI card cages  
   location • 1–6, 1–8  
 VDATA (VIC Data) register • 2–139  
 VIC (virtual instruction cache) • 2–31  
 VIC Data Register (VDATA) • 2–139  
 VIC Memory Address Register (VMAR) • 2–135  
 VIC Tag Register (VTAG) • 2–137  
 Virtual instruction cache (VIC) • 2–31  
 Virtual page number (VPN) field • 2–12  
 VMAR (VIC Memory Address) register • 2–135  
 VMB • 2–217  
 VMEbus • 1–3  
 VPN (Virtual page number) field • 2–12  
 VTAG (VIC Tag) register • 2–137

---

## W

---

Warm Start (WS) bit • 2–187  
 WBQFL (Writeback Queue Full) bit • 2–166  
 WCNAK0 (Writeback0 Command NO ACK) bit • 2–199  
 WCNAK1 (Writeback1 Command NO ACK) bit • 2–198  
 WDNAK (Write Data NO ACK) bit • 2–178  
 WEI (Write Error Interrupt) bit • 2–176  
 WEI INVINTR (Write Error Interrupt) bit • 2–184  
 WEI mask generation • 2–60  
 WFADR0 (Writeback 0 Failing Address Register) • 2–201  
 WFADR1 (Writeback 1 Failing Address Register) • 2–202  
 Writeback0 Command NO ACK (WCNAK0) bit • 2–199  
 Writeback 0 Failing Address Register (WFADR0) • 2–201  
 Writeback0 Second Error Occurred (WSEO0) bit • 2–199  
 Writeback0 Transaction Timeout (WTTO0) bit • 2–199

Writeback0 Write Data NO ACK (WWDNAK0) bit • 2–199  
 Writeback1 Command NO ACK (WCNAK1) bit • 2–198  
 Writeback 1 Failing Address Register (WFADR1) • 2–202  
 Writeback1 Second Error Occurred (WSEO1) bit • 2–198  
 Writeback1 Transaction Timeout (WTTO1) bit • 2–198  
 Writeback1 Write Data NO ACK (WWDNAK1) bit • 2–198  
 Writeback queue • 2–58  
 Writeback Queue Full (WBQFL) bit • 2–166  
 Write Data NO ACK (WDNAK) bit • 2–178  
 Write Error Interrupt (WEI) bit • 2–176  
 Write Error Interrupt (WEI INVINTR) bit • 2–184  
 Write Error IVINTR (WE IVINTR) Response • 2–61  
 Write packer • 2–6  
 Write ROM Error (WRT) bit • 2–165  
 Write Sequence Error (WSE) bit • 2–177, 3–11  
 WRT (Write ROM Error) bit • 2–165  
 WS (Warm Start) bit • 2–187  
 WSE (Write Sequence Error) bit • 2–177, 3–11  
 WSEO0 (Writeback0 Second Error Occurred) bit • 2–199  
 WSEO1 (Writeback1 Second Error Occurred) bit • 2–198  
 WTTO0 (Writeback0 Transaction Timeout) bit • 2–199  
 WTTO1 (Writeback1 Transaction Timeout) bit • 2–198  
 WWDNAK0 (Writeback0 Write Data NO ACK) bit • 2–199  
 WWDNAK1 (Writeback1 Write Data NO ACK) bit • 2–198

---

## X

---

XACLO (XMI AC LO) bit • 2–168  
 XBAD (XMI BAD) bit • 2–176  
 XBAD bit • 2–210  
 XBADD bit  
   See XMI BAD Drive bit  
 XBEER (Bus Error Extension) register • 2–197  
 XBER (Bus Error Register) • 2–174, 3–10  
 XCR register  
   See XMI Control Register  
 XDEV (Device Register) • 2–173, 3–8  
 XFADR (Failing Address Register) • 2–181  
 XFAER register (Failing Address Extension Register) • 2–194  
 XGPR (XMI General Purpose Register) • 2–185

## Index

- XMI AC LO (XACLO) bit • 2–168
- XMI AC LO L signal • 2–203
- XMI adapters • 1–10 to 1–11
- XMI BAD (XBAD) bit • 2–176
- XMI BAD Drive bit • 2–192
- XMI BAD L signal • 2–176, 2–207
- XMI bus • 1–3
- XMI card cage
  - location • 1–6, 1–8
- XMI Control Register • 2–188
- XMI DC LO L signal • 2–203
- XMI Device Interrupt Priority • 2–60
- XMI Failing Address Register (XFADR) • 2–181
- XMI Force Bad Parity<2:0> (XMIFP) field • 2–190
- XMIFP (XMI Force Bad Parity<2:0>) field • 2–190
- XMI General Purpose Register (XGPR) • 2–185
- XMI RESET L signal • 2–203
- XTC module
  - location • 1–8
- XTC power sequencer • 2–203