KA640 CPU Module Technical Manual

Order Number EK-KA640-TM-001

digital equipment corporation
maynard, massachusetts

First Edition, September 1988

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 ©1988 by Digital Equipment Corporation.

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

The following are trademarks of Digital Equipment Corporation:

DEC	MicroVAX 3300	RT
DECmate	MicroVAX 3400	ThinWire
DECUS	MicroVAX 3500	UNIBUS
DECwriter	PDP	VAX
DIBOL	P/OS	VAXstation
LSI-11	Professional	VMS
MASSBUS	Q-bus	VT
MicroPDP-11	Q22-bus	VT100
MicroVAX	Rainbow	Work Processor
MicroVAX I	RSTS	
MicroVAX II	RSX	

digital™

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

Contents

About This Manual xvii

1 Overview

1.1 Introduction 1
1.2 Central Processing Unit 3
1.3 Floating Point Accelerator 4
1.4 Cache Memory 5
1.5 Memory Controller 5
1.6 MS650-AA Memory Modules 5
1.7 DSSI Interface 5
1.8 Ethernet Interface 6
1.9 Q22-bus Interface 7
1.10 MicroVAX System Support Functions 7
1.11 Firmware 8
1.12 Clock Functions 8

2 Installation and Configuration

2.1 Introduction 9
2.2 Installing the KA640 9
2.3 Configuring the KA640 11
2.4 KA640 Connectors 11
2.4.1 System Support Connector (J1) 12
2.4.2 Memory/DSSI Connector (J2) 14
2.5 H3602-SA CPU Cover Panel 15
2.6 Compatible System Enclosures 17

3 Architecture

3.1 Central Processor 18
3.1.1 Processor State 19
3.1.1.1 General Purpose Registers 19
3.1.1.2 Processor Status Longword 20
3.1.1.3 Internal Processor Registers 21
3.1.2 Data Types 25
3.1.3 Instruction Set 25
3.1.4 Memory Management 26
3.1.4.1 Translation Buffer 26
3.1.4.2 Memory Management Control Registers 27
3.1.5 Exceptions and Interrupts 28
3.1.5.1 Interrupts 28
3.1.5.2 Exceptions 30
3.1.5.3 Information Saved On A Machine Check Exception . . . 33
3.1.5.4 System Control Block 39
3.1.5.5 Hardware Detected Errors 41
3.1.5.6 Hardware Halt Procedure 42
3.1.6 System Identification 44
3.1.7 CPU References 45
3.1.7.7 Instruction-Stream Read References 45
3.1.7.8 Data-Stream Read References 46
3.1.7.9 Write References 46
3.2 Floating Point Accelerator 46
3.2.1 Floating Point Accelerator Instructions 47
3.2.2 Floating Point Accelerator Data Types 47
3.3 Cache Memory 47
3.3.1 Cacheable References 47
3.3.2 Cache 48
3.3.2.1 Cache Organization 48
3.3.2.2 Cache Address Translation 49
3.3.2.3 Cache Data Block Allocation 50
3.3.2.4 Cache Behavior on Writes 51
3.3.2.5 Cache Disable Register 51
3.3.2.6 Memory System Error Register 54
3.3.2.7 Cache Error Detection 55

3.4 Main Memory System 56
3.4.1 Main Memory Organization 59
3.4.2 Main Memory Addressing 59
3.4.3 Main Memory Behavior on Writes 60
3.4.4 Main Memory Error Status Register 60
3.4.5 Main Memory Control and Diagnostic Status Register . . . 64
3.4.6 Main Memory Error Detection and Correction 66
3.5 Console Serial Line 68
3.5.1 Console Registers 68
3.5.1.1 Console Receiver Control/Status Register 68
3.5.1.2 Console Receiver Data Buffer 69
3.5.1.3 Console Transmitter Control/Status Register 70
3.5.1.4 Console Transmitter Data Buffer 72
3.5.2 Break Response 72
3.5.3 Baud Rate 72
3.5.4 Console Interrupt Specifications 73
3.6 Time of Year Clock and Timers 73
3.6.1 Time of Year Clock 74
3.6.2 Interval Timer 74
3.6.3 Programmable Timers 75
3.6.3.1 Timer Control Registers 75
3.6.3.2 Timer Interval Registers 77
3.6.3.3 Timer Next Interval Registers 78
3.6.3.4 Timer Interrupt Vector Registers 78
3.7 Boot and Diagnostic Facility 79
3.7.1 Boot and Diagnostic Register 79
3.7.2 Diagnostic LED Register 81
3.7.3 ROM Memory 82
3.7.3.1 ROM Socket 82
3.7.3.2 ROM Address Space 82
3.7.3.3 KA640 Resident Firmware Operation 83
3.7.4 Battery Backed-Up RAM 84

3.7.5 KA640 Initialization 84
3.7.5.1 Power-Up Initialization 84
3.7.5.2 Hardware Reset 84
3.7.5.3 I/O Bus Initialization 85
3.7.5.4 Processor Initialization 85
3.8 Q22-bus Interface 85
3.8.1 Q22-bus to Main Memory Address Translation 86
3.8.1.1 Q22-bus Map Registers 88
3.8.1.2 Accessing the Q22-bus Map Registers 89
3.8.1.3 Q22-bus Map Cache 90
3.8.2 CDAL Bus to Q22-bus Address Translation 91
3.8.3 Interprocessor Communication Register 92
3.8.4 Q22-bus Interrupt Handling 93
3.8.5 Configuring the Q22-bus Map 94
3.8.5.1 Q22-bus Map Base Address Register 94
3.8.6 System Configuration Register 95
3.8.7 DMA System Error Register 96
3.8.8 Q22-bus Error Address Register 98
3.8.9 DMA Error Address Register 99
3.8.10 Error Handling 100
3.9 Network Interface 101
3.9.1 Ethernet Overview 102
3.9.2 Network Interface Station Address ROM 103
3.9.3 LANCE Chip Overview 104
3.9.4 Network Interface Register Address Port 105
3.9.5 Network Interface Register Data Port 106
3.9.6 Network Interface Control and Status Register 0 107
3.9.7 Network Interface Control and Status Register 1 112
3.9.8 Network Interface Control and Status Register 2 112
3.9.9 Network Interface Control and Status Register 3 113
3.9.10 Network Interface Initialization Block 114
3.9.10.1 Network Interface Initialization Block Word 0 115
3.9.10.2 Network Interface Initialization Block Words 1-3 118
3.9.10.3 Network Interface Initialization Block Words 4-7 118
3.9.10.4 Network Interface Initialization Block Words 8,9 120
3.9.10.5 Network Interface Initialization Block Words 10,11 . . . 121

3.9.11 Buffer Management 123
3.9.12 Network Interface Receive Descriptor Ring 124
3.9.12.1 Receive Buffer Descriptors 124
3.9.13 Receive Buffers 128
3.9.14 Network Interface Transmit Descriptor Ring 129
3.9.14.1 Transmit Buffer Descriptors 130
3.9.15 Transmit Buffers 134
3.9.16 LANCE Operation 135
3.9.16.1 Switch Routine 136
3.9.16.2 Initialization Routine 136
3.9.16.3 Look-For-Work Routine 137
3.9.16.4 Receive Poll Routine 137
3.9.16.5 Receive Routine 137
3.9.16.6 Receive DMA Routine 138
3.9.16.7 Transmit Poll Routine 138
3.9.16.8 Transmit Routine 139
3.9.16.9 Transmit DMA Routine 139
3.9.16.10 Collision Detect Routine 139
3.9.17 LANCE Programming Notes 140
3.10 Mass Storage Interface 142
3.10.1 DSSI Bus Overview 143
3.10.2 Target Operation 145
3.10.3 Initiator Operation 146
3.10.3.1 Transmit Data Segment Links 146
3.10.4 Adding To A Buffer List 148
3.10.5 MSI Command Block (MSICB) 148
3.10.5.1 MSI Command Block Word 0 149
3.10.5.2 MSI Command Block Word 1 149
3.10.5.3 MSI Command Block Word 2 151
3.10.5.4 MSI Command Block Words 3-5 152
3.10.6 MSI Registers 152
3.10.6.1 MSI Control and Status Registers 152
3.10.6.2 List Pointer Registers 162
3.10.6.3 Diagnostic and Test Registers 163

4 KA640 Firmware

4.1 KA640 Firmware Features 171
4.1.1 Power Up Processing 173
4.1.2 Mode Switch Set to Test 173
4.1.3 Mode Switch Set to Language Inquiry 174
4.1.4 Mode Switch Set to Normal 174
4.1.5 KA640 ROM-Based Diagnostics 177
4.2 Halts 180
4.2.1 External Halts 180
4.2.2 Determination Of The Console Device 181
4.3 Console Emulation 181
4.3.1 Console Control Characters 181
4.3.2 Console Commands 183
4.3.2.1 Command Syntax 183
4.3.2.2 Address Specifiers 183
4.3.2.3 Symbolic Addresses 184
4.3.2.4 Console Command Qualifiers 187
4.3.2.5 Console Command Keywords 188
4.3.2.6 Conventions for Tables 4–5 and 4–6 189
4.3.2.7 References to Processor Registers and Memory 192
4.3.2.8 BOOT 193
4.3.2.9 CONFIGURE 195
4.3.2.10 CONTINUE 196
4.3.2.11 DEPOSIT 197
4.3.2.12 EXAMINE 199
4.3.2.13 FIND 201
4.3.2.14 HALT 202
4.3.2.15 HELP 203
4.3.2.16 INITIALIZE 205
4.3.2.17 MOVE 206
4.3.2.18 NEXT 208
4.3.2.19 REPEAT 209
4.3.2.20 SEARCH 210
4.3.2.21 SET 213
4.3.2.22 SHOW 216
4.3.2.23 START 219

4.3.2.24 TEST 220
4.3.2.25 UNJAM 223
4.3.2.26 X - Binary Load and Unload 224
4.3.2.27 ! - Comment 226
4.4 Bootstrapping 226
4.4.1 Preparing for the Bootstrap 227
4.4.1.1 Boot Devices 228
4.4.1.2 Boot Flags 229
4.4.2 Primary Bootstrap, VMB 230
4.4.3 Device Dependent Bootstrap Procedures 232
4.4.3.1 Disk and Tape Bootstrap Procedure 232
4.4.3.2 PROM Bootstrap Procedure 233
4.4.3.3 Network Bootstrap Procedure 234
4.5 Operating System Restart 237
4.5.1 Locating the RPB 238
4.6 Machine State When Halted 238
4.6.1 Main Memory Layout and State 238
4.6.1.1 Reserved Main Memory 239
4.6.1.2 Scatter/Gather Map 239
4.6.1.3 Bitmap 239
4.6.1.4 Contents of Main Memory 240
4.6.2 Cache Memory 240
4.6.3 Translation Lookaside Buffer 240
4.6.4 Halt Protect Space 241
4.7 Public Data Structures and Entry Points 241
4.7.1 Firmware EPROM Layout 241
4.7.2 Call Back Entry Points 242
4.7.2.1 CP$GETCHAR_R4 242
4.7.2.2 CP$MSG_OUT_NOLF_R4 243
4.7.2.3 CP$READ_WTH_PRMPT_R4 244
4.7.3 SSC RAM Layout 245
4.7.4 Public Data structures 245
4.7.4.1 Console Program Mailbox (CPMBX) 245
4.7.4.2 Firmware Stack 247
4.7.4.3 Diagnostic State 247
4.7.4.4 USER Area 247

4.8 Error Messages 247
4.8.1 Halt Messages 247
4.8.2 Console Error Messages 248
4.8.3 VMB Error Messages 249

A Specifications

A.1 Physical Specifications 251
A.2 Electrical Specifications 251
A.3 Environmental Specifications 252

B Address Assignments

B.1 General Local Address Space Map 253
B.2 Detailed Local Address Space Map 254
B.3 External IPRs 257
B.4 Global Q22-bus Address Space Map 258

C Q22-bus Specification

C.1 General Description 260
C.1.1 Master/Slave Relationship 261
C.2 Q22-bus Signal Assignments 262
C.3 Data Transfer Bus Cycles 265
C.3.1 Bus Cycle Protocol 265
C.3.2 Device Addressing 266
C.4 Direct Memory Access 276
C.4.1 DMA Protocol 276
C.4.2 Block Mode DMA 277
C.4.2.1 DATBI Bus Cycle 282
C.4.2.2 DATBO Bus Cycle 283
C.4.3 DMA Guidelines 285
C.5 Interrupts 285
C.5.1 Device Priority 286
C.5.2 Interrupt Protocol 286
C.5.3 Q22-bus 4-Level Interrupt Configurations 290
C.6 Control Functions 292

C.6.1 Memory Refresh 292
C.6.2 Halt 292
C.6.3 Initialization 292
C.6.4 Power Status 292
C.6.5 BDCOK H 293
C.6.6 BPOK H 293
C.6.7 Power-Up/Power-Down Protocol 293
C.7 Q22-bus Electrical Characteristics 294
C.7.1 Signal Level Specifications 294
C.7.2 Load Definition 294
C.7.3 120-Ohm Q22-bus 295
C.7.4 Bus Drivers 295
C.7.5 Bus Receivers 296
C.7.6 Bus Termination 296
C.7.7 Bus Interconnecting Wiring 297
C.7.7.1 Backplane Wiring 297
C.7.7.2 IntraBackplane Bus Wiring 298
C.7.7.3 Power and Ground 298
C.8 System Configurations 298
C.8.1 Power Supply Loading 300
C.9 Module Contact Finger Identification 301

D Acronyms

Examples

4-1 Language Selection Menu 174
4-2 Sample Output with No Errors 175
4-3 Sample Output with Errors 175

Figures

1-1 KA640 CPU Module 2
1-2 KA640 Block Diagram 3
1-3 MS650-AA Memory Module 6
2-1 CPU and Memory Module Placement 10
2-2 Cable Connections 10
2-3 KA640 Pin and LED Orientation 11
2-4 H3602-SA CPU Cover Panel 16
3-1 General Purpose Register Bit Map 19
3-2 PSL Bit Map 20
3-3 Interrupt Registers 30
3-4 Information Saved On A Machine Check Exception 33
3-5 System Control Block Base Register 39
3-6 System Identification Register 44
3-7 System Type Register 45
3-8 Cache Organization 48
3-9 Cache Entry 48
3-10 Cache Tag Block 49
3-11 Cache Data Block 49
3-12 Cache Address Translation 50
3-13 Cache Disable Register 51
3-14 Memory System Error Register 54
3-15 Format for MEMCSR16 60
3-16 Format for MEMCSR17 64
3-17 Console Receiver Control/Status Register 69
3-18 Console Receiver Data Buffer 69
3-19 Console Transmitter Control/Status Register 71
3-20 Console Transmitter Data Buffer 72
3-21 Time of Year Clock 74
3-22 Interval Timer 74
3-23 Timer Control Registers 76
3-24 Timer Interval Register 77
3-25 Timer Next Interval Register 78
3-26 Timer Interrupt Vector Register 78
3-27 Boot and Diagnostic Register 79
3-28 Diagnostic LED Register 81

3-29 Q22-bus to Main Memory Address Translation 87
3-30 Q22-bus Map Registers 88
3-31 Q22-bus Map Cache Entry 91
3-32 The Interprocessor Communication Register 92
3-33 Q22-bus Map Base Address Register 94
3-34 System Configuration Register 95
3-35 DMA System Error Register 97
3-36 Q22-bus Error Address Register 99
3-37 DMA Error Address Register 100
3-38 Ethernet Data Packet Format 102
3-39 Network Interface Station Address ROM Format 104
3-40 Network Interface Register Address Port 105
3-41 Network Interface Control and Status Register 107
3-42 Network Interface Control and Status Register 112
3-43 Network Interface Control and Status Register 2 113
3-44 Network Interface Control and Status Register 3 113
3-45 Network Interface Initialization Block 115
3-46 Network Interface Initialization Block Word 0 115
3-47 Network Interface Initialization Block Words 1-3 118
3-48 Network Interface Initialization Block Words 4-7 119
3-49 Network Interface Initialization Block Word 8 120
3-50 Network Interface Initialization Block Word 9 120
3-51 Network Interface Initialization Block Word 10 121
3-52 Network Interface Initialization Block Word 11 122
3-53 Network Interface Receive Descriptor Ring 124
3-54 Receive Buffer Descriptors 125
3-55 Receive Buffer Descriptor n Word 0 125
3-56 Receive Buffer Descriptor n Word 1 126
3-57 Receive Buffer Descriptor n Word 2 127
3-58 Receive Buffer Descriptor n Word 3 128
3-59 Receive Buffers 129
3-60 Network Interface Transmit Descriptor Ring 130
3-61 Transmit Buffer Descriptors 130
3-62 Transmit Buffer Descriptor n Word 0 131
3-63 Transmit Buffer Descriptor n Word 1 131
3-64 Transmit Buffer Descriptor n Word 2 133

3-65 Transmit Buffer Descriptor n Word 3 133
3-66 Transmit Buffers 135
3-67 DSSI Bus Sequences 144
3-68 Target Operation 145
3-69 Transmit Data Segment Links 146
3-70 Initiator Operation 147
3-71 MSI Command Block 148
3-72 MSI Command Block Word 0 149
3-73 MSI Command Block Word 1 149
3-74 MSI Command Block Word 2 151
3-75 MSI Control/Status Register 152
3-76 MSI DSSI Control Register 154
3-77 MSI DSSI Connection Register 156
3-78 MSI ID Register 160
3-79 MSI DSSI Timeout Register 161
3-80 MSI Target List Pointer Register 162
3-81 MSI Initiator List Pointer Register 163
3-82 MSI Diagnostic Control Register 164
3-83 MSI Diagnostic Register 0 165
3-84 MSI Diagnostic Register 1 167
3-85 MSI Diagnostic Register 2 169
3-86 MSI Clock Control Register 170
4-1 Boot Block Format 233
C-1 DATI Bus Cycle 268
C-2 DATI Bus Cycle Timing 270
C-3 DATO or DATOB Bus Cycle 271
C-4 DATO or DATOB Bus Cycle Timing 273
C-5 DATIO or DATIOB Bus Cycle 274
C-6 DATIO or DATIOB Bus Cycle Timing 275
C-7 DMA Protocol 278
C-8 DMA Request/Grant Timing 279
C-9 DATBI Bus Cycle Timing 280
C-10 DATBO Bus Cycle Timing 281
C-11 Interrupt Request/Acknowledge Sequence 287
C-12 Interrupt Protocol Timing 289
C-13 Position-Independent Configuration 291

C–14 Position-Dependent Configuration 291
C–15 Power-Up/Power-Down Timing 294
C–16 Bus Line Terminations 296
C–17 Single-Backplane Configuration 299
C–18 Multiple-Backplane Configuration 301
C–19 Typical Pin Identification System 302
C–20 Quad-Height Module Contact Finger Identification 303
C–21 Typical Q22-bus Module Dimensions 304

Tables

2–1 System Support Connector (J1) Pinouts 12
2–2 Memory/DSSI Connector (J2-Lower) Pinouts 14
2–3 DSSI Connector (J2-Upper) Pinouts 15
2–4 H3602-SA Features and Controls 16
3–1 KA640 Internal Processor Registers 22
3–2 VAX Standard IPRs 24
3–3 KA640 Unique IPRs 24
3–4 Interrupts 28
3–5 Exceptions 32
3–6 System Control Block Format 39
3–7 Unmaskable Interrupts That Can Cause a Halt 43
3–8 Exceptions That Can Cause A Halt 43
3–9 CPU Read Reference Timing 57
3–10 CPU Write Reference Timing 57
3–11 Q22-bus Interface Read Reference Timing 57
3–12 Q22-bus Interface Write Reference Timing 58
3–13 Error Syndromes 63
3–14 Console Registers 68
3–15 Baud Rate Select 73
3–16 Q22-bus Map 88
4–1 Actions Taken on a Halt 172
4–2 Diagnostic Tests and LED Codes 178
4–3 Console Symbolic Addresses 184
4–4 Command, Parameter, and Qualifier Keywords 188
4–5 Console Command Summary 190
4–6 Console Qualifier Summary 191

4-7 KA640 Supported Boot Devices 228
4-8 VMB Boot Flags 230
4-9 KA640 Network Maintenance Operations Summary 236
4-10 MOP/Ethernet Multicast Addresses and Protocols 236
4-11 NVR0 246
4-12 NVR1 246
4-13 NVR2 246
4-14 Halt Messages 247
4-15 Console Error Messages 248
4-16 VMB Error Messages 250
B-1 VAX Memory Space 253
B-2 VAX Input/Output Space 253
B-3 VAX Memory Space 254
B-4 VAX Input/Output Space 254
B-5 External IPRs 258
B-6 Q22-bus Memory Space 258
B-7 Q22-bus I/O Space with BBS7 Asserted 259
C-1 Data and Address Signal Assignments 262
C-2 Control Signal Assignments 263
C-3 Power and Ground Signal Assignments 264
C-4 Spare Signal Assignments 264
C-5 Data Transfer Operations 265
C-6 Bus Signals for Data Transfers 266
C-7 Bus Pin Identifiers 305

About This Manual

This *KA640 CPU Module Technical Manual* documents the functional, physical, and environmental characteristics of the KA640 CPU module, and also includes some information on the MS650 memory expansion modules. This manual also covers the KA640-BA CPU module, designed for workstations and VAXservers. The KA640-BA is functionally equivalent to the KA640-AA, except that it does not support multiuser VMS and ULTRIX operating system licenses.

This document is intended for a design engineer or applications programmer who is familiar with Digital's extended LSI-11 bus (Q22-bus) and the VAX instruction set. This manual should be used along with the *VAX Architecture Reference Manual* as a programmer's reference to the module.

The manual is divided into four chapters and four appendices.

Chapter 1, **Overview**, introduces the KA640 MicroVAX CPU module and MS650 memory modules, including module features and specifications.

Chapter 2, **Configuration and Installation**, describes the configuration and installation of the KA640 and MS650 modules in Q22-bus backplanes and system enclosures.

Chapter 3, **Architecture**, provides a description of KA640 registers, instruction set and memory.

Chapter 4, **KA640 Firmware**, describes the entry/dispatch code, boot diagnostics, device booting sequence, console program and console commands.

Appendix A, **KA640 Specifications**, describes the physical, electrical, and environmental specifications for the KA640 CPU module.

Appendix B, **Address Assignments**, provides a map of VAX memory space.

Appendix C, **Q22-bus Specification**, describes the low end member of Digital's bus family. All of Digital's microcomputers, such as the MicroVAX I, MicroVAX II, MicroVAX 3300, MicroVAX 3400, MicroVAX 3500, MicroVAX 3600, and MicroPDP-11, use the Q22-bus.

Appendix D, **Acronyms**, provides a list of the acronyms used in this manual.

CONVENTIONS

The following table lists the conventions used in this manual.

Convention	Meaning
<x:y>	Represents a bit field, a set of lines, or signals, ranging from x through y. For example, R0 <7:4> indicates bits 7 through 4 in general purpose register R0.
[x:y]	Represents a range of bytes, from y through x.
Return	A label enclosed in a box represents a key (usually a control or special character key) on the keyboard (in this case, the carriage return key).
Note	Contains general information.
Caution	Contains information to prevent damage to equipment.
n	Indicates variables.

RELATED DOCUMENTS

Manual	Order Number
Microcomputer Interfaces Handbook	EB-20175-20
Microcomputers and Memories Handbook	EB-18451-20
VAX Architecture Handbook	EB-19580-20
VAX Architecture Reference Manual	EY-3459E-DP
BA213 Enclosure Maintenance	EK-189AA-MG
BA215 Enclosure Maintenance	EK-191AA-MG

You can order these documents from:

Digital Equipment Corporation
Accessories and Supplies Group
P.O. Box CS2008
Nashua, NH 03061

Attention: Documentation Products

1
Overview

This chapter provides a brief overview of the KA640 CPU module and MS650 memory modules.

1.1 Introduction

The KA640, shown in Figure 1-1, is a quad-height VAX processor module for the Q22-bus (extended LSI-11 bus) with built-in DSSI and Ethernet controllers. It is designed for use in high speed, real-time applications and for multiuser, multitasking environments. There are two variants, KA640-AA and the KA640-BA. The KA640-AA runs multi-user software; the KA640-BA runs single-user software.

The KA640 is used in two systems, the MicroVAX 3300 and the MicroVAX 3400. The MicroVAX 3300 is in a BA215 enclosure. The MicroVAX 3400 is in a BA213 enclosure. Refer to the *BA215 Enclosure Maintenance* and the *BA213 Enclosure Maintenance* for a detailed description of each enclosure.

The KA640 CPU module and MS650 memory modules combine to form a VAX CPU/memory subsystem that uses the Q22-bus, DSSI bus, and Ethernet to communicate with mass storage and I/O devices. The KA640 and MS650 modules mount in standard Q22-bus backplane slots that implement the Q22-bus in the AB rows and the CD interconnect in the CD rows. A single KA640 can support up to three MS650 modules, if enough Q22/CD slots are available.

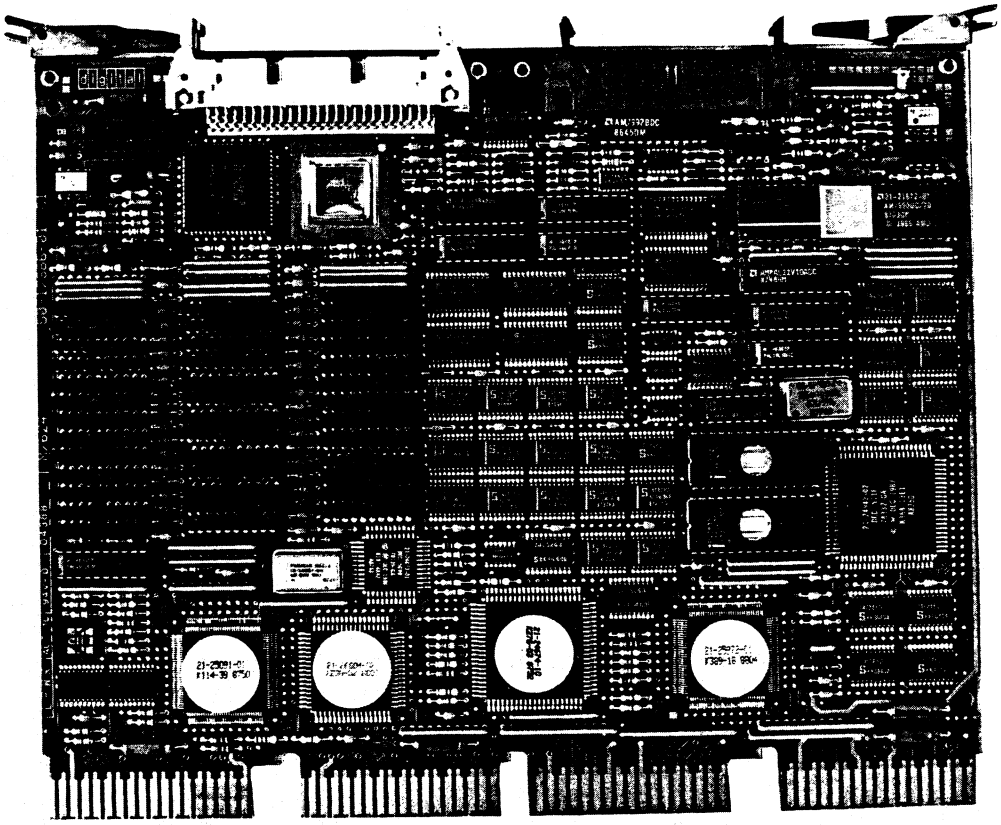


Figure 1–1 KA640 CPU Module

The KA640 communicates with the console device via the H3602-SA CPU cover panel, which also contains configuration switches, an LED display, and DESTA (Ethernet connector).

The major functional blocks of the KA640 CPU module are shown in Figure 1–2, and are described in the following paragraphs.

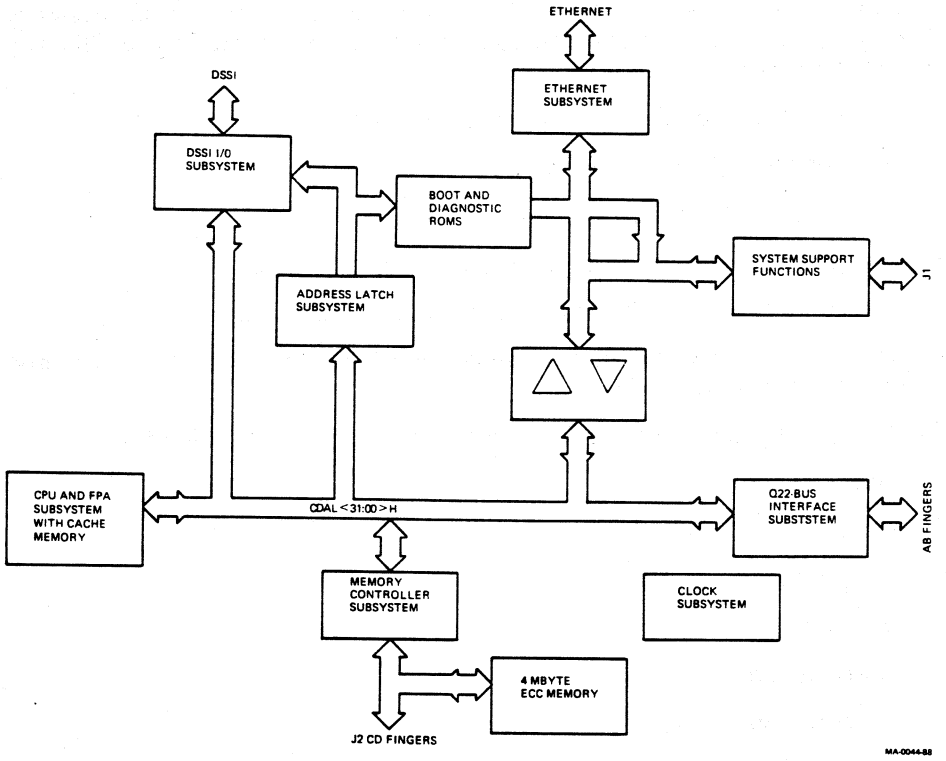


Figure 1–2 KA640 Block Diagram

1.2 Central Processing Unit

The central processing unit (CPU) is implemented by the CVAX chip. The CVAX chip contains approximately 180,000 transistors in an 84-pin CERQUAD surface mount package. It achieves a 100 ns microcycle and a 200 ns bus cycle at an operating frequency of 20 MHz. The CVAX chip supports full VAX memory management and a 4 Gigabyte virtual address space.

The CVAX chip contains all VAX visible general purpose registers (GPRs), several system registers (MSER, CADR, SCBB, etc.), the cache memory (1 Kbyte), and all memory management hardware including a 28-entry translation buffer.

The CVAX chip provides the following functions:

- Fetches all VAX instructions
- Executes 181 VAX instructions
- Assists in the execution of 21 additional instructions
- Passes 70 floating point instructions to the CFPA chip

The remaining 32 VAX instructions (including H-floating and octaword) must be emulated in macrocode.

The CVAX chip provides the following subset of the VAX data types:

- Byte
- Word
- Longword
- Quadword
- Character string
- Variable length bit field

Support for the remaining VAX data types can be provided via macrocode emulation.

1.3 Floating Point Accelerator

The floating point accelerator is implemented by the CFPA chip. The CFPA chip contains approximately 60,000 transistors in a 68-pin CERQUAD surface mount package. It executes 70 floating point instructions.

The CFPA chip receives opcode information from the CVAX chip, and receives operands directly from memory or from the CVAX chip. The floating point result is always returned to the CVAX chip.

1.4 Cache Memory

The KA640 module incorporates a cache memory within the CVAX chip to maximize CPU performance. The cache is a 1 Kbyte, two-way associative, write through cache memory, with a 100 ns cycle time.

1.5 Memory Controller

The main memory controller is implemented by a VLSI chip called the CMCTL. The CMCTL contains approximately 25,000 transistors in a 132-pin CERQUAD surface mount package. It supports up to 64 Mbytes of ECC memory, of which only 52 MBytes can be used on a KA640, with a 400 ns cycle time for longword transfers and a 600 ns cycle time for quadword transfers. This memory resides on the KA640 CPU module, and depending on the system configuration, one to three MS650 memory modules.

1.6 MS650-AA Memory Modules

The MS650-AA memory modules are 8 Mbyte, 450 ns, 39-bit wide arrays (32-bit data and 7-bit ECC) implemented with 256 Kb dynamic RAMs in zig-zag in-line packages (ZIPs). MS650-AA memory modules are single, quad-height, Q22-bus modules, as shown in Figure 1-3.

The MS650 modules communicate with the KA640 via the MS650 memory interconnect, which utilizes the CD rows of backplane slots 2 through 4, and a 50-pin ribbon cable. The KA640 memory subsystem supports a maximum of 3 memory modules.

1.7 DSSI Interface

An on-board digital small storage interconnect (DSSI) bus interface is implemented by the SII chip and four 32K by 8 static RAMs. The DSSI interface allows the KA640 to transmit packets of data to, and receive packets of data from up to seven other DSSI devices. The KA640 system configurations contain one or more RF30 fixed disk devices, connected through the DSSI bus.

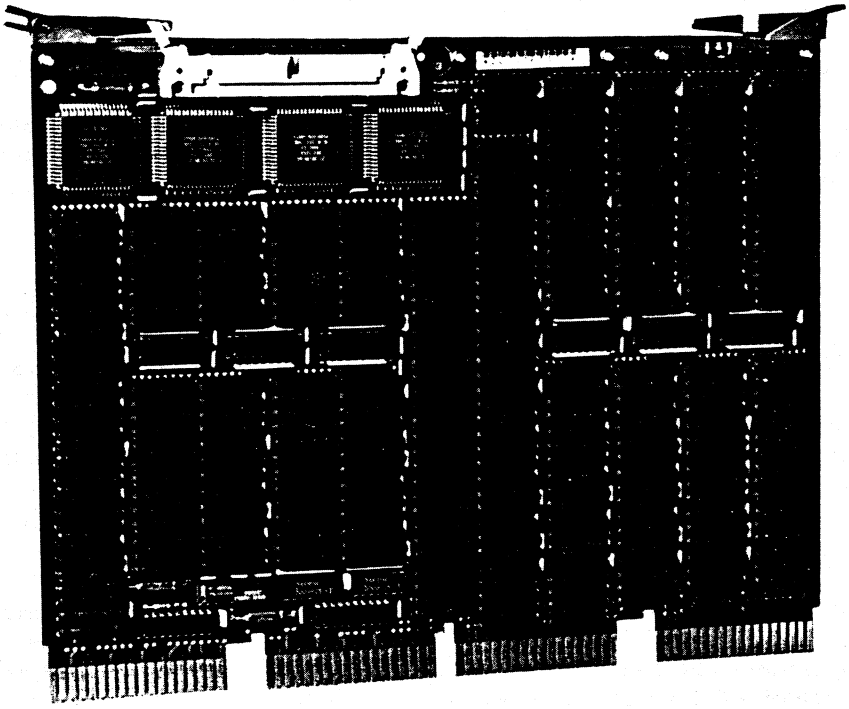


Figure 1–3 MS650-AA Memory Module

1.8 Ethernet Interface

The KA640 CPU module features an on-board network interface that is implemented via the LANCE chip and two 32K by 8 static RAMs. When used in conjunction with the H3602-SA CPU cover panel, this interface allows the KA640 to be connected to either a ThinWire or standard Ethernet network.

The Ethernet interface includes registers for control and status reporting as well as a DMA controller, a 24 word transmit silo and a 24 word receive silo.

1.9 Q22-bus Interface

The Q22-bus interface is implemented by the CQBIC chip. The CQBIC chip contains approximately 40,870 transistors in a 132-pin CERQUAD surface mount package. It supports up to 16-word, block mode transfers between a Q22-bus DMA device and main memory, and up to 2-word, block mode transfers between the CPU and Q22-bus devices. The Q22-bus interface contains the following:

- A 16-entry map cache for the 8,192-entry, main memory-resident "scatter-gather" map, used for translating 22-bit Q22-bus addresses into 26-bit main memory addresses
- Interrupt arbitration logic that recognizes Q22-bus interrupt requests BR7-BR4
- Q22-bus termination (240 Ω)

1.10 MicroVAX System Support Functions

System support functions are implemented by the system support chip (SSC). The SSC contains approximately 83,000 transistors in an 84-pin CERQUAD surface mount package. The SSC provides console and boot code support functions, operating system support functions, timers, and many extra features, including the following:

- Word-wide ROM unpacking
- 1 Kbyte battery backed-up RAM
- Halt arbitration logic
- Console serial line
- Interval timer with 10 ms interrupts
- VAX standard time of year (TOY) clock with support for battery back-up
- IORESET register
- Programmable CDAL bus timeout
- Two programmable timers similar in function to the VAX standard interval timer
- A register for controlling the dignostic LEDs

1.11 Firmware

The firmware consists of 128 Kbytes of 16 bit-wide ROM, located on two 27512 EPROMs. The firmware gains control when the processor halts, and contains programs that provide the following services:

- Board initialization
- Power-up self-testing of the KA640 and MS650 modules
- Emulation of a subset of the VAX standard console (automatic/manual bootstrap, automatic/manual restart, and a simple command language for examining/altering the state of the processor)
- Booting from supported Q22-bus devices
- Multilingual capability
- MOP support

1.12 Clock Functions

All clock functions are implemented by the CVAX clock chip. The CVAX clock chip is a 44-pin CERQUAD surface mount chip that contains approximately 350 transistors, and provides the following functions:

- Generates two MOS clocks for the CPU, the floating point accelerator, and the main memory controller
- Generates three auxiliary clocks for other miscellaneous TTL logic
- Synchronizes reset signal for the CPU, the floating point accelerator, and the main memory controller
- Synchronizes data ready and data error signals for the CPU, floating point accelerator, and the main memory controller

2
Installation and Configuration

2.1 Introduction

This chapter contains information required to install the KA640 in a system. The following topics are discussed:

- Installing the KA640
- Configuring the KA640
- KA640 connectors
- H3602-SA CPU cover panel
- Compatible system enclosures

2.2 Installing the KA640

The KA640 and MS650 modules must be installed in system enclosures having Q22/CD slots. These modules are not compatible with Q/Q backplane slots, and therefore should only be installed in Q22/CD backplane slots.

The KA640 CPU module must be installed in slot 1 of the Q22/CD backplane (Figure 2-1). MS650 memory modules must be installed in slots immediately adjacent to the CPU module. Up to three MS650 modules can be installed, occupying slots 2, 3, and 4 respectively. A 50-pin ribbon cable is used to connect the KA640 CPU module and the MS650 memory module(s), as shown in Figure 2-2.

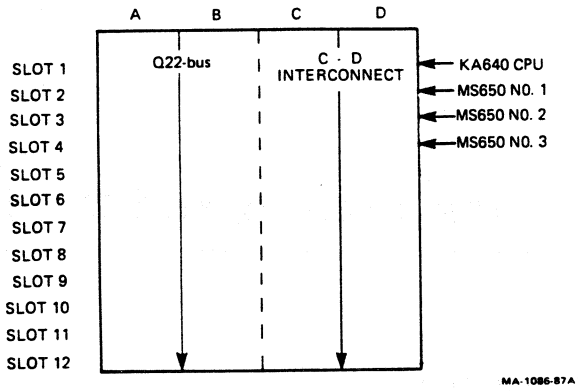


Figure 2–1 CPU and Memory Module Placement

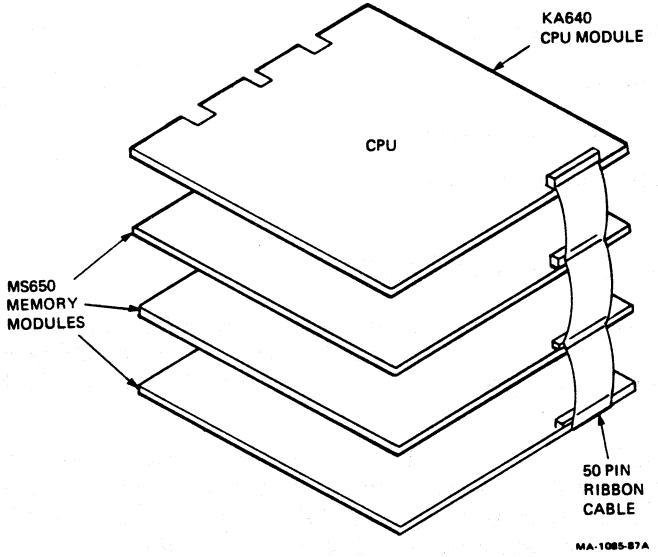


Figure 2–2 Cable Connections

2.3 Configuring the KA640

The following parameters must be configured on the KA640:

- Power-up mode
- Break enable switch
- Console serial line baud rate
- DSSI node ID
- Ethernet port connector selection

These parameters are configured using the H3602-SA CPU cover panel.

2.4 KA640 Connectors

The KA640 uses two connectors (J1 and J2) and four rows of module fingers (A,B,C, and D) to communicate with the console device, main memory, the Q22-bus, the DSSI controller, and the Ethernet controller. The contact finger identification of the KA640 module is described in Appendix C.

The orientation of connectors J1 and J2, and the LED indicators is shown in Figure 2-3.

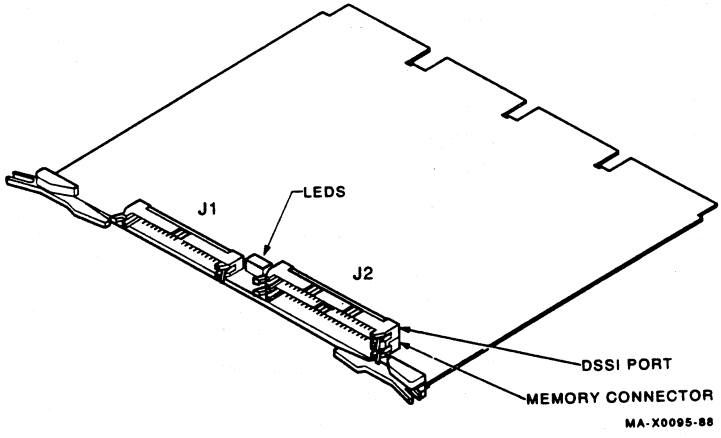


Figure 2-3 KA640 Pin and LED Orientation

2.4.1 System Support Connector (J1)

The system support connector (J1) is a 40-pin connector used to provide the connection to the Ethernet controller and the system console, and for configuration and display purposes. Table 2-1 gives the pinouts for the system support connector.

Table 2-1 System Support Connector (J1) Pinouts

Pin	Mnemonic	Meaning
01	XMIT- H	Transmit - output to the LAN interface
02	XMIT+ H	Transmit + output to the LAN interface
03	GND	Ground
04	GND	Ground
05	GND	Ground
06	RCV- H	Receive - input from the LAN interface
07	RCV+ H	Receive + input from the LAN interface
08	GND	Ground
09	GND	Ground
10	GND	Ground
11	COL- H	Collision
12	COL+ H	Collision
13	GND	Ground
14	GND	Ground
15	GND	Ground
16	GND	Ground
17	+12 V	Fused +12 Vdc
18	GND	Ground
19	DTR H	Data Terminal Ready.
20	GND	Ground
21	TXD L	Transmit Data.
22	SPID0 L	Not used with H3602-SA.
23	SPID1 L	Not used with H3602-SA.
24	RXD L	Receive Data.
25	RXD H	Receive Data.
26	SPID2 L	Not used with H3602-SA.
27	+5 V	Fused +5 Vdc
28	CONBITRATE2 L	Console Bit Rate <02:00>. These three bits
29	CONBITRATE1 L	determine the console baud rate. They are

Table 2–1 (Cont.) System Support Connector (J1) Pinouts

Pin	Mnemonic	Meaning
30	CONBITRATE0 L	configured using the select switch on the inside of the H3602-SA.
31	LEDCODE0 L	LED Code register bits <03:00>.
32	LEDCODE1 L	When asserted each of these four output signals
33	LEDCODE2 L	lights a corresponding LED on the module.
34	LEDCODE3 L	LEDCODE<03:00> are asserted (low) by power-up and by the negation of DCOK when the processor is halted. They are updated by boot and diagnostic programs from the BDR.
35	ENBHALT L	Halt Enable. This input signal controls the response to an external halt condition. If ENBHALT is asserted (low), then the KA640 halts and enters the console program if any of the following occur: • The program executes a halt instruction in kernel mode • The console detects a break character • The Q22-bus halt line is asserted If ENBHALT is negated (high), then the halt line and break character are ignored and the ROM program responds to a halt instruction by restarting or rebooting the system. ENBHALT is read by software from the BDR.
36	BDCODE1 L	Boot and Diagnostic code <01:00>. This 2-bit code
37	BDCODE0 L	indicates power-up mode, and is read by software from the BDR.
38	VBAT H	Battery Backup Voltage for the TOY clock.
39	CPUCODE1 L	CPU Code <01:00>. This 2-bit code
40	CPUCODE0 L	is read by software from the BDR. The configuration for the CPU code is as follows: 00 Normal operation 01 Reserved 10 Reserved 11 Reserved

2.4.2 Memory/DSSI Connector (J2)

J2 is a dual 50-pin connector (100 pins total) that provides two interfaces. The *lower* 50 pins are used as the memory interface between the KA640 and the MS650 memory modules; the *upper* 50 pins are used for the DSSI controller. The cables and connectors are keyed to prevent the memory cable from being installed into the DSSI connector and/or the DSSI cable from being installed into the memory connector.

Table 2-2 lists the pinouts for the memory portion of J2 (lower 50 pins). Table 2-3 lists the pinouts for the DSSI portion of J2 (upper 50 pins).

Table 2-2 Memory/DSSI Connector (J2-Lower) Pinouts

Pin	Mnemonic	Pin	Mnemonic
01	GND	26	D MD10 H
02	D MD9 H	27	GND
03	D MD8 H	28	D MD29 H
04	D MD7 H	29	D MD28 H
05	GND	30	D MD27 H
06	D MD6 H	31	GND
07	D MD5 H	32	D MD26 H
08	D MD4 H	33	D MD25 H
09	D MD3 H	34	D MD24 H
10	GND	35	D MD23 H
11	D MD2 H	36	GND
12	D MD1 H	37	D MD22 H
13	D MD0 H	38	D MD21 H
14	D MD19 H	39	D MD20 H
15	GND	40	D MD38 H
16	D MD18 H	41	GND
17	D MD17 H	42	D MD37 H
18	D MD16 H	43	D MD36 H
19	D MD15 H	44	D MD35 H
20	GND	45	D MD34 H
21	D MD14 H	46	GND
22	D MD13 H	47	D MD33 H
23	D MD12 H	48	D MD32 H
24	GND	49	D MD31 H
25	D MD11 H	50	D MD30 H

Table 2–3 DSSI Connector (J2-Upper) Pinouts

Pin	Mnemonic	Pin	Mnemonic
51	DSSIDATA0 L	76	VTERM H
52	GND	77	VTERM H
53	DSSIDATA1 L	78	VTERM H
54	GND	79	GND
55	DSSIDATA2 L	80	Unused
56	GND	81	GND
57	DSSIDATA3 L	82	Unused
58	GND	83	GND
59	DSSIDATA4 L	84	Unused
60	GND	85	GND
61	DSSIDATA5 L	86	DSSIBSY L
62	GND	87	GND
63	DSSIDATA6 L	88	DSSIACK L
64	GND	89	GND
65	DSSIDATA7 L	90	DSSIRST L
66	GND	91	GND
67	DSSIPARITY L	92	Unused
68	GND	93	GND
69	Unused	94	DSSISEL L
70	GND	95	GND
71	Unused	96	DSSIC/D L
72	GND	97	GND
73	VTERM H	98	DSSIREQ L
74	VTERM H	99	GND
75	VTERM H	100	DSSII/O L

2.5 H3602-SA CPU Cover Panel

The H3602-SA CPU cover panel is a special I/O panel that is used in BA213 and BA215 enclosures. A one-piece ribbon cable on the H3602-SA plugs into the system support connector (J1) on the KA640. The H3602-SA fits over backplane slots 1 and 2, covering both the KA640 CPU module and the first of three possible MS650 memory modules.

The H3602-SA CPU cover panel (Figure 2–4) includes the features and controls specified in Table 2–4.

Table 2–4 H3602-SA Features and Controls

Outside	Inside
Modified modular jack (MMJ) SLU connector	Baud rate rotary switch
Power-up mode switch	Battery backup unit (BBU) for TOY clock
Hexadecimal LED display	40-pin cable connector
Break enable switch	List of baud rate switch settings
Standard/ThinWire Ethernet connectors	
Standard/ThinWire Ethernet selector	
Indicator LEDs	

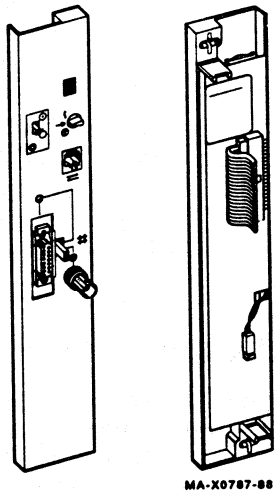


Figure 2–4 H3602-SA CPU Cover Panel

2.6 Compatible System Enclosures

The KA640 is compatible with the following Digital enclosures.

BA213

The BA213 contains a 4 row by 12 slot backplane, with the Q22-bus implemented in the AB rows of slots 1 through 12. The CD interconnect is implemented in the CD rows of slots 1 through 12, allowing up to 3 memory modules to be used. The BA213 has mounting space for up to four 13.2 cm (5.25 inch) mass storage devices. The BA213 is equipped with two modular power supplies. Each power supply delivers 7.0 A (maximum) at +12 Vdc and 33.0 A (maximum) at +5 Vdc. The combined maximum current at +12 Vdc and +5 Vdc must not exceed 230 watts of power for each supply.

BA215

The BA215 contains a 4 row by 6 slot backplane, with the Q22-bus implemented in the AB rows of slots 1 through 6. The CD interconnect is implemented in the CD rows of slots 1 through 6, allowing up to 3 memory modules to be used. The BA215 has mounting space for up to 3 mass storage devices (one full-height and two half-height). The BA215 is equipped with one power supply that delivers 7.0 A (maximum) at +12 Vdc and 33.0 A (maximum) at +5 Vdc. The maximum current at +12 Vdc and +5 Vdc must not consume more than 230 watts of power.

3
Architecture

This chapter describes the KA640 registers, instruction set, and memory. The chapter covers the following KA640 topics:

- Central processor
- Floating-point accelerator
- Cache memory
- Main memory system
- Console serial line
- Time of year clock and timers
- Boot and diagnostic facility
- Q22-bus interface
- Mass storage interface
- Network interface

3.1 Central Processor

The central processor of the KA640 supports the MicroVAX chip subset (plus six additional string instructions) of the VAX instruction set and data types and full VAX memory management. It is implemented by a single VLSI chip called the CVAX.

3.1.1 Processor State

The processor state consists of that portion of the state of a process that is stored in processor registers rather than in memory. The processor state is composed of sixteen general purpose registers (GPRs), the processor status longword (PSL), and the internal processor registers (IPRs).

Non-privileged software can access the GPRs and the processor status word (bits <15:00> of the PSL). The IPRs and bits <31:16> of the PSL can only be accessed by privileged software. The IPRs are explicitly accessible only by the move to processor register (MTPR) and move from processor register (MFPR) instructions that can be executed only while running in kernel mode.

3.1.1.1 General Purpose Registers

The KA640 implements 16 GPRs as specified in the *VAX Architecture Reference Manual*. These registers are used for temporary storage, as accumulators, and as base and index registers for addressing. These registers are denoted R0 - R15. The bits of a register are numbered from the right <0> through <31> (Figure 3-1).

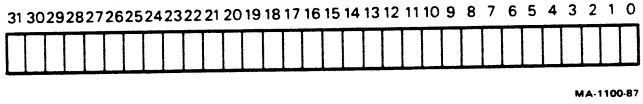


Figure 3-1 General Purpose Register Bit Map

Certain of these registers have been assigned special meaning by the VAX-11 architecture:

- R15 is the program counter (PC). The PC contains the address of the next instruction byte of the program.
- R14 is the stack pointer (SP). The SP contains the address of the top of the processor defined stack.
- R13 is the frame pointer (FP). The VAX-11 procedure call convention builds a data structure on the stack called a *stack frame*. The FP contains the address of the base of this data structure.
- R12 is the argument pointer (AP). The VAX-11 procedure call convention uses a data structure called an *argument list*. The AP contains the address of the base of this data structure.

Consult the *VAX Architecture Reference Manual* for more information on the operation and use of these registers.

3.1.1.2 Processor Status Longword

The KA640 processor status longword (PSL) is implemented per the *VAX Architecture Reference Manual,* which should be consulted for a detailed description of the operation of this register. The PSL is saved on the stack when an exception or interrupt occurs and is saved in the process control block (PCB) on a process context switch. Bits <15:00> may be accessed by non-privileged software, while bits <31:16> may only be accessed by privileged software. Processor initialization sets the PSL to 041F 0000 $_{16}$. Figure 3–2 shows the processor status longword bit map.

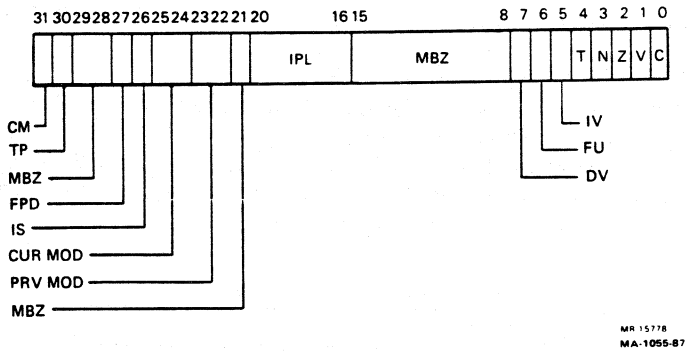


Figure 3–2 PSL Bit Map

Data Bit	Definition
PSL <31>	(CM) Compatibility mode. This bit always reads as zero; loading a 1 into this bit is a NOP.
PSL <30>	(TP) Trace pending
PSL <29:28>	Unused, must be written as zero.
PSL <27>	(FPD) First part done
PSL <26>	(IS) Interrupt stack
PSL <25:24>	(CUR) Current mode
PSL <23:22>	(PRV) Previous mode
PSL <21>	Unused, must be written as zero.
PSL <20:16>	(IPL) Interrupt priority level

Data Bit	Definition
PSL <15:8>	Unused, must be written as zero.
PSL <7>	(DV) Decimal overflow trap enable. This read/write bit has no effect on KA640 hardware; it can be used by macrocode which emulates VAX decimal instructions.
PSL <6>	(FU) Floating underflow fault enable
PSL <5>	(IV) Integer overflow trap enable
PSL <4>	(T) Trace trap enable
PSL <3>	(N) Negative condition code
PSL <2>	(Z) Zero condition code
PSL <1>	(V) Overflow condition code
PSL <0>	(C) Carry condition code

NOTE

VAX compatibility mode instructions can be emulated by macrocode, but the emulation software runs in native mode, so the CM bit is never set.

3.1.1.3 Internal Processor Registers

The KA640 IPRs can be accessed by using the MFPR and MTPR privileged instructions. Each IPR falls into one of the following seven categories:

1. Implemented by KA640 (in the CVAX chip) as specified in the *VAX Architecture Reference Manual*
2. Implemented by KA640 (in the SSC) as specified in the *VAX Architecture Reference Manual*
3. Implemented by KA640 (and all designs that use the CVAX chip) uniquely
4. Implemented by KA640 (and all designs that use the SSC) uniquely
5. Not implemented, timed out by the CDAL bus timer (in the SSC) after 4 μs. Read as zero, NOP on write.
6. Access not allowed; accesses result in a reserved operand fault
7. Accessible, but not fully implemented; accesses yield *unpredictable* results

Refer to Table 3-1 for a listing of each of the KA640 IPRs, along with its mnemonic, its access type (read or write) and its category number.

Table 3-1 KA640 Internal Processor Registers

Decimal	Hex	Register Name	Mnemonic	Type	Category
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
7:5	7:5	Reserved			5
8	8	P0 base register	P0BR	R/W	1
9	9	P0 length register	P0LR	R/W	1
10	A	P1 base register	P1BR	R/W	1
11	B	P1 length register	P1LR	R/W	1
12	C	System base register	SBR	R/W	1
13	D	System length register	SLR	R/W	1
15:14	F:E	Reserved			5
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 I*
19	13	AST level	ASTLVL	R/W	1 I*
20	14	Software interrupt request	SIRR	W	1
21	15	Software interrupt summary	SISR	R/W	1 I*
23:22	17:16	Reserved			5
24	18	Interval clock control/status	ICCS	R/W	3 I*
25	19	Next interval count	NICR	W	5
26	1A	Interval count	ICR	R	5
27	1B	Time of year	TODR	R/W	2
28	1C	Console storage receiver status	CSRS	R/W	7 I*
29	1D	Console storage receiver data	CSRD	R	7 I*
30	1E	Console storage transmit status	CSTS	R/W	7 I*
31	1F	Console storage transmit data	CSTD	W	7 I*
32	20	Console receiver control/status	RXCS	R/W	4 I*
33	21	Console receiver data buffer	RXDB	R	4 I*
34	22	Console transmit control/status	TXCS	R/W	4 I*
35	23	Console transmit data buffer	TXDB	W	4 I*

*An I following the category number indicates that the register is initialized on power-up and by the negation of DCOK when the processor is halted.

Table 3–1 (Cont.) KA640 Internal Processor Registers

Decimal	Hex	Register Name	Mnemonic	Type	Category
36	24	Translation buffer disable	TBDR	R/W	5
37	25	Cache disable	CADR	R/W	3 I*
38	26	Machine check error summary	MCESR	R/W	5
39	27	Memory system error	MSER	R/W	3 I*
41:40	29:28	Reserved			5
42	2A	Console saved PC	SAVPC	R	3
43	2B	Console saved PSL	SAVPSL	R	3
47:44	2F:2C	Reserved			5
48	30	SBI fault/status	SBIFS	R/W	5
49	31	SBI silo	SBIS	R	5
50	32	SBI silo comparator	SBISC	R/W	5
51	33	SBI maintenance	SBIMT	R/W	5
52	34	SBI error register	SBIER	R/W	5
53	35	SBI timeout address register	SBITA	R	5
54	36	SBI quadword clear	SBIQC	W	5
55	37	I/O bus reset	IORESET	W	4
56	38	Memory management enable	MAPEN	R/W	1
57	39	TB invalidate all	TBIA	W	1
58	3A	TB invalidate single	TBIS	W	1
59	3B	TB data	TBDATA	R/W	5
60	3C	Microprogram break	MBRK	R/W	5
61	3D	Performance monitor enable	PMR	R/W	5
62	3E	System identification	SID	R	1
63	3F	Translation buffer check	TBCHK	W	1
64:127	40:7F	Reserved			6

*An I following the category number indicates that the register is initialized on power-up and by the negation of DCOK when the processor is halted.

KA640 VAX Standard IPRs

The KA640 implements VAX standard IPRs as specified in the *VAX Architecture Reference Manual*. The *VAX Architecture Reference Manual* should be consulted for details on the operation and use of these registers.

The VAX standard IPRs listed in Table 3–2 are also referenced in other sections of this manual.

Table 3–2 VAX Standard IPRs

Number	Register Name	Mnemonic	Section
12	System base register	SBR	Section 3.1.5.3
13	System length register	SLR	Section 3.1.5.3
16	Process control block base	PCBB	Section 3.1.5
17	System control block base	SCBB	Section 3.1.5.4
18	Interrupt priority level	IPL	Section 3.1.5.1
20	Software interrupt request	SIRR	Section 3.1.5.1
21	Software interrupt summary	SISR	Section 3.1.5.1
27	Time of year clock	TODR	Section 3.6.1
56	Memory management enable	MAPEN	Section 3.1.4.2
57	Translation buffer invalidate all	TBIA	Section 3.1.4.2
58	Translation buffer invalidate single	TBIS	Section 3.1.4.2
62	System identification	SID	Section 3.1.6
63	Translation buffer check	TBCHK	Section 3.1.4.2

KA640 Unique IPRs

IPRs that are implemented uniquely on the KA640 (i.e., those that are not contained in, or do not fully conform to the standards in the *VAX Architecture Reference Manual*) are described in detail in this manual. Refer to the sections listed in Table 3–3 for a description of these registers.

Table 3–3 KA640 Unique IPRs

Number	Register Name	Mnemonic	Section
24	Interval clock control/status	ICCS	Section 3.6.2
32	Console receiver control/status	RXCS	Section 3.5.1.1
33	Console receiver data buffer	RXDB	Section 3.5.1.2
34	Console transmit control/status	TXCS	Section 3.5.1.3
35	Console transmit data buffer	TXDB	Section 3.5.1.4
37	Cache disable	CADR	Section 3.3.2.5
39	Memory system error	MSER	Section 3.3.2.6
42	Console saved PC	SAVPC	Section 3.1.5
43	Console saved PSL	SAVPSL	Section 3.1.5
55	I/O bus reset	IORESET	Section 3.7.5.3

3.1.2 Data Types

The KA640 CPU supports the following subset of the VAX data types:

- Byte
- Word
- Longword
- Quadword
- Character string
- Variable length bit field

Support for the remaining VAX data types can be provided via macrocode emulation.

3.1.3 Instruction Set

The KA640 CPU implements the following subset of the VAX instruction set types in microcode:

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

---

* These instructions were in the microcode assisted category on the KA630-AA (MicroVAX II) and therefore had to be emulated.

The KA640 CVAX chip provides special microcode assistance to aid the macrocode emulation of the following instruction groups:

- Character string (except MOVC3, MOVC5, CMPC3* CMPC5*, LOCC*, SCANC*, SKPC*, and SPANC*)
- Decimal string
- CRC
- EDITPC

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

- Octaword
- Compatibility mode instructions

3.1.4 Memory Management

The KA640 implements full VAX memory management as defined in the *VAX Architecture Reference Manual*. System space addresses are virtually mapped through single-level page tables, and process space addresses are virtually mapped through two-level page tables. See the *VAX Architecture Reference Manual* for descriptions of the virtual to physical address translation process, and the format for VAX page table entries (PTEs).

3.1.4.1 Translation Buffer

To reduce the overhead associated with translating virtual addresses to physical addresses, the KA640 employs a 28-entry, fully associative, translation buffer for caching VAX PTEs in modified form. Each entry can store a modified PTE for translating virtual addresses in either the VAX process space, or VAX system space. The translation buffer is flushed whenever memory management is enabled or disabled (i.e., by writes to IPR 56), any page table base or length registers are modified (i.e., by writes to IPRs 8 - 13) and by writing to IPR 57 (TBIA) or IPR 58 (TBIS).

Each entry is divided into two parts: a 23-bit tag register and a 31-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 PTE register stores the 21-bit PFN field, the PTE.V bit, the PTE.M bit and an 8-bit partially decoded representation of the 4-bit VAX PTE PROT field, from the corresponding VAX PTE, as well as a translation buffer valid (TB.V) bit.

---

* These instructions were in the microcode assisted category on the KA630-AA (MicroVAX II) and therefore had to be emulated.

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

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

3.1.4.2 Memory Management Control Registers

There are four IPRs that control the memory management unit (MMU): IPR 56 (MAPEN), IPR 57 (TBIA), IPR 58 (TBIS), and IPR 63 (TBCHK).

Memory management can be enabled/disabled via IPR 56 (MAPEN). Writing a 0 to this register with a MTPR instruction disables memory management, and writing a 1 to this register with a MTPR instruction enables memory management. Writes to this register flush the translation buffer. To determine whether or not memory management is enabled, IPR 56 is read using the MFPR instruction. Translation buffer entries that map a particular virtual address can be invalidated by writing the virtual address to IPR 58 (TBIS) using the MTPR instruction.

NOTE

Whenever software changes a valid PTE for the system or current process region, or a system PTE that maps any part of the current process page table, all process pages mapped by the PTE must be invalidated in the translation buffer.

The entire translation buffer can be invalidated by writing a 0 to IPR 57 (TBIA) using the MTPR instruction.

The translation buffer can be checked to see if it contains a valid translation for a particular virtual page by writing a virtual address within that page to IPR 63 (TBCHK) using the MTPR instruction. If the translation buffer contains a valid translation for the page, the condition code V bit (bit <1> of the PSL) is set.

NOTE

The TBIS, TBIA, and TBCHK IPRs are write only. The operation of a MFPR instruction from any of these registers is *undefined*.

3.1.5 Exceptions and Interrupts

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

3.1.5.1 Interrupts

Interrupts can be divided into two classes: non-maskable, and maskable.

Non-maskable interrupts cause a halt via the hardware halt procedure which saves the PC, PSL, MAPEN<0> and a halt code in IPRs, raises the processor IPL to 1F and then passes control to the resident firmware. The firmware dispatches the interrupt to the appropriate service routine based on the halt code and hardware event indicators. Non-maskable interrupts cannot be blocked by raising the processor IPL, but can be blocked by running out of the halt protected address space (except those non-maskable interrupts that generate a halt code of 3). Non-maskable interrupts with a halt code of 3 cannot be blocked since this halt code is generated after a hardware reset.

Maskable interrupts cause the PC and PSL to be saved, the processor IPL to be raised to the priority level of the interrupt (except for Q22-bus interrupts where the processor IPL is set to 17 independent of the level at which the interrupt was received) and the interrupt to be dispatched to the appropriate service routine through the system control block (SCB).

The various interrupt conditions for the KA640 are listed in Table 3-4 along with their associated priority levels and SCB offsets.

Table 3-4 Interrupts

Priority Level	Interrupt Condition	SCB Offset
Non-maskable	BDCOK and BPOK negated then asserted on Q22-bus (Power up)	*
	BDCOK negated then asserted while BPOK asserted on Q22-bus (SCR<7> clear)	
	BDCOK negated then asserted while BPOK asserted on Q22-bus (SCR<7> set)	†
	BHALT asserted on Q22-bus	†
	BREAK generated by the console device	†

*These conditions generate a hardware halt procedure with a halt code of 3 (hardware reset).
†These conditions generate a hardware halt procedure with a halt code of 2 (external halt).

Table 3–4 (Cont.) Interrupts

Priority Level	Interrupt Condition	SCB Offset
1F	Unused	
1E	BPOK negated on Q22-bus	0C
1D	CDAL bus parity error	60
	Q22-bus NXM on a write	60
	CDAL bus timeout during DMA	60
	Main memory NXM errors	60
	Uncorrectable main memory errors	60
1C - 1B	Unused	
1A	Correctable main memory errors	54
19 - 18	Unused	
17	BR7 L asserted	Q22-bus vector plus 200_{16}
16	Interval timer interrupt	C0
	BR6 L asserted	Q22-bus vector plus 200_{16}
15	BR5 L asserted	Q22-bus vector plus 200_{16}
14	Console terminal	$F8,F6_{16}$
	Programmable timers	78,7C
	Mass storage interface	C4
	Network interface	D4
	BR4 L asserted	Q22-bus vector plus 200_{16}
13 - 10	Unused	
0F - 01	Software interrupt requests	84-BC

NOTE

Because the Q22-bus does not allow differentiation between the four bus grant levels (i.e., a level 7 device could respond to a level 4 bus grant), the KA640 CPU raises the IPL to 17 after responding to interrupts generated by the assertion of either BR7 L, BR6 L, BR5 L, or BR4 L. The KA640 maintains the IPL at the priority of the interrupt for all other interrupts.

The interrupt system is controlled by three IPRs: IPR 18, the interrupt priority level register (IPL), IPR 20, the software interrupt request register (SIRR), and IPR 21, the software interrupt summary register (SISR).

The IPL is used for loading the processor priority field in the PSL (bits <20:16>). The SIRR is used for creating software interrupt requests. The SISR records pending software interrupt requests at levels 1 through 15. The format of these registers is shown in Figure 3-3. Refer to the *VAX Architecture Reference Manual* for more information on these registers.

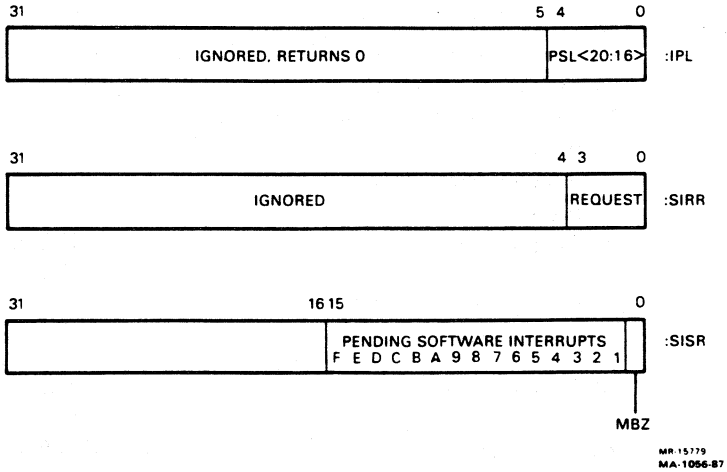


Figure 3-3 Interrupt Registers

3.1.5.2 Exceptions

Exceptions can be divided into three types:

- Trap
- Fault
- Abort

A *trap* is an exception that occurs at the end of the instruction that caused the exception. After an instruction traps, the PC saved on the stack is the address of the next instruction that would have normally been executed and the instruction can be restarted.

A *fault* is an exception that occurs during an instruction, and that leaves the registers and memory in a consistent state such that the elimination of the fault condition and restarting the instruction will give correct results. After an instruction faults, the PC saved on the stack points to the instruction that faulted.

An *abort* is an exception that occurs during an instruction, leaving the value of the registers and memory *unpredictable*, such that the instruction cannot necessarily be correctly restarted, completed, simulated or undone. After an instruction aborts, the PC saved on the stack points to the instruction that was aborted (which may or may not be the instruction that caused the abort) and the instruction may or may not be restarted depending on the class of the exception and the contents of the parameters that were saved.

Exceptions are grouped into six classes:

- Arithmetic exceptions
- Memory management exceptions
- Operand reference exceptions
- Instruction execution exceptions
- Tracing exception
- System failure exceptions

A list of exceptions grouped by class is given in Table 3-5. Exceptions save the PC and PSL and in some cases, one or more parameters, on the stack. Most exceptions do not change the IPL of the processor (except the exceptions in serious system failures class, which set the processor IPL to 1F) and cause the exception to be dispatched to the appropriate service routine through the SCB (except for the interrupt stack not valid exception, and exceptions that occur while an interrupt or another exception are being serviced, which cause the exception to be dispatched to the appropriate service routine by the resident firmware).

The exceptions listed in Table 3-5 (except machine check) are described in greater detail in the *VAX Architecture Reference Manual*. The machine check exception is described in greater detail in Section 3.1.5.3. Exceptions that can occur while servicing an interrupt or another exception are listed in Table 3-8 in Section 3.1.5.6.

Table 3-5 Exceptions

Arithmetic Exceptions	Type	SCB Offset
Integer overflow	Trap	34
Integer divide-by-zero	Trap	34
Subscript range	Trap	34
Floating overflow	Fault	34
Floating divide-by-zero	Fault	34
Floating underflow	Fault	34
Memory Management Exceptions	**Type**	**SCB Offset**
Access control violation	Fault	20
Translation not valid	Fault	24
Operand Reference Exceptions	**Type**	**SCB Offset**
Reserved addressing mode	Fault	1C
Reserved operand fault	Abort	18
Instruction Execution Exceptions	**Type**	**SCB Offset**
Reserved/privileged instruction	Fault	10
Emulated instruction	Fault	C8, CC
Change mode	Trap	40-4C
Breakpoint	Fault	2C
Tracing Exception	**Type**	**SCB Offset**
Trace	Fault	28
System Failure Exceptions	**Type**	**Offset**
Interrupt stack not valid	Abort	*
Kernel stack not valid	Abort	08

*Dispatched by resident firmware rather than through the SCB

Table 3-5 (Cont.) Exceptions

System Failure Exceptions	Type	Offset
Machine check	Abort	04
CDAL bus parity errors		
Cache parity errors		
Q22-bus NXM errors		
Q22-bus device parity errors		
Q22-bus NO GRANT errors		
CDAL bus timeout errors		
Main memory NXM errors		
Main memory uncorrectable errors		

3.1.5.3 Information Saved On A Machine Check Exception

In response to a machine check exception the PSL, PC, four parameters, and a byte count are pushed onto the stack, as shown in Figure 3-4.

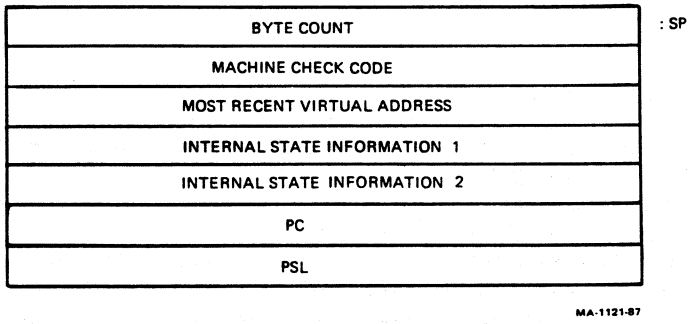


Figure 3-4 Information Saved On A Machine Check Exception

The meaning of this information and how it effects the recovery procedure is described in the following paragraphs.

Byte Count

<31:0> = 0000 0010 $_{16}$, 16 $_{10}$. This value indicates the number of bytes of information that follow on the stack (not including the PC and PSL).

Machine Check Code Parameter

Machine Check Code <31:0>—A code value that indicates the type of machine check that occurred. A list of the possible machine check codes (in hex) and their associated causes follows.

Floating Point Errors—These codes indicate that the floating point accelerator (FPA) chip detected an error while communicating with the CVAX CPU chip during the execution of a floating point instruction. The most likely cause(s) of these types of machine checks are: a problem internal to the CVAX CPU chip, a problem internal to the FPA, or a problem with the interconnect between the two chips. Machine checks due to floating point errors **may be recoverable**, depending on the state of the VAX CAN'T RESTART flag (captured in Internal State Information 2 <15>) and the FIRST PART DONE flag (captured in PSL <27>). If the FIRST PART DONE flag is set, the error is recoverable. If the FIRST PART DONE flag is cleared, then the VAX CAN'T RESTART flag must also be cleared for the error to be recoverable. Otherwise, the error is unrecoverable and depending on the current mode, either the current process or the operating system should be terminated. The information pushed onto the stack by this type of machine check is from the instruction that caused the machine check.

Hex Code	Error Description
1	A protocol error was detected by the FPA chip while attempting to execute a floating point instruction.
2	A reserved instruction was detected by the FPA while attempting to execute a floating point instruction.
3	An illegal status code was returned by the FPA while attempting to execute a floating point instruction. CPSTA<1:0>=10
4	An illegal status code was returned by the FPA while attempting to execute a floating point instruction. CPSTA<1:0>=01

Memory Management Errors—These codes indicate that the microcode in the CVAX CPU chip detected an impossible situation while performing functions associated with memory management. The most likely cause of this type of a machine check is a problem internal to the CVAX chip. Machine checks due to memory management errors are **non-recoverable**. Depending on the current mode, either the current process or the operating system should be terminated. The state of the P0BR, P0LR, P1BR, P1LR, SBR, and SLR should be logged.

Hex Code	Error Description
5	The calculated virtual address for a process PTE was in the P0 space instead of the system space when the CPU attempted to access a process PTE after a translation buffer *miss*.
6	The calculated virtual address space for a process PTE was in the P1 space instead of the system space when the CPU attempted to access a process PTE after a translation buffer *miss*.
7	The calculated virtual address for a process PTE was in the P0 space instead of the system space when the CPU attempted to access a process PTE to change the PTE <M> bit before writing to a previously unmodified page.
8	The calculated virtual address for a process PTE was in the P1 space instead of the system space when the CPU attempted to access a process PTE to change the PTE <M> bit before writing to a previously unmodified page.

Interrupt Errors—This code indicates that the interrupt controller in the CVAX CPU requested a hardware interrupt at an unused hardware IPL. The most likely cause of this type of a machine check is a problem internal to the CVAX chip. Machine checks due to unused IPL errors are **non-recoverable**. A non-vectored interrupt generated by a serious error condition (memory error, power fail, or processor halt) has probably been lost. The operating system should be terminated.

Hex Code	Error Description
9	A hardware interrupt was requested at an unused Interrupt Priority Level (IPL).

Microcode Errors—This code indicates that an impossible situation was detected by the microcode during instruction execution. Note that most erroneous branches in the CVAX CPU microcode cause random microinstructions to be executed. The most likely cause of this type of machine check is a problem internal to the CVAX chip. Machine checks due to microcode errors are **non-recoverable**. Depending on the current mode, either the current process or the operating system should be terminated.

Hex Code	Error Description
A	An impossible state was detected during a MOVC3 or MOVC5 instruction (not move forward, move backward, or fill).

Read Errors—These codes indicate that an error was detected while the CVAX CPU was attempting to read from either the cache, main memory, or the Q22-bus. The most likely cause of this type of machine check must be determined from the state of the MSER, DSER, MEMCSR16, QBEAR, DEAR, and CBTCR. Machine checks due to read errors **may be recoverable**, depending on the state of the VAX CAN'T RESTART flag (captured in Internal State Information 2 <15>) and the FIRST PART DONE flag (captured in PSL <27>). If the FIRST PART DONE flag is set, the error is recoverable. If the FIRST PART DONE flag is cleared, then the VAX CAN'T RESTART flag must also be cleared for the error to be recoverable. Otherwise, the error is unrecoverable and depending on the current mode, either the current process or the operating system should be terminated. The information pushed onto the stack by this type of machine check is from the instruction that caused the machine check.

Hex Code	Error Description
80	An error occurred while reading an operand, a process PTE during address translation, or on any read generated as part of an interlocked instruction.
81	An error occurred while reading a system page table entry (SPTE), during address translation, a process control block (PCB) entry during a context switch, or a system control block (SCB) entry while processing an interrupt.

Write Errors—These codes indicate that an error was detected while the CVAX CPU was attempting to write to either the cache, main memory, or the Q22-bus. The most likely cause of this type of machine check must be determined from the state of the MSER, DSER, MEMCSR16, QBEAR, DEAR, and CBTCR. Machine checks due to write errors are **non-recoverable** because the CPU is capable of performing many read operations out of the cache before a write operation completes. For this reason, the information that is pushed onto the stack by this type of machine check cannot be guaranteed to be from the instruction that caused the machine check.

Hex Code	Error Description
82	An error occurred while writing an operand, or a process page table entry (PPTE) to change the PTE <M> bit before writing a previously unmodified page.
83	An error occurred while writing a system page table entry (SPTE) to change the PTE <M> bit before writing a previously unmodified page, or a PCB entry during a context switch or during the execution of instructions that modify any stack pointers stored in the PCB.

Most Recent Virtual Address Parameter

Most Recent Virtual Address <31:0>—This field captures the contents of the virtual address pointer register at the time of the machine check. If a machine check other than a machine check 81 occurred on a read operation, this field represents the virtual address of the location that was being read when the error occurred, plus four. If machine check 81 occurred, this field represents the physical address of the location that was being read when the error occurred, plus four.

If a machine check other than a machine check 83 occurred on a write operation, this field represents the *virtual address* of a location that was being referenced either when the error occurred, or sometime after the error occurred, plus four. If a machine check 83 occurred, this field represents the *physical address* of the location that was being referenced either when the error occurred, or sometime after the error occurred, plus four. In other words, if the machine check occurred on a write operation, the contents of this field cannot be used for error recovery.

Internal State Information 1 Parameter

Internal State Information 1 is divided into four fields. The contents of these fields is described below.

<31:24>—This field captures the opcode of the instruction that was being read or executed at the time of the machine check.

<23:16>—This field captures the internal state of the CVAX CPU chip at the time of the machine check. The four most significant bits are equal to <1110> and the four least significant bits contain highest priority software interrupt <3:0>.

<15:8>—This field captures the state of CADR <7:0> at the time of the machine check. See Section 3.3.2.5 for an interpretation of the contents of this register.

<7:0>—This field captures the state of the MSER <7:0> at the time of the machine check. See Section 3.3.2.6 for an interpretation of the contents of this register.

Internal State Information 2 Parameter

Internal State Information 2 is divided into five fields. The contents of these fields is described below.

<31:24>—This field captures the internal state of the CVAX CPU chip at the time of the machine check. This field contains SC register <7:0>.

<23:16>—This field captures the internal state of the CVAX CPU chip at the time of the machine check. The two most significant bits are equal to 11 (binary) and the six least significant bits contain state flags <5:0>.

<15>—This field captures the state of the VAX CAN'T RESTART flag at the time of the machine check.

<14:8>—This field captures the internal state of the CVAX CPU chip at the time of the machine check. The three most significant bits are equal to <111> (binary) and the four least significant bits contain ALU condition codes.

<7:0>—This field captures the offset between the virtual address of the start of the instruction being executed at the time of the machine check (saved PC) and the virtual address of the location being accessed (PC) at the time of the machine check.

PC

PC <31:0>—This field captures the virtual address of the start of the instruction being executed at the time of the machine check.

PSL

PSL <31:0>—This field captures the contents of the PSL at the time of the machine check.

3.1.5.4 System Control Block

The system control block (SCB) consists of two pages in main memory that contain the vectors by which interrupts and exceptions are dispatched to the appropriate service routines. The SCB is pointed to by IPR 17, the system control block base register (SCBB), represented in Figure 3-5. The SCB format is presented in Table 3-6.

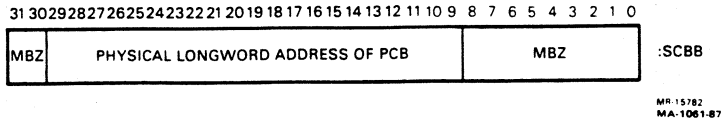


Figure 3-5 System Control Block Base Register

Table 3-6 System Control Block Format

SCB Offset	Interrupt/Exception	Type	Parameter	Notes
00	Unused			IRQ passive release on other VAXes
04	Machine check	Abort	4	Parameters depend on error type
08	Kernel stack not valid	Abort	0	Must be serviced on interrupt stack
0C	Power fail	Interrupt	0	IPL is raised to 1E
10	Reserved/privileged instruction	Fault	0	
14	Customer reserved instruction	Fault	0	XFC instruction
18	Reserved operand	Fault/Abort	0	Not always recoverable
1C	Reserved addressing mode	Fault	0	
20	Access control violation	Fault	2	Parameters are virtual address, status code
24	Translation not valid	Fault	2	Parameters are virtual address, status code
28	Trace pending (TP)	Fault	0	
2C	Breakpoint instruction	Fault	0	

Table 3-6 (Cont.) System Control Block Format

SCB Offset	Interrupt/Exception	Type	Parameter	Notes
30	Unused			Compatibility mode in other VAXes
34	Arithmetic	Trap/Fault	1	Parameter is type code
38:3C	Unused			
40	CHMK	Trap	1	Parameter is sign-extended operand word
44	CHME	Trap	1	Parameter is sign-extended operand word
48	CHMS	Trap	1	Parameter is sign-extended operand word
4C	CHMU	Trap	1	Parameter is sign-extended operand word
50	Unused			
54	Corrected read data	Interrupt	0	IPL is 1A (CRD L)
58:5C	Unused			
60	Memory error	Interrupt	0	IPL is 1D (MEMERR L)
64:6C	Unused			
78	Programmable timer 0	Interrupt	0	IPL is 14
7C	Programmable timer 1	Interrupt	0	IPL is 14
80	Unused			
84	Software level 1	Interrupt	0	
88	Software level 2	Interrupt	0	Ordinarily used for AST delivery
8C	Software level 3	Interrupt	0	Ordinarily used for process scheduling
90:BC	Software levels 4-15	Interrupt	0	
C0	Interval timer	Interrupt	0	IPL is 16 (INTTIM L)
C4	Mass storage interface	Interrupt	0	IPL is 14
C8	Emulation start	Fault	10	Same mode exception, FPD=0; parameters are opcode, PC, specifiers

Table 3-6 (Cont.) System Control Block Format

SCB Offset	Interrupt/Exception	Type	Parameter	Notes
CC	Emulation continue	Fault	0	Same mode exception, FPD=1: no parameters
D0	Unused			
D4	Network interface	Interrupt	0	IPL is 14
D8:DC	Unused			
E0:EC	Reserved for customer or CSS use			
F0:F4	Unused			Console storage registers on 11/750 and 11/730
F8	Console receiver	Interrupt	0	IPL is 14
FC	Console transmitter	Interrupt	0	IPL is 14
100:1FC	Adapter vectors	Interrupt	0	Not implemented by the KA640
200:3FC	Device vectors	Interrupt	0	Correspond to Q22-bus Vectors 000:1FC; KA640 appends the assertion of bit <9,0>
400:FFC	Unused	Interrupt	0	

3.1.5.5 Hardware Detected Errors

The KA640 is capable of detecting eleven types of error conditions during program execution.

1. CDAL bus parity errors indicated by MSER <6> (on a read) or MEMCSR16 <7> (on a write) being set.
2. Cache tag parity errors indicated by MSER <0> being set.
3. Cache data parity errors indicated by MSER <1> being set.
4. Q22-bus NXM errors indicated by DSER <7> being set.
5. Q22-bus NO SACK errors (no indicator).
6. Q22-bus NO GRANT errors indicated by DSER <2> being set.
7. Q22-bus device parity errors indicated by DSER <5> being set.
8. CDAL bus timeout errors indicated by DSER <4> (only on DMA) being set.

9. Main memory NXM errors indicated by DSER <0> (only on DMA) being set.
10. Main memory correctable errors indicated by MEMCSR16 <29> being set.
11. Main memory uncorrectable errors indicated by MEMCSR16 <31> and DSER <4> (only on DMA) being set.

These errors will cause either a machine check exception, a memory error interrupt, or a corrected read data interrupt, depending on the severity of the error and the reference type that caused the error.

3.1.5.6 Hardware Halt Procedure

The hardware halt procedure is the mechanism by which the hardware assists the firmware in emulating a processor halt. The hardware halt procedure saves the current value of the PC in IPR 42 (SAVPC), and the current value of the PSL, MAPEN<0>, and a halt code in IPR 43 (SAVPSL). The current stack pointer is saved in the appropriate internal register. The PSL is set to 041F 0000 $_{16}$ (IPL=1F, kernel mode, using the interrupt stack) and the current stack pointer is loaded from the interrupt stack pointer. Control is then passed to the resident firmware at physical address 2004 0000 $_{16}$ with the state of the CPU as follows:

Register	New Contents
SAVPC	Saved PC
SAVPSL<31:16, 7:0>	Saved PSL<31:16,7:0>
SAVPSL <15>	Saved MAPEN<0>
SAVPSL <14>	Valid PSL flag (unknown for halt code of 3)
SAVPSL <13:8>	Saved restart code
SP	Current interrupt stack
PSL	041F 0000 $_{16}$
PC	2004 0000 $_{16}$
MAPEN	0
ICCS	0 (for a halt code of 3)
MSER	0 (for a halt code of 3)
CADR	0 (for a halt code of 3, cache is also flushed)
SISR	0 (for a halt code of 3)
ASTLVL	0 (for a halt code of 3)
All else	Undefined

The firmware uses the halt code in combination with any hardware event indicators to dispatch the execution or interrupt that caused the halt to the appropriate firmware routine (either console emulation, power-up, reboot,

or restart). Table 3-7 and Table 3-8 list the interrupts and exceptions that can cause halts along with their corresponding halt codes and event indicators.

Table 3-7 Unmaskable Interrupts That Can Cause a Halt

Halt Code	Interrupt Condition	Event Indicators
2	**External Halt (CVAX HALTIN pin asserted)**	
	BHALT asserted on the Q22-bus.	DSER<15>
	BDCOK negated and asserted on the Q22-bus while BPOK stays asserted (Q22-bus REBOOT/RESTART) and SCR <7> is set.	DSER <14>
	BREAK generated by the console	RXDB <11>
3	**Hardware Reset (CVAX RESET pin negated)**	
	BDCOK and BPOK negated then asserted on the Q22-bus (Power-up)	-
	BDCOK negated and asserted on the Q22-bus while BPOK stays asserted (Q22-bus REBOOT/RESTART) and SCR <7> is clear.	-

Table 3-8 Exceptions That Can Cause A Halt

Halt Code	Exception Condition
6	HALT instruction executed in kernel mode
Exceptions While Servicing An Interrupt Or Exception	
4	Interrupt stack not valid during exception
5	Machine check during normal exception
7	SCB vector bits <1:0> = 11
8	SCB vector bits <1:0> = 10
A	CHMx executed while on interrupt stack
B	CHMx executed to the interrupt stack
10	ACV or TNV during machine check exception
11	ACV or TNV during kernel stack not valid exception
12	Machine check during machine check exception
13	Machine check during kernel stack not valid exception
19	PSL <26:24> = 101 during interrupt or exception

Table 3–8 (Cont.) Exceptions That Can Cause A Halt

Halt Code	Exception Condition
1A	PSL <26:24> = 110 during interrupt or exception
1B	PSL <26:24> = 111 during interrupt or exception
1D	PSL <26:24> = 101 during REI
1E	PSL <26:24> = 110 during REI
1F	PSL <26:24> = 111 during REI

3.1.6 System Identification

The system identification register (SID), IPR 62, is a read-only register implemented in the CVAX chip, as specified in the *VAX Architecture Reference Manual*. This 32-bit, read-only register is used to identify the processor type and its microcode revision level (Figure 3–6).



Figure 3–6 System Identification Register

Data Bit	Definition
SID <31:24>	(TYPE) Processor type. This field always reads as 10_{10}, indicating that the processor is implemented using the CVAX chip.
SID <23:8>	Reserved for future use.
SID <7:0>	(MICROCODE REV.) Microcode revision. This field reflects the microcode revision level of the CVAX chip.

In order to distinguish between different CPU implementations that use the same CPU chip, the KA640, as must all VAX processors that use the CVAX chip, implements a MicroVAX system type register (SYS_TYPE) at physical address $2004\ 0004_{16}$. This 32-bit read-only register is implemented in the KA640 ROM. The format of this register is shown in Figure 3–7.

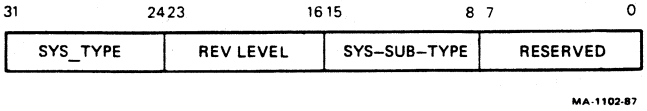


Figure 3–7 System Type Register

Data Bit	Definition
SYS_TYPE <31:24>	(SYS_TYPE) System type code. This field reads as 01_{16} for all single-processor Q22-bus based systems.
SYS_TYPE <23:16>	(REV LEVEL) Revision level. This field reflects the revision level of the KA640 firmware.
SYS_TYPE <15:8>	(SYS_SUB_TYPE) System sub-type code. This field reads as 10_{16} for the KA640.
SYS_TYPE <7:0>	Reserved for Digital use.

3.1.7 CPU References

All references by the CPU can be classified into one of three groups:

- Request instruction-stream read references
- Demand data-stream read references
- Write references

3.1.7.7 Instruction-Stream Read References

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

3.1.7.8 Data-Stream Read References

Whenever data is immediately needed by the CPU to continue processing, a demand data-stream (D-stream) read reference is generated. More specifically, demand D-stream references are generated on operand, PTE, SCB, and PCB references.

When interlocked instructions, such as branch on bit set and set interlock (BBSSI) are executed, a demand D-stream read-lock reference is generated. Since the CPU does not impose any restrictions on data alignment (other than the aligned operands of the add aligned word interlocked (ADAWI) and interlocked queue instructions) and since memory can only be accessed one aligned longword at a time, all data read references are translated into an appropriate combination of masked and unmasked, aligned longword read references.

If the required data is a byte, a word within a longword, or an aligned longword, then a single, aligned longword, demand D-stream read reference is generated. If the required data is a word that crosses a longword boundary, or an unaligned longword, then two successive aligned longword demand D-stream read references are generated. Data larger than a longword is divided into a number of successive aligned longword demand D-stream reads, with no optimization.

3.1.7.9 Write References

Whenever data is stored or moved, a write reference is generated. Since the CPU does not impose any restrictions on data alignment (other than the aligned operands of the ADAWI and interlocked queue instructions) and since memory can only be accessed one aligned longword at a time, all data write references are translated into an appropriate combination of masked and unmasked aligned longword write references.

If the required data is a byte, a word within a longword, or an aligned longword, then a single, aligned longword, write reference is generated. If the required data is a word that crosses a longword boundary, or an unaligned longword, then two successive aligned longword write references are generated. Data larger than a longword is divided into a number of successive aligned longword writes.

3.2 Floating Point Accelerator

The KA640 floating point accelerator is implemented via a single VLSI chip called the CFPA.

3.2.1 Floating Point Accelerator Instructions

The KA640 floating point accelerator processes F_Floating, D_Floating, and G_Floating format instructions and accelerates the execution of MULL, DIVL, and EMUL integer instructions.

3.2.2 Floating Point Accelerator Data Types

The KA640 floating point accelerator supports byte, word, longword, quadword, F_Floating, D_Floating, and G_Floating data types. The H_Floating data type is not supported, but may be implemented by macrocode emulation.

3.3 Cache Memory

To maximize CPU performance, the KA640 incorporates a cache memory, implemented within the CVAX chip.

3.3.1 Cacheable References

Any reference that can be stored by the cache is called a *cacheable reference*. The cache stores CPU read references to the VAX memory space (bit <29> of the physical address equals 0) only. It does not store references to the VAX I/O space, or DMA references by the Q22-bus interface. The type(s) of CPU references that can be stored (either request I-stream read references, or demand D-stream read references other than read-lock references) is determined by the state of cache disable register (CADR) bits <5:4>. The normal operating mode is for both I-stream and D-stream references to be stored.

Whenever the CPU generates a non-cacheable reference, a single longword reference of the same type is generated on the CDAL bus.

Whenever the CPU generates a cacheable reference that is stored in the cache, no reference is generated on the CDAL bus.

Whenever the CPU generates a cacheable reference that is not stored in the cache, a quadword transfer is generated on the CDAL bus. If the CPU reference was a request I-stream read, then the quadword transfer consists of two indivisible longword transfers, the first being a request I-stream read (prefetch), and the second being a request I-stream read (fill). If the CPU reference was a demand D-stream read, then the quadword transfer consists of two indivisible longword transfers, the first being a demand D-stream read, and the second being a request D-stream read (fill).

3.3.2 Cache

The KA640 includes a 1 KB, two-way associative, write through cache with a 100 ns cycle time. CPU read references access one longword at a time, while CPU writes can access one byte at a time. A single parity bit is generated, stored, and checked for each byte of data and each tag. The cache can be enabled/disabled by setting/clearing the appropriate bits in the CADR. The cache is flushed by any write to the CADR, as long as it is not in diagnostic mode.

3.3.2.1 Cache Organization

The cache is divided into two independent storage arrays called set 1 and set 2. Each set contains a 64 row x 22-bit tag array and a 64 row x 72-bit data array. The two sets are organized as shown in Figure 3-8.

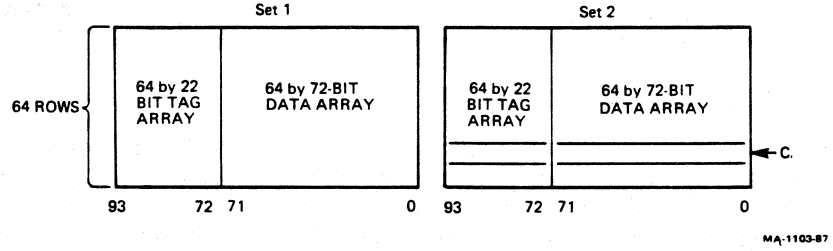


Figure 3-8 Cache Organization

A row within a set corresponds to a cache entry, so there are 64 entries in each set and a total of 128 entries in the entire cache. Each entry contains a 22-bit tag block and a 72-bit (eight-byte) data block. A cache entry is organized as shown in Figure 3-9.

A tag block consists of a parity bit, a valid bit, and a 20-bit tag. A tag block is organized as shown in Figure 3-10.

A data block consists of eight bytes of data, each with an associated parity bit. The total data capacity of the cache is 128 eight-byte blocks, or 1024 bytes. A data block is organized as shown in Figure 3-11.

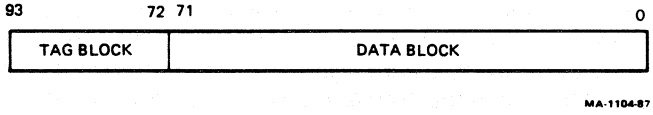


Figure 3-9 Cache Entry

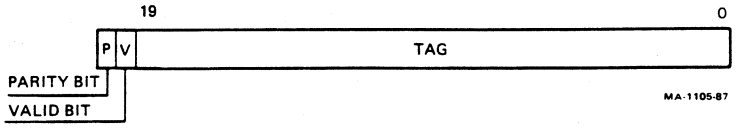


Figure 3–10 Cache Tag Block

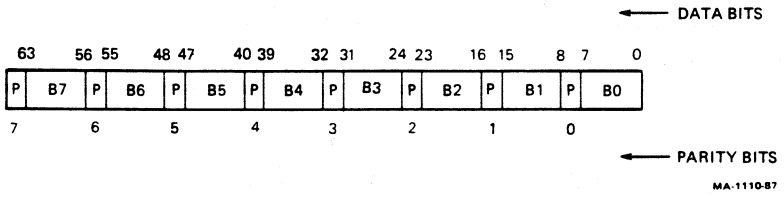


Figure 3–11 Cache Data Block

3.3.2.2 Cache Address Translation

Whenever the CPU requires an instruction or data, the contents of the cache is checked to determine if the referenced location is stored there. The cache contents is checked by translating the physical address as described in the following paragraphs.

On non-cacheable references, the reference is never stored in the cache, so a cache *miss* occurs and a single longword reference is generated on the CDAL bus.

On cacheable references, the physical address must be translated to determine if the contents of the referenced location is resident in the cache. The cache index field, bits <8:3> of the physical address, is used to select one of the 64 rows of the cache, with each row containing a single entry from each set. The cache tag field, bits <28:9> of the physical address, is then compared to the tag block of the entry from both sets in the selected row.

If a match occurs with the tag block of one of the set entries, and the valid bit within the entry is set, the contents of the referenced location is contained in the cache and a cache *hit* occurs. On a cache hit, the set match signals generated by the compare operation select the data block from the appropriate set. The cache displacement field, bits <2:0> of the physical address, is used to select the byte(s) within the block. No CDAL bus transfers are initiated on CPU references that *hit* the cache.

If no match occurs, then the contents of the referenced location is not contained in the cache and a cache *miss* occurs. In this case, the data must be obtained from the main memory controller, so a quadword transfer is initiated on the CDAL bus (Figure 3-12).

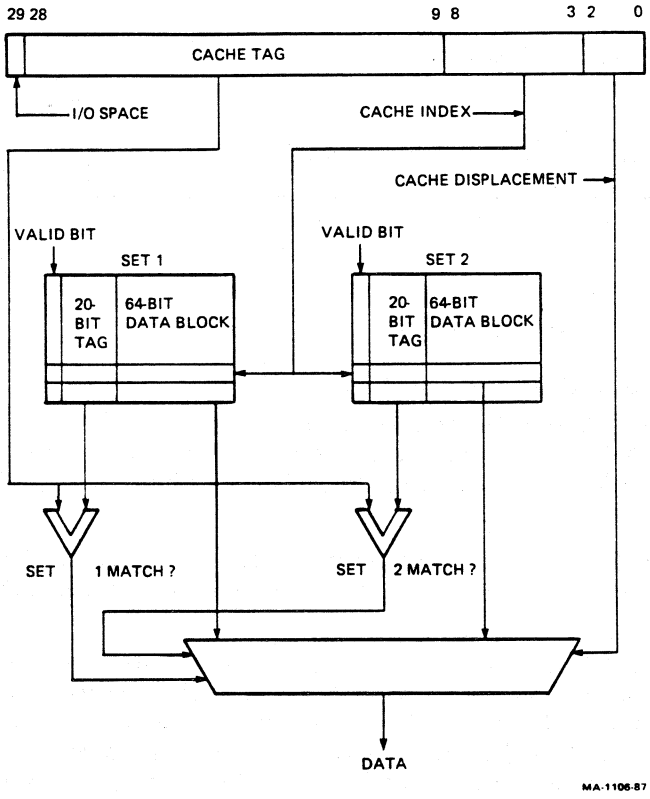


Figure 3-12 Cache Address Translation

3.3.2.3 Cache Data Block Allocation

Cacheable references that *miss* the cache, cause a quadword read to be initiated on the CDAL bus. When the requested quadword is supplied by the main memory controller, the requested longword is passed on to the CPU, and a data block is allocated in the cache to store the entire quadword.

Due to the fact that the cache is two-way associative, there are only two data blocks (one in each set) that can be allocated to a given quadword. These two data blocks are determined by the cache index field of the address of the quadword, which selects a unique row within the cache. Selection of

a data block within the row (i.e., set selection) for storing the new entry is random.

Since the KA640 supports 52MB (6.5M quadwords) of physical memory, up to 104K quadwords share each row (two data blocks) of the cache. Contiguous programs larger than 512 bytes or any non-contiguous programs separated by 512 bytes have a 50% chance of over-writing themselves when cache data blocks are allocated for the first time for data separated by 512 bytes (one page). After six allocations, there is a 97% probability both sets in a row will be filled.

3.3.2.4 Cache Behavior on Writes

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

On DMA write references that *hit* the cache, the cache entry containing the copy of the referenced location is invalidated. If the cache is configured to store only I-stream references, then the entire cache is also flushed whenever an REI instruction is executed. (The VAX Architecture requires that an REI instruction be executed before executing instructions out of a page of memory that has been updated.)

3.3.2.5 Cache Disable Register

The cache disable register (CADR), IPR 37, controls the cache, and is unique to CPU designs that use the CVAX chip (Figure 3-13).

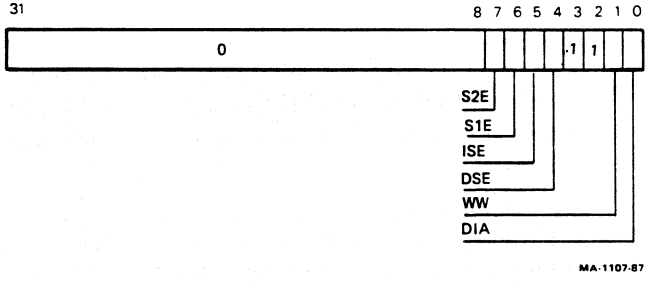


Figure 3-13 Cache Disable Register

Data Bit	Definition
CADR <31:8>	Unused. Always read as zeros. Writes have no effect.
CADR <7>	(S2E) Read/Write. This bit is used to selectively enable or disable set 2 within the cache. When set, set 2 of the cache is enabled. When cleared, set 2 of the cache is disabled. Cleared on power-up and by the negation of DCOK when the processor is halted.
CADR <6>	(S1E) Read/Write. This bit is used to selectively enable or disable set 1 within the cache. When set, set 1 of the cache is enabled. When cleared, set 1 of the cache is disabled. Cleared on power-up and by the negation of DCOK when the processor is halted.
CADR <5>	(ISE) Read/Write. This bit is used to selectively enable or disable storing I-stream references in the cache. When set, I-stream, memory space references are stored in the cache, if it is enabled. When cleared, I-stream memory references are not stored in the cache. Cleared on power-up and by the negation of DCOK when the processor is halted.
CADR <4>	(DSE) Read/Write. This bit is used to selectively enable or disable storing D-stream references in the cache. When set, D-stream, memory space references are stored in the cache, if it is enabled. When cleared, D-stream memory references are not stored in the cache. Cleared on power-up and by the negation of DCOK when the processor is halted.

NOTE

The cache can be disabled by either disabling both set 1 and set 2 (clearing CADR <7:6>), or by not storing either I-stream or D-stream references (clearing CADR <5:4>).

For maximum performance, the cache should be configured to store both I- and D-stream references. I-stream only mode suffers from a degradation in performance from what would normally be expected relative to I- and D-stream mode and D-stream only mode, due to the fact that invalidation of cache entries due to writes to memory by a DMA device are handled less efficiently.

Data Bit	Definition
	In I-stream only mode, the entire cache is flushed whenever an REI instruction is executed. (The *VAX Architecture Reference Manual* states that an REI instruction must be executed before executing instructions out of a page of memory that has been updated.) Whereas in the other two modes of operation, cache entries are invalidated on an individual basis, only if a DMA write operation results in a cache *hit*.
CADR <3:2>	Unused. Always read as 1s.
CADR <1>	(WWP) Write wrong parity. Read/Write. When set, incorrect parity is stored in the cache whenever it is written. When cleared, correct parity is stored in the cache whenever the cache is written. Cleared on power-up and by the negation of DCOK when the processor is halted.
CADR <0>	(DIA) Diagnostic mode. Read/Write. When cleared, the cache is in normal operating mode and writes to the CADR will cause the cache to be flushed, (all valid bits set to the invalid state) and the cache is configured for write-through operation. When set, the cache is in diagnostic mode and writes to the CADR will not cause the cache to be flushed. CPU write references with a longword destination (e.g., MOVL) will write the data into main memory (if it exists) as well as invalidate the corresponding cache entry regardless of whether or not a cache hit occurred. CPU write references with a quadword destination (e.g. MOVQ) will write the data into main memory (if it exists) as well as cause the SECOND longword of the quadword to be written into the longword of the cache data array that corresponds to the address of the FIRST longword of the destination, regardless of whether or not a cache hit occurred. The data in the longword of the cache data array that corresponds to the address of the second longword of the destination remains unaltered. In addition, errors generated during write references, that would normally cause a machine check, are ignored (they do not cause a machine check trap to be generated, or prevent data from being stored in the cache). Diagnostic mode is intended to allow the cache tag store to be fully tested without requiring 512 megabytes of main memory.

Data Bit	Definition
	This mode makes it possible for the tag block in a particular cache entry to be written with any pattern by executing a MOVQ instruction with bits <28:9> of the destination address equal to the desired pattern. Two MOVQ instructions, one with a quadword aligned destination address and one with the next longword aligned destination address, are required to write to both longwords in the data block of a cache entry. Diagnostic mode does not affect read references. Cleared on power-up and by the negation of DCOK when the processor is halted. **NOTE** **At least one read reference must occur between all write references made in diagnostic mode.** **Diagnostic mode should only be selected when one and only one of the two sets are enabled. Operation of this mode with both sets enabled or both sets disabled yields *unpredictable* results.**

3.3.2.6 Memory System Error Register

The memory system error register (MSER), IPR 39, records the occurrence of cache hits, as well as parity errors on the CDAL bus and in the cache. This register is unique to CPU designs that use the CVAX chip. MSER <6:4,1:0> are sticky in the sense that they remain set until explicitly cleared. Each bit is set on the first occurrence of the error it logs, and remains set for subsequent occurrences of that error. The MSER is explicitly cleared via the MTPR MSER instruction regardless of the write data (Figure 3-14).

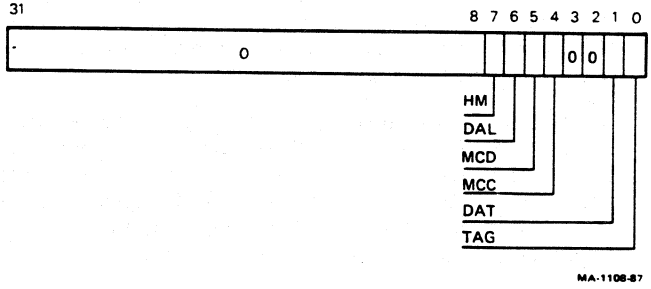


Figure 3-14 Memory System Error Register

Data Bit	Definition
MSER <31:8>	Unused. Always read as zeros. Writes have no effect.
MSER <7>	(HM) Hit/Miss. Read only. Writes have no effect. Cleared on all cacheable references that hit the cache. Set on all cacheable references that miss the cache. Cleared on power-up and by the negation of DCOK when the processor is halted.
MSER <6>	(DAL) DAL parity error. Read/Write to clear. This bit is set whenever a CDAL bus data store parity error is detected. Cleared on power-up and by the negation of DCOK when the processor is halted.
MSER <5>	(MCD) Machine check - DAL parity error. Read/Write to clear. This bit is set whenever a machine check is caused by a CDAL bus data parity error. These errors will only generate machine checks on demand D-stream read references. Cleared on power-up and by the negation of DCOK when the processor is halted.
MSER <4>	(MCC) Machine check - cache parity error. Read/Write to clear. This bit is set whenever a machine check is caused by a cache parity error in the tag or data store. These errors will only generate machine checks on demand D-stream read references. Cleared on power-up and by the negation of DCOK when the processor is halted.
MSER <3:2>	Unused. Always read zero. Writes have no effect.
MSER <1>	(DAT) Data parity error. Read/Write to clear. This bit is set when a parity error is detected in the data store of the cache. Cleared on power-up and by the negation of DCOK when the processor is halted.
MSER <0>	(TAG) Tag parity error. Read/Write to clear. This bit is set when a parity error is detected in the tag store of the cache. Cleared on power-up and by the negation of DCOK when the processor is halted.

3.3.2.7 Cache Error Detection

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

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

During a demand D-stream read reference, the entire cache is flushed, the CADR is cleared (which disables the cache). The cause of the error is logged in MSER <4,3:0> and a machine check abort is initiated.

During a request I-stream read reference, the entire cache is flushed (unless CADR <0> is set), the cause of the error is logged in MSER <1:0>, the prefetch is halted, but no machine check abort occurs, and both caches remain enabled.

During a masked or unmasked write reference, the entire cache is flushed (unless CADR <0> is set), the cause of the error is logged in MSER <0> (only tag parity is checked on CPU writes that hit the cache), there is no effect on CPU execution, and both caches remain enabled.

During a DMA write reference the cause of the error is logged in MSER<0> (only tag parity is checked on DMA writes that hit the cache), there is no effect on CPU execution, both caches remain enabled, and no invalidate operation occurs.

3.4 Main Memory System

The KA640 includes a main memory controller implemented via a single VLSI chip called the CMCTL. The KA640 main memory controller communicates with the MS650 memory boards over the MS650 memory interconnect, which utilizes the CD interconnect for the address and control lines and a 50-pin, ribbon cable for the data lines. It supports up to three MS650 memory boards, for a maximum of 52MB of ECC memory.

The controller supports synchronous longword read references, and masked or unmasked synchronous write references generated by the CPU as well as synchronous quadword read references generated by cacheable CPU references that miss the cache. Table 3-9 gives CPU read reference timing. Table 3-10 gives CPU write reference timing.

Table 3–9 CPU Read Reference Timing

Data Type	Timing
Longword	400 ns
Quadword	600 ns
First longword	400 ns
Second longword	200 ns
Aborted reference	400 ns
Longword (locked)	900 ns minimum
Aborted reference	400 ns
Retry (locked)	500 ns

Table 3–10 CPU Write Reference Timing

Data Type	Timing
Longword	200 ns
Longword (masked)	500 ns

The controller also supports asynchronous longword and quadword DMA read references and masked and unmasked asynchronous longword, quadword, hexword, and octaword DMA write references from the Q22-bus interface. Table 3–11 gives Q22-bus interface read reference timing. Table 3–12 gives Q22-bus interface write reference timing.

Table 3–11 Q22-bus Interface Read Reference Timing

Data Type	Timing
Longword	500 ns
Quadword	800 ns
First longword	500 ns
Second longword	300 ns
Longword (locked)	600 ns

Table 3–12 Q22-bus Interface Write Reference Timing

Data Type	Timing
Longword	400 ns
Longword (masked)	600 ns
Quadword	700 ns
First longword	400 ns
Second longword	300 ns
Quadword (masked)	1100 ns
First longword	400 ns
Second longword	700 ns
Hexword	1000 ns
First longword	400 ns
Second longword	300 ns
Third longword	300 ns
Hexword (masked)	1400 ns
First longword	400 ns
Second longword	300 ns
Third longword	700 ns
Octaword	1300 ns
First longword	400 ns
Second longword	300 ns
Third longword	300 ns
Fourth longword	300 ns
Octaword (masked)	1700 ns
First longword	400 ns
Second longword	300 ns
Third longword	300 ns
Fourth longword	700 ns

The timing in Table 3–12 assumes no exception conditions are encountered during the reference. Exception conditions will add the following amount of time if they are encountered during a reference:

Exception Condition	Time Added
Correctable error	100 ns
Uncorrectable error	200 ns-read
Uncorrectable error	100 ns-write
CDAL parity error	100 ns-write
Refresh collision	400 ns

The main memory controller contains eighteen registers. Sixteen registers are used to configure each of the sixteen possible banks in main memory. One register is used to control the operating mode of all memory banks and one register captures state on main memory errors.

3.4.1 Main Memory Organization

Main memory is logically and physically divided into 4 boards that correspond to the 3 possible MS650 memory expansion modules that can be attached to a KA640, plus the 4 Mbytes of on-board memory. Each board can contain zero (no memory module present), 1 (as on the KA640), or 2 (MS650-AA present) memory bank(s). Each bank contains 1,048,576 (1M) aligned longwords. Each aligned longword is divided into 4 data bytes and is stored with 7 ECC check bits, resulting in a memory array width of 39 bits.

3.4.2 Main Memory Addressing

The KA640 main memory controller is capable of controlling up to 13 banks of RAM, each bank containing 4MB of storage. Each bank of main memory has a programmable base address, determined by the state of bits <25:22> of the main memory configuration register associated with each bank.

A 4MB bank is accessed when bit <29> of the physical address is equal to 0, indicating a VAX memory space read/write reference, bits <28:26> of the physical address are equal to zero, indicating a reference within the range of the main memory controller, and the bank number of the bank matches bits <25:22> of the physical address. The remainder of the physical address (bits <21:2>) are used to determine the row and column of the desired longword within the bank. The byte mask lines are ignored on read operations, but are used to select the proper byte(s) within a longword during masked longword write references.

3.4.3 Main Memory Behavior on Writes

On unmasked CPU write references, the main memory controller operates in dump and run mode, terminating the CDAL bus transaction after latching the data, but before checking CDAL bus parity, calculating the ECC check bits, and transferring the data to main memory.

On unmasked DMA write references by the Q22-bus interface, the data is latched, CDAL bus parity is NOT checked, the CDAL bus transaction is terminated, the ECC check bits are calculated, and the data is transferred to main memory.

On single masked CPU or DMA write references, CDAL bus parity is checked (for CPU writes only), the referenced longword is read from main memory, the ECC code checked, the check bits recalculated to account for the new data byte(s), the CDAL transaction is terminated, and the longword is rewritten.

On multiple transfer masked DMA writes, each longword write is acknowledged, then the CDAL transaction is terminated.

3.4.4 Main Memory Error Status Register

The main memory status register (MEMCSR16), address 2008 0140 $_{16}$, is used to capture main memory error data. This register is unique to CPU designs that use the CMCTL memory controller chip (Figure 3-15).

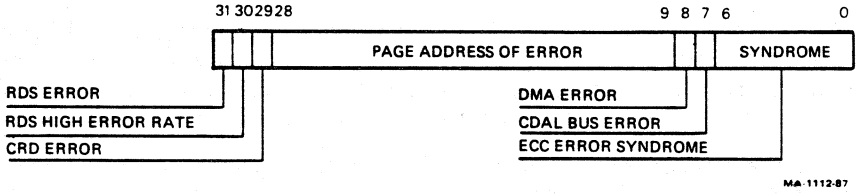


Figure 3-15 Format for MEMCSR16

Data Bit	Definition
MEMCSR16<31>	RDS error. Read/Write to clear. When set, an uncorrectable ECC error occurred during a memory read or masked write reference. Cleared by writing a 1 to it. Writing a 0 has no effect. Undefined if MEMCSR16<7> (CDAL bus error) is set. Cleared on power-up and the negation of DCOK when the processor is halted.
MEMCSR16<30>	RDS high error rate. Read/Write to clear. When set, an uncorrectable ECC error occurred while the RDS error log request bit was set, indicating multiple uncorrectable memory errors. Cleared by writing a 1 to it. Writing a 0 has no effect. Undefined if MEMCSR16<7> (CDAL bus error) is set. Cleared on power-up and the negation of DCOK when the processor is halted.
MEMCSR16<29>	CRD error. Read/Write to clear. When set, a correctable (single bit) error occurred during a memory read or masked write reference. Cleared by writing a 1 to it. Writing a 0 has no effect. Undefined if MEMCSR16<7> (CDAL bus error) is set. Cleared by writing a 1, on power-up and the negation of DCOK when the processor is halted.
MEMCSR16<28:9>	Page address of error. Read only. This field identifies the page (512 byte block) containing the location that caused the memory error. In the event of multiple memory errors, the types of errors are prioritized and the page address of the error with the highest priority is captured. Errors with equal priority do not overwrite previous contents. Writes have no effect. Cleared on power-up and the negation of DCOK when the processor is halted. The types of error conditions follow in order of priority: 1. CDAL bus parity errors during a CPU write reference, as logged by the CDAL bus error bit. 2. Uncorrectable ECC errors during a CPU or DMA read or masked write reference, as logged by the RDS error log bit. 3. Correctable ECC errors during a CPU or DMA read or masked write reference, as logged by CRD error bit.

Data Bit	Definition
MEMCSR16<8>	DMA error. Read/Write to clear. When set, an error occurred during a DMA read or write reference. Cleared by writing a 1 to it. Writing a 0 has no effect. Cleared on power-up and the negation of DCOK when the processor is halted.
MEMCSR16<7>	CDAL bus error. Read/Write to clear. When set, a CDAL bus parity error occurred on a CPU write reference. Cleared by writing a 1 to it. Writing a 0 has no effect. Cleared on power-up and the negation of DCOK when the processor is halted.
MEMCSR16<6:0>	Error syndrome. Read only. This field stores the error syndrome. A non-zero syndrome indicates a detectable error has occurred. A unique syndrome is generated for each possible single bit (correctable) error. A list of these syndromes and their associated single bit errors is given in Table 3-13. Any non-zero syndrome that is not contained in Table 3-13 indicates a multiple bit (uncorrectable) error has occurred. This field handles multiple errors in the same manner as MEMCSR16<28:9>. Cleared on power-up and the negation of DCOK when the processor is halted.

Table 3–13 Error Syndromes

Syndrome <6:0>	Bit Position in Error
0000000	no error detected
	Data Bits (0-32 decimal)
1011000	0
0011100	1
0011010	2
1011110	3
0011111	4
1011011	5
1011101	6
0011001	7
1101000	8
0101100	9
0101010	10
1101110	11
0101111	12
1101011	13
1101101	14
0101001	15
1110000	16
0110100	17
0110010	18
1110110	19
0110111	20
1110011	21
1110101	22
0110001	23
0111000	24
1111100	25
1111010	26
0111110	27
1111111	28
0111011	29
0111101	30
1111001	31
	Check Bits (32-38 decimal)
0000001	32

Table 3–13 (Cont.) Error Syndromes

Syndrome <6:0>	Bit Position in Error
0000010	33
0000100	34
0001000	35
0010000	36
0100000	37
1000000	38
0000111	Result of incorrect check bits written on detection of a CDAL parity error.
All others	Multi-bit errors

3.4.5 Main Memory Control and Diagnostic Status Register

The main memory control and diagnostic status register (MEMCSR17), address 2008 0144 $_{16}$, is used to control the operating mode of the main memory controller as well as to store diagnostic status information. This register is unique to CPU designs that use the CMCTL memory controller chip (Figure 3–16).

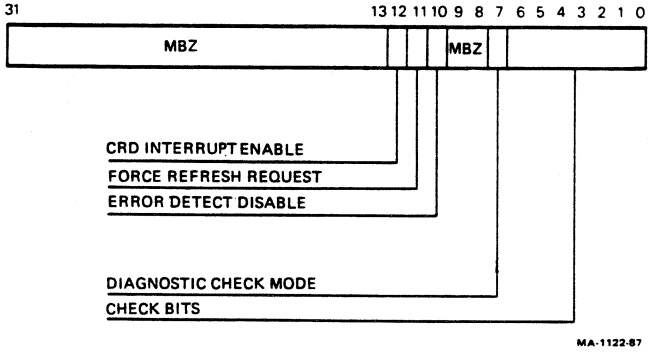


Figure 3–16 Format for MEMCSR17

Data Bit	Definition
MEMCSR17<31:13>	Unused. This field reads as zero and must be written as zero.
MEMCSR17<12>	CRD interrupt enable. Read/Write. When cleared, single-bit errors are corrected by the ECC logic, but no interrupt is generated. When set, single-bit errors are corrected by the ECC logic and they cause an interrupt to be generated at IPL 1A with a vector of 54_{16}. This bit has no effect on the capturing of error information in MEMCSR16, or on the reporting of uncorrectable errors. Cleared on power-up and the negation of DCOK when the processor is halted.
MEMCSR17 <11>	Force refresh request. Read/Write. When cleared, the refresh control logic operates in normal mode (refresh every 11.3 μs). When set, one memory refresh operation occurs immediately after the MEMCSR write reference that set this bit. Setting this bit provides a mechanism for speeding up the testing of the refresh logic during manufacturing test of the controller chip. This bit is cleared by the memory controller upon completion of the refresh operation. Cleared on power-up and the negation of DCOK when the processor is halted.
MEMCSR17 <10>	Memory error detect disable. Read/Write. When set, error detection and correction (ECC) is disabled, so all memory errors go undetected. When cleared, error detection, correction, state capture and reporting (via MEMCSR16) is enabled. Cleared on power-up and the negation of DCOK when the processor is halted.
MEMCSR17 <9:8>	Unused. This field reads as zero and must be written as zero.
MEMCSR17 <7>	Diagnostic check mode. Read/Write. When set, the contents of MEMCSR17 <6:0> are written into the 7 ECC check bits of the location (even if a CDAL parity error is detected) during a memory write reference. When cleared, the 7 check bits calculated by the ECC generation logic are loaded into the 7 ECC check bits of the location during a write reference and a memory read reference will load the state of the 7 ECC check bits of the location that was read into MEMCSR17 <6:0>. Cleared on power-up and the negation of DCOK when the processor is halted.

Data Bit	Definition
	NOTE **Diagnostic check mode is restricted to unmasked memory write references. No masked write references are allowed when diagnostic check mode is enabled.**
MEMCSR17 <6:0>	Check bits. Read/Write. When the diagnostic check mode bit is set, these bits are substituted for the check bits that are generated by the ECC generation logic during a write reference. When the diagnostic check mode bit is cleared, memory read references load the state of the 7 ECC check bits of the location that was read into MEMCSR16 <6:0>. Cleared on power-up and the negation of DCOK when the processor is halted.

3.4.6 Main Memory Error Detection and Correction

The KA640 main memory controller generates CDAL bus parity on CPU read references, and checks CDAL bus parity on CPU write references.

The actions taken following the detection of a CDAL bus parity error depend on the type of write reference.

For unmasked CPU write references, incorrect check bits are written to main memory (potentially masking an as yet undetected memory error) along with the data and an interrupt is generated at IPL 1D through vector 60_{16} on the next cycle and MCSR16 <7> is set. The incorrect check bits are determined by calculating the seven correct check bits, and complementing the three least significant bits.

For masked CPU write references, incorrect check bits are written to main memory (potentially masking an as yet undetected memory error) along with the data, unless an uncorrectable error is detected during the read portion, MEMCSR16 <7> is set, and a machine check abort is initiated. If an uncorrectable error is detected on the read portion, no write operation takes place. The incorrect check bits are determined by calculating the seven correct check bits, and complementing the three least significant bits.

The memory controller protects main memory by using a 32-bit modified Hamming code to encode the 32-bit data longword with seven check bits. This allows the controller to detect and correct single-bit errors in the data field and detect single bit errors in the check bit field and double-bit errors

in the data field. The most likely causes of these errors are failures in either the memory array or the 50-pin cable.

Upon detecting a correctable error on a read reference or the read portion of a masked write reference, the data is corrected (if it is in the data field), before placing it on the CDAL bus, or back in main memory, an interrupt is generated at IPL 1A through vector 54_{16}, bit <29> of MEMCSR16 is set, bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error, and bits <6:0> are loaded with the error syndrome which indicates which bit was in error. If the error was detected on a DMA reference, MEMCSR16 <8> is also set.

NOTE

The corrected data is not rewritten to main memory, so the single bit error will remain there until rewritten by software.

Upon detecting an uncorrectable error, the action depends on the type of reference being performed.

On a demand read reference, the affected row of the cache is invalidated, bit <31> of MEMCSR16 is set, bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error, and bits <6:0> are loaded with the error syndrome which indicates that the error was uncorrectable and a machine check abort is initiated. If the read was a local-miss, global- hit read, or a read of the Q22-bus map, MEMCSR16 <8> and DSER <4> are also set, and DEAR <12:0> are loaded with the address of the page containing the location that caused the error.

On a request read reference, the prefetch or fill cycle is aborted, but no machine check occurs, bit <31> of MEMCSR16 is set, bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error, and bits <6:0> are loaded with the error syndrome which indicates that the error was uncorrectable.

On the read portion of masked write reference, bit <31> of MEMCSR16 is set, bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error, and bits <6:0> are loaded with the error syndrome which indicates that the error was uncorrectable and a machine check abort is initiated.

On a DMA read reference, bit <31> and bit <8> of MEMCSR16 are set, bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error, and bits <6:0> are loaded with the error syndrome which indicates that the error was uncorrectable, DSER <4> is set, DEAR <12:0> are loaded with the address of the page containing the location that caused the error, BDAL <17:16> are asserted on the Q22-bus along with the data to notify the receiving device (unless it

was a map read by the Q22-bus interface during translation), and an interrupt is generated at IPL 1D through vector 60 $_{16}$.

On a DMA masked write reference, bit <31> and bit <8> of MEMCSR16 are set, bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error, and bits <6:0> are loaded with the error syndrome which indicates that the error was uncorrectable, DSER <4> is set, DEAR <12:0> are loaded with the address of the page containing the location that caused the error, IPCR <15> is set to notify the initiating device, and an interrupt is generated at IPL 1D through vector 60 $_{16}$.

3.5 Console Serial Line

The console serial line provides the KA640 processor with a full duplex, RS-423 EIA, serial line interface, which is also RS-232C compatible. The only data format supported is 8-bit data with no parity and one stop bit. The four IPRs that control the operation of the console serial line are a superset of the VAX console serial line registers described in the *VAX Architecture Reference Manual.*

3.5.1 Console Registers

There are four registers associated with the console serial line unit. They are implemented in the SSC and are accessed as IPRs 32 $_{10}$ through 35 $_{10}$. Refer to Table 3-14.

Table 3-14 Console Registers

IPR Number	Register Name	Mnemonic
32	Console receiver control/status	RXCS
33	Console receiver data buffer	RXDB
34	Console transmit control/status	TXCS
35	Console transmit data buffer	TXDB

3.5.1.1 Console Receiver Control/Status Register

The console receiver control/status register (RXCS), IPR 32, is used to control and report the status of incoming data on the console serial line (Figure 3-17).

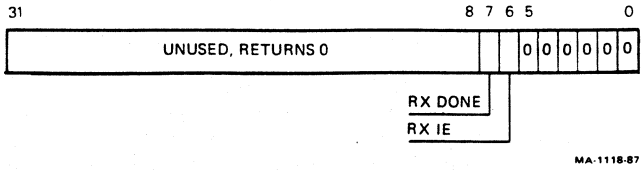


Figure 3–17 Console Receiver Control/Status Register

Data Bit	Definition
RXCS <31:8>	Unused. Read as zeros. Writes have no effect.
RXCS <7>	(RX DONE) Receiver done. Read only. Writes have no effect. This bit is set when an entire character has been received and is ready to be read from the RXDB Register. This bit is automatically cleared when RXDB is read. It is also cleared on power-up and the negation of DCOK when the processor is halted.
RXCS <6>	(RX IE) Receiver interrupt enable. Read/Write. When set, this bit causes an interrupt to be requested at IPL 14 with an SCB offset of F8 if RX DONE is set. When cleared, interrupts from the console receiver are disabled. This bit is cleared on power-up and the negation of DCOK when the processor is halted.
RXCS <5:0>	Unused. Read as zeros. Writes have no effect.

3.5.1.2 Console Receiver Data Buffer

The console receiver data buffer (RXDB), IPR 33, is used to buffer incoming data on the serial line and capture error information (Figure 3–18).

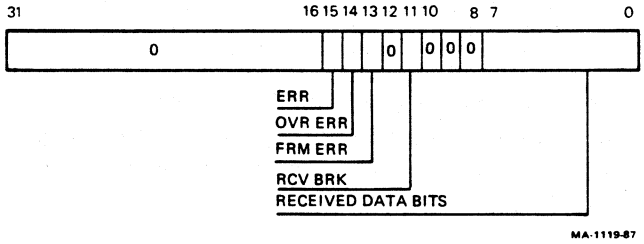


Figure 3–18 Console Receiver Data Buffer

Data Bit	Definition
RXDB <31:16>	Unused. Always read as zero. Writes have no effect.
RXDB <15>	(ERR) Error. Read only. Writes have no effect. This bit is set if RBUF <14> or <13> is set. It is clear if these two bits are clear. This bit cannot generate a program interrupt. Cleared on power-up and the negation of DCOK when the processor is halted.
RXDB <14>	(OVR ERR) Overrun error. Read only. Writes have no effect. This bit is set if a previously received character was not read before being overwritten by the present character. Cleared by reading the RXDB, on power-up and the negation of DCOK when the processor is halted.
RXDB <13>	(FRM ERR) Framing error. Read only. Writes have no effect. This bit is set if the present character did not have a valid stop bit. Cleared by reading the RXDB, on power-up and the negation of DCOK when the processor is halted.
	NOTE **Error conditions remain present until the next character is received, at which point, the error bits are updated.**
RXDB <12>	Unused. This bit always reads as 0. Writes have no effect.
RXDB <11>	(RCV BRK) Received break. Read only. Writes have no effect. This bit is set at the end of a received character for which the serial data input remained in the space condition for 20 bit times. Cleared by reading the RXDB, on power-up and the negation of DCOK when the processor is halted.
RXDB <10:8>	Unused. These bits always read as 0. Writes have no effect.
RXDB <7:0>	Received data bits. Read only. Writes have no effect. These bits contain the last received character.

3.5.1.3 Console Transmitter Control/Status Register

The console transmitter control/status register (TXCS), internal processor register 34, is used to control and report the status of outgoing data on the console serial line (Figure 3-19).

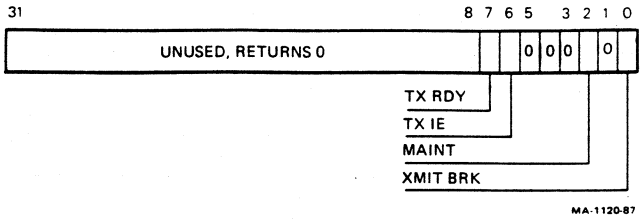


Figure 3-19 Console Transmitter Control/Status Register

Data Bit	Definition
TXCS <31:8>	Unused. Read as zeros. Writes have no effect.
TXCS <7>	(TX RDY) Transmitter ready. Read only. Writes have no effect. This bit is cleared when TXDB is loaded and set when TXDB can receive another character. This bit is set on power-up and the negation of DCOK when the processor is halted.
TXCS <6>	(TX IE) Transmitter interrupt enable. Read/Write. When set, this bit causes an interrupt to be requested at IPL 14 with an SCB offset of FC if TX RDY is set. When cleared, interrupts from the console receiver are disabled. This bit is cleared on power-up and the negation of DCOK when the processor is halted.
TXCS <5:3>	Unused. Read as zeros. Writes have no effect.
TXCS <2>	(MAINT) Maintenance. Read/Write. This bit is used to facilitate a maintenance self-test. When MAINT is set, the external serial input is set to MARK and the serial output is used as the serial input. This bit is cleared on power-up and the negation of DCOK when the processor is halted.
TXCS<1>	Unused. Read as zero. Writes have no effect.
TXCS<0>	(XMIT BRK) Transmit break. Read/Write. When this bit is set, the serial output is forced to the space condition after the character in TXB<7:0> is sent. While XMIT BRK is set, the transmitter will operate normally, but the output line will remain low. Thus, software can transmit dummy characters to time the break. This bit is cleared on power-up and the negation of DCOK when the processor is halted.

3.5.1.4 Console Transmitter Data Buffer

The console transmitter data buffer (TXDB), internal processor register 35, is used to buffer outgoing data on the serial line (Figure 3-20).

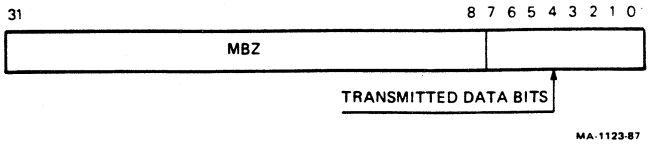


Figure 3-20 Console Transmitter Data Buffer

Data Bit	Definition
TXDB<31:8>	Unused. Writes have no effect.
TXDB<7:0>	Transmitted data bits. Write only. These bits are used to load the character to be transmitted on the console serial line.

3.5.2 Break Response

The console serial line unit recognizes a break condition which consists of 20 consecutively received space bits. If the console detects a valid break condition, the RCV BRK bit is set in the RXDB register. If the break was the result of 20 consecutively received space bits, the FRM ERR bit is also set. If halts are enabled (ENBHALT asserted on the 40-pin connector), the KA640 will halt and transfer program control to ROM location 2004 0000 when the RCV BRK bit is set. RCV BRK is cleared by reading RXDB. Another mark followed by 20 consecutive space bits must be received to set RCV BRK again.

3.5.3 Baud Rate

The receive and transmit baud rates are always identical and are controlled by the SSC configuration register bits <14:12>.

The user selects the desired baud rate through pins <30:28> on the the 40-pin system support connector (CONBITRATE <02:00>), which are configured using the select switch on the inside of the H3602-SA. The KA640 firmware reads this code from boot and diagnostic register bits <6:4> and loads it into SSC configuration register bits <14:12>. Operating systems will not cause the baud rate to be transferred. The baud rate is only set on power up.

Table 3-15 shows the baud rate select signal voltage levels (H or L), the corresponding INVERTED code as read in the boot and diagnostic register bits <6:4>, and the code that should be loaded into SSC configuration register bits <14:12>.

Table 3-15 Baud Rate Select

Baud Rate	CONBITRATE <2:0>	BDR <6:4>	SSC <14:12>
300	HHH	000	000
600	HHL	001	001
1200	HLH	010	010
2400	HLL	011	011
4800	LHH	100	100
9600	LHL	101	101
19200	LLH	110	110
38400	LLL	111	111

3.5.4 Console Interrupt Specifications

The console serial line receiver and transmitter both generate interrupts at IPL 14. The receiver interrupts with a vector of $F8_{16}$, while the transmitter interrupts with a vector of FC_{16}.

3.6 Time of Year Clock and Timers

The KA640 clocks include time of year clock (TODR) as defined in the *VAX Architecture Reference Manual,* a subset interval clock (subset ICCS), as defined in the *VAX Architecture Reference Manual,* and two additional programmable timers modeled after the VAX standard interval clock.

3.6.1 Time of Year Clock

The KA640 time of year clock (TODR), internal processor register 27, forms an unsigned 32-bit binary counter that is driven from a 100Hz oscillator, so that the least significant bit of the clock represents a resolution of 10 milliseconds, with less than .0025% error. The register counts only when it contains a non-zero value. This register is implemented in the SSC (Figure 3-21).



Figure 3-21 Time of Year Clock

The time of year clock is maintained during power failure by battery backup circuitry which interfaces, via the external connector, to a set of batteries which are mounted on the H3602-SA. The (TODR) will remain valid for greater than 162 hours when using the NiCad battery pack (three batteries in series).

The SSC configuration register contains a battery low (BLO) bit which, if set after initialization, the TODR is cleared, and will remain at zero until software writes a non-zero value into it.

NOTE

After writing a non-zero value into the TODR, software should clear the BLO bit by writing a 1 to it.

3.6.2 Interval Timer

The KA640 interval timer (ICCS), internal processor register 24, is implemented according to the *VAX Architecture Reference Manual* for subset processors. The interval clock control/status register (ICCS) is implemented as the standard subset of the standard VAX ICCS in the CVAX CPU chip, while NICR and ICR are not implemented (Figure 3-22).

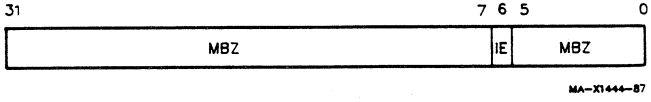


Figure 3-22 Interval Timer

Data Bit	Definition
ICCS<31:7>	Unused. Read as zeros, must be written as zeros.
ICCS<6>	(IE) Interrupt enable. Read/Write. This bit enables and disables the interval timer interrupts. When the bit is set, an interval timer interrupt is requested every 10 msec with an error of less than .01%. When the bit is clear, interval timer interrupts are disabled. This bit is cleared on power-up and the negation of DCOK when the processor is halted.
ICCS<5:0>	Unused. Read as zeros, must be written as zeros.

Interval timer requests are posted at IPL 16 with a vector of C0: the interval timer is the highest priority device at this IPL.

3.6.3 Programmable Timers

The KA640 features two programmable timers. Although they are modeled after the VAX standard interval clock, they are accessed as I/O space registers (rather than as internal processor registers) and a control bit has been added which stops the timer upon overflow. If so enabled, the timers will interrupt at IPL 14 upon overflow. The interrupt vectors are programmable and are set to 78 and 7C by the firmware.

Each timer is composed of four registers: a timer n control register, a timer n interval register, a timer n next interval register, and a timer n interrupt vector register, where n represents the timer number (0 or 1).

3.6.3.1 Timer Control Registers

The KA640 has two timer control registers, one for controlling timer 0 (TCR0), and one for controlling timer 1 (TCR1). TCR0 is accessible at address $2014\ 0100_{16}$, and TCR1 is accessible at $2014\ 0110_{16}$. These registers are implemented in the SSC (Figure 3-23).

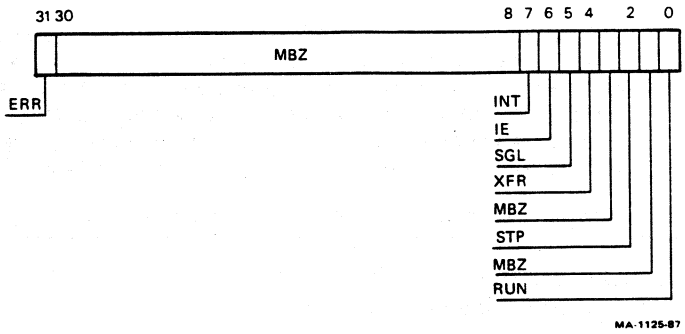


Figure 3-23 Timer Control Registers

Data Bit	Definition
TCRn<31>	(ERR) Error. Read/Write to clear. This bit is set whenever the timer interval register overflows and INT is already set. Thus, the ERR indicates a missed overflow. Writing a 1 to this bit clears it. Cleared on power-up and the negation of DCOK when the processor is halted.
TCRn<30:8>	Unused. Read as zeros, must be written as zeros.
TCRn<7>	(INT) Read/Write to clear. This bit is set whenever the timer interval register overflows. If IE is set when INT is set, an interrupt is posted at IPL 14. Writing a 1 to this bit clears it. Cleared on power-up and the negation of DCOK when the processor is halted.
TCRn<6>	(IE) Read/Write. When this bit is set, the timer will interrupt at IPL 14 when the INT bit is set. Cleared on power-up and the negation of DCOK when the processor is halted.
TCRn<5>	(SGL) Read/Write. Setting this bit causes the timer interval register to be incremented by 1 if the RUN bit is cleared. If the RUN bit is set, then writes to the SGL bit are ignored. This bit always reads as 0. Cleared on power-up and the negation of DCOK when the processor is halted.
TCRn<4>	(XFR) Read/Write. Setting this bit causes the timer next interval register to be copied into the timer interval register. This bit is always read as 0. Cleared on power-up and the negation of DCOK when the processor is halted.

Data Bit	Definition
TCRn<3>	Unused. Read as zeros, must be written as zeros.
TCRn<2>	(STP) Read/Write. This bit determines whether the timer stops after an overflow when the RUN bit is set. If the STP bit is set at overflow, the RUN bit is cleared by the hardware at overflow and counting stops. Cleared on power-up and the negation of DCOK when the processor is halted.
TCRn<1>	Unused. Read as zeros, must be written as zeros.
TCRn<0>	(RUN) Read/Write. When set, the timer interval register is incremented once every microsecond. The INT bit is set when the timer overflows. If the STP bit is set at overflow, the RUN bit is cleared by the hardware at overflow and counting stops. When the RUN bit is clear, the timer interval register is not incremented automatically. Cleared on power-up and the negation of DCOK when the processor is halted.

3.6.3.2 Timer Interval Registers

The KA640 has two timer interval registers, one for timer 0 (TIR0), and one for timer 1 (TIR1). TIR0 is accessible at address 2014 0104 $_{16}$, and TIR1 is accessible at 2014 0114 $_{16}$.

The timer interval register is a read only register containing the interval count. When the run bit is 0, writing a 1 increments the register. When the RUN bit is 1, the register is incremented once every microsecond. When the counter overflows, the INT bit is set, and an interrupt is posted at IPL 14 if the IE bit is set. Then, if the RUN and STP bits are both set, the RUN bit is cleared and counting stops. Otherwise, the counter is reloaded. The maximum delay that can be specified is approximately 1.2 hours. This register is cleared on power-up and the negation of DCOK when the processor is halted (Figure 3-24).

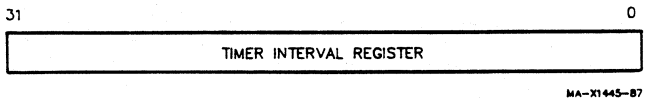


Figure 3-24 Timer Interval Register

3.6.3.3 Timer Next Interval Registers

The KA640 has two timer next interval registers, one for timer 0 (TNIR0), and one for timer 1 (TNIR1). TNIR0 is accessible at address 2014 0108 $_{16}$, and TNIR1 is accessible at 2014 0118 $_{16}$. These registers are implemented in the SSC.

This read/write register contains the value which is written into the timer interval register after overflow, or in response to a 1 written to the XFR bit. This register is cleared on power-up and the negation of DCOK when the processor is halted (Figure 3-25).

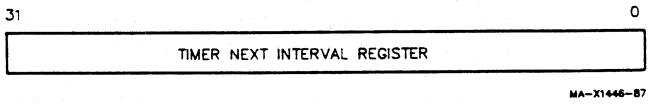


Figure 3-25 Timer Next Interval Register

3.6.3.4 Timer Interrupt Vector Registers

The KA640 has two timer interrupt vector registers, one for timer 0 (TIVR0), and one for timer 1 (TIVR1). TIVR0 is accessible at address 2014 010C $_{16}$, and TIVR1 is accessible at 2014 011C $_{16}$. These registers are implemented in the SSC and are set to 78 and 7C respectively by the resident firmware.

This read/write register contains the timer's interrupt vector. Bits <31:10> and <1:0> are read as 0 and must be written as 0. When TCRn<6> (IE) and TCRn<7> (INT) transition to 1, an interrupt is posted at IPL 14. When a timer's interrupt is acknowledged, the content of the interrupt vector register is passed to the CPU, and the INT bit is cleared. Interrupt requests can also be cleared by clearing either the IE or the INT bit. This register is cleared on power-up and the negation of DCOK when the processor is halted (Figure 3-26).

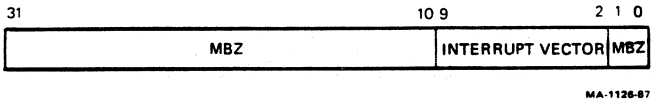


Figure 3-26 Timer Interrupt Vector Register

NOTE

Note that both timers interrupt at the same IPL (IPL 14) as the console serial line unit. When multiple interrupts are pending, the console serial line has priority over the timers, and timer 0 has priority over timer 1.

3.7 Boot and Diagnostic Facility

The KA640 boot and diagnostic facility features two registers, two 28-pin ROM sockets containing 128K bytes of EPROM, and 1KB of battery backed up RAM. The ROM and battery backed up RAM may be accessed via longword, word or byte references.

The KA640 CPU module populates the ROM sockets with 64K bytes of 16-bit ROM (or EPROM). This ROM contains the KA640 resident firmware. If this ROM is replaced for special applications, the new ROM must initialize and configure the board, provide halt and console emulation, as well as provide boot diagnostic functionality.

3.7.1 Boot and Diagnostic Register

The boot and diagnostic register (BDR) is a byte-wide register located in the VAX I/O page at physical address 2008 4004 $_{16}$. It is implemented uniquely on the KA640. It can be accessed by KA640 software, but not by external Q22-bus devices. The BDR allows the boot and diagnostic ROM programs to read various KA640 configuration bits. Only the low byte of the BDR should be accessed, bits <31:8> are undefined (Figure 3-27).

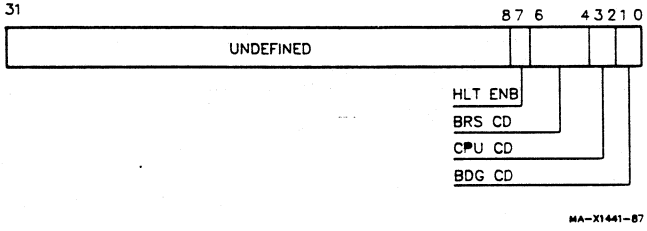


Figure 3-27 Boot and Diagnostic Register

Data Bit	Definition
BDR<31:8>	Undefined. Should not be read or written.
BDR<7>	(ENBHALT) Halt enable. Read only. Writes have no effect. This bit reflects the state of pin 35 (ENBHALT L) of the 40-pin connector. The assertion of this signal enables the halting of the CPU upon detection of a console break condition. On a power-up, the KA640 resident firmware reads the ENBHALT bit to decide whether to enter the console emulation program (ENBHALT set) or to boot the operating system (ENBHALT clear). On the execution of a HALT instruction while in kernel mode, the KA640 resident firmware reads the ENBHALT bit to decide whether to enter the console emulation program (ENBHALT set) or to restart the operating system (ENBHALT clear).
BDR<6:4>	(CONBITRATE) Console bit rate <02:00>. Read only. Writes have no effect. These three bits originate from pins <30:28> of the 40-pin connector. They reflect the setting of the baud rate select switch on the H3602-SA. These bits are read only on power up.

BDR<6:4>	Baud Rate
000	300
001	600
010	1200
011	2400
100	4800
101	9600
110	19200
111	38400

BDR<3:2>	(CPUCODE) CPU code <01:00>. Read only. Writes have no effect. These two bits originate from pins <40:39> of the 40-pin connector.

Data Bit	Definition

CPUCODE <01:00>	Configuration
00	Normal operation
01	Reserved
10	Reserved
11	Reserved

BDR<1:0> (BDCODE) Boot and diagnostic code <01:00>. Read only. Writes have no effect. This 2-bit code reflects the status of system support connector (J1) pins <37:36>. The KA640 ROM programs use BDCODE <01:00> to determine the power up mode as defined in the following:

BDCODE <01:00>	Power Up Mode
00	Run
01	Language inquiry
10	Test
11	Manufacturing

3.7.2 Diagnostic LED Register

The diagnostic LED register (DLEDR), address 2014 0030 $_{16}$, is implemented in the SSC and contains four read/write bits that control the external LED display. A 0 in a bit lights the corresponding LED; all four bits are cleared on power-up and the negation of DCOK when the processor is halted to provide a power-up lamp test (Figure 3-28).

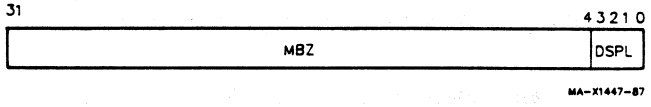


Figure 3-28 Diagnostic LED Register

Data Bit	Definition
DLEDR<31:4>	Unused. Read as zeros, must be written as zeros.
DLEDR<3:0>	(DSPL) Display <3:0>. Read/Write. These four bits update an external LED display. Writing a 0 to a bit lights the corresponding LED. Writing a 1 to a bit turns its LED off. The display bits are cleared (all LEDs are lit) on power-up and the negation of DCOK when the processor is halted.

3.7.3 ROM Memory

The KA640 supports up to 128KB of ROM memory for storing code for board initialization, VAX standard console emulation, board self-tests, and boot code. ROM memory may be accessed via byte, word and longword references. ROM accesses take 1300 ns. ROM is organized as a 64K x 8-bit array for one 64KB ROM, as a 32K by 16-bit array for two 32KB ROMs, and as a 64K by 16-bit array for two 64KB ROMs (ship configuration). CDAL bus parity is neither checked nor generated on ROM references.

3.7.3.1 ROM Socket

The KA640 provides two ROM sockets which contain two 64K by 8 EPROMs.

3.7.3.2 ROM Address Space

The entire 128KB boot and diagnostic ROM may be read from either the 128KB halt mode ROM space (hex addresses: 2004 0000 - 2005 FFFF), or the 128KB run mode ROM space (hex addresses: 2006 0000 - 2007 FFFF). Note that the run mode ROM space reads exactly the same ROM code as the halt mode ROM space.

Writes to either of these address spaces will result in a machine check.

Any I-stream read from the halt mode ROM space places the KA640 in halt mode. The Q22-bus SRUN signal is deasserted causing the front panel RUN light to extinguish and the CPU is protected from further halts.

Any I-stream read which does not access the halt mode ROM space, including reads from the run mode ROM space, places the KA640 in run mode. The Q22-bus SRUN signal is toggled causing the front panel RUN light to be lit and the CPU can be halted by asserting the Q22-bus BHALT line or by generating a break condition on the console serial line if BDR<7> (halt enable) is set.

Writes and D-stream reads to any address space have no effect on run mode/halt mode status.

3.7.3.3 KA640 Resident Firmware Operation

The KA640 CPU module populates the ROM socket with 128K bytes of 16-bit ROM (or EPROM). This ROM contains the KA640 resident firmware which can be entered by transferring program control to location 2004 0000 $_{16}$.

Section 3.1.5 lists the various halt conditions which cause the CVAX CPU to transfer program control to location 2004 0000 $_{16}$.

When running, the KA640 resident firmware provides the services expected of a VAX-11 console system. In particular, the following services are available:

- Automatic restart or bootstrap following processor halts or initial power up
- An interactive command language allowing the user to examine and alter the state of the processor
- Diagnostic tests executed on power up that check out the CPU, the memory system and the Q22-bus map
- Support of video or hardcopy terminals as the console terminal

Power Up Modes

The boot and diagnostic ROM programs use bits <1:0> of the BDR (Section 3.7.1) to determine the power up modes as follows:

Code	Mode
00	Run (factory setting). If the console terminal supports the multi-national character set (MCS), the user will be prompted for language only if the time-of-year clock battery backup has failed. Full startup diagnostics are run.
01	Language inquiry. If the console terminal supports MCS, the user will be prompted for language on every power up and restart. Full startup diagnostics are run.
10	Test. ROM programs run wrap-around serial line unit (SLU) tests.
11	Manufacturing. To provide for rapid startup during certain manufacturing test procedures, the ROM programs omit the power up memory diagnostics and set up the memory bit map on the assumption that all available memory is functional.

3.7.4 Battery Backed-Up RAM

The KA640 contains 1KB of battery backed-up static RAM, for use as a console *scratchpad*. The +12 Vdc power (fused) for the RAM is provided via pin 17 on the 40-pin system support connector (J1).

This RAM supports byte, word and longword references. Read operations take 700 ns to complete while write operations require 600 ns.

The RAM is organized as a 256 X 32-bit (one longword) array. The array appears in a 1KB block of the VAX I/O page at addresses 2014 0400 - 2014 07FF.

This array is not protected by parity, and CDAL bus parity is neither checked nor generated on reads or writes to this RAM.

3.7.5 KA640 Initialization

The VAX Architecture defines three kinds of hardware initialization:

1. Power-up initialization
2. Processor initialization
3. I/O bus initialization

3.7.5.1 Power-Up Initialization

Power-up initialization is the result of the restoration of power and includes a hardware reset, a processor initialization, an I/O bus initialization, as well as the initialization of several registers defined in the *VAX Architecture Reference Manual*.

3.7.5.2 Hardware Reset

A KA640 hardware reset occurs on power-up and the negation of DCOK when the processor is halted. A hardware reset causes the hardware halt procedure (Section 3.1.5.6) to be initiated with a halt code of 03. It also initializes some IPRs and most I/O page registers to a known state. Those IPRs that are affected by a module reset are noted in Section 3.1.1.3. The effect a hardware reset has on I/O space registers is documented in the description of the register.

3.7.5.3 I/O Bus Initialization

An I/O bus initialization occurs on power-up, the negation of DCOK when the processor is halted, or as the result of a MTPR to IPR 55 (IORESET) or console UNJAM command.

I/O Bus Reset Register

The I/O bus reset register (IORESET), internal processor register 55, is implemented in the SSC. A MTPR of any value to IORESET causes an I/O bus initialization.

3.7.5.4 Processor Initialization

A processor initialization occurs on power-up, the negation of DCOK when the processor is halted, as the result of a console INITIALIZE command, and after a halt caused by an error condition.

In addition to initializing those registers defined in the *VAX Architecture Reference Manual*, the KA640 firmware also configures main memory, the local I/O page, and the Q22-bus map during a processor initialization.

3.8 Q22-bus Interface

The KA640 includes a Q22-bus interface implemented via a single VLSI chip called the CQBIC. It contains a CDAL bus to Q22-bus interface that supports the following functions:

- A programmable mapping function (scatter-gather map) for translating 22-bit, Q22-bus addresses into 29-bit CDAL bus addresses that allows any page in the Q22-bus memory space to be mapped to any page in main memory.
- A direct mapping function for translating 29-bit CDAL addresses in the local Q22-bus address space and local Q22-bus I/O page into 22-bit, Q22-bus addresses.
- Masked and unmasked longword reads and writes from the CPU to the Q22-bus memory and I/O space and the Q22-bus interface registers. Longword reads and writes of the local Q22-bus memory space are buffered and translated into two-word, block mode, transfers on the Q22-bus. Longword reads and writes of the local Q22-bus I/O space are buffered and translated into two, single-word transfers on the Q22-bus.
- Up to sixteen-word, block mode, writes from the Q22-bus to main memory. These words are buffered then transferred to main memory using two asynchronous DMA octaword transfers. For block mode writes of less than sixteen words, the words are buffered and transferred to main memory using the most efficient combination of octaword,

quadword, and longword asynchronous DMA transfers. The maximum write bandwidth for block mode references is 3.3 MB per second. Block mode reads of main memory from the Q22-bus cause the Q22-bus interface to perform an asynchronous DMA quadword read of main memory and buffer all four words, so that on block mode reads, the next three words of the block mode read can be delivered without any additional CDAL bus cycles. The maximum read bandwidth for Q22-bus block mode references is 2.4 MB per second. Q22-bus burst mode DMA transfers result in single-word reads and writes of main memory.

- Transfers from the CPU to the local Q22-bus memory space, that result in the Q22-bus map translating the address back into main memory (local-miss, global-hit transactions).

The Q22-bus interface contains several registers for Q22-bus control and configuration, and error reporting.

The interface also contains Q22-bus interrupt arbitration logic that recognizes Q22-bus interrupt requests BR7-BR4 and translates them into CPU interrupts at levels 17-14.

The Q22-bus interface detects Q22-bus *no sack* timeouts, Q22-bus interrupt acknowledge timeouts, Q22-bus non-existent memory timeouts, main memory errors on DMA accesses from the Q22-bus and Q22-bus parity errors.

3.8.1 Q22-bus to Main Memory Address Translation

On DMA references to main memory, the 22-bit, Q22-bus address must be translated into a 29-bit main memory address. This translation process is performed by the Q22-bus interface by using the Q22-bus map. This map contains 8192 mapping registers, (one for each page in the Q22-bus memory space), each of which can map a page (512 bytes) of the Q22-bus memory address space into any of the 128K pages in main memory. Since local I/O space addresses cannot be mapped to Q22-bus pages, the local I/O page is inaccessible to devices on the Q22-bus.

Q22-bus addresses are translated to main memory addresses as shown in Figure 3–29.

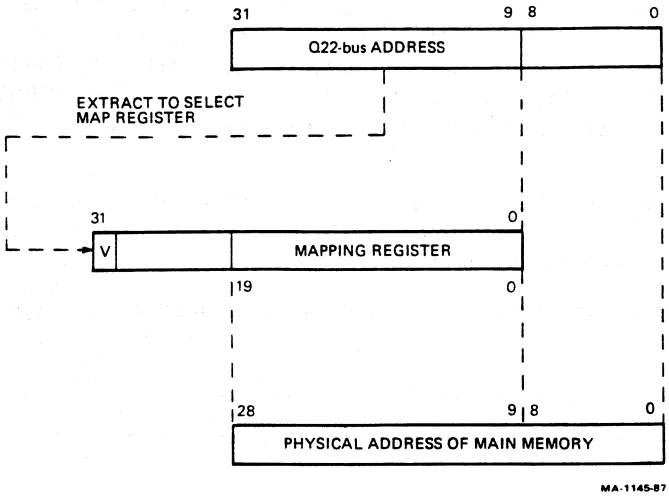


Figure 3–29 Q22-bus to Main Memory Address Translation

At power up time, the Q22-bus map registers, including the valid bits, are undefined. External access to main memory is disabled as long as the interprocessor communication register LM EAE bit is cleared. The Q22-bus interface monitors each Q22-bus cycle and responds if the following three conditions are met:

1. The interprocessor communication register LM EAE bit is set.
2. The valid bit of the selected mapping register is set.
3. During read operations, the mapping register must map into existent main memory, or a Q22-bus timeout occurs. (During write operations, the Q22-bus interface returns Q22-bus BRPLY before checking for existent local memory; the response depends only on conditions 1 and 2 above.)

NOTE

In the case of local-miss, global-hit, the state of the LM EAE bit is ignored.

If the map cache does not contain the needed Q22-bus map register, then the Q22-bus interface will perform an asychronous DMA read of the Q22-bus map register before proceeding with the Q22-bus DMA transfer.

3.8.1.1 Q22-bus Map Registers

The Q22-bus map contains 8192 registers (QMRs) that control the mapping of Q22-bus addresses into main memory. Each register maps a page of the Q22-bus memory space into a page of main memory (Table 3-16). These registers are implemented in a 32KB block of main memory, but are accessed through the CQBIC chip via a block of addresses in the I/O page.

The local I/O space address of each register was chosen so that register address bits <14:2> are identical to Q22-bus address bits <21:9> of the Q22-bus page which the register maps.

The Q22-bus map registers (QMRs) have the format shown in Figure 3-30.

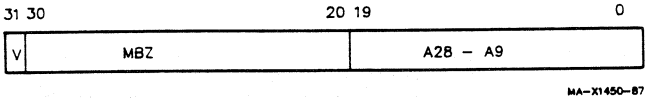


Figure 3-30 Q22-bus Map Registers

Table 3-16 Q22-bus Map

Register Address	Q22-bus Addresses Mapped (Hex)	Q22-bus Addresses Mapped (Octal)
2008 8000	00 0000 - 00 01FF	00 000 000 - 00 000 777
2008 8004	00 0200 - 00 03FF	00 001 000 - 00 001 777
2008 8008	00 0400 - 00 05FF	00 002 000 - 00 002 777
2008 800C	00 0600 - 00 07FF	00 003 000 - 00 003 777
2008 8010	00 0800 - 00 09FF	00 004 000 - 00 004 777
2008 8014	00 0A00 - 00 0BFF	00 005 000 - 00 005 777
2008 8018	00 0C00 - 00 0DFF	00 006 000 - 00 006 777
2008 801C	00 0E00 - 00 0FFF	00 007 000 - 00 007 777
.	.	.
.	.	.
.	.	.
2008 FFF0	3F F800 - 3F F9FF	17 774 000 - 17 774 777
2008 FFF4	3F FA00 - 3F FBFF	17 775 000 - 17 775 777
2008 FFF8	3F FC00 - 3F FDFF	17 776 000 - 17 776 777
2008 FFFC	3F FE00 - 3F FFFF	17 776 000 - 17 777 777

Data Bit	Definition
QMR<31>	(V) Valid. Read/Write. When a Q22-bus map register is selected by bits <21:9> of the Q22-bus address, the valid bit determines whether mapping is enabled for that Q22-bus page. If the valid bit is set, the mapping is enabled, and Q22-bus addresses within the page controlled by the register are mapped into the main memory page determined by bits <28:9>. If the valid bit is clear, the mapping register is disabled, and the Q22-bus interface does not respond to addresses within that page. This bit is *undefined* on power-up and the negation of DCOK when the processor is halted.
QMR<30:20>	Unused. These bits always read as zero and must be written as zero.
QMR<19:0>	(A28-A9) Address bits <28:9>. Read/Write. When a Q22-bus map register is selected by a Q22-bus address, and if that register's valid bit is set, then these 20 bits are used as main memory address bits <28:9>. Q22-bus address bits <8:0> are used as main memory address bits <8:0>. These bits are *undefined* on power-up and the negation of DCOK when the processor is halted.

3.8.1.2 Accessing the Q22-bus Map Registers

Although the CPU accesses the Q22-bus map registers via aligned, masked longword references to the local I/O page (addresses 2008 8000 $_{16}$ through 2008 FFFC $_{16}$), the map actually resides in a 32KB block of main memory. The starting address of this block is controlled by the contents of the Q22-bus map base register. The Q22-bus interface also contains a 16-entry, fully associative, Q22-bus map cache to reduce the number of main memory accesses required for address translation.

NOTE

The system software must protect the pages of memory that contain the Q22-bus map from direct accesses that will corrupt the map or cause the entries in the Q22-bus map cache to become stale. Either of these conditions will result in the incorrect operation of the mapping function.

When the CPU accesses the Q22-bus map through the local I/O page addresses, the Q22-bus interface reads or writes the map in main memory. The Q22-bus interface does not have to gain Q22-bus mastership when accessing the Q22-bus map. Since these addresses are in the local I/O space, they are not accessible from the Q22-bus.

On a Q22-bus map read by the CPU, the Q22-bus interface decodes the local I/O space address (2008 8000 - 2008 FFFC). If the register is in the Q22-bus map cache, the Q22-bus interface will internally resolve any conflicts between CPU and Q22-bus transactions (if both are attempting to access the Q22-bus map cache entries at the same time), then return the data. If the map register is not in the map cache, the Q22-bus interface will force the CPU to retry, acquire the CDAL bus, perform an asynchronous DMA read of the map register. On completion of the read, the CPU is provided with the data when its read operation is retried. A map read by the CPU does not cause the register that was read to be stored in the map cache.

On a Q22-bus map write by the CPU, the Q22-bus interface latches the data, then on the completion of the CPU write, acquires the CDAL bus and performs an asynchronous DMA write to the map register. If the map register is in the Q22-bus map cache, then the Cam Valid bit for that entry will be cleared to prevent the entry from becoming stale. A Q22-bus map write by the CPU does not update any cached copies of the Q22-bus map register.

3.8.1.3 Q22-bus Map Cache

To speed up the process of translating Q22-bus address to main memory addresses, the Q22-bus interface utilizes a fully associative, sixteen entry, Q22-bus map cache, which is implemented in the CQBIC chip.

If a DMA transfer ends on a page boundary, the Q22-bus interface will prefetch the mapping register required to translate the next page and load it into the cache, before starting a new DMA transfer. This allows Q22-bus block mode DMA transfers that cross page boundaries to proceed without delay. The replacement algorithm for updating the Q22-bus map cache is FIFO.

The cached copy of the Q22-bus map register is used for the address translation process. If the required map entry for a Q22-bus address (as determined by bits <21:9> of the Q22-bus address) is not in the map cache, then the Q22-bus interface uses the contents of the map base register to access main memory and retrieve the required entry. After obtaining the entry from main memory, the valid bit is checked. If it is set, the entry is stored in the cache and the Q22-bus cycle continues.

The format of a Q22-bus map cache entry is as shown in Figure 3–31.

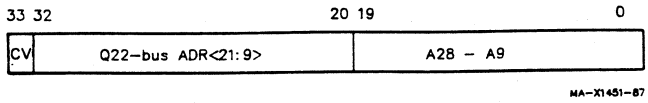


Figure 3–31 Q22-bus Map Cache Entry

Data Bit	Definition
CQMR<33>	(Cam Valid). When a mapping register is selected by a Q22-bus address, the Cam Valid bit determines whether the cached copy of the mapping register for that address is valid. If the Cam Valid bit is set, the mapping register is enabled, and addresses within that page can be mapped. If the Cam Valid bit is clear, the Q22-bus interface must read the map in local memory to determine if the mapping register is enabled. This bit is cleared on power-up, the negation of DCOK when the processor is halted, by setting the QMCIA (Q22-bus map cache invalidate all) bit in the interprocessor communication register, on writes to IPR 55 (IORESET), by a write to the Q22-bus map base register, or by writing to the QMR that is being cached.
CQMR<32:20>	(QBUS ADR). These bits contain the Q22-bus address bits <21:9> of the page that this entry maps. This is the content addressable field of the 16 entry cache for determining if the map register for a particular Q22-bus address is in the map cache. These bits are *undefined* on power-up.
CQMR<19:0>	(Address bits A28-A9). When a mapping register is selected by a Q22-bus address, and if that register's Cam Valid bit is set, then these 20 bits are used as main memory address bits 28 through 9. Q22-bus address bits 8 through 0 are used as local memory address bits 8 through 0. These bits are *undefined* on power-up.

3.8.2 CDAL Bus to Q22-bus Address Translation

CDAL bus addresses within the local Q22-bus I/O space, addresses 2000 0000 - 2000 1FFF $_{16}$, are translated into Q22-bus I/O space addresses by using bits <12:0> of the CDAL address as bits <12:0> of the Q22-bus address and asserting BBS7. Q22-bus address bits <21:13> are driven as zeros.

CDAL bus addresses within the local Q22-bus memory space, addresses 3000 0000 - 303F FFFF $_{16}$, are translated into Q22-bus memory space addresses by using bits <21:0> of the CDAL address as bits <21:0> of the Q22-bus address.

3.8.3 Interprocessor Communication Register

The interprocessor communication register (IPCR), address 2000 1F40 $_{16}$, is a 16-bit register which resides in the Q22-bus I/O page address space and can be accessed by any device which can become Q22-bus master (including the KA640 itself). The IPCR, implemented in the CQBIC chip, is byte accessible, meaning that a write byte instruction can write to either the low or high byte without affecting the other byte.

The IPCR also appears at Q22-bus address 17 777 500 (Figure 3-32).

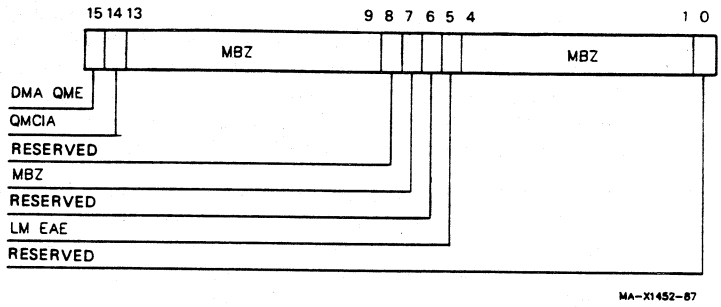


Figure 3-32 The Interprocessor Communication Register

Data Bit	Definition
IPCR<15>	(DMA QME) DMA Q22-bus address space memory error. Read/Write to clear. This bit indicates that an error occurred when a Q22-bus device was attempting to read main memory. It is set if DMA system error register bit DSER<4> (main memory error) is set, or the CDAL bus timer expires. The main memory error bit indicates that an uncorrectable error occurred when an external device (or CPU) was accessing the KA640 local memory. The CDAL bus timer expiring indicates that the memory controller did not respond when the Q22-bus interface initiated a DMA transfer. This bit is cleared by writing a 1 to it, on power-up, by the negation of DCOK when the processor is halted, by writes to IPR 55 (IORESET), and whenever DSER<4> is cleared.
IPCR<14>	(QMCIA) Q22-bus invalidate all. Write only. Writing a 1 to this bit clears the Cam Valid bits in the cached copy of the map. This bit always reads as zero. Writing a 0 has no effect.
IPCR<13:9>	(Unused) Read as zeros. Must be written as zeros.
IPCR<8>	Reserved for Digital use.
IPCR<7>	Unused. Read as zero. Must be written as zero.
IPCR<6>	Reserved for Digital use.
IPCR<5>	(LM EAE) Local memory external access enable. Read/Write when the KA640 is Q22-bus master. Read only when another device is Q22-bus master. When set, this bit enables external access to local memory (via the Q22-bus map). Cleared on power-up and by the negation of DCOK when the processor is halted.
IPCR<4:1>	Unused. Read as zeros. Must be written as zeros.
IPCR<0>	Reserved for Digital use.

3.8.4 Q22-bus Interrupt Handling

The KA640 responds to interrupt requests BR7-4 with the standard Q22-bus interrupt acknowledge protocol (DIN followed by IAK). The console serial line unit, the programmable timers, and the interprocessor doorbell request interrupts at IPL 14 and have priority over all Q22-bus BR4 interrupt requests. After responding to any interrupt request BR7-4, the CPU sets the processor priority to IPL 17. All BR7-4 interrupt requests are disabled unless software lowers the interrupt priority level.

Interrupt requests from the KA640 interval timer are handled directly by the CPU. Interval timer interrupt requests have a higher priority than BR6 interrupt requests. After responding to an interval timer interrupt request, the CPU sets the processor priority to IPL 16. Thus, BR7 interrupt requests remain enabled.

3.8.5 Configuring the Q22-bus Map

The KA640 implements the Q22-bus map in an 8K longword (32KB) block of main memory. This map must be configured by the KA640 firmware during a processor initialization by writing the base address of the uppermost 32KB block of good main memory into the Q22-bus map base register. The base of this map must be located on a 32KB boundary.

NOTE

This 32KB block of main memory must be protected by the system software. The only access to the map should be through local I/O page addresses 2008 8000 - 2008 FFFC $_{16}$.

3.8.5.1 Q22-bus Map Base Address Register

The Q22-bus map base address register (QBMBR), address 2008 0010 $_{16}$, controls the main memory location in of the 32KB block of Q22-bus map registers.

This read/write register is accessible by the CPU on a longword boundary only. Bits <31:29,14:0> are unused and should be written as zero and will return zero when read.

A write to the map base register will flush the Q22-bus map cache by clearing the Cam Valid bits in all the entries.

The contents of this register are *undefined* on power up and the negation of DCOK when the processor is halted, and are not affected by BINIT being asserted on the Q22-bus (Figure 3-33).

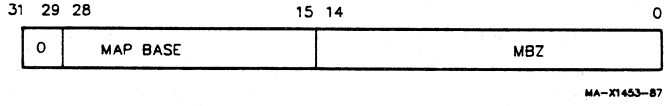


Figure 3-33 Q22-bus Map Base Address Register

3.8.6 System Configuration Register

The system configuration register (SCR), address 2008 0000 $_{16}$, contains a BHALT enable bit and a power OK flag.

The system configuration register (SCR) is longword, word, and byte accessible. Programmable option fields are cleared on power-up and by the negation of DCOK when the processor is halted. The format of the SCR register is shown in Figure 3-34.

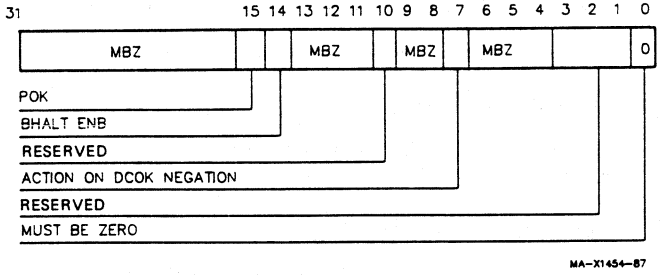


Figure 3-34 System Configuration Register

Data Bit	Definition
SCR<31:16>	Unused. Read as zero. Must be written as zero.
SCR<15>	(POK) Power OK. Read only. Writes have no effect. This bit is set if the Q22-bus BPOK signal is asserted and clear if it is negated. This bit is cleared on power-up and by the negation of DCOK when the processor is halted.
SCR<14>	(BHALT EN) BHALT enable. Read/Write. This bit controls the effect the Q22-bus BHALT signal has on the CPU. When set, asserting the Q22-bus BHALT signal will halt the CPU and assert DSER<15>. When cleared, the Q22-bus BHALT signal will have no effect. This bit is cleared on power-up and by the negation of DCOK when the processor is halted.
SCR<13:11>	Unused. Read as zero. Must be written as zero.
SCR<10>	Reserved for Digital use.
SCR<9:8>	Unused. Read as zero. Must be written as zero.

Data Bit	Definition
SCR<7>	(ACTION ON DCOK NEGATION) Read/Write. When cleared, the Q22-bus interface asserts SYSRESET (causing a hardware reset of the board and control to be passed to the resident firmware via the hardware halt procedure with a halt code of 3) when DCOK is negated on the Q22-bus. When set, the Q22-bus interface asserts HALTIN (causing control to be passed to the resident firmware via the hardware halt procedure with a halt code of 2) when DCOK is negated on the Q22-bus. Cleared on power-up and the negation of DCOK when the processor is halted.
SCR<6:4>	Unused. Read as zero. Must be written as zero.
SCR<3:1>	Reserved for Digital use.
SCR<0>	Unused. Read as 0. Must be written as 0.

3.8.7 DMA System Error Register

The DMA system error register (DSER), address 2008 0004 $_{16}$, is one of three registers associated with Q22-bus Interface error reporting. These registers are located in the local VAX I/O address space and can only be accessed by the local processor.

The DMA system error register is implemented in the CQBIC chip, and logs main memory errors on DMA transfers, Q22-bus parity errors, Q22-bus non-existent memory errors, and Q22-bus no-grant errors. The Q22-bus error address register contains the address of the page in Q22-bus space which caused a parity error during an access by the local processor. The DMA error address register contains the address of the page in local memory which caused a memory error during an access by an external device or the processor during a local miss global hit transaction. An access by the local processor which the Q22-bus interface maps into main memory will provide error status to the processor when the processor does a RETRY for a READ local miss-global hit, or by an interrupt in the case of a local-miss global-hit write.

The DSER is a longword, word, or byte accessable read/write register available to the local processor. The bits in this register are cleared to 0 on power-up, by the negation of DCOK when the processor is halted, and by writes to IPR 55 (IORESET). All bits are set to 1 to record the occurrence of an event. They are cleared by writing a 1, writing zeros has no effect (Figure 3-35).

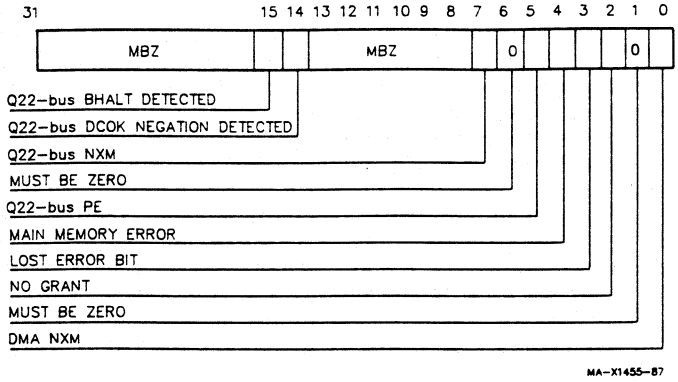


Figure 3-35 DMA System Error Register

Data Bit	Definition
DSER<31:16>	Unused. Read as 0. Must be written as 0.
DSER<15>	Q22-bus BHALT detected. Read/Write to clear. Set when the Q22-bus interface detects that the Q22-bus BHALT line was asserted and SCR<14> (BHALT ENABLE) is set. Cleared by writing a 1, writes to IPR 55 (IORESET), on power-up and the negation of DCOK when the processor is halted.
DSER<14>	Q22-bus DCOK negation detected. Read/Write to clear. Set when the Q22-bus interface detects the negation of DCOK on the Q22-bus and SCR<7> (action on DCOK negation) is set. Cleared by writing a 1, writes to IPR 55 (IORESET), on power-up and the negation of DCOK when the processor is halted.
DSER<13:8>	Unused. Read as zero. Must be written as zero.
DSER<7>	Master DMA NXM. Read/Write to clear. This bit is set when the CPU performs a demand Q22-bus read cycle or write cycle that does not reply after 10 μs. During interrupt acknowledge cycles, or request read cycles, this bit is not set. It is cleared by writing a 1, on power-up, by the negation of DCOK when the processor is halted and by writes to IPR 55 (IORESET).
DSER<6>	Unused. Read as zero. Must be written as zero.

Data Bit	Definition
DSER<5>	Q22-bus parity error. Read/Write to clear. This bit is set when the CPU performs a Q22-bus demand read cycle which returns a parity error. During interrupt acknowledge cycles or request read cycles, this bit is not set. It is cleared by writing a 1, on power-up, by the negation of DCOK when the processor is halted and by writes to IPR 55 (IORESET).
DSER<4>	Main memory error. Read/Write to clear. This bit is set if an external Q22-bus device or local miss global hit receives a memory error while reading local memory. The IPCR<15> reports the memory error to the external Q22-bus device. It is cleared by writing a 1, on power-up, by the negation of DCOK when the processor is halted and by writes to IPR 55 (IORESET).
DSER<3>	Lost error. Read/Write to clear. This bit indicates that an error address has been lost because of DSER<7,5,4,0> having been previously set and a subsequent error of either type occurs which would have normally captured an address and set either DSER<7,5,4,0> flag. It is cleared by writing a 1, on power-up, by the negation of DCOK when the processor is halted and by writes to IPR 55 (IORESET).
DSER<2>	No grant timeout. Read/Write to clear. This bit is set if the Q22-bus does not return a bus grant within 10ms of the bus request from a CPU demand read cycle, or write cycle. During interrupt acknowledge or request read cycles this bit is not set. It is cleared by writing a 1, on power-up, by the negation of DCOK when the processor is halted and by writes to IPR 55 (IORESET).
DSER<1>	Unused. Read as zero. Must be written as zero.
DSER<0>	DMA NXM. Read/Write to clear. This bit is set on a DMA transfer to a non-existent main memory location. This includes local-miss global-hit cycles and map accesses to non-existent memory. It is cleared by writing a 1, on power-up, by the negation of DCOK when the processor is halted and by writes to IPR 55 (IORESET).

3.8.8 Q22-bus Error Address Register

The Q22-bus error address register (QBEAR), address 2008 0008 $_{16}$, is a read only, longword accessible register which is implemented in the CQBIC chip. Its contents are valid only if DSER <5> (Q22-bus parity error) is set or if DSER<7> (Q22-bus timeout) is set.

Reading this register when DSER<5> and DSER<7> are clear will return *undefined* results. Additional Q22-bus parity errors that could have set DSER<5> or Q22-bus timeout errors that could have caused DSER<7> to set, will cause DSER<3> to set.

The QBEAR contains the address of the page in Q22-bus space which caused a parity error during an access by the on-board CPU which set DSER<5> or a master timeout which set DSER<7>.

Q22-bus address bits <21:9> are loaded into QBEAR bits <12:0>. QBEAR bits <31:13> always read as zeros (Figure 3-36).

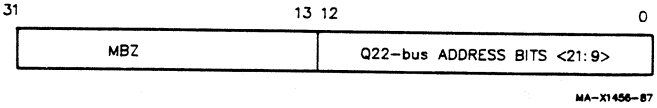


Figure 3-36 Q22-bus Error Address Register

NOTE

This is a read only register, if a write is attempted a machine check will be generated.

3.8.9 DMA Error Address Register

The DMA error address register (DEAR), address 2008 000C $_{16}$, is a read only, longword accessible register which is implemented in the CQBIC chip. It contains valid information only when DSER<4> (main memory error) is set or when DSER<0> (DMA NXM) is set. Reading this register when DSER<4> and DSER<0> are clear will return *undefined* data.

The DEAR contains the map translated address of the page in local memory which caused a memory error or non existent memory error during an access by an external device or the Q22-bus interface for the CPU during a local-miss global-hit transaction or Q22-bus map access.

The contents of this register are latched when DSER<4> or DSER<0> is set. Additional main memory errors or non-existent memory errors have no effect on the DEAR until software clears DSER<4> and DSER<0>.

Mapped Q22-bus address bits <28:9> are loaded into DEAR bits <19:0>. DEAR bits <31:20> always read as zeros (Figure 3-37).

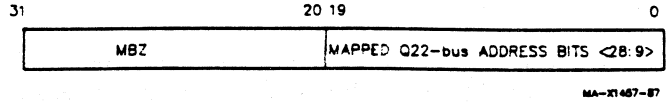


Figure 3-37 DMA Error Address Register

NOTE

This is a read only register, if a write is attempted a machine check will be generated.

3.8.10 Error Handling

The Q22-bus interface does not generate or check CDAL bus parity.

The Q22-bus interface checks all CPU references to Q22-bus memory and I/O spaces to insure that nothing but masked and unmasked longword accesses are attempted. Any other type of reference will cause a machine check abort to be initiated.

The Q22-bus interface maintains several timers to prevent incomplete accesses from hanging the system indefinitely. These include a 10 μs non-existent memory timer for accesses to the Q22-bus memory and I/O spaces, a 10 μs *no sack* timer for acknowledgement of Q22-bus DMA grants, and a 10 ms *no grant* timer for acquiring the Q22-bus.

If there is a non-existent memory (NXM) error (10 μs timeout) while accessing the Q22-bus on a demand read reference, the associated row in the cache is invalidated, DSER<7> is set, the address of the Q22-bus page being accessed is captured in QBEAR<12:0>, and a machine check abort is initiated.

If there is a NXM error on a prefetch read, or an interrupt acknowledge vector read, then the prefetch or interrupt acknowledge reference is aborted but no information is captured and no machine check occurs.

If there is a NXM error on a masked write reference, then DSER<7> is set, the address of the Q22-bus page being accessed is captured in QBEAR<12:0>, and an interrupt is generated at IPL 1D through vector 60_{16}.

If the Q22-bus interface does not receive an acknowledgement within 10 μs after it has granted the Q22-bus, then the grant is withdrawn, no errors are reported, and the Q22-bus interface waits 500 ns to clear the Q22-bus grant daisy chain before beginning arbitration again.

If the Q22-bus interface tries to obtain Q22-bus mastership on a CPU demand read reference and does not obtain it within 10 ms, then the associated row in the cache is invalidated, DSER<2> is set, and a machine check abort is initiated.

The Q22-bus interface also monitors Q22-bus signals BDAL<17:16> while reading information over the Q22-bus so that parity errors detected by the device being read from are recognized.

If a parity error is detected by another Q22-bus device on a CPU demand read reference to Q22-bus memory or I/O space, then the associated row in the cache is invalidated, DSER<5> is set, the address of the Q22-bus page being accessed is captured in QBEAR<12:0>, and a machine check abort is initiated.

If a parity error is detected by another Q22-bus device on a prefetch request read by the CPU, the prefetch is aborted, the associated row in the cache is invalidated, DSER<5> is set, the address of the Q22-bus page being accessed is captured in QBEAR<12:0>, but no machine check is generated.

The Q22-bus interface also monitors the backplane BPOK signal to detect power failures. If BPOK is negated on the Q22-bus, a power fail trap is generated, and the CPU traps through vector $0C_{16}$. The state of the Q22-bus BPOK signal can be read from SCR<15>. The Q22-bus interface continues to operate after generating the powerfail trap, until DCOK is negated.

3.9 Network Interface

The KA640 includes a network interface that is implemented via the LANCE chip, a 32 by 8 bit ROM and two 32K x 8 static RAMs. When used in conjunction with the H3602-SA, this interface allows the KA640 to be connected to either a thinwire or standard Ethernet network. It supports the Ethernet data link layer.

The network interface includes a word-wide 64KB NI buffer (the two 32 x 8 static RAMs) as well as four registers for control and status reporting, a DMA controller, a 24 word transmit silo and a 24 word receive silo (all resident in the LANCE chip). The DMA controller reads control information and writes status information from/to the 64 KB NI buffer as well as transfers data between the NI buffer and either the transmit or receive silo. The DMA controller can perform bursts of up to eight longword references.

Each reference (between the LANCE and the NI buffer) takes 600 ns and contains either a byte or word of data, resulting in a maximum burst duration of 4.8 μs. The minimum time between bus requests is 8 μs.

The CPU moves data between main memory and the NI buffer via programmed transfers.

3.9.1 Ethernet Overview

Ethernet is a serial bus that can support up to 1,024 nodes with a maximum separation of 2.8 kilometers (1.7 miles). Data is passed over the Ethernet in Manchester encoded format at a rate of 10 million bits per second in variable-length packets. Each packet has the format shown in Figure 3-38.

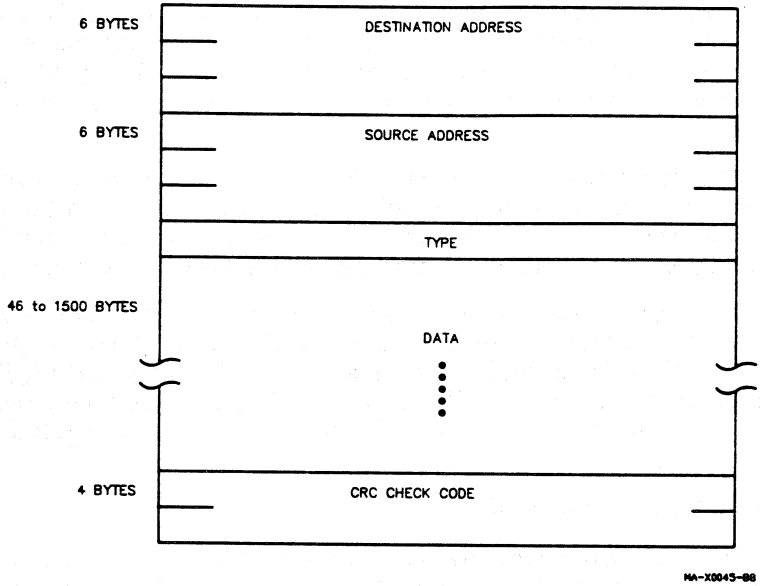


Figure 3-38 Ethernet Data Packet Format

The minimum size of a packet is 64 bytes, which implies a minimum data length of 46 bytes. Packets shorter than this are called *runt packets* and are treated as erroneous when received by the network controller.

All nodes on the Ethernet have equal priority. The technique used to control access to the bus is carrier sense, multiple access, with collision detection (CSMA/CD). To access the bus, devices must first wait for the bus to clear (no carrier sensed). Once the bus is clear, all devices that want to access the bus have equal priority (multi-access), so they all attempt to transmit. After starting transmission, devices must monitor the bus for collisions (collision detection). If no collision is detected, the device may continue with transmission. If a collision is detected, then the device waits for a random amount of time and repeats the access sequence.

Ethernet allows point to point communication between two devices, as well as simultaneous communication between multiple devices. To support these two modes of communication, there are two types of network addresses, **physical** and **multicast**. These two types of addresses are both 48 bits (6 bytes) long and are described below.

A **Physical address** is the unique address associated with a particular station on the Ethernet, which should be distinct from the physical address of any other station on any other Ethernet.

A **Multicast address** is a multi-destination address associated with one or more stations on a given Ethernet (sometimes called a logical address).

Further, there are two kinds of multicast addresses, the *Multicast-group address* and the *Broadcast address*.

The *Multicast-group address* is an address associated by higher-level convention with a group of logically related stations.

The *Broadcast address* is a predefined multicast address which denotes the set of all the stations on the Ethernet.

Bit 0 (the least significant bit of the first byte) of an address denotes the type: it is 0 for physical addresses and 1 for multicast addresses. In either case the remaining 47 bits form the address value. A value of 48 ones is always treated as the broadcast address.

The hardware address of the KA640 module is determined at the time of manufacture and is stored in the network interface station address (NISA) ROM. Since every device that is intended to connect to an Ethernet network must have a unique physical address, the bit pattern blasted into the NISA ROM must be unique for each KA640. The multicast addresses to which the KA640 will respond are determined by the multicast address filter mask in the network interface initialization block.

3.9.2 Network Interface Station Address ROM

The network interface includes a byte-wide, 32-byte, socketed ROM called the network interface station address ROM (NISA ROM). One byte of this ROM appears in the low-order byte of each of 32 consecutive longwords in the address range 2008 4200 - 2008 427C $_{16}$. Bytes two and three of each longword are *undefined*. The low-order byte of the first six longwords contain the 48-bit network physical address (NPA) of the KA640. The low-order byte in the remaining 26 longwords are unused. This address range is read only. Writes to this address range will result in a CDAL bus timeout and a machine check 83. The format for the NISA ROM is shown in Figure 3-39.

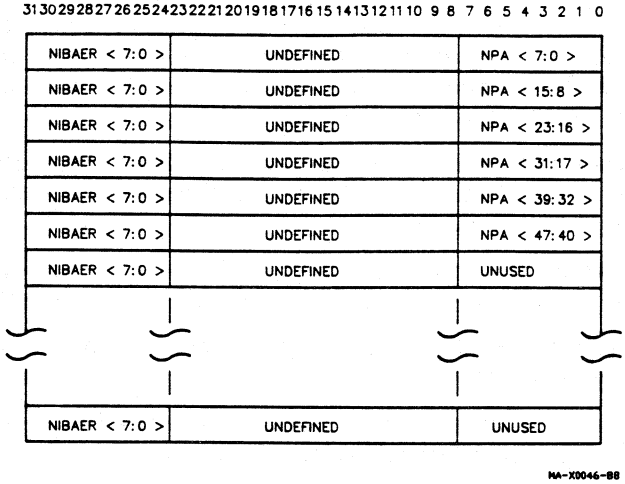


Figure 3-39 Network Interface Station Address ROM Format

3.9.3 LANCE Chip Overview

The LANCE chip is a microprogrammed controller which can conduct extensive operations independently of the central processor. There are four control and status registers (CSRs) within the LANCE chip which are programmed by the central processor (i.e. the MicroVAX CPU chip) to initialize the LANCE chip and start its independent operation. Once started, the LANCE uses its built-in DMA controller to directly access NI buffer RAM to get additional operating parameters and to manage the buffers it uses to transfer packets to and from the Ethernet. The LANCE uses three structures in the NI buffer:

1. **Network Interface Initialization Block**—24 bytes of contiguous memory starting on a word boundary. The initialization block is set up by the central processor and is read by the LANCE when the processor starts the LANCE's initialization process. The initialization block contains the system's network address and pointers to the receive and transmit descriptor rings.

2. **Descriptor Rings**—two logically circular rings of buffer descriptors, one ring used by the chip receiver for incoming data (the network interface receive descriptor ring) and one ring used by the chip transmitter for outgoing data (the network interface transmit descriptor ring). Each buffer descriptor in a ring is 8 bytes long and starts on a quadword boundary. It points to a data buffer elsewhere in memory, contains

the size of that buffer, and holds various status information about the buffer's contents.

3. **Data Buffers**—contiguous portions of memory to buffer incoming packets (Receive Data Buffers) or outgoing packets (transmit data buffers). Data buffers must be at least 64 bytes long (100 bytes for the first buffer of a packet to be transmitted) and may begin on any byte boundary.

When the system is ready to begin network operation, the central processor sets up the network interface initialization block, the network interface receive descriptor ring, the network interface transmit descriptor ring, and their data buffers in the NI buffer RAM, and then starts the LANCE by writing to its CSRs. The LANCE performs its initialization process and then enters its polling loop. In this loop, it listens to the network for packets whose destination addresses are of interest and it scans the network interface transmit descriptor ring for descriptors which have been marked by the central processor to indicate that they contain outgoing data packets.

When the LANCE detects a network packet of interest, it receives and stores that packet in one or more receive buffers and marks their descriptors accordingly. When the LANCE finds a packet to be transmitted, it transmits the packet to the network and marks its descriptor when transmission is complete. Whenever the LANCE chip completes a reception or transmission (or encounters an error condition), it sets flags in NICSR0 to signal the central processor (usually by an interrupt) that it has done something of interest.

3.9.4 Network Interface Register Address Port

The network interface register address port (NIRAP), address 2008 4404 $_{16}$, is a word-wide register that is implemented on all designs that use the LANCE chip. It is used to select which of the four CSRs is accessed via the network interface register data port. The format for the NIRAP is shown in Figure 3-40.

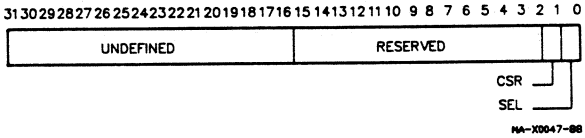


Figure 3-40 Network Interface Register Address Port

Data Bit	Definition
NIRAP <31:16>	Undefined. Should not be read or written.
NIRAP <15:2>	Reserved. Ignored on write; read as zeros.
NIRAP <1:0>	(CSRSEL) CSR select <1:0>. Read/Write. These bits select which of the four control and status registers (NICSR0-NICSR3) is accessible via the register data port. Cleared on power-up, by writing NICSR0 <2>, and by the negation of DCOK when SCR <7> is clear. Values are as follows:

Bits 1:0	Register
0 0	NICSR0
0 1	NICSR1
1 0	NICSR2
1 1	NICSR3

3.9.5 Network Interface Register Data Port

The network interface register data port (NIRDP), address 2008 4400 $_{16}$, is a word-wide register that is implemented on all designs that use the LANCE chip. It is used as a 16-bit window through which the CPU can read and write the control and status register (NICSR0-NICSR3) designated by the NIRAP.

Note that registers NICSR1, NICSR2, and NICSR3 are accessible only while NICSR0 <2>(STOP) is set. If NICSR0 <2> is clear (i.e., the LANCE chip is active), attempts to read from those registers will return *undefined* data and attempts to write to them will be ignored. Accesses to a command and status register via the NIRDP do not alter the contents of the NIRAP. In normal operation only NICSR0 can be accessed, so the NIRAP should be configured so that NICSR0 is accessible through the NIRDP and left that way.

3.9.6 Network Interface Control and Status Register 0

The network interface control and status register 0 (NICSR0), address 2008 4600 $_{16}$ when NIRAP <1:0> are set to 00, is a word-wide register that is implemented on all designs that use the LANCE chip. This register is used to start and stop the operation of the LANCE chip and to monitor its status. All of its bits can be read at any time and none of its bits are affected by reading the register. The effects of a write operation are described individually for each bit. When power is applied to the system, all the bits in this register are cleared except the STOP bit which is set. The format for NICSR0 is shown in Figure 3–41.

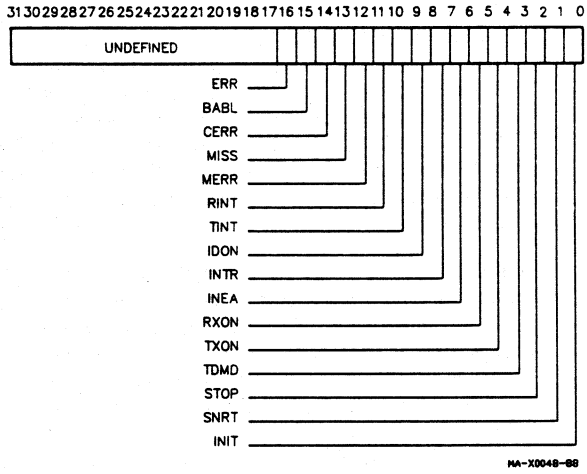


Figure 3–41 Network Interface Control and Status Register

Data Bit	Definition
NICSR0 <31:16>	Undefined. Should not be read or written.
NICSR0 <15>	(ERR) Error summary. Read only. Writes have no effect. This bit is set whenever NICSR0 <14> (BABL), NICSR0 <13>(CERR), NICSR0 <12> (MISS), or NICSR0 <11> (MERR) are set. Cleared by clearing BABL, CERR, MISS and MERR, by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <14>	(BABL) Transmitter timeout error. Read/Write to clear. This bit is set when the transmitter has been on the channel longer than the time required to send the maximum length packet. It will be set after 1519 data bytes have been transmitted (the LANCE will continue to transmit until the whole packet is transmitted or there is a failure). When this bit is set, NICSR0 <15> (ERR) and NICSR0 <7> (INTR) will also be set. Writing a 0 has no effect. Cleared by writing a 1, by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <13>	(CERR) Collision error. Read/Write to clear. This bit is set when the collision input to the LANCE chip failed to activate within 2 microseconds after a LANCE initiated transmission is completed. This *collision after transmission* is a transceiver test feature. This function is also known as heartbeat or signal quality error test (SQE). When this bit is set, NICSR0 <15> is also set. Writing a 0 has no effect. Cleared by writing a 1, by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <12>	(MISS) Missed packet. Read/Write to clear. This bit is set when the receiver loses a packet because it does not own a receive buffer. The MISS bit is not valid in internal loopback mode. When this bit is set, NICSR0 <15> and NICSR0 <7> bits are also set. Writing a 0 has no effect. Cleared by writing a 1, by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.

Data Bit	Definition
NICSR0 <11>	(MERR) Memory error. Read/Write to clear. This bit is set when the LANCE chip attempts a DMA transfer and does not receive a ready response from the network interface buffer RAM within 25.6 microseconds after beginning the memory cycle. When MERR is set, the receiver and transmitter are turned off (NICSR0 <5:4> cleared). When this bit is set, NICSR0 <15> and NICSR0 <7> bits are also set. Writing a 0 has no effect. Cleared by writing a 1, by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <10>	(RINT) Receive interrupt. Read/Write to clear. This bit is set when the LANCE chip updates an entry in the receive descriptor ring for the last buffer received or when reception is stopped due to a failure. When this bit is set, NICSR0 <7> is also set. Writing a 0 has no effect. Cleared by writing a 1, by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <9>	(TINT) Transmitter interrupt. Read/Write to clear. This bit is set when the LANCE chip updates an entry in the transmit descriptor ring for the last buffer sent or when transmission is stopped due to a failure. When this bit is set, NICSR0 <7> is also set. Writing a 0 has no effect. Cleared by writing a 1, by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <8>	(IDON) Initialization done. Read/Write to clear. This bit is set when the LANCE chip completes the initialization process which was started by setting NICSR0 <0> (INIT). When IDON is set, the LANCE chip has read the initialization block from memory and stored the new parameters. When this bit is set, NICSR0 <7> is also set. Writing a 0 has no effect. Cleared by writing a 1, by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <7>	(INTR) Interrupt request. Read only. This bit is set whenever any of the bits NICSR0 <14> (BABL), NICSR0 <12> (MISS), NICSR0 <11> (MERR), NICSR0 <10> (RINT), NICSR0 <9> (TINT), or NICSR0 <8> (IDON) are set. When both this bit and NICSR0 <6> (INEA) are set, an interrupt request is posted at IPL 14 with vector offset of $D4_{16}$. Writing to this bit has no effect. Cleared by clearing BABL, MISS, MERR, RINT, TINT, and IDON, by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.

Data Bit	Definition
NICSR0 <6>	(INEA) Interrupt enable. Read/Write. This bit controls whether the setting of the NICSR0 <7> (INTR) bit generates an interrupt request. When both this bit and NICSR0 <6> (INEA) are set, an interrupt request is posted at IPL 14 with vector offset of $D4_{16}$. Cleared by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <5>	(RXON) Receiver on. Read only. When set, this bit indicates that the receiver is enabled. This bit is set when initialization is completed (i.e., when IDON is set, unless the DRX bit of the initialization block mode register was one) and then NICSR0 <1> (STRT) is set. Writing to this bit has no effect. Cleared by setting NICSR0 <2> or NICSR0 <11>, on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <4>	(TXON) Transmitter on. Read only. When set, this bit indicates that the transmitter is enabled. This bit is set when initialization is completed (i.e., when IDON is set, unless the DTX bit of the initialization block mode register was one) and then NICSR0 <1> (STRT) is set. Writing to this bit has no effect. Cleared by setting NICSR0 <2>, NICSR0 <11>, NITMD2 <31> (UFLO), NITMD2 <30> (BUFF), or NITMD2 <26> (RTRY), on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <3>	(TDMD) Transmit demand. Read/Write. Setting this bit signals the LANCE chip to access the transmit descriptor ring without waiting for the polltime interval to elapse. This bit need not be set to transmit a packet; setting it merely hastens the chip's response to the insertion of a transmit descriptor ring entry by the host program. This bit is cleared by the LANCE chip when it recognizes the bit has been set (the bit may read as one for a short time after it is set, depending upon the level of activity in the LANCE chip). Writing a zero has no effect. This bit is also cleared by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.

Data Bit	Definition
NICSR0 <2>	(STOP) Stop external activity. Read/Write. Setting this bit stops all external activity and clears the internal logic of the LANCE chip; this has the same effect on the LANCE chip as a hardware reset does. When set, the LANCE chip remains inactive until NICSR0 <1> (STRT) or NICSR0 <0> (INIT) are set. Setting STOP clears all the other bits in this register. After STOP has been set, the other three command and status registers (NICSR1, NICSR2, and NICSR3) must be reloaded before setting INIT or STRT (note that NICSR1, NICSR2, and NICSR3 may be accessed *only* while STOP is set). If the processor attempts to set STOP, INIT, and STRT at the same time, STOP takes precedence and neither STRT nor INIT is set. Writing zero has no effect. This bit is set on power-up and the negation of DCOK when SCR <7> is clear. It is cleared by setting either INIT or STRT.
NICSR0 <1>	(STRT) Start operation. Read/Write. Setting this bit enables the LANCE chip to send and receive packets, perform DMA and manage its buffers. The STOP bit must be set prior to setting the STRT bit (setting STRT then clears STOP). Writing a 0 has no effect. Cleared by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.
NICSR0 <0>	(INIT) Initialize. Read/Write. Setting this bit causes the LANCE chip to perform its initialization process, which reads the initialization block from the area in the network interface buffer RAM addressed by the contents of NICSR1 and NICSR2 via DMA. The STOP bit must be set prior to setting the INIT bit (setting INIT then clears STOP). Writing a zero has no effect. Cleared by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear.

NOTE

The INIT and STRT bits must not be set at the same time. The proper initialization procedure is as follows:

- **Set STOP in NICSR0**
- **Set up the initialization block in memory**
- **Load NICSR1 and NICSR2 with the starting address of the initialization block**
- **Set INIT in NICSR0**
- **Wait for IDON in NICSR0 to become set**
- **Set STRT in NICSR0 to begin operation**

3.9.7 Network Interface Control and Status Register 1

The network interface control and status register 1 (NICSR1), address 2008 4604 $_{16}$ when NIRAP <1:0> are set to 01, is a word-wide register that is implemented on all designs that use the LANCE chip. This register is used in conjunction with NICSR2 to supply the network interface buffer RAM address of the initialization block which the chip reads when it performs its initialization process. This register is accessible only if the STOP bit in NICSR0 is set. Bits <31:16> are undefined and bits <15:0> are read/write. On power-up, all the bits in this register are *undefined.*

The format NICSR1 is shown in Figure 3–42.

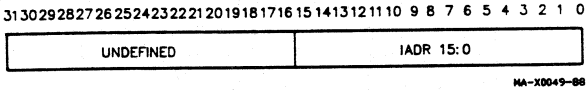


Figure 3–42 Network Interface Control and Status Register

Data Bit	Definition
NICSR1 <31:16>	Undefined. Should not be read or written.
NICSR1 <15:0>	(IADR15:0) Initialization block address bits <15:0>. Read/Write. These are the low-order sixteen bits of the network interface buffer RAM address of the first byte of the initialization block. Note that since the block must be aligned on a word boundary bit <0> must be zero.

3.9.8 Network Interface Control and Status Register 2

The network interface control and status register 2 (NICSR2), address 2008 4608 $_{16}$ when NIRAP <1:0> are set to 10, is a word-wide register that is implemented on all designs that use the LANCE chip. This register is used in conjunction with NICSR1 to supply the network interface buffer RAM address of the initialization block which the chip reads when it performs its initialization process. This register is accessible only if the STOP bit in NICSR0 is set. Bits <31:16> are undefined bits <15:8> are reserved and bits <15:0> are read/write. On power-up, all the bits in this register are *undefined.*

The format for NICSR2 is shown in Figure 3–43.

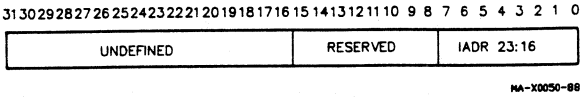


Figure 3–43 Network Interface Control and Status Register 2

Data Bit	Definition
NICSR2 <31:16>	Undefined. Should not be read or written.
NICSR2 <15:8>	Reserved. Should not be read or written.
NICSR2 <7:0>	(IADR23:16) Initialization block address bits <23:16>. Read/Write. These are bits 23:16 of the network interface buffer RAM address of the first byte of the initialization block.

3.9.9 Network Interface Control and Status Register 3

The network interface control and status register 3 (NICSR3), address 2008 460C $_{16}$ when NIRAP <1:0> are set to 11, is a word-wide register that is implemented on all designs that use the LANCE chip. This register controls certain aspects of the electrical interface between the LANCE chip and the system. It must be set by the on-board firmware as indicated for each bit. This register is accessible only if the STOP bit in NICSR0 is set. Bits <31:16> are undefined, bits <15:3> are reserved and bits <3:0> are read/write.

The format for NICSR3 is shown in Figure 3–44.

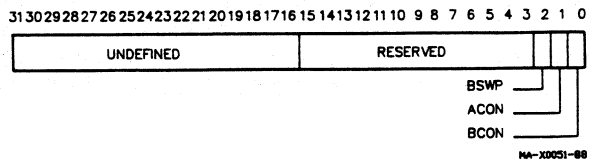


Figure 3–44 Network Interface Control and Status Register 3

Data Bit	Definition
NICSR3 <31:16>	Undefined. Should not be read or written.
NICSR3 <15:3>	Reserved. Read as zeros. Writes have no effect.
NICSR3 <2>	(BSWP) Byte swap. Read/Write. When this bit is set, the LANCE chip will swap the high and low bytes for DMA data transfers between the silo and the network interface buffer RAM in order to accomodate processors which consider address bits <15:08> to be the least significant byte of data. Cleared by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear. This bit must be set to 0 by the on-board firmware.
NICSR3 <1>	(ACON) ALE control. Read/Write. This bit controls the polarity of the signal emitted on the LANCE chip's ALE/AS pin during DMA operation. Cleared by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear. This bit must be set to 0 by the on-board firmware.
NICSR3 <0>	(BCON) Byte control. Read/Write. This bit controls the configuration of the byte mask and hold signals on the LANCE chip's pins during DMA operation. Cleared by setting NICSR0 <2>, on power-up and the negation of DCOK when SCR <7> is clear. This bit must be set to 0 by the on-board firmware.

3.9.10 Network Interface Initialization Block

When the LANCE chip is initialized (by setting the INIT bit in NICSR0), it reads a 24-byte block of data called the network interface initialization block (NIIB) from the network interface buffer RAM using DMA accesses. The base address of the initialization block is formed by concatenating the contents of the NICSR1 and NICSR2. Since the NIIB must start on a word boundary, the low-order bit of the address must be zero. The initialization block is made up of twelve 16-bit words, (NIIBW0-NIIBW11), arranged as shown in Figure 3-45.

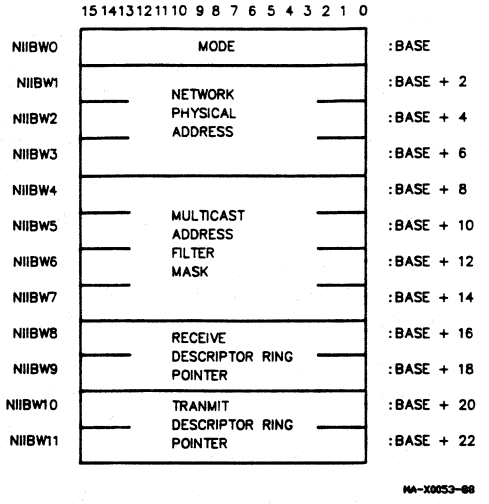


Figure 3–45 Network Interface Initialization Block

3.9.10.1 Network Interface Initialization Block Word 0

Word 0 of the network interface initialization block (NIIBW0), also referred to as the mode word, resides in the network interface buffer RAM at the base address of the initialization block. The mode word of the initialization block allows alteration of the LANCE chip's normal operation for testing and special applications. For normal operation the mode word is entirely zero.

The format for NIIBW0 is shown in Figure 3–46.

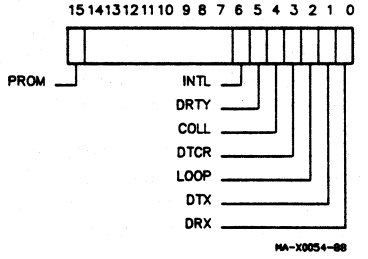


Figure 3–46 Network Interface Initialization Block Word 0

Data Bit	Definition
NIIBW0<15>	(PROM) Promiscuous mode. When set, all incoming packets are accepted regardless of their destination addresses. When cleared, only incoming packets with a destination address that matches the KA640's address are accepted (normal operating mode).
NIIBW0<14:7>	Reserved. Should be written with zeros.
NIIBW0<6>	(INTL) Internal loopback. This bit is used in conjunction with the NIIBW0<2> (LOOP) to control loopback operation. See the description of the LOOP bit, below.
NIIBW0<5>	(DRTY) Disable retry. When set, the LANCE chip will attempt only one transmission of a packet. If there is a collision on the first transmission attempt, a retry error (RTE) will be reported in the transmit buffer descriptor.
NIIBW0<4>	(COLL) Force collision. When set, the collision logic can be tested. The LANCE chip must be in internal loopback mode for COLL to be used. When COLL is set to one, a collision will be forced during the subsequent transmission attempt. This will result in 16 transmission attempts and a retry error (RTE) being reported in the transmit buffer descriptor.
NIIBW0<3>	(DTCR) Disable transmit CRC. When cleared, the transmitter will generate and append a 4-byte CRC to each transmitted packet (normal operation). When DTCR is one the CRC logic is allocated instead to the receiver and no CRC is sent with a transmitted packet. During loopback, setting DTCR to zero will cause a CRC to be generated and sent with the transmitted packet, but no CRC check can be done by the receiver since the CRC logic is shared and cannot both generate and check a CRC at the same time. The CRC transmitted with the packet will be received and written into memory following the data where it can be checked by software. If DTCR is set to one during loopback, the driving software must compute and append a CRC value to the data to be transmitted. The receiver will check this CRC upon reception and report any error.

Data Bit	Definition
NIIBW0 <2>	(LOOP) Loopback control. This bit is used in conjunction with NIIBW0 <6> to perform internal loopback tests on the LANCE chip. During loopback the LANCE chip operates in full duplex mode. The maximum packet size is limited to 32 data bytes (in addition to which 4 CRC bytes may be appended). During loopback, the runt packet filter is disabled because the maximum packet is forced to be smaller than the minimum size Ethernet packet (64 bytes). Setting LOOP to one allows simultaneous transmission and reception for a packet constrained to fit within the silo. The chip waits until the entire packet is in the silo before beginning serial transmission. The incoming data stream fills the silo from behind as it is being emptied. Moving the received packet out of the silo into memory does not begin until reception has ceased. In loopback mode, transmit data chaining is not possible. Receive data chaining is allowed regardless of the receive buffer length. (In normal operation, the receive buffers must be 64 bytes long, to allow time for buffer lookahead.) Valid loopback bit settings are:

LOOP	INTL	Operation
0	x	Normal on-line operation
1	0	External loopback
1	1	Internal loopback

Internal loopback allows the LANCE chip to receive its own transmitted packet without disturbing the network. The LANCE chip will not receive any packets from the network while it is in internal loopback mode.

External loopback allows the LANCE chip to transmit a packet through the transceiver out to the network cable to check the operability of all circuits and connections between the LANCE chip and the network cable. Multicast addressing in external loopback is valid only when DTCR is one (user needs to append the 4 CRC bytes). In external loopback, the LANCE chip also receives packets from other nodes.

Data Bit	Definition
NIIBW0<1>	(DTX) Disable transmitter. When set, the LANCE chip will not set the TXON bit in NICSR0 at the completion of initialization. This will prevent the LANCE chip from attempting to access the transmit descriptor ring, hence no transmissions will be attempted.
NIIBW0<0>	(DRX) Disable receiver. When set, the LANCE chip will not set the RXON bit in NICSR0 at the completion of initialization. This will cause the LANCE chip to reject all incoming packets and to not attempt to access the receive descriptor ring.

3.9.10.2 Network Interface Initialization Block Words 1-3

Words 1-3 of the network interface initialization block (NIIBW1-3), reside in the network interface buffer RAM at the base address of the NIIB plus 2, 4, and 6 respectively. These three words contain the 48-bit NPA of the KA640 and are loaded by the resident firmware from the NISA ROM.

This address identifies the KA640 to the Ethernet network and must be unique within the domain of the network. The low-order bit (bit 0) of this address must be zero to indicate it is a physical rather than multicast address.

The format for network interface initialization block words 1 through 3 is shown in Figure 3–47.

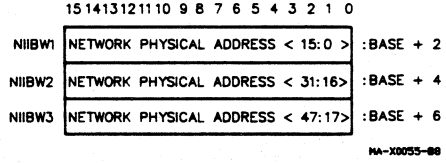


Figure 3–47 Network Interface Initialization Block Words 1-3

3.9.10.3 Network Interface Initialization Block Words 4-7

Words 4-7 of the network interface initialization block (NIIBW4-7), reside in network interface buffer RAM at the base address of the NIIB plus 8, 10, 12, and 14 respectively. These four words contain the 64-bit multicast address filter mask.

The format for network interface initialization block words 4 through 7 is shown in Figure 3-48.

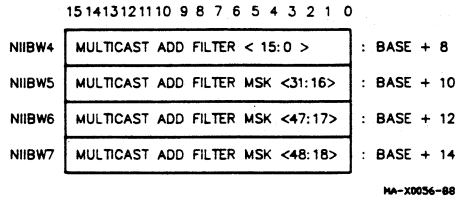


Figure 3-48 Network Interface Initialization Block Words 4-7

Multicast Ethernet addresses are distinguished from physical network addresses by the presence of a one in bit 0 of the 48-bit address field. If an incoming packet contains a physical destination address (bit 0 is zero), then its entire 48 bits are compared with the network physical address and the packet is ignored if they are not equal. If the packet contains a multicast destination address which is all ones (the broadcast address), it is always accepted and stored regardless of the contents of the multicast address filter mask.

All other multicast addresses are processed through the multicast address filter to determine whether the incoming packet will be stored in a receive buffer. This filtering is performed by passing the multicast address field through the CRC generator. The high-order 6 bits of the resulting 32-bit CRC are used to select one of the 64 bits of the multicast address filter mask. (These high-order six bits represent in binary the number of the bit in the multicast address filter mask.) If the bit selected from the multicast address filter mask is one, the packet is stored in a receive buffer; otherwise it is ignored. This mechanism effectively splits the entire domain of 2**47 multicast addresses into 64 parts, and multicast addresses falling into each part will be accepted or ignored according to the value of the corresponding bit in the multicast address filter mask. The driver program must examine the addresses of the packets accepted by this partial filtering to complete the filtering task.

3.9.10.4 Network Interface Initialization Block Words 8,9

Words 8 and 9 of the network interface initialization block (NIIBW8,9), also referred to as the receive descriptor ring pointer, reside in network interface buffer RAM at the base address of the NIIB plus 16 and 18 respectively. These two words contain the starting address and the number of descriptors in the receive descriptor ring.

The format for NIIBW8 is shown in Figure 3-49.

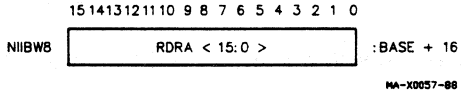


Figure 3-49 Network Interface Initialization Block Word 8

Data Bit	Definition
NIIBW8 <15:0>	(RDRA <15:0>) Receive descriptor ring address <15:0>. This field contains bits <15:0> of the base address of the receive descriptor ring. Since the receive descriptor ring must start on a a quadword boundary, bits <2:0> of this field must be zero.

The format for NIIBW9 is shown in Figure 3-50.

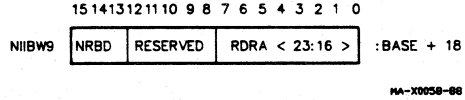


Figure 3-50 Network Interface Initialization Block Word 9

Data Bit	Definition
NIIBW9<15:13>	(NRBD) Number of receive buffer descriptors. This field gives the number of receive buffer descriptors in the receive descriptor ring, expressed as a power of two:

Value	Number of Descriptors
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

NIIBW9<12:8>	Reserved; should be zeros.
NIIBW9<7:0>	(RDRA <23:16>) Receive descriptor ring address <23:16>. This field contains bits <23:16> of the base address of the receive descriptor ring.

3.9.10.5 Network Interface Initialization Block Words 10,11

Words 10 and 11 of the network interface initialization block (NIIBW10,11), also referred to as the transmit descriptor ring pointer, reside in network interface buffer RAM at the base address of the NIIB plus 20 and 22 respectively. These two words contain the starting address and the number of transmit buffer descriptors in the transmit descriptor ring.

The format for NIIBW10 is shown in Figure 3-51.

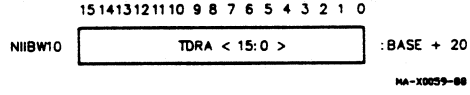


Figure 3-51 Network Interface Initialization Block Word 10

Data Bit	Definition
NIIBW10<15:0>	(TDRA <15:0>) Transmit descriptor ring address <15:0>. This field contains bits <15:0> of the base address of the transmit descriptor ring. Since the transmit descriptor ring must start on a a quadword boundary, bits <2:0> of this field must be zero.

The format for NIIBW11 is shown in Figure 3-52.

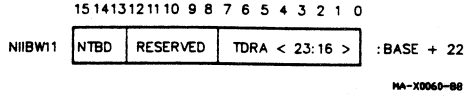


Figure 3-52 Network Interface Initialization Block Word 11

Data Bit	Definition
NIIBW11<15:13>	(NTBD) Number of transmit buffer descriptors. This field gives the number of transmit buffer descriptors in the transmit descriptor ring, expressed as a power of two:

Value	Number of Descriptors
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

NIIBW11 <12:8>	Reserved; should be zeros.
NIIBW11 <7:0>	(TDRA <23:16>) Transmit descriptor ring address <23:16>. This field contains bits <23:16> of the base address of the transmit descriptor ring.

3.9.11 Buffer Management

The LANCE chip manages its data buffers by using two rings of buffer descriptors which are stored in the network interface buffer RAM: the network interface receive descriptor ring and the network interface transmit descriptor ring. Each buffer descriptor points to a data buffer elsewhere in the network interface buffer RAM, contains the size of that buffer, and contains status information about that buffer's contents.

The starting location in the network interface buffer RAM of each ring and the number of descriptors in it are given to the LANCE chip via the NIIB during the chip initialization process. Each descriptor is 8 bytes long and must be aligned on a quadword boundary (the three low-order bits of its address must be zero). The descriptors in a ring are physically contiguous in the network interface buffer RAM and the number of descriptors must be a power of two. The LANCE keeps an internal index to its current position in each ring which it increments modulo the number of descriptors in the ring as it advances around each ring.

Once started, the LANCE chip polls each ring to find descriptors for buffers in which to receive incoming packets and from which to transmit outgoing packets, and revises the status information in buffer descriptors as it processes their associated buffers. When polling, the LANCE chip is limited to looking only one ahead of the descriptor with which it is currently working. The high speed of the data stream requires that each buffer be at least 64 bytes long to allow time to chain buffers for packets which are larger than one buffer. (The first buffer of a packet to be transmitted should be at least 100 bytes to avoid problems in case a late collision is detected.)

Each descriptor in a ring is "owned" either by the LANCE chip or by the host processor; this status is indicated by the OWN bit in each descriptor. Mutual exclusion is accomplished by the rule that each device can only relinquish ownership of a descriptor to the other device, it can never take ownership; and that each device cannot change any field in a descriptor or its associated buffer after it has relinquished ownership. When the host processor sets up the rings of descriptors before starting the LANCE chip, it sets the OWN bits such that the LANCE chip will own all the descriptors in the network interface receive descriptor ring (to be used by the LANCE to receive packets from the network) and the host will own all the descriptors in the network interface transmit descriptor ring (to be used by the host to set up packets to be transmitted to the network).

3.9.12 Network Interface Receive Descriptor Ring

The network interface receive descriptor ring (NIRDR) contains a receive buffer descriptor for each receive buffer (Figure 3-53). It is located in a contiguous block of the network interface buffer RAM whose base address is formed by concatenating the contents of NIIBW8 and NIIBW9 <7:0> (RDRA<23:0). Since the NIRDR must start on a quadword boundary, bits <2:0> of this address must be zero. The size of the network interface receive descriptor ring can vary between 8 and 1024 bytes depending on the number of 8-byte descriptors it contains (Figure 3-53). The number of descriptors must be a power of two between one and 128 and is determined by NIIBW9 <15:13> (NRBD).

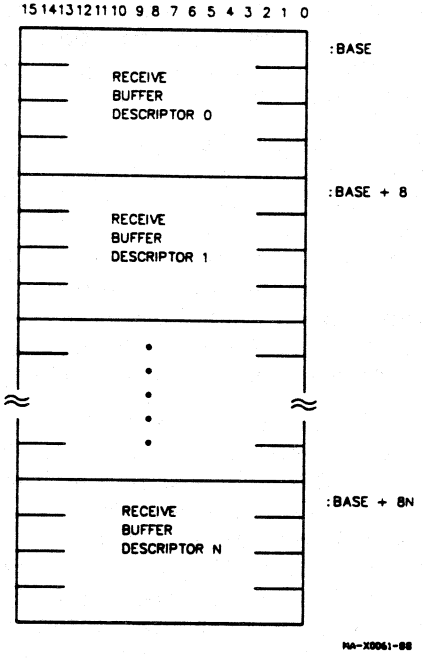


Figure 3-53 Network Interface Receive Descriptor Ring

3.9.12.1 Receive Buffer Descriptors

Receive buffer descriptor n contains the base address and size of a receive buffer as well as status and error information. It is four words (eight bytes) in length and is located in the receive descriptor ring at base + 8n.

A representation of a typical receive buffer descriptor (RBD) is shown in Figure 3-54.

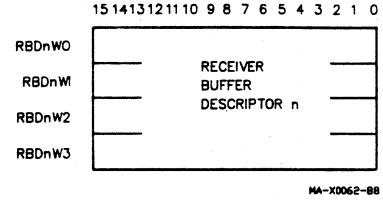


Figure 3-54 Receive Buffer Descriptors

3.9.12.1.1 Receive Buffer Descriptor n Word 0

Word 0 of RBD n (RBDnW0) resides in the network interface buffer RAM at the base address of the NIIRDR +8n. This word contains a portion of the base address of the associated receive buffer.

The format for receive buffer descriptor n word 0 is shown in Figure 3-55.

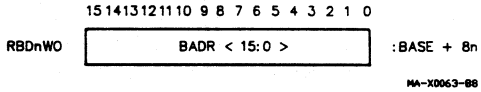


Figure 3-55 Receive Buffer Descriptor n Word 0

Data Bit	Definition
RBDnW0 <15:0>	(BADR) Buffer address. This field contains bits <15:0> of the 24-bit network address buffer RAM address of the start of the buffer associated with this descriptor. Written by the host; unchanged by the LANCE.

3.9.12.1.2 Receive Buffer Descriptor n Word 1

Word 1 of RBD n (RBDnW1) resides in the network interface buffer RAM at the base address of the NIIRDR +8n+2. This word contains a portion of the base address of the associated receive buffer as well as status and error information.

The format for receive buffer descriptor n word 1 is shown in Figure 3-56.

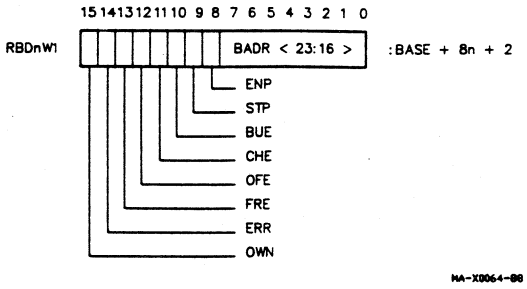


Figure 3-56 Receive Buffer Descriptor n Word 1

Data Bit	Definition
RBDnW1 <15>	(OWN) Owned flag. This bit indicates whether the descriptor is owned by the host (OWN = 0) or by the LANCE chip (OWN = 1). The LANCE clears OWN after filling the buffer associated with the descriptor with an incoming packet. The host sets OWN after emptying the buffer. In each case, this must be the last bit changed by the current owner, since changing OWN passes ownership to the other party and the relinquishing party must not thereafter alter anything in the descriptor or its buffer.
RBDnW1 <14>	(ERR) Error summary. This is the logical OR of the FRE, OFE, CHE and BUE bits in this word. Set by the LANCE chip and cleared by the host.
RBDnW1 <13>	(FRE) Framing error. This bit is set by the LANCE chip to indicate that the incoming packet stored in the buffer had both a non-integral multiple of 8 bits and a checksum error (CHE). It is cleared by the host.
RBDnW1 <12>	(OFE) Overflow error. This bit is set by the LANCE chip to indicate that the receiver has lost part or all of an incoming packet because it could not store it in the buffer before the chip's silo overflowed. Cleared by the host.
RBDnW1 <11>	(CHE) Checksum error. This bit is set by the LANCE chip to indicate that the received packet has an invalid CRC checksum. Cleared by the host.

Data Bit	Definition
RBDnW1 <10>	(BUE) Buffer error. This bit is set by the LANCE chip when it has used all its owned receive descriptors or when it could not get the next descriptor in time while attempting to chain to a new buffer in the midst of a packet. When a buffer error occurs, an overflow error (bit OFLO) also occurs because the LANCE continues to attempt to get the next buffer until its silo overflows. BUE is cleared by the host.
RBDnW1 <9>	(STP) Start of packet. This bit is set by the LANCE chip to indicate that this is the first buffer used for this packet. Cleared by the host.
RBDnW1 <8>	(ENP) End of packet. This bit is set by the LANCE chip to indicate that this is the last buffer used for this packet. When both STP and ENP are set in a descriptor, its buffer contains an entire packet; otherwise two or more buffers have been chained together to hold the packet. ENP is cleared by the host.
RBDnW1 <7:0>	(BADR <23:16>) Buffer address <23:16>. This field contains bits <23:16> of the 24-bit the NI buffer RAM address of the start of the buffer associated with this descriptor. Written by the host; unchanged by the LANCE.

3.9.12.1.3 Receive Buffer Descriptor n Word 2

Word 2 of RBD n (RBDnW2) resides in the network interface buffer RAM at the base address of the NIIRDR +8n+4. This word contains the size of the associated receive buffer.

The format for receive buffer descriptor n word 2 is shown in Figure 3-57.

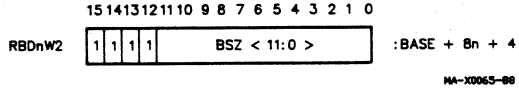


Figure 3-57 Receive Buffer Descriptor n Word 2

Data Bit	Definition
RBDnW2<15:12>	These bits must be set by the host to ones. Unchanged by the LANCE chip.
RBDnW2<11:0>	(BSZ <11:0>) Buffer size. This field contains the size (in bytes) of the associated receive buffer in two's complement form. Note that the minimum buffer size is 64 bytes (to allow enough time for chaining buffers) and the maximum buffer size is 1518 bytes (the largest legal Ethernet packet). Written by the host; unchanged by the LANCE chip.

3.9.12.1.4 Receive Buffer Descriptor n Word 3

Word 3 of RBD n (RBDnW3) resides in the network interface buffer RAM at the base address of the NIIRDR +8n+6. This word contains the size of the packet that was received.

The format for receive buffer descriptor n word 3 is shown in Figure 3-58.

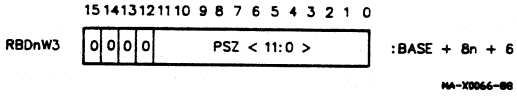


Figure 3-58 Receive Buffer Descriptor n Word 3

Data Bit	Definition
RBDnW3<15:12>	These bits are reserved. They should be set to zeros by the host when it constructs the descriptor.
RBDnW3<11:0>	(PSZ <11:0>) Packet size. This field contains the size (in bytes) of the received packet. This field is valid only in a descriptor in which ENP is set (last buffer) and ERR is clear (no error). Set by the LANCE chip and cleared by the host.

3.9.13 Receive Buffers

Receive buffers are set up by the host by adding a receive buffer descriptor to the network interface receive buffer descriptor ring. These buffers are used for storing incoming Ethernet packets. An Ethernet packet may span multiple buffers, but a buffer cannot contain more than one Ethernet packet. The base address of a receive buffer is formed by concatenating the contents of RBDnW0 and RBDnW1 <7:0> (BADR). The size of a receive buffer is determined by RBDnW2 <11:0>.

Receive buffers are structured as shown in Figure 3–59.

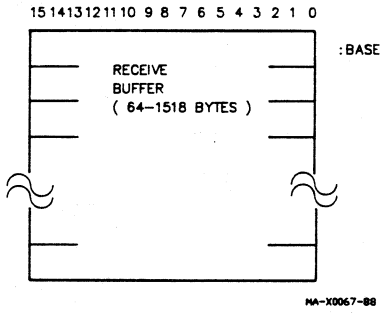


Figure 3–59 Receive Buffers

3.9.14 Network Interface Transmit Descriptor Ring

The network interface transmit descriptor ring (NITDR) contains a transmit buffer descriptor for each transmit buffer (Figure 3–60). It is located in a contiguous block of the network interface buffer RAM whose base address is formed by concatenating the contents of the NIIBW10 and NIIBW11 <7:0> (TDRA<23:0>). Since the NITDR must start on a quadword boundary, bits <2:0> of this address must be zero. The size of the network interface transmit descriptor ring can vary between 8 and 1024 bytes depending on the number of 8-byte descriptors it contains. The number of descriptors must be a power of two between one and 128 and is determined by NIIBW11 <15:13> (NTBD).

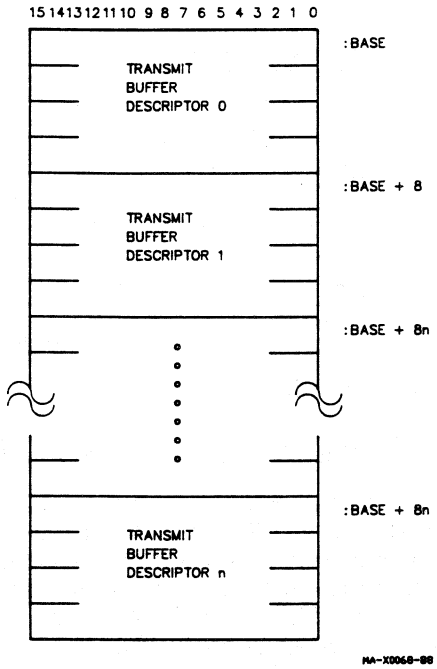


Figure 3–60 Network Interface Transmit Descriptor Ring

3.9.14.1 Transmit Buffer Descriptors

Transmit buffer descriptor n contains the base address, size, of a transmit buffer as well as status and error information. It is four words (eight bytes) in length and is located in the transmit descriptor ring at base + 8n.

A representation of a typical transmit buffer descriptor (TBD) is shown in Figure 3–61.

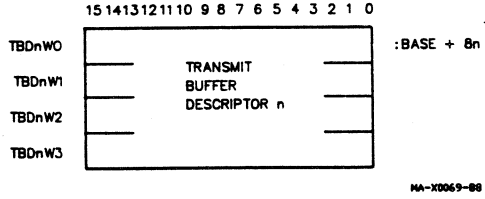


Figure 3–61 Transmit Buffer Descriptors

3.9.14.1.1 Transmit Buffer Descriptor n Word 0

Word 0 of TBD n (TBDnW0) resides in the network interface buffer RAM at the base address of the NIIRDR +8n. This word contains a portion of the base address of the associated transmit buffer.

The format for transmit buffer descriptor n word 0 is shown in Figure 3–62.

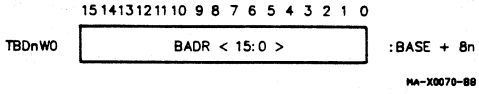


Figure 3–62 Transmit Buffer Descriptor n Word 0

Data Bit	Definition
TBDnW0<15:0>	(BADR) Buffer address. This field contains bits <15:0> of the 24-bit network interface buffer RAM address of the start of the buffer associated with this descriptor. Written by the host; unchanged by the LANCE chip.

3.9.14.1.2 Transmit Buffer Descriptor n Word 1

Word 1 of TBD n (TBDnW1) resides in the network interface buffer RAM at the base address of the NIIRDR +8n+2. This word contains a portion of the base address of the associated transmit buffer as well as status and error information.

The format for transmit buffer descriptor n word 1 is shown in Figure 3–63.

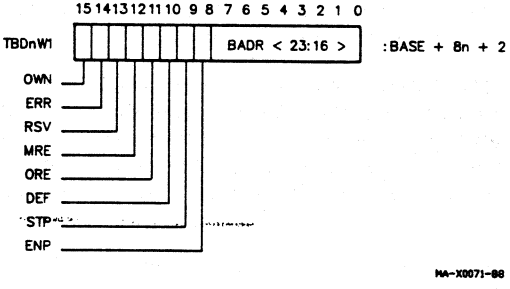


Figure 3–63 Transmit Buffer Descriptor n Word 1

Data Bit	Definition
TBDnW1 <15>	(OWN) Owned flag. This bit indicates whether the descriptor is owned by the host (OWN = 0) or by the LANCE chip (OWN = 1). The LANCE clears OWN after filling the buffer associated with the descriptor with an incoming packet. The host sets OWN after emptying the buffer. In each case, this must be the last bit changed by the current owner, since changing OWN passes ownership to the other party and the relinquishing party must not thereafter alter anything in the descriptor or its buffer.
TBDnW1 <14>	(ERR) Error summary. This bit is the logical OR of the COE, CAE, UFE and RTE bits in this descriptor. Set by the LANCE chip and cleared by the host.
TBDnW1 <13>	(RSV) Reserved. The LANCE chip will write a zero in this bit.
TBDnW1 <12>	(MRE) More retries. The LANCE chip sets this bit when more than one retry was required to transmit the packet. Cleared by the host.
TBDnW1 <11>	(ORE) One retry. The LANCE chip sets this bit when exactly one retry was required to transmit the packet. Cleared by the host.
TBDnW1 <10>	(DEF) Deferred. The LANCE chip sets this bit when it had to defer while trying to transmit the packet. This occurs when the network is busy when the LANCE is ready to transmit. Cleared by the host.
TBDnW1 <9>	(STP) Start of packet. This bit is set by the host to indicate that this is the first buffer used for this packet. STP is not changed by the LANCE chip.
TBDnW1 <8>	(ENP) End of packet. This bit is set by the host to indicate that this is the last buffer used for this packet. When both STP and ENP are set in a descriptor, its buffer contains an entire packet; otherwise two or more buffers have been chained together to hold the packet. ENP is not changed by the LANCE chip.
TBDnW1 <7:0>	(BADR <23:16>) Buffer address <23:16>. This field contains bits <23:16> of the 24-bit network interface buffer RAM address of the start of the buffer associated with this descriptor. Written by the host; unchanged by the LANCE chip.

3.9.14.1.3 Transmit Buffer Descriptor n Word 2

Word 2 of TBD n (TBDnW2) resides in the network interface buffer RAM at the base address of the NIITDR +8n+4. This word contains the size of the associated transmit buffer.

The format for transmit buffer descriptor n word 2 is shown in Figure 3-64.

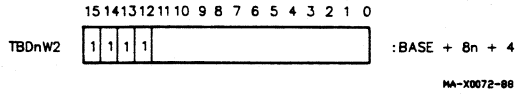


Figure 3-64 Transmit Buffer Descriptor n Word 2

Data Bit	Definition
TBDnW2<15:12>	These bits must be set by the host to ones. Unchanged by the LANCE chip.
TBDnW2 <11:0>	(BSZ <11:0>) Buffer size. This field contains the size (in bytes) of the associated transmit buffer in two's complement form. Note that the minimum buffer size is 64 bytes (to allow enough time for chaining buffers) and the maximum buffer size is 1518 bytes (the largest legal Ethernet packet). Written by the host; unchanged by the LANCE chip.

3.9.14.1.4 Transmit Buffer Descriptor n Word 3

Word 3 of TBD n (TBDnW3) resides in the network interface buffer RAM at the base address of the NIITDR +8n+6. This word contains error information and a time domain reflectometer. The contents of this word are valid only when the ERR bit in TBDnW2 has been set by the LANCE chip.

The format for transmit buffer descriptor n word 3 is shown in Figure 3-65.

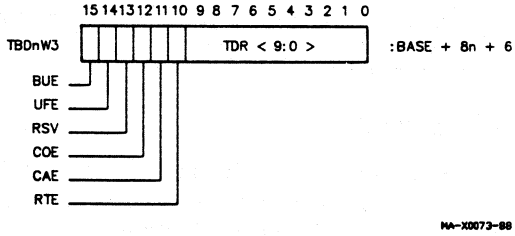


Figure 3-65 Transmit Buffer Descriptor n Word 3

Data Bit	Definition
TBDnW3 <15>	(BUE) Buffer error. This bit is set by the LANCE chip during transmission when it does not find the ENP bit set in the current descriptor and it does not own the next descriptor. When BUE is set, the UFE bit (below) is also set because the LANCE chip continues to transmit until its silo becomes empty. BUE is cleared by the host.
TBDnW3 <14>	(UFE) Underflow error. This bit is set by the LANCE chip when it truncates a packet being transmitted because it has drained its silo before it was able to obtain additional data from a buffer in memory. UFE is cleared by the host.
TBDnW3 <13>	(RSV) Reserved. The LANCE chip will write a zero in this bit.
TBDnW3 <12>	(COE) Late collision error. This bit is set by the LANCE chip to indicate that a collision has occurred after the slot time of the network channel has elapsed. The LANCE chip does not retry after a late collision. COE is cleared by the host.
TBDnW3 <11>	(CAE) Loss of carrier error. This bit is set by the LANCE chip when the carrier present input to the chip becomes false during a transmission initiated by the LANCE. The LANCE chip does not retry after such a failure. CAE is cleared by the host.
TBDnW3 <10>	(RTE) Retries exhausted. This bit is set by the LANCE chip after 16 attempts to transmit a packet have failed due to repeated collisions on the network. (If the DRTY bit of network interface initialization block word 0 (mode word) is set, RTE will instead be set after only one failed transmission attempt.) RTE is cleared by the host.
TBDnW3 <9:0>	(TDR) Time domain reflectometer. These bits are the value of an internal counter which is set by the LANCE chip to count system clocks from the start of a transmission to the occurrence of a collision. This value is useful in determing the approximate distance to a cable fault; it is valid only when the RTE bit in this word is set.

3.9.15 Transmit Buffers

Transmit buffers are set up by the host by adding a transmit buffer descriptor to the network interface transmit buffer descriptor ring. These buffers are used for storing incoming Ethernet packets. An Ethernet packet may span multiple buffers, but a buffer cannot contain more than one Ethernet packet. The base address of a transmit buffer is formed by concatenating the contents of the TBDnW0 and TBDnW1 <7:0> (BADR). The size of a transmit buffer is determined by TBDnW2 <11:0>.

Transmit buffers are structured as shown in Figure 3-66.

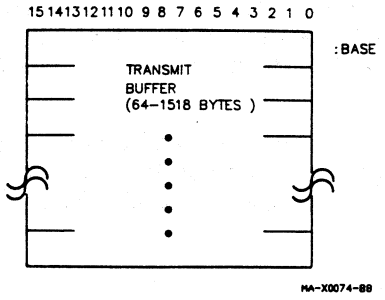


Figure 3-66 Transmit Buffers

3.9.16 LANCE Operation

The LANCE chip operates independently of the host under control of its own internal microprogram. This section is a simplified description of the operation of the LANCE in terms of its principal microcode routines (these should not be confused with device driver programming in the host, which is not a part of this specification). These microcode routines make use of numerous temporary storage cells within the LANCE chip; most of these are not accessible from outside the chip but they are mentioned here when necessary to clarify the operation of the microcode.

Two such (conceptual) internal variables are of central importance: the pointers to the "current" entry in the receive descriptor ring and in the transmit descriptor ring, which are referred to below as TXP and RXP. Each of these designates the descriptor which the LANCE will use for the next operation of that type. If the descriptor designated by one of these pointers is not owned by the LANCE (the OWN bit is 0), then the LANCE can neither perform activity of that type nor advance the pointer.

For the transmit ring, the LANCE will do nothing until the host sets up a packet in the buffer and sets the OWN bit in the descriptor designated by the LANCE's TXP. (The host must keep track of the position of the TXP, since setting up a packet in some other descriptor will not be detected by the LANCE.) For the receive ring, if the LANCE does not own the descriptor designated by RXP, it cannot receive a packet. In both rings, when the LANCE finishes with a descriptor and relinquishes it to the host by clearing OWN, it then advances the ring pointer (modulo the number of entries in the ring).

When the LANCE begins activity using the current descriptor (i.e., begins receiving or transmitting a packet), it may look ahead at the next descriptor and attempt to read its first three words in advance so it can chain to the next buffer in mid-packet without losing data. However, it does not actually advance its RXP or TXP until it has cleared the OWN bit in the current descriptor.

The LANCE is a very complex chip and this specification does not attempt to cover all the details of its operation. The chip purchase specification and the chip vendor's literature should also be consulted for more detailed information.

3.9.16.1 Switch Routine

Upon power on, the STOP bit is set and the INIT and STRT bits are cleared in NICSR0. The LANCE microprogram begins execution in the switch routine, which tests the INIT, STRT, and STOP bits. When the host sets either INIT or STRT, STOP is cleared. While STOP is set, if the host writes to NICSR1 and NICSR2, that data is stored for use by the initialization routine.

When the microprogram sees STOP cleared, it tests first the INIT bit and then the STRT bit. If INIT is set, it performs the initialization routine. Then if STRT is set, it begins active chip operation by jumping to the look-for-work routine. Control returns to the switch routine whenever the host again sets the STOP bit (which also clears the INIT and STRT bits). Note that the ring pointers RXP and TXP are not altered by the setting of either STOP or START; they are reset to the start of their rings only when INIT is set.

3.9.16.2 Initialization Routine

The initialization routine is called from the switch routine when the latter finds the INIT bit set. It reads the initialization block from the memory addressed by NICSR1 and NICSR2 and stores its data within the LANCE chip. This routine also sets the ring pointers RXP and TXP to the start of their rings (i.e., to point to the descriptor at the lowest memory address in the ring).

3.9.16.3 Look-For-Work Routine

The look-for-work routine is executed while the LANCE is active and looking for work. It is entered from the switch routine when the STRT bit is set, and is returned to from the receive and transmit routines after they have received or transmitted a packet.

This routine begins by testing whether the receiver is enabled (bit RXON of NICSR0 is set). If so, it tries to have a receive buffer available for immediate use when a packet addressed to this system arrives. It tests its internal registers to see whether it has already found a receive descriptor owned by the LANCE and, if not, calls the receive poll routine to attempt to get a receive buffer.

Next the routine tests whether the transmitter is enabled (bit TXON of NICSR0 is set). If so, it calls the transmit poll routine to see whether there is a packet to be transmitted and to transmit it.

If there was no transmission and the TDMD bit of NICSR0 is not set, the microprogram delays 1.6 milliseconds and then goes to check the receive descriptor status again. If a packet was transmitted or the host has set TDMD, the delay is omitted so that multiple packets will be transmitted as quickly as possible.

If at any point in this routine the receiver detects an incoming packet whose destination address matches the station's physical address, is the broadcast address, or passes the multicast address filter (or if the PROM bit of NIIBW0 is set), the receive routine is called.

3.9.16.4 Receive Poll Routine

The receive poll routine is called whenever the receiver is enabled and the LANCE needs a free buffer from the receive descriptor ring. The routine reads the second word of the descriptor designated by RXP and, if the OWN bit in it is set, reads the first and third words also.

3.9.16.5 Receive Routine

The receive routine is called when the receiver is enabled and an incoming packet's destination address field matches one of the criteria described above. The routine has three sections: initialization, lookahead, and descriptor update.

In initialization, the routine checks whether a receive ring descriptor has already been acquired by the receive poll routine. If not, it makes one attempt to get the descriptor designated by RXP (if OWN is not set in it, MISS and ERR are set in NICSR0 and the packet is lost). The buffer thus acquired is used by the receive DMA routine to empty the silo.

In lookahead, the routine reads the second word of the next descriptor in the receive ring and, if the OWN bit is set, reads the rest of the descriptor and holds it in readiness for possible data chaining.

The descriptor update section is performed when either the current buffer is filled or the packet ends. If the packet ends but its total length is less than 64 bytes, it is an erroneous "runt packet" and is ignored: no status is posted in the descriptor, RXP is not moved, and the buffer will be reused for the next incoming packet (this is why a receive buffer must be at least 64 bytes long; otherwise the runt might be detected after advancing RXP).

If the packet ends (with or without error), the routine writes the packet length into MCNT, sets ENP and other appropriate status bits and clears OWN in the current descriptor, and sets RINT in NICSR0 to signal the host that a complete packet has been received. Then it advances RXP and returns to the look-for-work routine.

If the buffer is full and the packet has not ended, chaining is required. The routine releases the current buffer by writing status bits into its descriptor (clearing OWN and ENP, in particular), makes current the next descriptor data acquired in the lookahead section, advances RXP, and goes to the lookahead section to prepare for possible additional chaining. Note that RINT is not set in NICSR0, although the host would find OWN cleared if it looked at the descriptor, and it could begin work on that section of the packet, since the mutual exclusion rule prevents the LANCE from going back and altering it.

3.9.16.6 Receive DMA Routine

The receive DMA routine is invoked asynchronously by the chip hardware during execution of the receive routine whenever the silo contains 16 or more bytes of incoming data or when the packet ends and the silo is not empty. It executes DMA cycles to drain data from the silo into the buffer designated by the current descriptor.

3.9.16.7 Transmit Poll Routine

The transmit poll routine is called by the look-for-work routine to see whether a packet is ready for transmission. It reads the second word of the descriptor designated by TXP and tests the OWN bit. If OWN is zero, the LANCE does not own the buffer and this routine returns to its caller. If OWN is set, the routine tests the STP bit, which should be set to indicate the start of a packet. If STP is clear, this is an invalid packet; the LANCE sets its OWN bit to return it to the host, sets TINT in NICSR0 to notify the host, and advances TXP to the next transmit descriptor. If both OWN and STP are set, this is the beginning of a packet, so the transmit poll routine reads the rest of the descriptor and then calls the transmit routine to transmit the

packet. During this time the chip is still watching for incoming packets from the network and it will abort the transmit operation if one arrives.

3.9.16.8 Transmit Routine

The transmit routine is called from the transmit poll routine when the latter finds the start of a packet to be transmitted. This routine has three sections: initialization, lookahead, and descriptor update.

In initialization, the routine sets the chip's internal buffer address and byte count from the transmit descriptor, enables the transmit DMA engine, and starts transmission of the packet preamble. It then waits until the transmitter is actually sending the bit stream (including possible backoff-and-retry actions in case of collisions).

In lookahead, the transmit routine test the current descriptor to see whether it is the last in the packet (the ENP bit is set). If so, no additional buffer is required so the routine waits until all the bytes from the current packet have been transmitted. If not, the routine attempts to get the next descriptor and hold it in readiness for data chaining, and then waits until all the bytes from the current buffer have been transmitted.

Descriptor update is entered when all the bytes from a buffer have been transmitted or an error has occurred. If there is no error and the buffer was not the last of the packet, the pre-fetched descriptor for the next buffer is made current for use by the transmit DMA routine. The routine writes the appropriate status bits and clears the OWN bits in the current descriptor and advances TXP. If this was the last buffer in the packet, the routine sets the TINT bit in NICSR0 to notify the host and returns to the look-for-work routine. Otherwise it goes back to the lookahead section in this routine.

3.9.16.9 Transmit DMA Routine

The transmit DMA routine is invoked asynchronously by the chip hardware during execution of the transmit routine whenever the silo has 16 or more empty bytes. It executes DMA cycles to fill the silo with data from the buffer designated by the current descriptor.

3.9.16.10 Collision Detect Routine

This routine is invoked asynchronously by the chip hardware during execution of the transmit routine when a collision is detected on the network. It ensures that the jam sequence is transmitted, then backs up the chip's internal buffer address and byte count registers, waits for a pseudo-random backoff time, and then attempts the transmission again. If 15 retransmission attempts fail (a total of 16 attempts), it sends the microcode to the descriptor update routine to report an error in the current transmit descriptor (bits RTRY and ERR are set).

3.9.17 LANCE Programming Notes

The following are LANCE programming notes:

1. The interrupt signal is simply the OR of the interrupt-causing conditions. If another such condition occurs while the interrupt signal is already asserted, there will not be another active transition of the interrupt signal and another interrupt will not be generated. An interrupt service routine should use logic similar to the following to avoid losing interrupts:

   a. Read NICSR0 and save the results in a register, say R0.

   b. Clear the interrupt enable bit INEA in the saved data in R0.

   c. Write NICSR0 with the saved data in R0. This will make the interrupt signal false because INEA is clear and will clear all the write-one-to-reset bits such as RINT, TINT and the error bits; it will not alter the STRT, INIT or STOP bits nor any interrupt-cause bits which came true after NICSR0 was read.

   d. Write NICSR0 with only INEA to enable interrupts again.

   e. Service all the interrupt and error conditions indicated by the flags in the data in R0.

   f. Exit from the interrupt service routine.

2. Be sure to access NICSR0 only with instructions which do a single access, such as MOVE. Instructions such as BIS which do a read-modify-write operation can have unintended side effects.

3. An interrupt is signalled to the host only when the last buffer of a multibuffer (chained) packet is received or transmitted. However, the OWN bit in each descriptor is cleared as soon as the LANCE has finished with that portion of the packet, and the mutual exclusion rule makes it safe for the host to process such a descriptor and its buffer.

4. When a transmitter underflow occurs (UFE is set in a transmit descriptor because the silo is not filled fast enough), the LANCE will turn off its transmitter and the LANCE must be restarted to turn the transmitter back on again. This can be done by setting STOP in NICSR0 and then setting STRT in NICSR0 (DTX will still be clear in the chip's internal copy of NIIBW0). It is not necessary to set INIT to reread the initialization block.

Note that setting STOP will immediately terminate any reception which is in progress. If the status of a receive descriptor has been updated and its OWN bit is now clear, then the contents of its buffer are valid. If the incoming packet was chained into more than one buffer, however, the packet is only valid if its last buffer has been completed (the one with the ENP bit set).

5. The network controller hardware requires up to five seconds after power on to become stable. Self-test routines must delay at least this time before attempting to use the controller for either internal or external testing.

6. The CAE bit (loss of carrier) may be set in the transmit descriptor when a packet is sent in internal loopback mode. When the LANCE is operating in internal loopback mode and a transmission is attempted with a non-matching address, the LANCE will correctly reject that packet. If the next operation is an internal loopback transmission without first resetting the LANCE, the packet will not be sent and LCAR will be set in the transmit descriptor for that packet. The receive descriptor will still be owned by the LANCE. To avoid this problem, the LANCE should be reinitialized after each internal loopback packet.

7. The ONE flag is occasionally set in a transmit descriptor after a late collision. The LANCE does not attempt a retransmission even though ONE may be set. The host should disregard ONE if the COE flag is also set.

8. The chip's internal copy of NICSR1 may become invalid when the chip is stopped. The NICSR1 and NICSR2 registers should always be loaded prior to setting INIT to initialize the LANCE chip.

9. Attempting an external loopback test on a busy network can cause a silo pointer misalignment if a transmit abort occurs while the chip was preparing to transmit the loopback packet. The resulting retransmission may cause the transmitter enable circuit to hang, and the resulting illegal length transmission must be terminated by the jabber timer in the transceiver. It is unlikely that there will be a corrupted receive buffer because the reception that caused the transmit abort will usually not pass address recognition.

   Since external loopback is a controlled situation it is possible to implement a software procedure to detect a silo pointer misalignment problem and prevent continuous transmissions. Since the test is being done in loopback the exact length and contents of the receive packet are known; thus the software can determine whether the data in the receive buffer has been corrupted.

On transmission the diagnostic software should allow up to 32 retries before a hard error is flagged. This is not to say that 32 errors are allowed for each condition; the sum of all errors encountered in the test should not exceed 32. The diagnostic software should expect to get a transmit done interrupt with 1 millisecond of passing the transmit packet to the LANCE. If this does not occur, it should reset the LANCE and retry the test. This prevents a continuous transmission (babble) longer than the longest legal packet in case the LANCE has become hung.

10. When the chip is in internal loopback mode and a CRC error is forced, a framing error will also be indicated along with the CRC error. In external loopback, when a CRC error is forced only that error is indicated; a framing error is indicated only if the LANCE actually receives extra bits.

11. When transmit data chaining, a buffer error will be set in the current transmit descriptor if a late collision or retry error occurred while the LANCE was still transmitting data from the previous buffer. The BUE error in this case is an invalid error indication and should be ignored. BUE is valid only when UFE is also set.

12. When the host program sets up a packet for transmission in chained buffers, it should set the OWN bits in all the transmit buffers except the first one (i.e., the one containing the STP bit), and then as its last act set the OWN bit in the first descriptor. Once that bit is set, the LANCE will start packet transmission and may encounter an underflow error if the subsequent descriptors for the packet are not available.

13. Do not set INIT and STRT in NICSR0 at the same time. After stopping the chip, first set INIT and wait for IDON, then set STRT. If both are set at once, corrupt transmit or receive packets can be generated if RENA becomes true during the initialization process.

3.10 Mass Storage Interface

The KA640 contains a DSSI bus interface which is implemented via the SII chip and four 32K x 8 static RAMs. The interface allows the KA640 to transmit packets of data to, and receive packets of data from, up to seven other DSSI devices (typically RF type disk drives and TF type streaming tape drives). The KA640 also provides for the DSSI bus termination with removable resistors.

This interface contains 27 registers (of which only 16 are used) and 128K bytes of 32 bit wide RAM (MSI buffer RAM). The SII chip transfers data between the DSSI bus and the MSI buffer RAM, and the processor transfers data between the MSI buffer RAM and main memory (typically using MOVC instructions).

3.10.1 DSSI Bus Overview

Some of the major characteristics of the DSSI bus are:

- Eight bit data path
- Eight devices supported
- Parity checking
- Distributed arbitration
- Synchronous operation
- Maximum bandwidth of 4M bytes/sec

Communication on the DSSI bus is limited to two devices at a time. Each device has an unique ID assigned to it.

When two devices communicate on the DSSI bus, one acts as the initiator, the other as the target. The initiator is the device that starts a DSSI bus transaction. The target device controls the remainder of the DSSI bus transaction. The direction of data flow is from the initiator to the target

A DSSI bus transaction consists of six phases:

1. WAIT—During this phase the initiator waits for the bus to become free.
2. ARBITRATION—During this phase control of the bus is taken by the initiator with the highest ID.
3. SELECTION—During this phase the initiator tries to make a logical connection with the target.
4. COMMAND OUT—During this phase the initiator sends the six bytes of command information specified in the command block to the target (Section 3.10.5).
5. DATA OUT—During this phase the initiator sends from one to 4K bytes of data to the target.
6. STATUS IN—During this phase the target sends one byte of status information on the transaction to the initiator. The initiator writes this byte to the status word in the command block.

A block diagram of DSSI bus sequences, showing these six phases, is given in Figure 3-67.

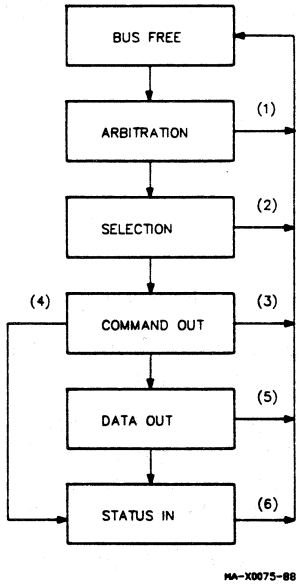


Figure 3–67 DSSI Bus Sequences

The normal path follows vertically downward. Exception paths are listed below:

1. The initiator arbitrates and loses.
2. The target failed to respond or responded with an unexpected bus phase.
3. The operation was timed out or the target responded with unexpected phase.
4. The target detected a parity error or information mismatch in the command, or the target did not have any buffer space available.
5. The operation was timed out or the target responded with an unexpected phase.

3.10.2 Target Operation

When the KA640 is functioning as a target device, the SII chip expects receive buffers to be established in the 128KB MSI buffer RAM (addresses 2010 0000 $_{16}$ through 201F FFFF $_{16}$). Receive buffers must be set up by the processor and start on quadword boundaries. These buffers consist of a command block (Section 3.10.5) and a receive data block. These buffers are linked together by the first word in the command block, and the MSI_TLP register is used to point to the first buffer in the list (Figure 3-68).

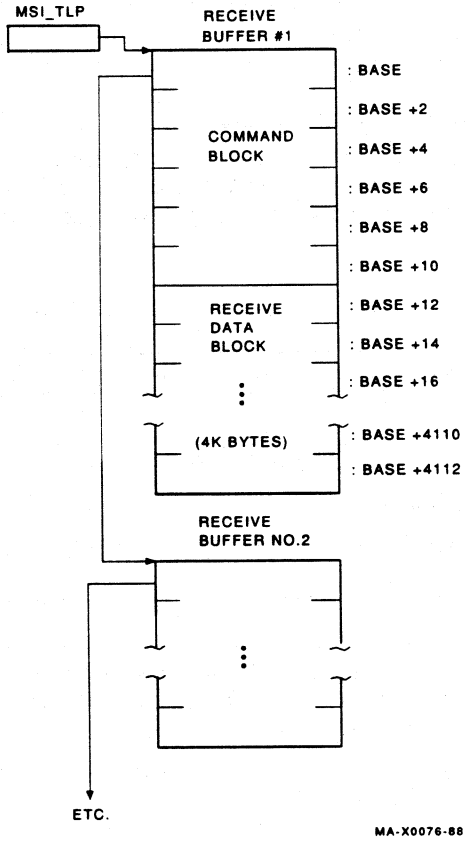


Figure 3-68 Target Operation

During target operation, the SII chip uses the MSI_TLP register to determine the address of the next free receive buffer to be used for this DSSI bus transaction. As the SII chip fills the buffer, it will reload the MSI_TLP for the next target transaction with the buffer's *thread word* (the first word in the command block). The target then places the DSSI bus in the status in

phase, sends a status byte to the initiator and updates the status byte in its buffer's command block.

3.10.3 Initiator Operation

When the KA640 is functioning as a initiator device, the SII chip expects transmit buffers to be established in the 128K Byte MSI BUFFER RAM (addresses 2010 0000 $_{16}$ through 201F FFFF $_{16}$). These buffers must be set up by the processor and start on quadword boundaries. These buffers consist of a command block (Section 3.10.5) and a transmit data block. These buffers are linked together by the first word in the command block, and the MSI_ ILP register is used to point to the first buffer in the list.

3.10.3.1 Transmit Data Segment Links

The transmit data block is broken into one or more segments. These segments need not reside in contiguous locations in the MSI buffer RAM and are connected together by the link. Pictorially, the link appears as shown in Figure 3-69.

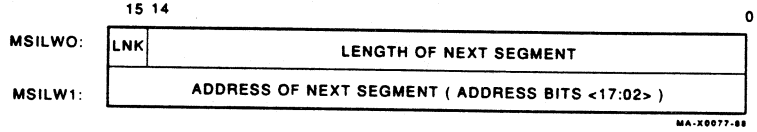


Figure 3-69 Transmit Data Segment Links

3.10.3.1.1 MSI Link Word 0

A definition of the bit fields of MSI link word 0 (MSILW0) is given below.

Data Bit	Definition
MSILW0 <15>	LNK. When set, this bit indicates that there is a data segment following the next one. When clear the next data segment is the last in this data block.
MSILW0 <14:0>	Length of next segment. This field contains the number of bytes in the next data segment.

3.10.3.1.2 MSI Link Word 1

A definition of the bit field of MSI link word 1 (MSILW1) is given below.

Data Bit	Definition
MSILW1 <15:0>	Address of next segment. This field contains bits <17:2> of the next quadword aligned data segment.

Each segment of data must be preceded by the above described link. The number of linked segments is only limited by the maximum size of the data block (4K bytes) (Figure 3-70).

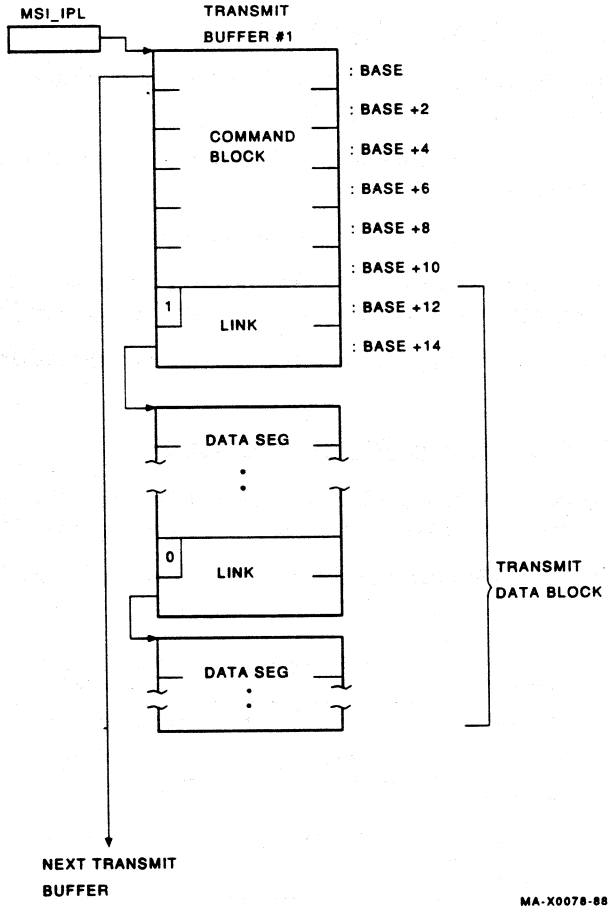


Figure 3-70 Initiator Operation

When the KA640 is the initiator, the SII chip uses the MSI_IPL register to determine the address of the transmit buffer to be used for this DSSI

transaction. As the SII chip is processing a transmit buffer, it loads an internal register with the second word of the link word as long as the LNK is enabled. This chaining continues until a LNK value of zero is encountered. The SII will then transfer the next segment and deposit the status of the entire transfer in the status area of the command block. The MSI_IPL register is then loaded with the buffer's thread word (the first word in the command block) for the next initiator operation. If an error of any kind occurs during the processing of a transmit buffer, the SII will stop the transmit operation by clearing the output enable bit MSI_DSCTRL <14>.

3.10.4 Adding To A Buffer List

The following enumerates the method required to dynamically add new buffers to the MSI_TLP and MSI_IPL lists:

1. Fill in the new buffer command block and ensure that the MSB of the status word is zero.
2. Make the thread word of the new buffer zero.
3. Replace the thread word of the last item on either the MSI_IPL or MSI_TLP list with the new thread word, pointing to the new buffer.
4. If the MSI_IPL or MSI_TLP is zero, load it with the address of the new buffer.

3.10.5 MSI Command Block (MSICB)

The MSI command block is a 12 byte data structure that the processor has to build at the start of all transmit and receive buffers in the MSI buffer RAM.

The format for the MSI command block is shown in Figure 3-71.

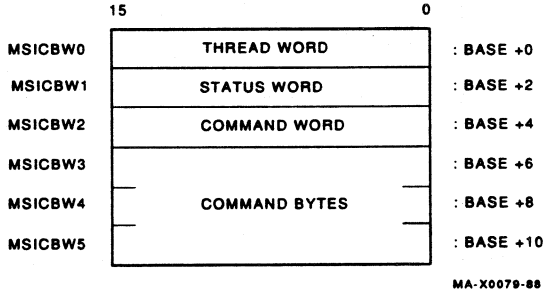


Figure 3-71 MSI Command Block

3.10.5.1 MSI Command Block Word 0

Word 0 of the MSI command block (MSICBW0), also referred to as the thread word, resides in the MSI buffer RAM at the base address of the MSI command block. The thread word contains bits <17:2> of the base address of the next buffer. Bit 0 of this field must always be set to 0, since buffers must start on a quadword boundary. A thread word of zero indicates that there are no more buffers.

The format for MSICBW0 is shown in Figure 3–72.



Figure 3–72 MSI Command Block Word 0

3.10.5.2 MSI Command Block Word 1

Word 1 of the MSI command block (MSICBW1), also referred to as the status word, resides in the MSI buffer RAM at the base+2 address of each MSI command block. This word indicates the status of the current DSSI transaction and is used by the processor to find out which buffers the SII chip has finished processing.

The format for MSICBW1 is shown in Figure 3–73.

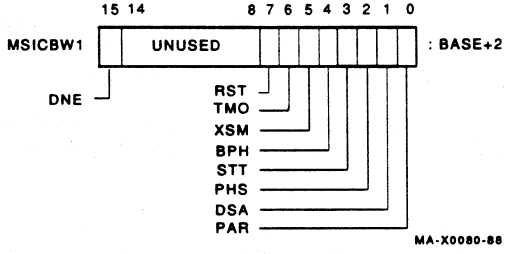


Figure 3–73 MSI Command Block Word 1

The bit fields in MSICBW1 represent the following:

Data Bit	Definition
MSICBW1<15>	(DNE) Done. When set, this indicates that the SII chip has used this buffer (either successfully or not). When clear the SII chip has not used this buffer. Note, if this bit is set when the SII chip begins processing a buffer, the buffer is not used.
MSICBW1<14:8>	Unused.
MSICBW1<7>	(RST). Reset. When set, a DSSI device reset the DSSI bus during this buffer's transaction. **NOTE** **If a DSSI bus reset occurred before the SII chip reached status in phase, the SII chip will clear MSI_DSCTRL <7> (output enable bit) and interrupt the processor without writing any status.**
MSICBW1<6>	(TMO) Timeout. When set, one of the MSI_DSTMO timers has expired. **NOTE** **If the timeout occurred before the SII chip reached status in phase, the SII chip will clear MSI_ DSCTRL<7> (output enable bit) and interrupt the processor without writing any status.**
MSICBW1<5>	(XSM) Checksum. When set, the received checksum does not agree with that computed by the SII chip. Note the XSM bit is only valid when the KA640 is a target.
MSICBW1<4>	(BPH). Bad phase. When set, an illegal DSSI phase was entered by the target. Note the BPH bit is only valid when the KA640 is the initiator.
MSICBW1<3>	(STT) Status. When set, ACK was not returned by the target. Note the STT bit is only valid when the KA640 is the initiator.
MSICBW1<2>	(PHS) Phase. When set, the DSSI bus phase changed before the initiator expected. Note the PHS bit is only valid when the KA640 is the initiator.
MSICBW1<1>	(DSA). DSSI. When set, the target detected an error in the command bytes. Note the DSA bit is only valid when the KA640 is the target.

Data Bit	Definition
MSICBW1<0>	(PAR). Parity. When set, a parity error on the DSSI bus was detected.

Please note that the following cases will not cause status to be written in memory:

- DSSI bus reset occurs before status in phase is reached.
- Initiator selects a non-existent device (initiator timeout will cause a DSSI bus reset).
- Target disconnects from the DSSI bus before status in phase is reached.

3.10.5.3 MSI Command Block Word 2

Word 2 of the MSI command block (MSICBW2), also referred to as the command word, resides in the MSI buffer RAM at the base+4 address of each MSI command block. This word contains information regarding the transfer.

The format for MSICBW2 is shown in Figure 3-74.

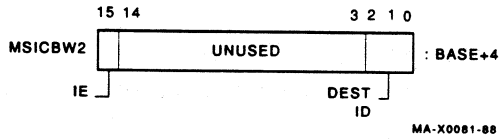


Figure 3-74 MSI Command Block Word 2

The bit fields in this memory word represent the following:

Data Bit	Definition
MSICBW2<15>	(IE). Interrupt enable. When set, the SII chip will interrupt the KA640 upon the completion (successful or not) of this transaction. When clear the SII chip will not generate an interrupt. Interrupts are posted at IPL14 with a vector offset of $C4_{16}$.
MSICBW2<14:3>	Unused.
MSICBW2<2:0>	(DEST ID). Destination ID. The ID of the target to be selected. This field is only used when the KA640 is the initiator.

3.10.5.4 MSI Command Block Words 3-5

Words 3-5 of the MSI command block (MSICBW3-5), also referred to as command bytes, reside in the MSI buffer RAM at the base+6 through base+10 address of each MSI command block. These 6 bytes are sent out during the command out phase by the initiator. Some of the information contained in these bytes are:

- The target and initiator IDs
- The number of data bytes which will be transferred by the initiator in the data out phase
- The DSSI opcode.

3.10.6 MSI Registers

The SII chip is very powerful and diverse. The KA640 does not use all its functionality. As a result of this the KA640 does not use all of the SII's twenty-seven processor visible registers. The following is a description of the 16 registers needed to control the SII chip during DSSI bus operations.

NOTE

The other 11 registers are not used during DSSI operations and should not be accessed.

3.10.6.1 MSI Control and Status Registers

These five registers are used to configure, control and monitor the SII chip.

3.10.6.1.1 MSI Control/Status Register

The mass storage interface control/status register (MSI_CSR), address 2008 460C $_{16}$, contains control and status information about the general operation of the SII chip in regard to the DSSI bus, including various enable bits. The format of the mass storage control/status register is shown in Figure 3–75.

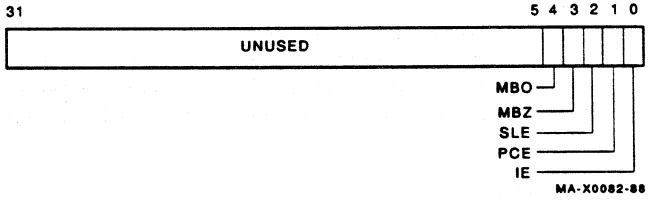


Figure 3–75 MSI Control/Status Register

Data Bit	Definition
MSI_CSR <31:5>	Unused. Reads return undefined results, writes have no effect.
MSI_CSR <4>	(MBO) Must be one. Read/Write. These bits must read as zero and be written as one.
MSI_CSR <3>	(MBZ) Must be zero. Read/Write. These bits must read as zero and be written as zero.
MSI_CSR <2>	(SLE) Selections. Read/Write. When set, the SII chip will respond to selections. When clear the SII chip will not respond to an initiator trying to select it. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSR <1>	(PCE) Parity check. Read/Write. When set, the SII chip reports parity errors. When clear the SII chip will continue to check parity but will not report any errors during the status in phase. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSR <0>	(IE) Interrupt enable. Read/Write. When set, interrupts are enabled. The SII chip posts interrupts when an error occurs or at the end of a transaction (successful or not). Interrupts are posted at IPL14 with an offset of $C4_{16}$. When clear interrupts are disabled. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

3.10.6.1.2 MSI DSSI Control Register

The mass storage interface DSSI control register (MSI_DSCTRL), address $2008\ 4644_{16}$, contains information to control the SII chip. The format of the mass storage interface DSSI control register is shown in Figure 3-76.

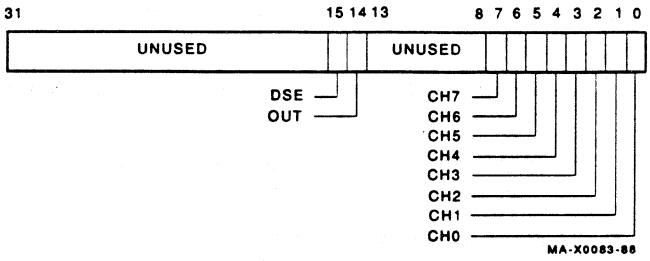


Figure 3-76 MSI DSSI Control Register

Data Bit	Definition
MSI_DSCTRL <31:16>	Unused. Reads return undefined results. Writes have no effect.
MSI_DSCTRL <15>	(DSE) DSSI Enable. Read/Write. This bit must be set to one by the processor for the SII chip to work on a DSSI bus. This bit is cleared by the SII chip if: the SII chip selects or is selected by a non-DSSI device, the SII chip is selected with Attention. It is also cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DSCTRL <14>	(OUT) Output enable. Read/Write. When set, the SII chip is enabled to send transmit buffers. This bit is cleared by the SII chip if: the MSI_IPL becomes zero, the initiator timer MSI_DSTMO <3:0> expires, or a transmit buffer is not terminated with ACK. It is also cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DSCTRL <13:8>	Unused. Reads return undefined results, writes have no effect.
MSI_DSCTRL <7>	(CH7) Channel 7. Read/Write. This bit is used to determine if device 7 is an DSSI device. This bit must be set to one by the processor. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

Data Bit	Definition
MSI_DSCTRL <6>	(CH6) Channel 6. Read/Write. This bit is used to determine if device 6 is a DSSI device. This bit must be set to one by the processor. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DSCTRL <5>	(CH5) Channel 5. Read/Write. This bit is used to determine if device 5 is a DSSI device. This bit must be set to one by the processor. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DSCTRL <4>	(CH4) Channel 4. Read/Write. This bit is used to determine if device 4 is a DSSI device. This bit must be set to one by the processor. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DSCTRL <3>	(CH3) Channel 3. Read/Write. This bit is used to determine if device 3 is a DSSI device. This bit must be set to one by the processor. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DSCTRL <2>	(CH2) Channel 2. Read/Write. This bit is used to determine if device 2 is a DSSI device. This bit must be set to one by the processor. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DSCTRL <1>	(CH1) Channel 1. Read/Write. This bit is used to determine if device 1 is a DSSI device. This bit must be set to one by the processor. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DSCTRL <0>	(CH0) Channel 0. Read/Write. This bit is used to determine if device 0 is a DSSI device. This bit must be set to one by the processor. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

3.10.6.1.3 MSI DSSI Connection Register

The mass storage interface DSSI connection register (MSI_CSTAT), address 2008 4648 $_{16}$, contains interrupt status related to SII chip connections. The format of the mass storage interface DSSI connection register is shown in Figure 3–77.

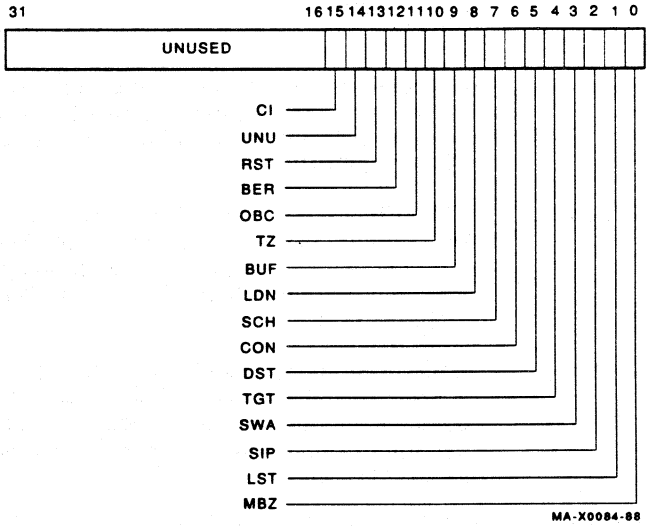


Figure 3–77 MSI DSSI Connection Register

Data Bit	Definition
MSI_CSTAT <31:16>	Unused. Reads return undefined results, writes have no effect.
MSI_CSTAT <15>	(CI) Composite interrupt. Read only. This bit is the composite error bit of the MSI_CSTAT register. It is the logical OR of bits MSI_CSTAT <13:11> and MSI_CSTAT <9:7>. When set, the processor will be interrupted at IPL14 with an offset of $C4_{16}$ if interrupts are enabled. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <14>	(UNU) Unused. Reads return undefined results, writes have no effect.
MSI_CSTAT <13>	(RST) Reset asserted. Read/Write one to clear. When set, the DSSI bus was reset by one of the eight DSSI devices. The SII chip will automatically disconnect itself from the bus and interrupt the processor at IPL14 with an offset of $C4_{16}$. This bit is write one to clear and is also cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <12>	(BER) Bus error. Read/Write one to clear. This bit is set to one on any of the following conditions: • Buffer overflow • Req/Ack offset exceeded • Illegal phase change While this bit is asserted, the SII chip will not receive or transmit data. This bit is write one to clear and is also cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

Data Bit	Definition
MSI_CSTAT <11>	(OBC) OUT_EN Bit cleared. Read/Write one to clear. This bit is set to one on any of the following conditions: • The SII chip has received RSTIN. (The DSSI bus has been reset). • The MSI_DSTMO (MSI_DSTMO <3:0> or MSI_DSTMO <7:4>) has expired. • As an initiator, the attached target disconnects unexpectedly. This bit is write one to clear and is also cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <10>	(TZ) Target pointer zero. Read only. When set, the MSI_TLP register contains a value of zero. This bit is set on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <9>	(BUF) Buffer service. Read/Write one to clear. When set, the SII chip has begun processing a transmit buffer destined for non-DSSI device. Note, this bit should always be zero since all devices must be DSSI. This bit is write one to clear and is also cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <8>	(LDN) List element done. Read/Write one to clear. When interrupts are enabled, this bit is set if the SII chip has completed a buffer, successfully or not. This bit is write one to clear and is also cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <7>	(SCH) State change. Read/Write one to clear. Set if MSI_DSCTRL <15> is cleared causing the SII chip to leave DSSI mode. This bit is write one to clear and is also cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

Data Bit	Definition
MSI_CSTAT <6>	(CON) Connected. Read only. When set, the SII is connected to another device on the DSSI bus. Clear while the SII chip is not connected to another device on the DSSI bus. This bit is cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <5>	(DST) Destination. Read only. When set, the SII is the destination of the current transaction. In other words, this bit is set if the SII chip was selected by another device on the DSSI bus. This bit is cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <4>	(TGT) Target. Read only. When set, the SII chip is operating as a target during the current transaction. This bit is cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <3>	(SWA) Selected with attention. Read only. When set, the SII chip was selected with attention. This bit is write one to clear and is also cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <2>	(SIP) Selection in progress. Read only. When set, the SII chip is currently in a selection process. This is useful in determining if the desired target is unavailable. This bit is write one to clear and is also cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <1>	(LST) Lost. Read only. When set, the SII lost arbitration. It is cleared by the SII chip when it begins a selection process and on power-up the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_CSTAT <0>	(MBZ) Must be zero. Read Only. This bit will be read as zero.

3.10.6.1.4 MSI ID Register

The mass storage interface ID register (MSI_ID), address 2008 4610 $_{16}$, contains the three bit ID number of the KA640 on the DSSI bus. This value is placed on the DSSI bus during the selection phase so the target knows who selected it.

The format of the mass storage interface ID register is shown in Figure 3–78.

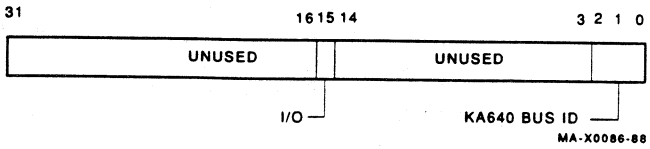


Figure 3–78 MSI ID Register

Data Bit	Definition
MSI_ID <31:16>	Unused. Reads return undefined results, writes have no effect.
MSI_ID <15>	(I/O) Input/Output. Read/Write. When set, the KA640's ID is determined by MSI_ID <2:0>. When clear the KA640's ID is determined by on board jumpers and MSI_ID <2:0> will reflect the one's complement of the KA640's DSSI ID. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET). Note that if this bit is cleared, writing to this register has no effect.
MSI_ID <14:3>	Unused. Reads return undefined results, writes have no effect.
MSI_ID <2:0>	KA640 Bus ID. Read/Write. When MSI_ID <31> is clear (the normal operation configuration), this field contains the DSSI ID of the KA640, as determined by the on board jumpers. When MSI_ID <31> is set, any DSSI ID value may be input (used, for example, to temporarily override the on board jumpers for test or diagnostic purposes). Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

3.10.6.1.5 MSI DSSI Timeout Register

The mass storage interface DSSI timeout register (MSI_DSTMO), address $2008\,461C_{16}$, contains the timeout values of the SII chip for both the initiator and target roles. Also contained in this register is a single enable bit that governs both timers.

The format of the mass storage interface DSSI timeout register is shown in Figure 3–79.

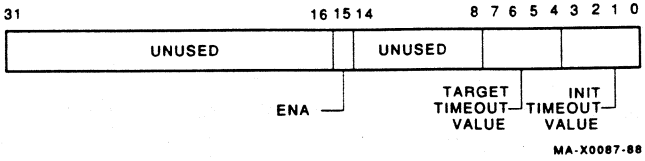


Figure 3–79 MSI DSSI Timeout Register

Data Bit	Definition
MSI_DSTMO <31:16>	Unused. Reads return undefined results, writes have no effect.
MSI_DSTMO <15>	(ENA) Enable. Read/Write. When set, both the DSSI target and DSSI initiator timers are enabled. When clear, both the DSSI target and DSSI initiator timer are disabled. Cleared on power up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DSTMO <14:8>	Unused. Reads return undefined results, writes have no effect.
MSI_DSTMO <7:4>	Target timeout value. Read/Write. This field contains the number of 200 microsecond intervals which may elapse while the KA640 is the target. The timer starts from the point when the KA640 was selected ends at the next observed bus free phase. Cleared on power_up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DSTMO <3:0>	Initiator timeout value. Read/Write. This field contains the number of 200 microsecond intervals which may elapse, from the last observed bus free phase, until the next observed bus free phase, while the KA640 is in the initiator role; or the number of 200 microsecond intervals which may elapse before the KA640, acting as a potential initiator, detects a bus free phase. Should the timer expire under either of these two conditions the SII chip will assert a DSSI bus reset. Cleared on power_up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

3.10.6.2 List Pointer Registers

These are the two registers used as address pointers for next incoming and outgoing data buffers.

3.10.6.2.1 MSI Target List Pointer Register

The mass storage interface target list pointer register (MSI_TLP), address 2008 463C $_{16}$, contains the address to which the SII chip will write the next free receive buffer. The SII chip will automatically reload the register with the receive buffer's thread word upon completion of the current transaction. Note this register must contain bits <17:2> of a quadword aligned address, therefore bit 0 will always be zero. The SII chip will interpret an address of 0000 $_{16}$ as the end of a linked list.

The format of the mass storage interface target list pointer register is shown in Figure 3-80.

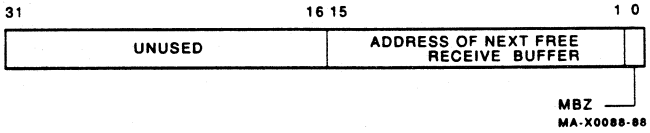


Figure 3-80 MSI Target List Pointer Register

Data Bit	Definition
MSI_TLP <31:16>	Unused. Reads return undefined results, writes have no effect.
MSI_TLP <15:1>	Address of next incoming buffer. Read/Write. This field contains bits 17:3 of the quadword aligned address to where the SII chip will find the next free receive buffer. Cleared on powerup, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_TLP <0>	(MBZ) Must be zero. Read/Write. This bit is read as zero and must be written as zero.

NOTE

This register can only be written by the processor when the register is zero or MSI_DSCTRL <15> (DSE) is clear; all other attempts to write to this register have no effect.

3.10.6.2.2 MSI Initiator List Pointer Register

The mass storage interface initiator list pointer register (MSI_IPL), address 2008 4640 $_{16}$, contains the address from which the SII chip will find the next transmit buffer. The SII chip will automatically reload this register with the transmit buffer's thread word upon completion of the current transaction. Note this register must contain bits <17:2> of a quadword aligned address, therefore bit 0 must always be zero. The SII chip will interpret an address of 0000 $_{16}$ as the end of a linked list. The format of the mass storage interface initiator list pointer register is shown in Figure 3–81.

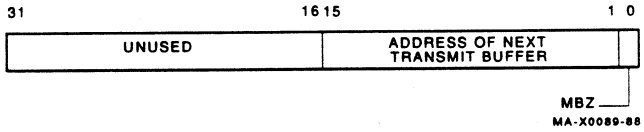


Figure 3–81 MSI Initiator List Pointer Register

Data Bit	Definition
MSI_IPL <31:16>	Unused. Reads return undefined results, writes have no effect.
MSI_IPL <15:1>	Address of next outgoing buffer. Read/Write. This field contains bits 17:3 of the quadword aligned address of where the SII chip will find the next transmit buffer. Cleared on power_up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_TLP <0>	(MBZ) Must be zero. Read/Write. This bit will read as zero and must be written as zero.

NOTE

This register can only be written to by the processor when the register is zero or MSI_DSCTRL <15> (DSE) is clear; all other attempts to write to this register have no effect.

3.10.6.3 Diagnostic and Test Registers

This group of registers is used for test and diagnostic purposes only. They should never be used during normal operation.

3.10.6.3.1 MSI Diagnostic Control Register

The mass storage interface diagnostic control register (MSI_DICTRL), at address 2008 4654 $_{16}$, allows the SII chip to be placed in one of three diagnostic test modes. The format of the mass storage interface diagnostic control register is shown in Figure 3-82.

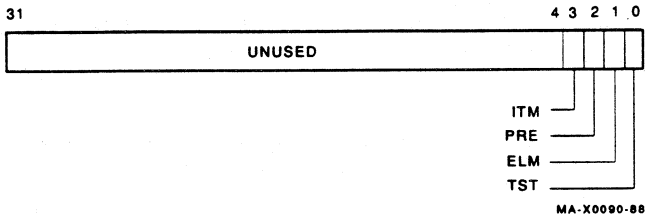


Figure 3-82 MSI Diagnostic Control Register

Data Bit	Definition
MSI_DICTRL<31:4>	Unused. Reads return undefined results. Writes have no effect.
MSI_DICTRL<3>	(ITM) Internal test mode. Read/Write. When set, the values written to MSI_DR0, MSI_DR1 and MSI_DR2 are to be looped back into the chip. This will enable the processor to insert test vectors into the chip during power-up diagnostics. Note that the MSI_DICTRL<1> (ELM) must be deasserted for this test to be meaningful. This bit is cleared on power-up, the negation of DCOK when SCR<7> is clear or writes to IPR55 (IORESET).
MSI_DICTRL<2>	(PRE) Port enable. Read/Write. When set, the off-chip drivers to the DSSI port are enabled. After a reset, the KA640 will be disconnected from the bus (this bit will be zero). The primary purpose of this bit is to allow SII chip diagnostics to run without affecting the rest of the DSSI bus (PRE=0). This bit is cleared on power-up, the negation of DCOK when SCR<7> is clear or writes to IPR55 (IORESET).

Data Bit	Definition
MSI_DICTRL<1>	(ELM) External loopback mode. Read/Write. When set, the SII chip is in external loopback mode. In this mode, MSI_DR0, MSI_DR1 and MSI_DR2 are used to directly control the DSSI data and control lines, as well as the external bus transceiver. Note an external loopback connector must be in place when using this test mode. This bit is cleared on power-up, then negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DICTRL<0>	(TST) Test mode. Read/Write. When set, to one (1), the SII chip is in test mode. This enables the user to replace the 20 MHz clock. The new clock is pulsed each time the MSI_CLOCK register is written. This bit is cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

3.10.6.3.2 MSI Diagnostic Register 0

The mass storage interface diagnostic register 0 (MSI_DR0), address 2008 4600 $_{16}$ is used during internal and external loopback diagnostic tests. The fields in this register are used to emulate the data lines of the DSSI.

The format of mass storage interface diagnostic register 0 is shown in Figure 3-83.

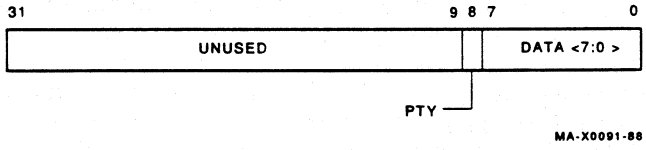


Figure 3-83 MSI Diagnostic Register 0

Data Bit	Definition
MSI_DR0 <31:9>	Unused. Reads return undefined results, writes have no effect.
MSI_DR0 <8>	(PTY) Parity. Read/Write. This bit contains the parity bit for the data byte MSI_DR0 <7:0>. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET). Note, parity checking is only enabled if MSI_CSR <1> PCE is set to 1. The SII chip chip uses odd parity checking.
MSI_DR0 <7:0>	(DATA) Read/Write. This field contains the current byte on the data bus. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

NOTE

This register should NOT be used during normal operation.

3.10.6.3.3 MSI Diagnostic Register 1

The mass storage interface diagnostic register 1 (MSI_DR1), address 2008 4604 $_{16}$ is used during internal and external loopback tests. In external loopback mode an external loopback connector in place allows values written into MSI_DR0 to be read back in MSI_DR1 and values written into MSI_DR1 to be read back in MSI_DR0. In internal loopback mode it acts as the DSSI bus emulating some of the DSSI control lines. Note that all the control lines are asserted high in internal loopback test mode. For more information on the SII chip modes see the description of the mass storage interface diagnostic control register (MSI_DICTRL) Section 3.10.6.3.1. The format of mass storage interface diagnostic register 1 is shown in Figure 3-84.

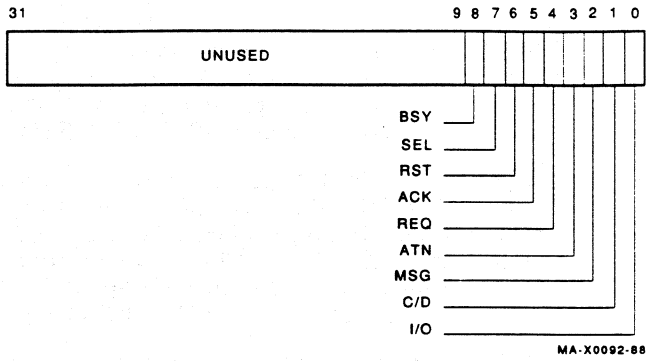


Figure 3–84 MSI Diagnostic Register 1

Data Bit	Definition
MSI_DR1 <31:9>	Unused. Reads return undefined results, writes have no effect.
MSI_DR1 <8>	(BSY) Busy. Read/Write. In internal loopback test mode, MSI_DICTRL <3> set, this bit emulates the DSSI BSY bus signal. In external loopback mode this bit is linked to MSI_DR0 <8> (PTY) for driver testing. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DR1 <7>	(SEL) Select. Read/Write. In internal loopback test mode, MSI_DICTRL <3> set, this bit emulates the DSSI SEL bus signal. In external loopback mode this bit is linked to MSI_DR0 <7> (DATA <7>) for driver testing. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DR1 <6>	(RST) Reset. Read/Write. In internal loopback test mode, MSI_DICTRL <3> set, this bit emulates the DSSI RST bus signal. In external loopback mode this bit is linked to MSI_DR0 <6> (DATA <6>) for driver testing. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

Data Bit	Definition
MSI_DR1 <5>	(ACK) Acknowledge. Read/Write. In internal loopback test mode, MSI_DICTRL <3> set, this bit emulates the DSSI ACK bus signal. In external loopback mode this bit is linked to MSI_DR0 <5> (DATA <5>) for driver testing. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DR1 <4>	(REQ) Request. Read/Write. In internal loopback test mode, MSI_DICTRL <3> set, this bit emulates the DSSI REQ bus signal. In external loopback mode this bit is linked to MSI_DR0 <4> (DATA <4>) for driver testing. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DR1 <3>	(ATN) Attention. Read/Write. In internal loopback test mode, MSI_DICTRL <3> set, this bit emulates the DSSI ATN bus signal. In external loopback mode this bit is linked to MSI_DR0 <3> (DATA <3>) for driver testing. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DR1 <2>	(MSG) Message. Read/Write. In internal loopback test mode, MSI_DICTRL <3> set, this bit emulates the DSSI MSG bus signal. In external loopback mode this bit is linked to MSI_DR0 <2> (DATA <2>) for driver testing. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DR1 <1>	(C/D) Control/Data. Read/Write. In internal loopback test mode, MSI_DICTRL <3> set, this bit emulates the DSSI C/D bus signal. In external loopback mode this bit is linked to MSI_DR0 <1> (DATA <1>) for driver testing. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DR1 <0>	(I/O) Input/Output. Read/Write. In internal loopback test mode, MSI_DICTRL <3> set, this bit emulates the DSSI I/O bus signal. In external loopback mode this bit is linked to MSI_DR0 <0> (DATA <0>) for driver testing. Indeterminate on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

NOTE

The data written to this register in internal test mode may differ from that read back from it, since only certain bits are driven when configured as a target or initiator. See the register description of MSI_DR2 for more information on the internal test mode.

3.10.6.3.4 MSI Diagnostic Register 2

The mass storage interface diagnostic register 2 (MSI_DR2), address 2008 4608 $_{16}$ is used by diagnostics to directly control the DC563 transceiver chip. The format of the mass storage interface diagnostic register 2 is shown in Figure 3-85.

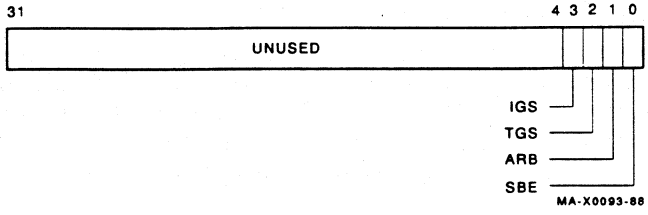


Figure 3-85 MSI Diagnostic Register 2

Data Bit	Definition
MSI_DR2 <31:4>	Unused. Reads return undefined results. Writes have no effect.
MSI_DR2 <3>	(IGS) Read/Write. This bit enables the DSSI bus drivers for ACK and ATN, placing the SII chip in the initiator role. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DR2 <2>	(TGS) Read/Write. When set, this bit enables the DSSI bus drivers for I/O, C/D, MSG and ATN, placing the SII chip in the target role. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DR2 <1>	(ARB) Arbitrate. Read/Write. This bit enables the decoding of ID0..ID2, putting the SII chip in the arbitration phase. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).
MSI_DR2 <0>	(SBE) Read/Write. When set, the DC563 transceiver drives the DSSI data bus and parity lines. Cleared on power-up, the negation of DCOK when SCR <7> is clear or writes to IPR55 (IORESET).

NOTE

Special care should be taken when writing to this register to avoid disturbing the DSSI bus during power-up diagnostics. This register should only be used when an external loopback connector is in place and not during normal operation.

3.10.6.3.5 MSI Clock Control Register (MSI_CLOCK)

Writing to the mass storage interface clock control register, address 2008 4658 $_{16}$, generates a pulse which, in test mode (MSI_DICTRL <0> set to one), replaces the 20 MHz clock input. This can be used to allow the CVAX CPU to observe and sequence the various state machines inside the SII chip. The format of the mass storage interface clock control register is shown in Figure 3-86.



Figure 3-86 MSI Clock Control Register

Data Bit	Definition
MSI_CLOCK <31:0>	Unused. Write only. Writing to this register generates a pulse which, in test mode (MSI_DICTRL <0> set to one), replaces the 20 MHz clock input.

3.10.6.3.6 MSI Internal State Registers (0-3)

These registers, at addresses 2008 465C $_{16}$, 2008 4660 $_{16}$, 2008 4664 $_{16}$, and 2008 4668 $_{16}$, reflect the status of the SII chip's internal state machine when used in test mode. (I.E. MSI_DICTRL <0> set to one.)

4

KA640 Firmware

This chapter describes the functional operation of the KA640 firmware. The KA640 firmware gains control of the processor whenever the KA640 performs a processor restart (also called a processor halt). A halt means only that the control is transferred to the firmware. It does not mean that the processor actually stops executing instructions.

4.1 KA640 Firmware Features

The firmware is located in two 64-Kbyte EPROMS on the KA640. The firmware image is duplicated in the local I/O space of the KA640 from 2004 0000 to 2007 FFFF inclusive. The firmware displays diagnostic progress and error reports on the KA640 LEDs and on the console terminal. It provides the following features:

- Automatic/manual operating system restart or bootstrap of customer application images at power-up, on reset, or conditionally after processor halts. (Restart in this context is not the same as restarting or resetting the hardware.)
- Automatic/manual bootstrap of an operating system on power-up.
- An interactive command language that allows the user to examine and alter the state of the processor.
- Diagnostic tests that test all components on the board and verify that the module is working correctly.
- Support of various terminals and devices as the system console.
- Multilingual support. The firmware can be configured to issue its messages in one of several languages.

In order for the console program to operate, the processor must be functioning at a level able to execute instructions from the console program ROM.

The firmware consists of four major parts:

1. Entry/dispatch code
2. Diagnostics
3. Console emulation
4. Virtual memory bootstrap (VMB)

The entry/dispatch code, located at physical address 2004 0000, is entered whenever a halt occurs. The processor can halt for a variety of reasons, including power-up. The reason for the halt is found in <13:08> of internal processor register (IPR) 43 (decimal) which is the SAVPSL. IPR 42 (decimal), which is the SAVPC, also contains the value in the program counter (PC) when the processor was halted. On power-up, the contents of SAVPC are undefined.

After a halt, the firmware saves the current LED code, then writes an "E" to the LED. This operation occurs within several instructions upon entry into entry/dispatch code. It indicates that at least several instructions have been successfully executed, although if the CPU is functioning properly, this occurs too quickly to be seen.

The entry code determines what action is to be taken based on the state of SAVPSL <13:08>, the halt enable bit, and the processor halt action (CPMBX <01:00>).

Table 4-1 lists the actions taken on a halt.

Table 4-1 Actions Taken on a Halt

Halt Enable	Power-up Halt?	Halt Action	Action
T*	T	X	Diagnostics, Halt
T	F	0	Halt
F	T	X	Diagnostics, Bootstrap, Halt
F	F	0	Restart, Bootstrap, Halt
X	F	1	Restart, Halt
X	F	2	Bootstrap, Halt
X	F	3	Halt

*T = condition is true, F = condition is false, X = doesn't matter

Multiple actions mean the next action is taken if and only if the previous action fails. Diagnostics are an exception; if diagnostics fail, then the processor enters console emulation.

Because the KA640 does not support battery backed-up main memory, a restart operation is not attempted on power-up. Restart in this context means the continuation of the operating system. Operating systems provide a mechanism for continuing operation after a halt condition.

The halt action is a 2-bit field (CPMBX <01:00>) used by operating systems to force the firmware to enter console emulation, or to restart or reboot following a halt, regardless of the setting of the halt enable switch.

4.1.1 Power Up Processing

On power-up, the firmware performs several unique actions. It runs initial power-up tests (IPT), locates and identifies the console device, performs a language query, and runs the remaining diagnostics.

Power-up actions differ, depending on the state of the mode switch on the H3602-SA (Figure 2-4), which has three settings: test, query, and normal. The differences are described in Sections 4.1.2 through 4.1.4.

The purpose of the IPT is to verify that the console private nonvolatile RAM (NVRAM) is valid (battery is charged). If it is invalid (battery is discharged), then the IPT tests and initializes the NVRAM. Prior to checking the NVRAM, the IPT waits for power to stabilize by monitoring SCR<5>(POK). Once power is stable, the IPT tests to see if the backup batteries failed by checking SSCCR<31>(BLO). If the batteries failed, then the IPT initializes certain nonvolatile data (such as the default boot device) to a known state. It then initializes other data structures and performs a processor initialization.

Table 4-2 lists the tests that correspond to the LED display.

4.1.2 Mode Switch Set to Test

The purpose of the test position on the H3602-SA is to verify that the connection between the KA640 and the console terminal is intact.

The firmware toggles between two states, active and passive. During the active state (3 seconds) the LED is set to 6. The firmware reads the baud rate and mode switch, then transmits and receives a character sequence.

During the passive state (7 seconds), the LED is set to 3.

If at any time the firmware detects an error (parity, framing, overflow or no characters), the display hangs at 6. If the configuration switch is moved from the test position, the firmware continues as if on a normal power-up.

4.1.3 Mode Switch Set to Language Inquiry

If the H3602-SA mode switch is set to language inquiry, or the firmware detects that the contents of NVRAM are invalid, the firmware queries the user for the language to be used for displaying critical system messages.

The language inquiry menu is shown in Example 4-1. If no response is received within 30 seconds, the firmware uses English as the console language.

```
 1) Dansk
 2) Deutsch (Deutschland/Osterreich)
 3) Deutsch (Schweiz)
 4) English (United Kingdom)
 5) English (United States/Canada)
 6) Espanol
 7) Francais (Canada)
 8) Francais (France/Belgique)
 9) Francais (Suisse)
10) Italiano
11) Nederlands
12) Norsk
13) Portugues
14) Suomi
15) Svenska
 (1..15):
```

Example 4-1 Language Selection Menu

After the language inquiry, the firmware proceeds as if the mode switch were set to normal.

4.1.4 Mode Switch Set to Normal

If the mode switch on the H3602-SA is set to normal, and the firmware detects that the contents of NVRAM are invalid then the language query menu is displayed, regardless of the setting of the mode switch.

If the mode switch is set to normal and the contents of NVRAM are valid, then the saved console language is used.

The firmware then prints out the banner message (first line of Example 4-2). The letter in the firmware revision number indicates whether the firmware is pre-field test (X), field test (T), or an official release (V).

```
KA640-A V4.1, VMB 2.4

Performing Normal System Tests

41..40..39..38..37..36..35..34..33..32..31..30..29..28..27..26..
25..24..23..22..21..20..19..18..17..16..15..14..13..12..11..10..
09..08..07..06..05..04..03..

Tests Completed
>>>
```

Example 4–2 Sample Output with No Errors

The first line contains the firmware revision (V4.1 in this example) and the virtual memory bootstrap (VMB) revision (V2.4 in this example).

Before a console is established, the only error reporting is via the KA640 diagnostic LEDs (and any LEDs on other boards). Once a console has been established, all errors detected by the diagnostics are also reported by the console. When possible, the diagnostics issue an error summary on the console. For example, diagnostic test failures, if specified in the firmware script, produce an error display in the format shown in Example 4–3.

```
Performing normal system tests.
38..37..36..35..34..33..32..31..

?34    2 08 FF 00 0000

P1=00000000  P2=00000003  P3=00000031  P4=00000011  P5=00002000
P6=FFFFFFFF  P7=00000000  P8=00000000  P9=00000000 P10=2005438F
r0=00114B98  r1=FFFFFFFF  r2=2005D2F0  r3=55555555  r4=AAAAAAAA
r5=00000000  r6=AAAAAAAA  r7=00000000  r8=00000000 ERF=80000180
30..29..28..27..26..25..24..23..
22..21..20..19..18..17..16..15..14..13..12..11..10..09..08..07..
06..05..04..03..
Normal operation not possible.
```

Example 4–3 Sample Output with Errors

Errors are printed according to the following syntax:

```
Test  Severity  Error  De_error  Vector  Count
P1       P2       P3       P4       P5
P6       P7       P8       P9       P10
R0       R1       R2       R3       R4
R5       R6       R7       R8       hardware_register_summary
```

The fields have the following meaning:

- *Test* identifies the diagnostic test.
- *Severity* is the severity level of a test failure, as dictated by the script. Failure of a severity level 2 test causes the display of this five-line error printout, and halts an autoboot to console I/O mode. An error of severity level 1 displays the first line of the error printout, but does not interrupt an autoboot. Most tests have a severity level of 2.
- *Error* is two hex digits identifying, within 10 instructions, where in the diagnostic the error occurred. This field is also called the *subtestlog*.
- *De_error* is a code with which the diagnostic executive signals the diagnostic's state and any illegal behavior. This field indicates a condition that the diagnostic expects on detecting a failure. FE or EF in this field means that an unexpected exception or interrupt was detected. FF indicates an error as a result of normal testing, such as a miscompare. The possible codes are:

  FF - Normal error exit from diagnostic
  FE - Unanticipated interrupt
  FD - Interrupt in cleanup routine
  FC - Interrupt in interrupt handler
  FB - Script requirements not met
  FA - No such diagnostic
  EF - Unanticipated exception in executive

- *Vector* identifies the SCB vector (10 in the example above) through which the unexpected exception or interrupt trapped, when the *de_error* field detects an unexpected exception or interrupt (FE or EF).
- *Count* is four hex digits. It shows the number of previous errors that have occurred.

Lines 2 and 3 of the error printout are parameters 1 through 10. An exception to this format is when an unexpected exception or machine check occurs during the executive. In that case, the stack is saved in the parameters.

Lines 4 and 5 of the error printout are general registers R0 through R8 and the hardware error summary register.

4.1.5 KA640 ROM-Based Diagnostics

The ROM-based diagnostics are the primary tools for troubleshooting and testing of the CPU, memory, Ethernet and DSSI subsystems.

The diagnostics run automatically on power-up. While the diagnostics are running, the LEDs on the H3602-SA display a hexadecimal countdown of the tests from F to 3 before booting the operating system, and 2 to 0 while booting the operating system. A different countdown appears on the console terminal. Table 4–2 lists the LED codes at power-up.

The ROM-based diagnostics are a collection of individual tests with user selectable *parameters*. A data structure called a *script* points to the tests.

A program called the *diagnostic executive* determines which of the available scripts to invoke, depending on the environment of the KA640 CPU (manufacturing or non-manufacturing). The diagnostic executive interprets the script to determine what tests to run, the right order to run the tests, and the right parameters to use for each test.

The diagnostic executive also controls tests so that errors can be detected and reported. It also ensures that when the tests are run, the machine is left in a consistent and well-defined state.

Table 4–2 shows a list of the ROM-based tests and their parameters. To get a similar listing, enter T 9E (T is the abbreviation of TEST) at the console prompt.

Each test accepts up to ten parameters. The asterisks (*) represent parameters that are used by the tests but that you cannot specify individually. These parameters are encoded in ROM and are provided by the diagnostic executive.

Parameters that you can specify are written out, as shown in the following examples:

```
54  2004E557   Virtual mode     ******
30  20053C6D   MEM_bitmap       *** Mark_hard_SBEs ******
```

The virtual mode test on the first line above contains several parameters, but you cannot specify any of them. To run this test individually, enter:

```
>>>T  54
```

The MEM_bitmap test on the second line above accepts ten parameters, but the fourth one (Mark_hard_SBEs) is the only one that you can specify. To mark pages bad in the bitmap for single or multiple bit errors, enter a 1 in the fourth parameter field as shown:

```
>>>T 59 0 0 0 1
```

You must enter a value of either 0 or 1 for the first three parameters (0 is used in this example). The values have no effect on the test; they are simply place holders for the first three parameters. You do not have to specify a value for parameters that follow the user-defined parameter.

When running tests interactively on an individual basis, users should be aware that certain tests may be dependent on some state set up from a previous test. In general, tests should not be run out of order.

Table 4–2 Diagnostic Tests and LED Codes

Test Number	LED Code	Test Name	Parameters
C1	C	SSC RAM	*
C2	C	SSC RAM ALL	*
C5	C	SSC regs	*
C6	C	SSC_powerup	**********
C7	C	CBTCR timeout	***
34	B	ROM logic test	*
33	A	CMCTL_ powerup	*
32	A	CMCTL regs	MEMCSR0_addr *********
91	9	CQBIC_powerup	**
90	9	CQBIC regs	*
80	8	CQBIC-memory	*********
60	6	Console serial	Start_baud end_baud ******
61	6	Console QVSS	Mark_not_present ***
62	6	Console QDSS	Mark_not_present selftest_r0 selftest_r1 *****
63	6	QDSS self-test	Input_csr selftest_r0 selftest_r1 ******
51	5	CFPA	*****
52	5	Prog timer	Which_timer wait_ time_us ***
53	5	TOY clock	Repeat_count_250ms_ea ****
54	5	Virtual mode	****

Table 4–2 (Cont.) Diagnostic Tests and LED Codes

Test Number	LED Code	Test Name	Parameters
55	5	Interval timer	*
56	5	SII_ext_loopbck	***
5C	5	SII_initiatior	******
5D	5	SII target	*******
5A	5	VAX CMCTL CDAL	Dont_report_memory_bad repeat_count *
57	5	SII_memory	Incr test_pattern *****
5B	5	SII_registers	****
5E	5	NI_memory	Incr data_pattern ***
5F	5	NI_Test	Do_extl ********
41	4	Board Reset	**
44	4	Cache_memory	Addr_incr *********
45	4	Cache_mem_ cqbic	Start_addr end_addr addr_incr ****
46	4	Cache1_diag_md	Addr_incr *********
31	A	MEM_Setup_ CSRs	**********
30	A	MEM_Bitmap	*** Mark_Hard_SBEs ******
4F	4	MEM_Data	Start_add end_add add_incr cont_on_ err ******
4E	4	MEM_Byte	Start_add end_add add_incr cont_on_ err ******
4D	4	MEM_Address	Start_add end_add add_incr cont_on_ err ******
4C	4	MEM_ECC_Error	Start_add end_add add_incr cont_on_ err ******
4B	4	MEM_Maskd_ Errs	Start_add end_add add_incr cont_on_ err ******
4A	4	MEM_Correction	Start_add end_add add_incr cont_on_ err ******
49	4	MEM_FDM_Logic	*** Cont_on_err ******
48	4	MEM_Addr_shrts	Start_add end_add * cont_on_err pat1 pat2 ****
47	4	MEM_Refresh	Start end incr cont_on_err time_ seconds *****
40	4	MEM_Count_ Errs	First_board Last_board ******* Soft_ errs_allowed

4.2 Halts

The H3602-SA, which contains the console baud rate, console serial line connector and language inquiry switches, is read by the firmware only when the processor halts. For this reason, changing the baud rate or any other switch on the panel will not take effect until the next power up. This is different from the KA630, in which the switches are hardwired into the hardware.

Note that a powerup is a halt condition, so that on powerup, the panel settings will correctly configure the hardware. The firmware may poll these switches at more frequent intervals, but the connection is guaranteed to be made only on a halt.

Once the firmware gives control to the operating system, the connection between the panel's switches and the hardware is lost, and changing the switches has no effect. Current Digital operating systems do not read the switches on the panel.

4.2.1 External Halts

Several conditions can trigger an external halt (SAVPSL<13:8>(HALT_CODE) = 2), and different actions are taken depending on the condition. An external halt can be caused by:

1. The halt enable switch is set to enable, and you press Break on the system console terminal. The firmware identifies a console Break condition by checking the RXDB<BRK> bit in the SSC.
2. Assertion of the BHALT line on the Q-bus. The halt is delivered to the processor if the BHALT ENB bit in the CQBIC is set.
3. Negation of DCOK. A halt is delivered if the processor is not running out of halt protected space, and the BHALT ENB bit is set. The system restart switch negates DCOK. DCOK may also be negated by the DELQA sanity timer, or any other Q22-bus module that chooses to implement the QBUS restart/reboot protocol.

When in console I/O mode, the KA640 cannot detect the negation of DCOK, so no action is taken. More importantly, however, the negation of DCOK destroys system state without notifying the firmware.

CAUTION

Do not press the restart button while in console I/O mode. Doing so will destroy system state without notifying the firmware.

The action taken by the firmware on a console Break or Q22-bus BHALT is the same: the firmware enters console I/O mode if halts are enabled.

The firmware, which runs in halt protected space after it is halted, distinguishes between the negation of DCOK and BHALT by assuming that BHALT must be asserted for at least 10 msec, and that DCOK is negated for at most 9 μsec. To determine if the BHALT line is asserted, the firmware steps out into halt unprotected space after 9 msec. If the processor halts again, the firmware concludes that the halt was caused by the BHALT and not the negation of DCOK. The firmware keeps a halt in progress flag to tell if it is halting due to the stepping out into halt unprotected space. This flag is cleared on power-up.

4.2.2 Determination Of The Console Device

After the battery check, the firmware tries to find out where and what the system console is. Normally, the system console is whatever terminal is attached to the console serial line.

4.3 Console Emulation

The system is by definition halted when the firmware is in control of the KA640. When halted, the KA640 provides most of the services of a standard VAX console.

4.3.1 Console Control Characters

In console I/O mode several characters have special meaning:

- Return—Also <CR>. The carriage return ends a command line. No action is taken on a command until after it is terminated by a carriage return. A null line terminated by a carriage return is treated as a valid, null command. No action is taken, and the console re-prompts for input. Carriage return is echoed as carriage return, line feed.
- Rubout—When you type rubout, the console deletes the previously typed character. What appears on the console terminal depends on whether it is a video or a hardcopy terminal.

  For hard copy terminals, the console echoes with a backslash (\), followed by the character being deleted. If you type additional rubouts, the additional deleted characters are echoed. If you type a non-rubout character, the console echoes another backslash, followed by the character typed. The result is to echo the characters deleted, surrounding them with backslashes. For example:

  EXAMI;E Rubout Rubout NE<CR>

  The console echoes: EXAMI;E\E;\NE<CR>

  The console sees the command line: EXAMINE<CR>

For video terminals, the previous character is erased from the screen and the cursor is restored to its previous position.

The console does not delete characters past the beginning of a command line. If you type more rubouts than there are characters on the line, the extra rubouts are ignored. If a rubout is typed on a blank line, it is ignored.

- Ctrl U—the console echoes ^U<CR>, and deletes the entire line. If Ctrl U is typed on an empty line, it is echoed, and otherwise ignored. The console prompts for another command.
- Ctrl S— stops output to the console terminal until Ctrl Q is typed. Ctrl S and Ctrl Q are not echoed. Ctrl C Ctrl O and Ctrl P also clear Ctrl S.
- Ctrl Q— resumes output to the console terminal. Additional Ctrl Qs are ignored. Ctrl S and Ctrl Q are not echoed.
- Ctrl O—causes the console to throw away transmissions to the console terminal until the next Ctrl O is entered. Ctrl O is echoed as ^O<CR> when it disables output, but is not echoed when it reenables output. Output is reenabled if the console prints an error message, or prompts for a command from the terminal. Displaying a REPEAT command does not reenable output. When output is reenabled for reading a command, the console prompt is displayed. Output is also enabled by entering program I/O mode, by Ctrl P and by Ctrl C. Ctrl O clears Ctrl Ss.
- Ctrl R—causes the console to echo <CR><LF> followed by the current command line. This function can be used to improve the readability of a command line that has been heavily edited.
- Ctrl C—causes the console to echo ^C and to abort processing of a command. Ctrl C clears Ctrl S and reenables output stopped by Ctrl O. When Ctrl C is typed as part of a command line, the console deletes the line as it does with Ctrl U.
- Ctrl P—if in console I/O mode, causes the console to echo ^P and to abort processing of a command. If the console is in program I/O mode and halt is disabled, Ctrl P is passed to the operating system.

4.3.2 Console Commands

This section describes the console I/O mode commands. Enter the commands at the console I/O mode prompt >>>.

4.3.2.1 Command Syntax

The console accepts commands up to 80 characters long. Longer commands produce error messages. The character count does not include rubouts, rubbed-out characters, or the Return at the end of the command.

You can abbreviate a command by dropping characters from the end of its keyword. Most commands can be recognized from their first character (Table 4–4).

The console treats two or more consecutive spaces and tabs as a single space. Leading and trailing spaces and tabs are ignored. You can place command qualifiers after the command keyword, or after any symbol or number in the command.

All numbers (addresses, data, counts) are hexadecimal, but symbolic register names number the registers in decimal. The hexadecimal digits are 0 through 9, and A through F. You can use uppercase and lowercase letters in hexadecimal numbers (A through F) and commands.

NOTE

The following conventions are used to describe command syntax:

[ ] indicates an optional command element

{ } indicates a command element

... indicates a list of command elements

4.3.2.2 Address Specifiers

Several commands take an address or addresses as arguments. In the context of the console, an address has two components: the address space, and the offset into that space. The console supports six address spaces:

- Physical memory (/P qualifier)
- Virtual memory (/V qualifier)
- General purpose registers (/G qualifier)
- Internal processor registers (/I qualifier)
- Protected memory (/U qualifier)
- The PSL (/M qualifier).

The address space that the console references is inherited from the previous console reference, unless explicitly specified. The initial address space is physical memory.

4.3.2.3 Symbolic Addresses

The console supports symbolic references to addresses. A symbolic reference simultaneously defines the address space, and the offset into that space. Table 4–3 lists symbolic references supported by the console, grouped according to address space.

Table 4–3 Console Symbolic Addresses

Symbol	Address	Symbol	Address
General Purpose Registers			
R0	0	R1	1
R2	2	R3	3
R4	4	R5	5
R6	6	R7	7
R8	8	R9	9
R10	0A	R11	0B
R12	0C	R13	0D
R14	0E	R15	0F
AP	0C	FP	0D
SP	0D	PC	0E
PSL	—	—	-
Internal Processor Registers			
PR$_KSP	00	PR$_ESP	01
PR$_SSP	02	PR$_USP	03
PR$_ISP	04	PR$_P0BR	08
PR$_P0LR	09	PR$_P1BR	0A
PR$_P1LR	0B	PR$_SBR	0C
PR$_SLR	0D	PR$_PCBB	10
PR$_SCBB	11	PR$_IPL	12
PR$_ASTLV	13	PR$_SIRR	14
PR$_SISR	15	PR$_ICCR	18
PR$_NICR	19	PR$_ICR	1A
PR$_TODR	1B	PR$_RXCS	20
PR$_RXDB	21	PR$_TXCS	22

Table 4-3 (Cont.) Console Symbolic Addresses

Symbol	Address	Symbol	Address
Internal Processor Registers			
PR$_TXDB	23	PR$_TBDR	24
PR$_CADR	25	PR$_MCESR	26
PR$_MSER	27	PR$_SAVPC	2A
PR$_SAVPSL	2B	PR$_IORESET	37
PR$_MAPEN	38	PR$_TBIA	39
PR$_TBIS	3A	PR$_SID	3E
PR$_TBCHK	3F		
Physical (VAX I/O Space)			
QBIO	2000 0000	QBMEM	3000 0000
QBMBR	2008 0010		
ROM	2004 0000		
CACR	2008 4000	BDR	2008 4004
DSCR	2008 0000	DSER	2008 0004
DMEAR	2008 0008	DSEAR	2008 000C
IPCR0	2000 1F40	IPCR1	2000 1F42
IPCR2	2000 1F44	IPCR3	2000 1F46
SSC_RAM	2014 0400	SSC_CR	2014 0010
SSC_CDAL	2014 0020	SSC_DLEDR	2014 0030
SSC_ AD0MAT	2014 0130	SSC_ AD0MSK	2014 0134
SSC_ AD1MAT	2014 0140	SSC_ AD1MSK	2014 0144
SSC_TCR0	2014 0100	SSC_TIR0	2014 0104
SSC_TNIR0	2014 0108	SSC_TIVR0	2014 010C
SSC_TCR1	2014 0110	SSC_TIR1	2014 0114
SSC_TNIR1	2014 0118	SSC_TIVR1	2014 011C
MEMCSR0	2008 0100	MEMCSR1	2008 0104
MEMCSR2	2008 0108	MEMCSR3	2008 010C
MEMCSR4	2008 0110	MEMCSR5	2008 0114
MEMCSR6	2008 0118	MEMCSR7	2008 011C
MEMCSR8	2008 0120	MEMCSR9	2008 0124
MEMCSR10	2008 0128	MEMCSR11	2008 012C
MEMCSR12	2008 0130	MEMCSR13	2008 0134
MEMCSR14	2008 0138	MEMCSR15	2008 013C

Table 4–3 (Cont.) Console Symbolic Addresses

Symbol	Address	Symbol	Address
Physical (VAX I/O Space)			
MEMCSR16	2008 0140	MEMCSR17	2008 0144
NISAROM	2008 4200	NIRDP	2008 4400
NIRAP	2008 4404	NIBUF	2012 0000
MSI_SBB	2008 4600	MSI_SC1	2008 4604
MSI_SC2	2008 4608	MSI_CSR	2008 460C
MSI_ID	2008 4610	MSI_SLCSR	2008 4614
MSI_DESTAT	2008 4618	MSI_DSTMO	2008 461C
MSI_DATA	2008 4620	MSI_ DMCTRL	2008 4624
MSI_ CMLOTC	2008 4628	MSI_ DMADDRL	2008 462C
MSI_ DMADDRH	2008 4630	MSI_ DMABYTE	2008 4634
MSI_STLP	2008 4638	MSI_LTLP	2008 463C
MSI_ILP	2008 4640	MSI_DSCTRF	2008 4644
MSI_CSTAT	2008 4648	MSI_DSTAT	2008 464C
MSI_COMM	2008 4650	MSI_DICTRL	2008 4654
MSI_CLOCK	2008 4658	MSI_BHDIAG	2008 465C
MSI_SIDIAG	2008 4660	MSI_ DMDIAG	2008 4664
MSI_ MCDIAG	2008 4668	MSI_RAM	2010 0000
Any Address Space			

*	The location last referenced in an examine or deposit command.
+	The location immediately following the last location referenced in an examine or deposit command. For references to physical or virtual memory spaces, the location referenced is the last address, plus the size of the last reference (1 for byte, 2 for word, 4 for longword). For other address spaces, the address is the last address referenced plus one.

Table 4-3 (Cont.) Console Symbolic Addresses

Any Address Space	
–	The location immediately preceding the last location referenced in an examine or deposit command. For references to physical or virtual memory spaces, the location referenced is the last address minus the size of this reference (1 for byte, 2 for word, 4 for longword). For other address spaces, the address is the last addressed referenced minus one.
@	The location addressed by the last location referenced in an examine or deposit command.

4.3.2.4 Console Command Qualifiers

All qualifers in the console command syntax are global, that is, they may appear in any order on the command line after the command keyword.

All qualifiers have unique meanings throughout the console regardless of the command. For example, the /B qualifier always means byte.

The following qualifers are recognized by the console:

- /xxxx — xxxx an unsigned hexadecimal integer that is evaluated into a longword. An error message of VALUE TOO BIG occurs if the number overflows 32 bits. This qualifier is only used on the bootstrap command to specify the value that is to be put into R5.
- /N:xxxx —xxxx an unsigned hexadecimal integer that is evaluated into a longword. An error message occurs if the number overflows 32 bits. This qualifier determines the number of additional operations that are to take place on EXAMINE, DEPOSIT, MOVE and SEARCH commands.
- /R5:xxxx—Functionally equivalent to /xxxx.
- /WRONG—Used to override or set error bits when referencing main memory.
- /STEP:xxxx — The step qualifier overrides the default incrementing of the console current reference. Commands that manipulate memory, such as the examine, deposit, move and search commands normally increment the console current reference by the size of the data being used.
- /B — The data size is byte.
- /W—The data size word.

- /L—The data size is longword.
- /Q—The data size is quadword.
- /G—The address space the general register space.
- /I—The address space is the IPR space.
- /V—The address space is virtual memory.
- /P—The address space is physical memory
- /M—The address space is the PSL space.
- /RPB—Used as a qualfier on the FIND command to search for a restart parameter block (RPB).
- /U—Access memory locations that are normally protected from access.
- /MEM— On the FIND command, search for a good memory block.
- /NOT— invert the sense of the match on the search command.

4.3.2.5 Console Command Keywords

Table 4-4 lists command, parameter and qualifier keywords. Table 4-5 is a summary of the console commands. Table 4-6 is a summary of the console qualifiers.

Table 4-4 Command, Parameter, and Qualifier Keywords

Command Keywords

Processor Control	Data Transfer	Console Control
B*OOT	E*XAMINE	CONF*IGURE
C*ONTINUE	D*EPOSIT	F*IND
H*ALT	M*OVE	R*EPEAT
I*NITIALIZE	SEA*RCH	SET
N*EXT	X	SH*OW
S*TART		T*EST
U*NJAM		!

Table 4–4 (Cont.) Command, Parameter, and Qualifier Keywords

SET & SHOW Parameter Keywords

BO*OT	BF*LAG	DE*VICE
DS*SI	E*THERNET	H*OST
L*ANGUAGE	M*EMORY	Q*BUS
U*QSSP	V*ERSION	R*LV12

Qualifier Keywords

Data Control	Address Space Control	Command Specific
/B	/G	/IN*STRUCTION
/W	/I	/NO*T
/L	/P	/R5: or /
/Q	/V	/RP*B or /ME*M
/N:	/M	/F*ULL
/S*TEP:	/U	/DS*SI xor /U*QSSP
/WR*ONG		/DI*SK xor /T*APE
		/DU*P xor /MA*INTENANCE
		/SE*RVICE

* Indicates the minimum number of characters required to uniquely identify the keyword.

4.3.2.6 Conventions for Tables 4–5 and 4–6

The following is a list of conventions used in Tables 4–5 and 4–6.

- UPPERCASE denotes the command or qualifier keyword.
- {} denotes a mandatory item which must be syntactically correct.
- [] denotes an optional item.
- ! denotes an or condition.
- *bitmap, count, size, address, & parameters* denote hex longword values.
- *device_string* denotes a legal boot device string.
- *csr* denotes a Q22-bus I/O page CSR address.
- *node* denotes a DSSI node name upto 8 characters or number from 0 to 7.
- *controller_number* denotes a controller number from 0 to 255.

- *task* denotes a DUP or MAINTENANCE task.
- *language_type* denotes the language value, 1..15.
- *command* denotes a console command other than REPEAT.
- *data, pattern, & mask* denote hex values of the current size.
- *major* denotes hex byte test number.

Table 4–5 Console Command Summary

Command	Qualifiers	Argument	Other(s)
BOOT	/R5:{bitmap} /{bitmap}	[device_string]	—
CONTINUE	—	—	—
DEPOSIT	/B /W /L /Q /G /I /V /P /M /U /N:{count} /STEP:{size} /WRONG	{address}	{data} [data]
EXAMINE	/B /W /L /Q /G /I /V /P /M /U /N:{count} /STEP:{size} /WRONG /INSTRUCTION	[address]	—
FIND	/MEM /RPB	—	—
HALT	—	—	—
HELP	—	—	—
INITIALIZE	—	—	—
MOVE	/B /W /L /Q /V /P /U /N:{count} /STEP:{size} /WRONG	{src_address}	{dest_ address}
NEXT	—	[count]	—
REPEAT	—	{command}	—
SEARCH	/B /W /L /Q /V /P /U /N:{count} /STEP:{size} /WRONG /NOT	{start_address}	{pattern} [mask]
SET BFLAG	—	{bitmap}	—
SET BOOT	—	{device_string}	—
SET HOST	/DUP {/DSSI ! /UQSSP} {/DISK ! /TAPE ! csr} /MAINTENANCE /UQSSP {/SERVICE ! csr}	{node} ! {controller_ number}	[task]

Table 4–5 (Cont.) Console Command Summary

Command	Qualifiers	Argument	Other(s)
SET LANGUAGE	—	{language_ type}	—
SHOW BFLAG	—	—	—
SHOW BOOT	—	—	—
SHOW DSSI	—	—	—
SHOW ETHERNET	—	—	—
SHOW LANGUAGE	—	—	—
SHOW MEMORY	/FULL	—	—
SHOW QBUS	—	—	—
SHOW RLV12	—	—	—
SHOW UQSSP	—	—	—
SHOW VERSION	—	—	—
START	—	{address}	—
TEST	—	{major}	[parameters]
UNJAM	—	—	—
X	—	{address}	{count}

Table 4–6 Console Qualifier Summary

Data Control	
/B	Byte, legal for memory references only.
/W	Word, legal for memory references only.
/L	Longword, the default for GPR and IPR references.
/Q	Quadword, legal for memory references only.
/N:{count}	Specify number of additional operations.
/STEP:{size}	Override the default step incrementing size with the value specified for the current reference.
/WRONG	Use 1's complement of the ECC bits on writes to main memory. Ignore ECC errors on reads of main memory.

Table 4-6 (Cont.) Console Qualifier Summary

Address Space Control	
/G	General Purpose Registers
/I	Internal Processor Registers
/V	Virtual memory
/P	Physical memory, both VAX memory and I/O spaces
/U	Protected memory (ROMs, SSC RAM, PFN bitmap, etc.)
/M	Machine state (PSL)

Command Specific	
/INSTRUCTION	EXAMINE command only. Disassemble the instruction at the specified address.
/NOT	SEARCH command only. Invert the sense of the match.
/R5:{bitmap}, /{bitmap}	BOOT command only. Specify a function bitmap to pass to VMB through R5. Refer to the BOOT command listing for a bit description of R5. Either form of the command is acceptable.
/RPB, /MEM	FIND command only. Search for valid RPB or good block of memory.
/DUP, /DSSI, /UQSSP, /DISK, /TAPE, /MAINTENANCE, /SERVICE	SET HOST command only. Refer to command description for usage.

4.3.2.7 References to Processor Registers and Memory

The KA640 console is implemented by macrocode executing from EPROM. Actual processor registers can not be modified by the console command interpreter. When the console is entered, the console saves the processor registers in console memory. All command references to the processor registers are directed to the corresponding saved values, not to the registers themselves.

When the console reenters program mode, the saved registers are restored and any changes become operative only then. References to processor memory are handled normally. The binary load and unload command can not reference the console memory pages

The following registers are saved by the console. Any direct reference to these registers is intercepted by the console and redirected to the saved copies:

- R0—R15, the general purpose registers

- PR$_IPL, the interrupt priority level register
- PR$_SCBB, the system control block base register
- PR$_ISP, the interrupt stack pointer
- PR$MAPEN, the memory management enable register

The following registers are also saved, yet can be accessed directly through console commands. Writing values to these registers may make the console inoperative.

- PR$_SAVPC, the halt PC
- PR$_SAVPSL, the halt PSL
- ADxMCH/ADxMSK, the SSC address decode and match registers
- SSCCR, the SSC configuration register
- DLEDR, the SSC diagnostic LED register

4.3.2.8 BOOT

Format :

BOOT [qualifier-list] [{device_name}]

Description :

The console initializes the processor and transfers execution to VMB. VMB attempts to boot the operating system from the specified device or the default boot device if none is specified. The console qualifies the bootstrap operation by passing a boot flags bitmap to VMB in R5.

If either the qualifier or the device name is absent, then the default value is used. Explicitly stating the boot flags or the boot device overrides but does not reset the corresponding default value.

The default boot device and and boot flags are set in three ways:

1. The operating system can write a default boot device and flags into the appropriate locations in NVRAM.
2. You can set the default boot device and boot flags explicitly with the console SET BOOT and SET BFLAG commands respectively.
3. The console prompts you for the default boot device under any of the following conditions:
   - The power-up mode switch is set to query mode.

- The console detects that there is a dead battery, and therefore the contents of NVRAM are no longer valid.
- The console detects that you have not explicitly set the default boot device within 30 seconds (causing a device timeout) or neither (1) nor (2) has been performed.

  The console prompts the user for a default boot device on every powerup, until the request has been satisfied.

If no default boot device is specified on power-up, the console issues a list of bootable devices. The devices may or may not contain bootable images. The console then prompts you for a device name.

If you do not enter a device name within 30 seconds, the console times out and a default device name of ESA0 is used, although not stored in the NVRAM.

Qualifiers :

- /R5:{bitmap}—Bitmap is a 32 bit hex value that is passed to VMB in R5. The console does not interpret this value. You can specifiy a default boot flags longword by using the SET BFLAG command. You can display the longword with the SHOW BFLAG command. Table 4–8 lists the supported R5 boot flags.)
- /{bitmap}—Same as /R5:{bitmap})
- [{device_name}]—The device name can be any character string up to 39 characters long. Longer strings cause a VAL TOO BIG error message. Apart from length, the console makes no attempt to interpret or validate the device name. The console converts the string to upper case, then passes VMB a string descriptor to this device name in R0. The default boot device can be specified using the SET BOOT command and displayed with the SHOW BOOT command. The factory default device is the onboard Ethernet port, ESA0.)

Examples :

```
>>>show boot
ESA0
>>>show bflag
0
>>> b                    ! Boot using default boot flags and device.
(BOOT/R5:0 ESA0)

  2..
-ESA0
```

```
>>>b xqa0                    ! Boot from XQA0 using default boot flags.
(BOOT/R5:0 XQA0)

  2..
-XQA0
>>> b/10                     ! Boot using supplied boot flags
                             ! and default device.
(BOOT/R5:10 ESA0)

  2..
-ESA0
>>> boot /r5:220 xqa0        ! Boot using supplied boot
                             ! flags and device.
(BOOT/R5:220 XQA0)

  2..
-XQA0
```

4.3.2.9 CONFIGURE

Format :

CONFIGURE

Description :

The CONFIGURE command invokes an interactive mode that permits you to enter Q22-bus device names, then generates a table of Q22-bus I/O page device CSR addresses and interrupt vectors. This feature simplifies field configuration by providing information that is typically available only with a running operating system.

CONFIGURE is similar to the VMS SYSGEN CONFIG utility.

Qualifiers :

None.

Arguments :

None.

Examples :

```
>>>config
Enter device configuration, HELP, or EXIT
Device,Number? help
Devices:
 LPV11     KXJ11       DLV11J      DZQ11       DZV11       DFA01
 RLV21     TSV05       RXV21       DRV11W      DRV11B      DPV11
 DMV11     DELQA       DEQNA       RQDX3       KDA50       RRD50
 RQC25                 TQK50       TQK70       TU81E       RV20
           KMV11       IEQ11       DHQ11       DHV11       CXA16
 CXB16     CXY08       VCB01       QVSS        LNV11       LNV21
 QPSS      DSV11       ADV11C      AAV11C      AXV11C      KWV11C
 ADV11D    AAV11D      VCB02       QDSS        DRV11J      DRQ3B
 VSV21     IBQ01       IDV11A      IDV11B      IDV11C      IDV11D
 IAV11A    IAV11B      MIRA
Numbers:
 1 to 255, default is 1
Device,Number? rqdx3,2
Device,Number? dhv11
Device,Number? qdss
Device,Number? tqk50
Device,Number? tqk70
Device,Number? exit
Address/Vector Assignments
-772150/154 RQDX3
-760334/300 RQDX3
-774500/260 TQK50
-760444/304 TQK70
-760500/310 DHV11
-777400/320 QDSS
>>>
```

4.3.2.10 CONTINUE

Format :

CONTINUE

Description :

The processor begins instruction execution at the address currently contained in the program counter. Processor initialization is not performed. The console enters program I/O mode.

Qualifiers :

None.

Arguments :

None.

Examples :

```
>>> continue
```

4.3.2.11 DEPOSIT

Format :

DEPOSIT [qualifier_list] {address} {data} [{data}...]

Description :

Deposits data into the address specified. If no address space or data size qualifiers are specified, the defaults are the last address space and data size used in a DEPOSIT, EXAMINE, MOVE or SEARCH command. After processor initialization, the default address space is physical memory, the default data size is a longword and the default address is zero. If conflicting address space or data sizes are specified, the console ignores the command and issues an error message.

Qualifiers :

- **/B** —The data size is byte.
- **/W** —The data size is word.
- **/L** —The data size is longword.
- **/Q** —The data size is quadword.
- **/G** —The address space is the general purpose register set, R0 through R15. The data size is always long.
- **/I** —The address space is internal processor registers (IPRs). These are the registers only accessible by the MTPR and MFPR instructions. The data size is always long.
- **/M** —The address space is the PSL space.
- **/P** —The address space is physical memory.
- **/V** —The address space is virtual memory. All access and protection checking occur. If access to a program running with the current PSL is not allowed, the console issues an error message. Virtual space DEPOSITs cause the PTE<M> bit to be set. If memory mapping is not enabled, virtual addresses are equal to physical addresses.

- /U —Access to console private memory is allowed. This qualifier also disables virtual address protection checks. On virtual address writes, the PTE<M> bit will not be set if the /U qualifier is present. This qualifier is not inherited, and must be respecified on each command.
- /N:{count} —The address is the first of a range. The console deposits to the first address, then to the specified number of succeeding addresses. Even if the address is the symbolic address, the succeeding addresses are at larger addresses. The symbolic address specifies only the starting address, not the direction of succession. For repeated references to preceding addresses, use REPEAT DEPOSIT - <DATA>.
- /STEP:{size} —The number to add to the current address. Normally this defaults to the data size, but the /STEP qualifier overrides this default. This qualifier is not inherited.
- /WRONG —The data error correction bits will be written with the one's complement of the correct check bits.

Arguments :

- {address} —A longword address that specifies the first location into which data is deposited. The address can be an actual address or a symbolic address.
- {data} —The data to be deposited. If the specified data is larger than the deposit data size, the firmware ignores the command and issues an error response. If the specified data is smaller than the deposit data size, it is extended on the left with zeros.
- [{data}] —Additional data to be deposited (as many as can fit on the command line).

Examples :

```
>>> D/P/B/N:1FF 0 0          ! Clear first 512 bytes of physical
                             !  memory.
>>> D/V/L/N:3 1234 5         ! Deposit 5 into four longwords
                             ! starting at virtual memory
                             ! address 1234.
>>> D/N:8 R0 FFFFFFFF        ! Loads GPRs R0 through R8 with -1.
>>> D/N:200 - 0              ! Starting at previous address,
                             ! clear 513 bytes.
>>> D/L/P/N:10/S:200 0 8     ! Deposit 8 in the first longword of
                             ! the first 17 pages in physical memory.
```

4.3.2.12 EXAMINE

Format :

EXAMINE [qualifier_list] [{address}]

Description :

Examines the contents of the memory location or register specified by the address. If no address is specified, + is assumed. The display line consists of a single character address specifier, the hexadecimal physical address to be examined, and the examined data, also in hexadecimal.

EXAMINE uses the same qualifiers as DEPOSIT. However, the /WRONG qualifier causes examines to ignore ECC errors on reads from physical memory. The EXAMINE command also supports an /INSTRUCTION qualifier, which will disassemble the instructions at the current address.

Qualifiers :

- **/B** —The data size is byte.
- **/W** —The data size is word.
- **/L** —The data size is longword.
- **/Q** —The data size is quadword.
- **/G** —The address space is the general purpose register set, R0 through R15. The data size is always long.
- **/I** —The address space is internal processor registers (IPRs). These are the registers only accessible by the MTPR and MFPR instructions. The data size is always long.
- **/P** —The address space is physical memory. Note that when virtual memory is examined, the address space and address in the response are the translated physical address.
- **/V** —The address space is virtual memory. All access and protection checking occur. If the access would not be allowed to a program running with the current PSL, the console issues an error message. If memory mapping is not enabled, virtual addresses are equal to physical addresses.
- **/M** —The address space and display are the PSL. The data size is always long.
- **/U** —Access to console private memory is allowed. This qualifier also disables virtual address protection checks. This qualifier is not inherited, and must be respecified on each command.

- /N:{count} —The address is the first of a range. The console deposits to the first address, then to the specified number of succeeding addresses. Even if the address is the symbolic address -, the succeeding addresses are at larger addresses. The symbolic address specifies only the starting address, not the direction of succession.
- /STEP:{size} —The number to add to the current address. Normally this defaults to the data size, but the /STEP qualifier overrides this default. This qualifier is not inherited.
- /WRONG —ECC errors on this read access to main memory are ignored.
- /INSTRUCTION —Disassemble and display the VAX Macro-32 instruction at the specified address.

Arguments :

- [{address}] —A longword address that specifies the first location to be examined. The address can be an actual address or a symbolic address. If no address is specified, + is assumed.

Examples :

```
>>>ex pc                                  ! Examine the PC.
  G 0000000F FFFFFFFC
>>>ex sp                                  ! Examine the SP.
  G 0000000E 00000200
>>>ex psl                                 ! Examine the PSL.
  M 00000000 041F0000
>>>e/m                                    ! Examine PSL another way.
  M 00000000 041F0000
>>>e r4/n:5                               ! Examine R4 through R9.
  G 00000004 00000000
  G 00000005 00000000
  G 00000006 00000000
  G 00000007 00000000
  G 00000008 00000000
  G 00000009 801D9000
>>>ex pr$_scbb                            ! Examine the SCBB, IPR 17.
  I 00000011 2004A000
>>>e/p 0                                  ! Examine local memory 0.
  P 00000000 00000000
>>>ex /ins 20040000                       ! Examine 1st byte if ROM.
  P 20040000   11 BRB     20040019
>>>ex /ins/n:5 20040019                   ! Disassemble from branch.

  P 20040019   D0 MOVL    I^#20140000,@#20140000
  P 20040024   D2 MCOML   @#20140030,@#20140502
  P 2004002F   D2 MCOML   S^#0E,@#20140030
  P 20040036   7D MOVQ    R0,@#201404B2
  P 2004003D   D0 MOVL    I^#201404B2,R1
  P 20040044   DB MFPR    S^#2A,B^44(R1)
>>>e/ins                                 ! Look at next instruction.
  P 20040048   DB MFPR    S^#2B,B^48(R1)
>>>
```

4.3.2.13 FIND

Format :

FIND [qualifier-list]

Description :

The console searches main memory starting at address zero for a page-aligned 128 Kbyte segment of good memory, or a restart parameter block (RPB). If the segment or block is found, its address plus 512 is left in SP (R14). If the segment or block is not found, an error message is issued, and the contents of SP are preserved. If no qualifier is specified, /RPB is assumed.

Qualifiers :

- **/MEMORY** —Search memory for a page aligned block of good memory, 128 Kbytes in length. The search looks only at memory that is deemed usable by the bitmap. This command leaves the contents of memory unchanged.
- **/RPB** —Search all of physical memory for an RPB. The search does not use the bitmap to qualifiy which pages are looked at. The command leaves the contents of memory unchanged.

Arguments :

None.

Examples :

```
>>>ex sp                                  ! Check the SP.
  G 0000000E 00000000
>>>find /mem                              ! Look for a valid 128 Kb.
>>>ex sp                                  ! Note where it was found.
  G 0000000E 00000200
>>>find /rpb                              ! Check for valid RPB.
?2C FND ERR 00C00004                      ! None to be found here.
>>>
```

4.3.2.14 HALT

Format :

HALT

Description :

This command has no effect. It is included for compatability with other consoles.

Qualifiers :

None.

Arguments :

None.

Examples :

```
>>> halt                                  ! Pretend to halt.
>>>
```

4.3.2.15 HELP

Format :

HELP

Description :

This command has been included to help the console operator answer simple questions about command syntax and usage.

Qualifiers :

None.

Arguments :

None.

Examples :

```
>>>help
Following is a brief summary of all the commands
supported by the console:

        UPPERCASE  denotes a keyword that you must type in
        |          denotes an OR condition
        []         denotes optional parameters
        <>         denotes a field that must be filled in
                   with a syntactically correct value

Valid qualifiers:
    /B /W /L /Q /INSTRUCTION
    /G /I /V /P /M
    /STEP: /N: /NOT
    /WRONG /U
```

```
Valid commands:
    DEPOSIT [<qualifiers>] <address> [<datum>
[<datum>]]
    EXAMINE [<qualifiers>] [<address>]
    MOVE [<qualifiers>] <address> <address>
    SEARCH [<qualifiers>] <address> <pattern>
[<mask>]
    SET BFLAG <boot_flags>
    SET BOOT <boot_device>
    SET HOST/DUP/DSSI <node_number> [<task>]
    SET HOST/DUP/UQSSP </DISK | /TAPE> <controller_number> [<task>]
    SET HOST/DUP/UQSSP <physical_CSR_address> [<task>]
    SET HOST/MAINTENANCE/UQSSP/SERVICE <controller_number> [<task>]
    SET HOST/MAINTENANCE/UQSSP <physical_CSR_address> [<task>]
    SET LANGUAGE <language_number>
    SHOW BFLAG
    SHOW BOOT
    SHOW DEVICE
    SHOW DSSI
    SHOW ETHERNET
    SHOW LANGUAGE
    SHOW MEMORY [/FULL]
    SHOW QBUS
    SHOW RLV12
    SHOW UQSSP
    SHOW VERSION
    HALT
    INITIALIZE
    UNJAM
    CONTINUE
    START <address>
    REPEAT <command>
    X <address> <count>
    FIND [/MEMORY | /RPB]
    TEST [<test_code> [<parameters>]]
    BOOT [/R5:<boot_flags> | /<boot_flags>] [<boot_device>]
    NEXT [count]
    CONFIGURE
    HELP
>>>
```

4.3.2.16 INITIALIZE

Format :

INITIALIZE

Description :

A processor initialization is performed. The following registers are initialized:

Register	Initialized Value
PSL	041F 0000
IPL	1F
ASTLVL	4
SISR	0
ICCS	Bits <6> and <0> are clear, the rest are unpredictable
RXCS	0
TXCS	80
MAPEN	0
Cache memory	Disabled, all entries invalid
Instruction buffer	Unaffected
Console previous reference	Longword, physical, address 0
TODR	Unaffected
Main memory	Unaffected
General registers	Unaffected
Halt code	Unaffected
Bootstrap in progress flag	Unaffected
Internal restart in progress flag	Unaffected

The firmware performs the following additional initialization:

- The CDAL bus timer is initialized.
- The address decode and match registers are initialized.
- The programmable timer interrupt vectors are initialized.
- The BDR registers are read to determine the baud rate, and then the SSCCR is configured accordingly.
- All error status bits are cleared.

Qualifiers :

None.

Arguments :

None.

Examples :

```
>>>init
>>>
```

4.3.2.17 MOVE

Format :

MOVE [qualifier-list] {src_address} {dest_address}

Description :

The console copies the block of memory starting at the source address to a block beginning at the destination address. Typically, this command has a /N qualifier so that more than one data is transferred. The destination correctly reflects the contents of the source, regardless of the overlap between the source and the data.

The MOVE command actually performs byte, word, longword, and quadword reads and writes as needed in the process of moving the data. Moves are only supported for the PHYSICAL and VIRTUAL address spaces.

Qualifiers :

- **/B** —The data size is byte.
- **/W** —The data size is word.
- **/L** —The data size is longword.

- **/Q** —The data size is quadword.
- **/P** —The address space is physical memory.
- **/V** —The address space is virtual memory. All access and protection checking occur. If the access would not be allowed to a program running with the current PSL, the console issues an error message. Virtual space MOVEs cause the PTE<M> bit to be set. If memory mapping is not enabled, virtual addresses are equal to physical addresses.
- **/U** —Access to console private memory is allowed. This qualifier also disables virtual address protection checks. On virtual address writes, the PTE<M> bit will not be set if the /U qualifier is present. This qualifier is not inherited, and must be respecified on each command.
- **/N:{count}** —The address is the first of a range. The console deposits to the first address, then to the specified number of succeeding addresses. Even if the address is the symbolic address -, the succeeding addresses are at larger addresses. The symbolic address specifies only the starting address, not the direction of succession.
- **/STEP:{size}** —The number to add to the current address. Normally this defaults to the data size, but the /STEP qualifier overrides this default. This qualifier is not inherited.
- **/WRONG** —The data written will be written with the ECC check bits cc. If no check bits are specified, the one's complement of the correct check bits will be written. ECC errors on read accesses to main memory are ignored.

Arguments :

- **{src_address}** —A longword address that specifies the first location of the source data to be copied.
- **{dest_address}** —A longword address that specifies the destination of the first byte of data. These addresses may be an actual address or a symbolic address. If no address is specified, + is assumed.

Examples :

```
>>>ex /n:4 0                                  ! Observe destination.
  P 00000000 00000000
  P 00000004 00000000
  P 00000008 00000000
  P 0000000C 00000000
  P 00000010 00000000
```

```
>>>ex/n:4 200                          ! Observe source data.
  P 00000200 58DD0520
  P 00000204 585E04C1
  P 00000208 00FF8FBB
  P 0000020C 5208A8D0
  P 00000210 540CA8DE
>>>mov/n:4 200 0                      ! Move the data.
>>>ex /n:4 0                           ! Voila. ?!
  P 00000000 58DD0520
  P 00000004 585E04C1
  P 00000008 00FF8FBB
  P 0000000C 5208A8D0
  P 00000010 540CA8DE
>>>
```

4.3.2.18 NEXT

Format :

NEXT {count}

Description :

The console causes the processor to execute the specified number of macro instructions. If no count is specified, 1 is assumed.

After the the last macro instruction is executed, the console does enters console I/O mode again.

The console uses the trace bit and trace pending in the PSL, and the trace trap in the SCB to implement the NEXT function. This creates the following restrictions:

- If memory management is enabled, the NEXT command works if and only if the first page in SSC RAM is mapped somewhere in S0 (system) space.
- The NEXT command does not work where time critical code is being executed due to the instructions executed in implementation.
- The NEXT command elevates the IPL to 31 for long periods of time (milliseconds) while single stepping over several commands.
- Unpredictable results occur if the macro instruction being stepped over modifies the SCBB, or the trace trap entry. This means that the NEXT command can not be used in conjunction with other debuggers.

Qualifiers :

None.

Arguments :

- {**count**} —A value representing the number of macro instructions to execute.

Examples :

```
>>>
>>>mov /n:100 rom 0                    ! Copy a few instructions from
                                       ! EPROM.
>>>ex /ins 0                           ! Verify the first instruction.
  P 00000000   11 BRB     00000019
>>>
>>>dep pr$_scbb 200                    ! Setup a user SCB.
>>>ex pr$_scbb
  I 00000011 00000200
>>>
>>>dep pc 0                            ! Set the start PC.
>>>ex pc
  G 0000000F 00000000
>>>                                    ! Single (or multiple) step.
>>>next
  P 00000019   D0 MOVL    I^#20140000,@#20140000
>>>next 3
  P 00000024   D2 MCOML   @#20140030,@#20140502
  P 0000002F   D2 MCOML   S^#0E,@#20140030
  P 00000036   7D MOVQ    R0,@#201404B2
>>>n
  P 0000003D   D0 MOVL    I^#201404B2,R1
>>>
>>>! Warning..the state of the console may be
   ! corrupted with this example.
>>>
```

4.3.2.19 REPEAT

Format :

REPEAT {command}

Description :

The console repeatedly displays and executes the specified command. The repeating is stopped by the operator pressing Ctrl C. Any valid console command can be specified for the command with the exception of the REPEAT command.

Qualifiers :

None.

Arguments :

- **{command}** —A valid console command other than REPEAT.

Examples :

```
>>>repeat ex pr$_todr                          ! Watch the clock.
  I 0000001B 5AFE78CE
  I 0000001B 5AFE78D1
  I 0000001B 5AFE78FD
  I 0000001B 5AFE7900
  I 0000001B 5AFE7903
  I 0000001B 5AFE7907
  I 0000001B 5AFE790A
  I 0000001B 5AFE790D
  I 0000001B 5AFE7910
  I 0000001B 5AFE793C
  I 0000001B 5AFE793F
  I 0000001B 5AFE7942
  I 0000001B 5AFE7946
  I 0000001B 5AFE7949
  I 0000001B 5AFE794C
  I 0000001B 5AFE794F
  I 0000001B 5^C
>>>
```

4.3.2.20 SEARCH

Format :

SEARCH [qualifier_list] {address} {pattern} [{mask}]

Description :

The search command finds all occurences of a pattern, and reports the addresses where the pattern was found. If the /NOT qualifier is present, all addresses where the pattern didn't match are reported.

The command accepts an optional mask that indicates bits to be ignored ("don't care" bits). For example, to ignore bit 0 in the comparison, specify a mask of 1. The mask, if not present, defaults to 0.

A match occurs if pattern and not mask equals data and not mask, where:

pattern is the target data
mask is the optional don't care bitmask (which defaults to 0)
data is the data (byte, word, longword , quadword) at the current address

The command reports the address under the following conditions:

/NOT qualifier	Match condition	Action
Absent	True	Report address
Absent	False	No report
Present	True	No report
Present	False	Report address

The address is advanced by the size of the pattern (byte, word, longword or quadword), unless overriden by the /STEP qualifier.

Qualifiers :

- **/B** —The data size is byte.
- **/W** —The data size is word.
- **/L** —The data size is longword.
- **/Q** —The data size is quadword.
- **/P** —The address space is physical memory. Note that when virtual memory is examined, the address space and address in the response are the translated physical address.
- **/V** —The address space is virtual memory. All access and protection checking occur. If the access would not be allowed to a program running with the current PSL, the console issues an error message. If memory mapping is not enabled, virtual addresses are equal to physical addresses.
- **/U** —Access to console private memory is allowed. This qualifier also disables virtual address protection checks. This qualifier is not inherited, and must be respecified on each command.
- **/N:{count}** —The address is the first of a range. The first access is to the address specified, then subsequent accesses are made to succeeding addresses. Even if the address is the symbolic address -, the succeeding addresses are at larger addresses. The symbolic address specifies only the starting address, not the direction of succession.
- **/STEP:{size}** —The number to add to the current address. Normally this defaults to the data size, but is overriden by the presence of this qualifier. This qualifier is not inherited.
- **/WRONG** —ECC errors on read accesses to main memory are ignored.

- /NOT —Inverts the sense of the match.

Arguments :

- {**start_address**} —A longword address that specifies the first location subject to the search. This address can be an actual address or a symbolic address. If no address is specified, + is assumed.
- {**pattern**} —The target data.
- [{**mask**}] —A value specifying bits to be masked out.

Examples :

```
>>>search/p/b/not/
n:ffff 0 0                      ! Find all nonzero bytes in the first
                                ! 64 Kbytes of physical memory.

>>>search/v/w/n:ffff/
step:1 0 fe11                   ! Find all two byte sequences that
                                ! could be interpreted as a
                                ! "branch to self" (10$: brb 10$)
                                ! (brb assembles to FE11)

>>>search/l/p/n:ffff
0 1 FFFFFFFE                    ! Find all odd numbers in the first
                                ! 64 Kbytes of physical memory.
                                ! Masks all bits except bit 0,
                                ! compares to pattern of 1, displays
                                ! match with 1.

>>>search/l/p/not/n:ffff
0 0 FFFFFFFE                    ! Find all odd numbers in the first
                                ! 64 Kbytes of physical memory.
                                ! Same as previous example, using /not
                                ! qualifier. Masks all bits except bit
                                ! 0, compares to pattern of 0, displays
                                ! NOT match with 0.
```

4.3.2.21 SET

Format :

SET {parameter} {value}

Description :

Sets the indicated console parameter to the indicated value. The following are console parameters and their acceptable values:

Parameters :

- **BFLAG** —Set the default R5 boot flags. The value must be a hexadecimal number of up to 8 hex digits.
- **BOOT** —Set the default boot device. The value must be a valid device name as specified in Section 4.3.2.8 on the BOOT command.
- **HOST** —Invoke the DUP or MAINTENANCE driver on the selected node. Only SET HOST /DUP accepts a value parameter.

NOTE

The hierarchy of the SET HOST qualifiers listed below suggests the appropriate usage. Each qualifier only supports the additional qualifiers at levels below it.

/DUP —Use the DUP protocol to examine/modify parameters of a device on either the DSSI bus or the Q22-bus. The optional value for SET HOST is a task name for the selected driver to execute.

/DSSI node —Select the DSSI node, where node is a number from 0 to 7.

/UQSSP —Select the Q22-bus device using one of the following three methods:

/DISK n —Specify the disk controller number, where n is from 0 to 255. (The resulting fixed address for n=0 is 20001468 and the floating rank for n>0 is 26.)

/TAPE n —Specify the tape controller number, where n is from 0 to 255. (The resulting fixed address for n=0 is 20001940 and the floating rank for n>0 is 30.)

csr_address —Specify the Q22-bus I/O page CSR address for the device.

/MAINTENANCE —Use the MAINTENANCE protocol to examine/modify configuration parameters. Note that SET HOST /MAINTENANCE does not accept a task value.

/UQSSP —

/SERVICE n -
csr_address —

- **LANGUAGE** —Set console language and keyboard type. If the current console terminal does not support the Digital Multinational Character Set (MCS), then this command has no effect and the console remains in English. Acceptable values are 1 through 15:

  1) Dansk
  2) Deutsch (Deutschland/Österreich)
  3) Deutsch (Schweiz)
  4) English (United Kingdom)
  5) English (United States/Canada)
  6) Español
  7) Français (Canada)
  8) Français (France/Belgique)
  9) Français (Suisse)
  10) Italiano
  11) Nederlands
  12) Norsk
  13) Português
  14) Suomi
  15) Svenska

Qualifiers :

On a per parameter basis.

Arguments :

None.

Examples :

```
>>>set bflag 220
>>>set boot xqa0
>>>set language 2
>>>set host /dup/dssi 0
Starting DUP server...

DSSI Node 0 (SUSAN)
DRVEXR V1.0  D 25-APR-1988 10:01:35
DRVTST V1.0  D 25-APR-1988 10:01:35
HISTRY V1.0  D 25-APR-1988 10:01:35
ERASE  V1.0  D 25-APR-1988 10:01:35
PARAMS V1.0  D 25-APR-1988 10:01:35
DIRECT V1.0  D 25-APR-1988 10:01:35
Copyright © 1988  Digital Equipment Corporation
```

```
Task Name? params
Copyright © 1988  Digital Equipment Corporation

PARAMS> stat path

 ID  Path Block      Remote Node          DGS_S     DGS_R     MSGS_S    MSGS_R
 --  ------------   ---------------       -----     -----     ------    ------
  0  PB   FF811EC8    Internal Path           0         0          0         0
  5  PB   FF8120D0   BETTY   RFX T311         0         0          0         0
  4  PB   FF8121D4   WILMA   RFX T311         0         0          0         0
  3  PB   FF8122D8   DSSI2   VMS V5.0         0         0        816      3045
  2  PB   FF8123DC   2       VMB BOOT         0         0         50        52
  1  PB   FF8125E4   KAREN   RFX T311         0         0          0         0

PARAMS> exit
Exiting...

Task Name?

Stopping DUP server...

>>>set host /dup/dssi 0 params
Starting DUP server...

DSSI Node 0 (SUSAN)
Copyright © 1988  Digital Equipment Corporation

PARAMS> show node

Parameter      Current       Default       Type      Radix
---------      -------       -------       ----      -----
NODENAME       SUSAN         RF30          String    Ascii    B

PARAMS> show allclass

Parameter      Current       Default       Type      Radix
---------      -------       -------       ----      -----
ALLCLASS             1             0       Byte      Dec      B

PARAMS> exit
Exiting...

Stopping DUP server...
>>>
```

4.3.2.22 SHOW

Format :

SHOW {parameter}

Description :

Displays the console parameter indicated.

Parameters :

- **BFLAG** —Show the default R5 boot flags.
- **BOOT** —Show the default boot device.
- **DEVICE** —Show a list of all devices in the system.
- **DSSI** —Show the status of all nodes that can be found on the DSSI bus. For each node on the DSSI bus, the firmware displays the node number, the node name, and the boot name and type of the device, if available. The command does not indicate the bootability of the device.

  The node that issues the command reports a node name of * .

  The device information is obtained from the media type field of the MSCP command GET UNIT STATUS. In the case where the node is not running, or is not capable of running, an MSCP server, then no device information is displayed.
- **ETHERNET** —Show hardware Ethernet address for all Ethernet adapters that can be found, both on board and the Q22-bus. Displays as blank if no Ethernet adapter is present.
- **LANGUAGE** —Show console language and keyboard type. Refer to the corresponding SET LANGUAGE command for the meaning.
- **MEMORY** —Show main memory configuration on a board by board basis. Also report the addresses of bad pages, as defined by the bitmap.

  /FULL Additionally show the normally inaccessible areas of memory, such as, the PFN bitmap pages, the console scratch memory pages, and the Q22-bus scatter/gather map pages.
- **QBUS** —Show all Q22-bus I/O addresses that respond to an aligned word read. For each address, the console displays the address in the VAX I/O space in hex, the address as it would appear in the Q22-bus I/O space in octal, and the word data that was read in hex.

This command may take several minutes to complete, so the user may want to issue a CONTROL-C to terminate the command. The command disables the scatter/gather map for the duration of the command.

- **RLV12** —Show all RL01 and RL02 disks which appear on the Q22-bus.
- **UQSSP** —Show the status of all disks and tapes that can be found on the Q22-bus which support the UQSSP protocol. For each such disk or tape on the Q22-bus, the firmware displays the controller number, the controller CSR address, and the boot name and type of each device connected to the controller. The command does not indicate the bootability of the device.

  The device information is obtained from the media type field of the MSCP command GET UNIT STATUS. In the case where the node is not running, or is not capable of running, an MSCP server, then no device information is displayed.

- **VERSION** —Show the current firmware version.

Qualifiers :

On a per parameter basis.

Arguments :

None.

Examples :

```
>>>show bflag
00000220
>>>show boot
XQA0
>>>show device
DSSI Node 0 (SUSAN)
-DIA0 (RF30)

DSSI Node 1 (KAREN)
-DIA1 (RF30)

DSSI Node 2 (*)

DSSI Node 4 (WILMA)
-DIA4 (RF30)

DSSI Node 5 (BETTY)
-DIA5 (RF30)
```

```
UQSSP Disk Controller 0 (772150)
-DUA4 (RD53)
-DUA5 (RX50)
-DUA6 (RX50)

UQSSP Tape Controller 0 (774500)
-MUA0 (TK50)

Ethernet Adapter
-ESA0 (AA-00-03-01-2E-3F)
>>>show dssi
DSSI Node 0 (SUSAN)
-DIA0 (RF30)

DSSI Node 1 (KAREN)
-DIA1 (RF30)

DSSI Node 2 (*)

DSSI Node 4 (WILMA)
-DIA4 (RF30)

DSSI Node 5 (BETTY)
-DIA5 (RF30)
>>>show ether
Ethernet Adapter
-ESA0 (AA-00-03-01-2E-3F)
>>>show lang
English (United States/Canada)
>>>show memory
Memory 0: 00000000 to 003FFFFF, 4MB, 0 bad pages
Memory 1: 00400000 to 00BFFFFF, 8MB, 0 bad pages
Memory 2: 00C00000 to 013FFFFF, 8MB, 0 bad pages

Total of 20MB, 0 bad pages, 106 reserved pages
>>>show memory/full
Memory 0: 00000000 to 003FFFFF, 4MB, 0 bad pages
Memory 1: 00400000 to 00BFFFFF, 8MB, 0 bad pages
Memory 2: 00C00000 to 013FFFFF, 8MB, 0 bad pages

Total of 20MB, 0 bad pages, 106 reserved pages

Memory Bitmap
-013F2C00 to 013F3FFF, 10 pages

Console Scratch Area
-013F4000 to 013F7FFF, 32 pages

Qbus Map
-013F8000 to 013FFFFF, 64 pages
```

```
Scan of Bad Pages
>>>show qbus
Scan of Qbus I/O Space
-20001468 (772150) = 4000 (154) RQDX3/KDA50/RRD50/RQC25
-2000146A (772152) = 0B40
-20001940 (774500) = 0000 (260) TQK50/TQK70/TU81E/RV20
-20001942 (774502) = 0BC0
-20001F40 (777500) = 0020 (004) IPCR

Scan of Qbus Memory Space
>>>show uqssp
UQSSP Disk Controller 0 (772150)
-DUA4 (RD53)
-DUA5 (RX50)
-DUA6 (RX50)

UQSSP Tape Controller 0 (774500)
-MUA0 (TK50)
>>>show version
KA640-A X2.8-13, VMB 2.2
>>>
```

4.3.2.23 START

Format :

START [{address}]

Description :

The console starts instruction execution at the specified address. If no address is given, the current PC is used. If memory mapping is enabled, macro instructions are executed from virtual memory, and the address is treated as a virtual address. The START command is equivalent to a DEPOSIT to PC, followed by a CONTINUE. No INITIALIZE is performed.

Qualifiers :

None.

Arguments :

- **[{address}]** —The address at which to begin execution and is loaded in the user's PC.

Examples :

```
>>>start  1000
```

4.3.2.24 TEST

Format :

TEST [{test_number} [{test_arguments}]]

Description :

The console invokes a diagnostic test program specified by the test number. If a test number of 0 is specified, the power-up script is executed. The console accepts an optional list of up to five additional hexadecimal arguments.

Qualifiers :

None.

Arguments :

- **{test_number}** —A two digit hexadecimal number specifying the test to be executed.
- **{test_arguments}** —Up to five additional test arguments. These arguments are accepted but no meaning is attached to them by the console. For the interpretation of these arguments, consult the test specification for each inidividual test.

Examples :

```
>>>
>>>              ! Execute the power-up diagnostic script
>>>              ! Warning...this has the same affect as a power-up!
>>>
>>>test 0
41..40..39..38..37..36..35..34..33..32..31..30..29..28..27..26..
25..24..23..22..21..20..19..18..17..16..15..14..13..12..11..10..
09..08..07..06..05..04..03..
>>>
>>>              ! List all of the diagnostic tests.
>>>
>>>t 9e
```

Test #	Address	Name	Parameters
C1	2004D987	SSC RAM	*
C2	2004DB4E	SSC RAM ALL	*
C5	2004DCBE	SSC regs	*
C6	2004DDB8	SSC_powerup	*********
C7	2004DE7C	CBTCR timeout	***
34	2004DF38	ROM logic test	*
33	2004E000	CMCTL_powerup	*
32	2004E048	CMCTL regs	MEMCSR0_addr *********
91	2004E16C	CQBIC_powerup	**
90	2004E1FE	CQBIC regs	*
80	2004E257	CQBIC-memory	**********
60	2004E739	Console serial	start_baud end_baud ******
61	2004EA95	Console QVSS	mark_not_present ***
62	2004EB44	console QDSS	mark_not_present selftest_r0 self test_r1 *****
63	2004EDD8	QDSS self-test	input_csr selftest_r0 selftest_r1 ******
51	2004EF3F	CFPA	*****
52	2004F12B	Prog timer	which_timer wait_time_us ***
53	2004F3F8	TOY clock	repeat_test_250ms_ea Tolerance ********
54	2004F663	Virtual mode	******
55	2004F990	Interval timer	*
56	2004FA0C	SII_ext_loopbck	***
5C	2004FD11	SII_initiatior	******
5D	200509E6	SII target	*******
58	200521A6	DSSI reset	port_no time_secs *
5A	20052590	VAX CMCTL CDAL	dont_report_memory_bad repeat_count *
57	200526A8	SII_memory	incr test_pattern *****
5B	20052A60	SII_registers	****
5E	20052BA8	NI_memory	incr data_pattern ***
5F	20052CF8	NI_Test	do_ext1 where *******
41	20053874	Board Reset	***
42	20053A02	Check_for_intrs	***
44	20053A44	Cache_memory	addr_incr *********
45	20053D98	cache_mem_cqbic	start_addr end_addr addr_incr ****
46	2005407C	Cache1_diag_md	addr_incr *********
31	200546C8	MEM_Setup_CSRs	**********
30	20054DC9	MEM_Bitmap	*** mark_Hard_SBEs ******
4F	20054EC5	MEM_Data	start_add end_add add_incr cont_on_err ******
4E	2005508A	MEM_Byte	start_add end_add add_incr cont_on_err ******
4D	200551AC	MEM_Address	start_add end_add add_incr cont_on_err ******
4C	20055355	MEM_ECC_Error	start_add end_add add_incr cont_on_err ******

```
4B  200556F1  MEM_Maskd_Errs  start_add end_add add_incr cont_on_err
                              ******
4A  200558D5  MEM_Correction  start_add end_add add_incr cont_on_err
                              ******
49  20055AF1  MEM_FDM_Logic   *** cont_on_err ******
48  2005612A  MEM_Addr_shrts  start_add end_add * cont_on_err pat1
                              pat2 ****
47  20056566  MEM_Refresh     start end incr cont_on_err time_seconds
                              *****
40  20056708  MEM_Count_Errs  First_board Last_board Soft_errs_allowed
                              *******
9C  20056919  List CPU regs   *
9D  20057128  Utilities       Expnd_err_msg get_mode init_LEDs
                              clr_ps_cnt
9E  200571FC  List diags      *
9F  20057222  Create script   ******
81  20057940  MSCP-QBUS test  IP_csr ******
82  20057B07  DELQA           device_num_addr ****
>>>
>>>             ! Show the diagnostic state.
>>>
>>>t fe

bitmap=00BF3400, length=0C00, checksum=007E
busmap=00BF8000
return_stack=201406A4
subtest_pc=2004EBB0
timeout=00000001, error=00, de_error=00
de_error_vector=00, severity_code=02, total_error_count=0000
previous_error=00000000, 00000000, 00000000, 00000000, 00000000
last_exception_pc=2004EBDA
flags=21FFFD7F, test_flags=20
highest_severity=00
led_display=06
console_display=00
save_mchk_code=80, save_err_flags=000000
parameter_1=00000000  2=00000000  3=00000000
          4=00000000  5=00000000

parameter_6=00000001  7=00000000  8=2004EBE0
          9=00000000 10=20056056
```

```
>>>
>>>              ! Display the CPU registers.
>>>
>>>t 9c
TOY  =76BA1D75  ICCS =00000000
TCR0 =00000000  TIR0 =00000000  TNIR0=00000000  TIVR0=00000078
TCR1 =00000001  TIR1 =02BD7971  TNIR1=0000000F  TIVR1=0000007C
RXCS =00000000  RXDB =0000000D  TXCS =00000000  TXDB =00000030
MSER =00000000  CADR =0000000C
BDR  =FFFFFFD0  DLEDR=0000000C  SSCCR=00D45033  CBTCR=C0000004
SCR  =0000C000  DSER =00000080  QBEAR=0000000F  DEAR =00000000
QBMBR=00000000  IPCRn=0020

MEM_FRU 1  MEMCSR_0=80000015   1=00000015   2=00000015   3=00000015
MEM_FRU 2  MEMCSR_4=80400016   5=80800016   6=00000016   7=00000016
MEM_FRU 3  MEMCSR_8=00000000   9=00000000  10=00000000  11=00000000
MEM_FRU 4  MEMCSR12=00000000  13=00000000  14=00000000  15=00000000
           MEMCSR16=00000044  17=0000203C

Ethernet  SA = 08-00-2B-0B-25-65    NICSR0=0004

SII      MSIDR0 =01FF  MSIDR1 =0002  MSIDR2 =0000  MSICSR =0010
         MSIIDR =8007  MSITR  =0000  MSITLP =0000  MSIILP =0000
         MSIDSCR=80FF  MSIDSSR=8500  MSIDCR =0008
>>>
>>>
>>>
```

4.3.2.25 UNJAM

Format :

UNJAM

Description :

An I/O bus reset is performed. This is implemented by writing a 1 to IPR 55 (decimal).

Qualifiers :

None.

Arguments :

None.

Examples :

```
>>>
>>>
>>>unjam
>>>
>>>
```

4.3.2.26 X - Binary Load and Unload

Format :

X {address} {count} <CR> {line_checksum}
{data} {data_checksum}

Description :

The X command is for use by automatic systems communicating with the console. It is not intended for use by operators.

The console loads or unloads (that is, writes to memory, or reads from memory) the specified number of data bytes, starting at the specified address through the console serial line, regardless of which device is serving as the system console.

If bit 31 of the count is clear, data is to be received by the firmware, and deposited into memory. If bit 31 of the count is set, data is to be read from memory and sent by the firmware. The remaining bits in the count are a positive number indicating the number of bytes to load or unload.

The firmware accepts the command upon receiving the carriage return. The next byte the firmware receives is the command checksum, which is not echoed. The command checksum is verified by adding all command characters, including the checksum and separating space, (but not including the terminating carriage return, rubouts or characters deleted by rubout), into an 8-bit register initially set to zero. If no errors occur, the result is zero. If the command checksum is correct, the console responds with the input prompt and either sends data to the requester or prepares to receive data. If the command checksum is in error, the console responds with an error message. The intent is to prevent inadvertent operator entry into a mode where the console is accepting characters from the keyboard as data, with no escape mechanism possible.

If the command is a load (bit 31 of the count is clear), the firmware responds with the input prompt, then accepts the specified number of bytes of data for depositing to memory, and an additional byte of received data checksum. The data is verified by adding all data characters and the checksum character into an 8-bit register initially set to zero. If the final content of the register

is non-zero, the data or checksum are in error, and the firmware responds with an error message.

If the command is a binary unload (bit 31 of the count is set), the firmware responds with the input prompt, followed by the specified number of bytes of binary data. As each byte is sent, it is added to a checksum register initially set to zero. At the end of the transmission, the two's complement of the low byte of the register is sent.

If the data checksum is incorrect on a load, or if memory errors or line errors occur during the transmission of data, the entire transmission is completed, then the console issues an error message. If an error occurs during loading, the contents of the memory being loaded are unpredictable.

Echo is suppressed during the receiving of the data string and checksums.

To avoid treating flow control characters from the terminal as valid command line checksums, all flow control is terminated at the reception of the carriage return terminating the command line.

You can use control characters (Ctrl C, Ctrl S, Ctrl O, etc.) to control the console serial line during a binary unload. It is not possible during a binary load, since all received characters are valid binary data.

The firmware must receive data being loaded with a binary load command at a rate of at least one byte every 60 seconds. It must receive the command checksum that precedes the data within 60 seconds of the carriage return that terminates the command line. It must receive the data checksum within 60 seconds of the last data byte. If any of these timing requirements are not met, then the firmware aborts the transmission by issuing an error message and returning to the console prompt.

The entire command, including the checksum, can be sent to the firmware as a single burst of characters at the console serial lines's specified character rate. The firmware is able to receive at least 4 Kbytes of data in a single X command.

Qualifiers :

None.

Arguments :

None.

Examples :

None.

4.3.2.27 ! - Comment

Format :

!

Description :

The comment command is used to document command sequences. The comment character can appear anywhere on the command line. All characters following the comment character are ignored.

Qualifiers :

None.

Arguments:

None.

Examples :

```
>>>! The console ignores this line.
>>>
```

4.4 Bootstrapping

Bootstrapping is the process of loading and transferring control to an operating system. The KA640 supports bootstrap of the following operating systems: VAX/VMS, Ultrix-32, and VAXELN. Additionally, the KA640 will boot MDM diagnostics and any user application image which conforms to the boot formats described herein.

On the KA640, a bootstrap occurs whenever a BOOT command is issued at the console or whenever the processor halts and the conditions specified in the Table 4–1 for automatic bootstrap are satisfied.

4.4.1 Preparing for the Bootstrap

Prior to dispatching to the primary bootstrap (VMB), the firmware initializes the system to a known state. The initialization sequence follows:

1. Check CPMBX<2>(BIP). If it is set, bootstrap fails.
2. If this is an automatic bootstrap, print the message "Loading system software" on the console terminal.
3. Validate the boot device name. If none exists, supply a list of available devices and prompt user for a device. If no device is entered within 30 seconds, use ESA0.
4. Write a form of this BOOT request including the active boot flags and boot device on the console, for example "(BOOT/R5:0 ESA0)".
5. Set CPMBX<2>(BIP).
6. Initialize the Q22-bus Scatter/Gather map.
   a. Clear IPCR<5>(LMEAE).
   b. Perform an UNJAM.
   c. Map all vacant Q22-bus pages to the corresponding page in local memory and validate each entry if that page is good.
   d. Perform an INIT.
   e. Set IPCR<5>(LMEAE).
7. Validate the PFN bitmap. If invalid, rebuild it.
8. Search for a 128Kb contiguous block of good memory as defined by the PFN bitmap. If 128Kb cannot be found, the bootstrap fails.
9. Initialize the general purpose registers.

   R0 = address of descriptor of the boot device name or 0 if none specified
   R2 = length of PFN bitmap in bytes
   R3 = address of PFN bitmap
   R4 = time of day from PR$_TODR at power-up
   R5 = boot flags
   R10 = halt PC value
   R11 = halt PSL value (without halt code and map enable)
   AP = halt code
   SP = base of 128Kb good memory block + 512
   PC = base of 128Kb good memory block + 512
   R1, R6, R7, R8, R9, FP = 0

10. Copy the VMB image from EPROM to local memory beginning at the base of the 128Kb good memory block + 512.

11. Exit from the firmware to memory resident VMB.

On entry to VMB the processor is running at IPL 31 on the interrupt stack with memory management disabled.

4.4.1.1 Boot Devices

The KA640 firmware passes the address of a descriptor of the boot device name to VMB through R0. This device name used for the bootstrap operation is either

- ESA0, if no default boot device has been specified, or
- The default boot device specified at initial power-up or via a SET BOOT command, or
- The boot device name explicitly specified in a BOOT command line.

The device name may be any arbitrary character string, with a maximum length of 17 characters. Longer strings cause a "VAL TOO BIG" error message to be issued from the console. Otherwise the console makes no attempt at interpreting or validating the device name. The console converts the string to all upper case, and passes VMB the address of a string descriptor for the device name in R0.

Table 4–7 correlates the boot device names expected in a BOOT command with the corresponding supported devices.

Table 4–7 KA640 Supported Boot Devices

Boot Name*	Controller Type	Device Type(s)
Disk:		
[node$]DIAn	On-board DSSI	RF30,
DUcn	RQDX3 MSCP	RD52, RD53, RD54, RX33, RX50
	KDA50 MSCP	RA70, RA80, RA81, RA82,
	KLESI	RC25
DLcn	RLV12	RL01, RL02

* Boot device names consist of minimally a two letter device code, followed by a single character controller letter (A...Z), and terminating in a device unit number (0...65535). DSSI device names may optionally include a node prefix, consisting of either a node number (0...7) or a node name (a string of up to 8 characters), terminating in a "$".

Table 4–7 (Cont.) KA640 Supported Boot Devices

Boot Name*	Controller Type	Device Type(s)
Tape:		
[node$]MIAn	On-board DSSI	—-
MUcn	TQK50 MSCP	TK50
	TQK70 MSCP	TK70
	KLESI	TU81E
Network:		
ESA0	On-board Enet	—-
XQcn	DEQNA	—-
	DELQA	—-
PROM:		
PRA0	MRV11	—-

* Boot device names consist of minimally a two letter device code, followed by a single character controller letter (A...Z), and terminating in a device unit number (0...65535). DSSI device names may optionally include a node prefix, consisting of either a node number (0...7) or a node name (a string of up to 8 characters), terminating in a "$".

NOTE

Table 4–7 presents a definitive list of boot devices which the KA640 supports. However, the KA640 will likely boot other devices which adhere to the MSCP standards.

4.4.1.2 Boot Flags

The action of VMB is qualified by the value passed to it in R5. R5 contains boot flags that specify conditions of the bootstrap. The firmware passes to VMB either the R5 value specified in the BOOT command or the default boot flag value specified with a SET BFLAG command.

Table 4–8 describes the boot flags used by VMB in the boot flag longword.

Table 4–8 VMB Boot Flags

Field	Name	Description
3	RPB$V_ BBLOCK	Secondary bootstrap from bootblock. When this bit is set, VMB reads logical block number 0 of the boot device and tests it for conformance with the bootblock format. If in conformance, the block is executed to continue the bootstrap. No attempt to perform a Files-11 bootstrap is made.
4	RPB$V_DIAG	Diagnostic bootstrap. When this bit is set, the load image requested over the network is [SYS0.SYSMAINT]DIAGBOOT.EXE.
5	RPB$V_ BOOBPT	Bootstrap breakpoint. If this flag is set, a breakpoint instruction is executed in VMB and control is transferred to XDELTA prior to boot.
6	RPB$V_ HEADER	Image header. If this bit is set, VMB transfers control to the address specified by the file's image header. If this bit is not set, VMB transfers control to the first location of the load image.
8	RPB$V_ SOLICT	File name solicit. When this bit is set, VMB prompts the operator for the name of the application image file. A maximum of a 39 character file specification is permitted.
9	RPB$V_HALT	Halt before transfer. When this bit is set, VMB halts before transferring control to the application image.
31:28	RPB$V_ TOPSYS	This field can be any value from 0 through F. This flag changes the top level directory name for the system disks with multiple operating systems. For example, if TOPSYS is 1, the top level directory name is [SYS1...].

4.4.2 Primary Bootstrap, VMB

Virtual memory boot (VMB) is the primary bootstrap for booting VAX processors. On the KA640, VMB is resident in the firmware and is copied into main memory before control is transferred to it. VMB then loads the secondary bootstrap image and transfers control to it.

NOTE

In certain cases, such as VAXELN, VMB actually loads the operating system directly. However, for the purpose of this discussion "secondary bootstrap" refers to any VMB loadable image.

VMB inherits a well defined environment and is responsible for further initialization. The following summarizes the operation of VMB:

1. Initialize a two page SCB on the first page boundary above VMB.

2. Allocate a three page stack above the SCB.
3. Initialize the restart parameter block (RPB).
4. Initialize the secondary bootstrap argument list.
5. If not a PROM boot, locate a minimum of three consecutive valid QMRs.
6. Write "2" to the diagnostic LEDs and display "2.." on the console to indicate that VMB is searching for the device.
7. Optionally, solicit from the console a "Bootfile: " name.
8. Write the name of the boot device from which VMB will attempt to boot on the console, for example, "-ESA0".
9. Copy the secondary bootstrap from the boot device into local memory above the stack. If this fails, the bootstrap fails.
10. Write "1" to the diagnostic LEDs and display "1.." on the console to indicate that VMB has found the secondary bootstrap image on the boot device and has loaded the image into local memory.
11. Clear CPMBX<2>(BIP) and CPMBX<3>(RIP).
12. Write "0" to the diagnostic LEDs and display "0.." on the console to indicate that VMB is now transferring control to the loaded image.
13. Transfer control to the loaded image with the following register usage.

    R5 = transfer address in secondary bootstrap image
    R10 = base address of secondary bootstrap memory
    R11 = base address of RPB
    AP = base address of secondary boot parameter block
    SP = current stack pointer

If the bootstrap operation fails, VMB relinquishes control to the console by halting with a HALT instruction.

NOTE

VMB makes no assumptions about the location of Q22-bus memory. However, VMB searches through the Q22-bus map registers (QMRs) for the first QMR marked "valid." VMB requires minimally 3 and maximally 129 contiguous "valid" maps to complete a bootstrap operation. If the search exhausts all map registers or there are fewer than the required number of "valid" maps, a bootstrap cannot be performed. It is recommended that a suitable block of Q22-bus memory address space be available (unmapped to other devices) for proper operation.

The following is a sample console display of a successful automatic bootstrap:

```
Loading system software.
(BOOT/R5:0 DUA0)

 2..
-DUA0
 1..0..
```

After a successful bootstrap operation, control is passed to the secondary bootstrap image. In the event that an operating system has an extraordinarily large secondary bootstrap which overflows the 128Kb of "good" memory, VMB loads the remainder of the image in memory above the "good" block. However, if there are not enough contiguous "good" pages above the block to load the remainder of the image, the bootstrap fails.

4.4.3 Device Dependent Bootstrap Procedures

As mentioned earlier the KA640 supports bootstrapping from a variety of boot devices. The following sections describe the various device dependent boot procedures.

4.4.3.1 Disk and Tape Bootstrap Procedure

The disk and tape bootstrap supports Files-11 lookup (supporting only the ODS level 2 file structure) or the boot block mechanism (used in PROM boot also). Of the standard DEC operating systems VMS and ELN use the Files-11 bootstrap procedure and Ultrix-32 uses the boot block mechanism.

VMB first attempts a Files-11 lookup, unless the RPB$V_BBLOCK boot flag is set. If VMB determines that the designated boot disk is a Files-11 volume, it searches the volume for the designated boot program, usually [SYS0.SYSEXE]SYSBOOT.EXE. However, VMB can request a diagnostic image or prompt the user for an alternate file specification (Section 4.4.1.2). If the boot image cannot be found, VMB fails.

If the volume is not a Files-11 volume or the RPB$V_BBLOCK boot flag was set, the boot block mechanism proceeds as follows:

1. Read logical block 0 of the selected boot device (this is the boot block).
2. Validate that the contents of the boot block conform to the boot block format (Figure 4-1).
3. Use the boot block to find and read in the secondary bootstrap.
4. Transfer control to the secondary bootstrap image, just as for a Files-11 boot.

The format of the boot block must conform to that shown in Figure 4-1, where 18 (hex) indicates this is a VAX instruction set and 18 (hex) + k = the one's complement of "CHK".

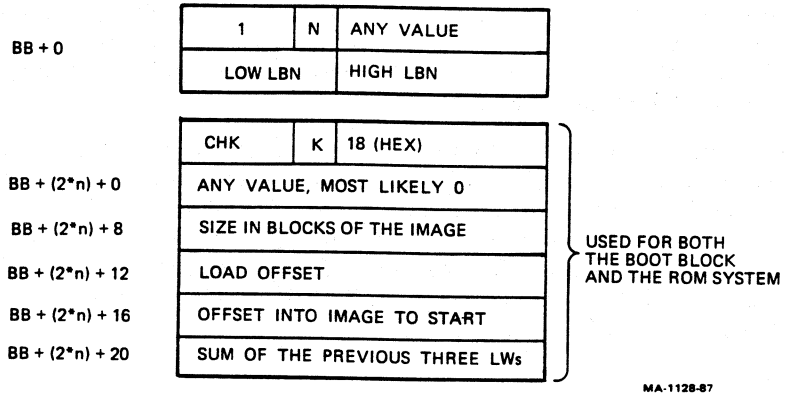


Figure 4-1 Boot Block Format

4.4.3.2 PROM Bootstrap Procedure

The PROM bootstrap uses a variant of the boot block mechanism. VMB searches through Q22-bus memory on 16Kb boundaries for a valid PROM *signature block*, the second segment of the boot block defined in Figure 4-1.

At each boundary, VMB :

- Validates the readibility of that Q22-bus memory page.
- If readable, checks to see if it contains a valid PROM signature block.

If verification passes, the PROM image will be copied into main memory and VMB will transfer control to that image at the offset specified in the PROM bootblock. If not, the next page will be tested.

NOTE

It is not necessary that the boot image actually reside in PROM. Any boot image in Q22-bus memory space with a valid signature block on a 16KB boundary is a candidate.

The PROM image is copied into main memory in 127 page *chunks* until the entire PROM is moved. All destination pages beyond the primary 128Kb block are verified to make sure they are marked good in the PFN bitmap. The PROM must be copied contiguously and if all required pages cannot fit into the memory immediately following the VMB image, the boot fails.

4.4.3.3 Network Bootstrap Procedure

Whenever a network bootstrap is selected on a KA640, VMB makes continuous attempts to boot from the network. VMB uses the DNA maintenance operations protocol (MOP) as the transport protocol for network bootstraps and other network operations. Once a network boot has been invoked, VMB turns on the designated network link and retries MOP load attempts every 8 to 12 minutes, until either a successful boot occurs or it is halted from the operator console.

The KA640 supports the load of a standard operating system, a diagnostic image, or a user designated program via network bootstraps. The default image is the a standard operating system, however, a user may select an alternate image by setting either the RPB$V_DIAG bit or in the RPB$V_SOLICT bit in the boot flag longword R5. Note, that the RPB$V_SOLICT bit has precedence over the RPB$V_DIAG bit. Hence, if both bits are set, then the solicited file is requested.

NOTE

VMB accepts a maximum of a 39 character file specification for solicited boots. If the network server is running VMS the following defaults apply to the file specification: the directory MOM$LOAD:, and an extention .SYS. Therefore, the 39 character file specification need only consist of the filename if the default directory and extension attributes are used.

The KA640 VMB uses the MOP program load sequence for bootstrapping the module and the MOP "dump/load" protocol type for load related message exchanges. The MOP message types used in the exchange are listed in Table 4-9.

VMB, the requester, starts by sending a REQ_PROGRAM message to the MOP "dump/load" multicast address (Table 4-10). It then waits for a response in the form of a VOLUNTEER message from another node on the network, the MOP server. If a response is received, then the destination address is changed from the multicast address to the node address of the server and the same REQ_PROGRAM message is retransmitted to the server as an acknowledge.

Next, VMB begins sending REQ_MEM_LOAD messages to the server. The server responds with either:

- MEM_LOAD message, while there is still more to load.
- MEM_LOAD_w_XFER, if it is the end of the image.

- PARAM_LOAD_w_XFER, if it is the end of the image and operating system parameters are required.

The "load number" field in the load messages is used to synchronize the load sequence. At the beginning of the exchange, both the requester and server initialize the load number. The requester only increments the load number if a load packet has been successfully received and loaded. This forms the acknowledge to each exchange. The server will resend a packet with a specific load number, until it sees the load number incremented. The final acknowledge is sent by the requester and has a load number equivalent to the load number of the appropriate LOAD_w_XFER message + 1.

During a network boot sequence, transmit messages are sent a maximum of four times when there is no response to the message. If there is no response after the four transmit attempts, then the mode is reset to multicast address mode (regardless of the current mode) and the boot sequence is repeated up to three times. This gives any node ample opportunity to respond to one of these requests.

A timeout on a receive request is considered "No Response" to the transmit request. Timeouts on receives take 15 seconds, therefore, each retransmit attempt is made at 15 second intervals. Since there are three boot sequence attempts consisting of four transmit attempts, a maximum of three minutes can elapse during a single bootstrap attempt.

4.4.3.3.1 Network Listening

While VMB is waiting for a load volunteer, it listens on the network for other maintenance messages directed to the node and periodically identifies itself at the end of each 8 to 12 minute interval prior to a bootstrap retry. In particular, this listener supplements the MOP functions of the VMB load requester typically found in bootstrap firmware and supports:

- A remote console server that generates unsolicited SYSTEM_ID messages every 8 to 12 minutes and solicited SYSTEM_ID messages in response to REQUEST_ID messages.
- A loopback server that responds to LOOP_DATA messages by echoing the LOOPED_DATA to the requester.

The KA640 remains in the listener mode until a volunteer is found and a load sequence is initiated. This mode is also reentered, whenever a boot sequence fails.

The MOP functions and message types which are supported by the KA640 are summarized in Table 4-9. During network operation VMB listens only to MOP "Load/Dump," MOP "Remote Console," and Ethernet "Loopback Assistance" message protocols (listed in Table 4-10). All other Ethernet protocols are filtered by the network device driver.

Table 4–9 KA640 Network Maintenance Operations Summary

		MOP/Ethernet Messages		
Function	**Role**	**Transmit**		**Receive**
Dump	Requester	—		—
	Server	—		—
Load	Requester	REQ_PROGRAM*	to solicit	VOLUNTEER
		REQ_MEM_LOAD	to solicit	MEM_LOAD
			or	MEM_LOAD_w_XFER
			or	PARAM_LOAD_w_XFER
	Server	—		—
Console	Requester	—		—
	Server	SYSTEM_ID†	in response to	REQUEST_ID
Loopback	Requester	—		—
	Server	LOOPED_DATA‡	in response to	LOOP_DATA

*The initial REQ_PROGRAM message is sent to the dumpload multicast address. If an assistance VOLUNTEER message is received, then the responder's address is used as the destination to repeat the REQ_PROGRAM message and for all subsequent REQ_MEM_LOAD messages.

†SYSTEM_ID messages are sent out every 8 to 12 minutes to the remote console multicast address and on receipt of a REQUEST_ID message they are sent to the initiator.

‡LOOPED_DATA messages are sent out in response to LOOP_DATA messages. These messages are actually in Ethernet LOOP TEST format, not in MOP format, and are sent without the additional length field (padding is disabled).

Table 4–10 MOP/Ethernet Multicast Addresses and Protocols

Function	Address	Protocol	Owner
Dump/Load	AB-00-00-01-00-00	60-01	Digital
Remote Console	AB-00-00-02-00-00	60-02	Digital
Loopback Assistance	CF-00-00-00-00-00	90-00	Cross Company

4.5 Operating System Restart

An operating system restart is the process of bringing up the operating system from a known initialization state following a processor halt. This procedure is often called *restart* or *warmstart*, and should not be confused with a CVAX processor restart which results in firmware entry.

On the KA640, a restart occurs if the conditions specified in Table 4-1 are satisfied.

To restart a halted operating system, the firmware searches system memory for the Restart Parameter Block (RPB), a data structure constructed for this purpose by VMB. If a valid RPB is found, the firmware passes control to the operating system at an address specified in the RPB.

The firmware keeps a *restart in progress* (RIP) flag in the CPMBX which it uses to avoid repeated attempts to restart a failing operating system. An additional RIP flag is maintained by the operating system in the RPB.

The firmware uses the following algorithm to restart the operating system:

1. Check CPMBX<3>(RIP). If it is set, restart fails.
2. Print the message "Restarting system software" on the console terminal.
3. Set CPMBX<3>(RIP).
4. Search for a valid RPB. If none is found, restart fails.
5. Check the operating system RPB$L_RSTRTFLG<0>(RIP) flag. If it is set, restart fails.
6. Write "0" on the diagnostic LEDs.
7. Dispatch to the restart address, RPB$L_RESTART, with:

   SP = the physical address of the RPB plus 512
   AP = the halt code
   PSL = 041F0000
   PR$_MAPEN = 0.

If the restart is successful, the operating system must clear CPMBX<3>(RIP).

If restart fails, the firmware prints "Failure" on the system console.

4.5.1 Locating the RPB

The RPB is a page aligned data structure created by the bootstrap. The firmware uses the following algorithm to find a valid RPB:

1. Search for a page of memory that contains its address in the first longword. If none is found, the search for a valid RPB has failed.
2. Read the second longword in the page (the physical address of the restart routine). If it is not a valid physical address, or if it is zero, return to step 1. The check for zero is necessary to ensure that a page of zeros does not pass the test for a valid RPB.
3. Calculate the 32 bit twos-complement sum (ignoring overflows) of the first 31 longwords of the restart routine. If the sum does not match the third longword of the RPB, return to step 1.
4. A valid RPB has been found.

4.6 Machine State When Halted

This section describes the state of the KA640 after a power-up halt.

The descriptions in this section assume a machine with no errors, that the machine has just been turned on and that only the power-up diagnostics have been run. The state of the machine is not defined if individual diagnostics are run or during any other halts other than a power-up halt (SAVPSL<14:8>(HALT_CODE) = 3).

The following sections describe data structures that are guaranteed to be constant over future versions of the KA640 firmware. Placement and/or existance of any other structure(s) is not implied.

4.6.1 Main Memory Layout and State

Main memory is tested and initialized by the firmware on power-up.

4.6.1.1 Reserved Main Memory

In order to build the scatter/gather map and the bitmap, the firmware attempts to find a physically contiguous 64 K Bytes section of memory at the highest possible address that has no multiple bit errors. Single bit errors are tolerated in this section.

This algorithm has the side effect of leaving possibly good memory above the bitmap. This memory, due to the placement algorithm, will not have any contiguous section larger than 64K Bytes-1 bytes long. There may also be bad memory above this section.

While the full 64 K Bytes is used by the diagnostics on power-up, on machines with less than 64MB of main memory, the lower 32 K Bytes that is not used by the bitmap is rolled into the remainder of main memory.

4.6.1.2 Scatter/Gather Map

On power-up, the scatter/gather map is set by the firmware to map to the first 4 M Bytes of main memory. Main memory pages will not be mapped if there is a corresponding page in Q22-bus memory, or if the page is flagged bad by the bitmap.

On a processor halt other than power-up, the contents of the scatter/gather map is undefined, and is dependant on the individual operating systems.

Operating systems should not move the location of the scatter/gather map, and should access the map only on aligned longwords through the local I/O space of 2008 8000 - 2008 FFFC inclusive.

The Q22-bus map base register (QMBR) is set up by the firmware to point to this area and should not be changed by software.

4.6.1.3 Bitmap

The bitmap is a data structure that indicates which pages in memory are deemed useable by operating systems. The bitmap is built by the diagnostics as a side effect of the memory tests on power-up.

Each bit in the bitmap corresponds to a page in main memory. There is a one to one correspondance between a page frame number (origin 0) and a bit index in the bitmap. A one in the bitmap indicates that the page may be used, with a zero indicating that the page has an error(s). The bitmap does not map itself or the scatter/gather map. There may be memory above the bitmap which has both good and bad pages.

By default, a page is flagged as bad if there are multiple bit errors in the page. Single bit errors, regardless of frequency, do not flag the pageas bad. Via the SET command, users may change the semantics of how the bitmap is built so that any and all errors will flag the page being tested as being

bad. Users should be aware that the later techinique may unduly reduce availability of memory to the system.

The bitmap is protected by a checksum stored in the SSC RAM. The checksum is a simple byte wide, two's complement checksum. The sum of all bytes in the bitmap, and the bitmap checksum, should have the lower 8 bits zero. Operating systems that map out pages are encouraged to use this bitmap to facilitate diagnosis by service personnel.

The bitmap always starts on a page boundary and is typically found just below the scatter/gather map, although future versions of the firmware will not guarantee this. The bitmap takes up 2 K Bytes for every 8 M Bytes of main memory, so a 64 M Bytes machine will have a 16 K Bytes of bitmap and an 8 M Bytes will have a 2 K Bytes bitmap. The location of the bitmap can be found by invoking test 9E.

4.6.1.4 Contents of Main Memory

The contents of main memory are undefined after the diagnostics have run. Typically, non zero test patterns will be left in memory.

The diagnostics will scrub all of main memory so that no power-up induced errors remain in the memory system. On the KA640 memory subsystem, the state of the ECC bits and the data bits are undefined on initial power-up. This can result in single and multiple bit errors if the locations are read before written because the ECC bits are not in agreement with their correspsonding data bits. An aligned longword write to every location (done by diagnostics) eliminates all power-up induced errors.

4.6.2 Cache Memory

The cache is tested during the power-up diagnostics, flushed and then turned off. The cache is again turned on by the BOOT and by the INIT command. Otherwise, the state of the cache is not touched.

4.6.3 Translation Lookaside Buffer

The translation lookaside buffer (TLB) is tested by diagnostics on power-up, but not used otherwise as the firmware runs in physical mode. The TLB is invalidated via IPR 57 prior to the exit from a halt.

4.6.4 Halt Protect Space

Halt protect space is from 2004 0000 - 2005 FFFF inclusive. The halt unprotected space is from 2006 0000 - 2008 FFFF inclusive for the 128 K Bytes of firmware code that is currently on the KA640.

The firmware always runs in halt protect space. When passing control to the bootstrap, the firmware exits the halt protected space, so if halts are enabled, and the halt line is asserted, the processor will then halt before booting.

The address decode space that the SSC is set to twice the size of the ROMs in actual use. This then allows the ROMs to appear twice in the assigned address space. The halt protect space is then set to the size of the ROMs, thus setting up the SSC to recognize the lower half of the addressable space as halt protected, and the upper half as halt unprotected.

4.7 Public Data Structures and Entry Points

This section describes data structures and subroutine entry points that are public and are guaranteed to be compatible over future versions of the KA640 firmware.

4.7.1 Firmware EPROM Layout

The KA640 uses two 64KB EPROMs for a total of 128Kb of EPROM memory. Of this, approximately 120KB is used for code, with the remaining reserved for future expansion and customer usage. There are two copies of the firmware, one in halt protected space, and one in halt unprotected space. Both copies are identical.

The first instruction executed on halts is a branch around the system ID extension (SIE) and the callback entry points. This allows these public data structures to reside in fixed locations in the EPROM.

The callback area entry points provide a simple interface to the currently defined console for VMB and secondary bootstraps (Section 4.7.2).

The EPROM checksum is a longword checksum from 2004 0000 to the checksum inclusive. The diagnostics use this to determine that the EPROMs can correctly be read.

The memory between the checksum and the four page user area at the end of the EPROMs is reserved by Digital for future expansion of the KA640 firmware. The contents of this area is set to FF.

The four pages reserved for customer use are at the top of the EPROMs, and start at address 2005 F800 (halt protected space) or 2007 F800 (halt unprotected space). These areas are not burned and may be reburned by OEMs or end users. The area is not tested by the KA640 firmware and is not included in the checksum.

4.7.2 Call Back Entry Points

The KA640 firmware provides several entry points that facilitate I/O to the designated console device. Users of these entry points do not need to be aware of the console device type, be it a video terminal or workstation.

The primary intent of these routines is to provide a simple console device to VMB and secondary bootstraps before operating systems load their own terminal drivers.

These are JSB (subroutine as opposed to procedure) entry points located in fixed locations in the firmware. These locations branch to code that in turn calls the appropriate routines.

All of the entry points are designed to run at IPL 31 on the interrupt stack in physcial mode. Virtual mode is not supported. Due to internal firmware architectural restrictions, users are encouraged to only call into the halt protected entry points. These entry points are listed below.

Entry Point	Address
CP$GET_CHAR_R4	2004 0008
CP$MSG_OUT_NOLF_R4	2004 000C
CP$READ_WTH_PRMPT_R4	2004 0010

4.7.2.1 CP$GETCHAR_R4

This routine returns the next character entered by the operator in R0. A timeout interval can be specified. If the timeout interval is zero, no timeout is generated. If a timeout is specified and if timeout occurs, a value of 18 (CAN) is returned instead of normal input.

Registers R0, R1, R2, R3 and R4 are modified by this routine, all others are preserved.

```
;-------------------------------------------------------------
; Usage with timeout:

movl    #timeout_in_tenths_of_second,r0 ; Specify timeout.
jsb     @#CP$GET_CHAR_R4                ; Call routine.
cmpb    r0,#^x18                        ; Check for timeout.
beql    timeout_handler                 ; Branch if timeout.
; Input is in R0.

;-------------------------------------------------------------
; Usage without timeout:

clrl    r0                              ; Specify no timeout.
jsb     @#CP$GET_CHAR_R4                ; Call routine.
; Input is in R0.

;-------------------------------------------------------------
```

4.7.2.2 CP$MSG_OUT_NOLF_R4

This routine outputs a message to the console. The message is specified either by a message code or a string descriptor. The routine distinguishes between message codes and descriptors by requiring that any descriptor be located outside of the first page of memory. Hence, message codes are restricted to values between 0 and 511.

Registers R0, R1, R2, R3 and R4 are modified by this routine, all others are preserved.

```
;-------------------------------------------------------------
; Usage with message code:

movzbl  #console_message_code,r0        ; Specify message code.
jsb     @#CP$MSG_OUT_NOLF_R4            ; Call routine.

;-------------------------------------------------------------
; Usage with a message descriptor (position dependent).

movaq   5$,r0                           ; Specify address of desc.
jsb     @#CP$MSG_OUT_NOLF_R4            ; Call routine.
.
.
5$:     .ascid  /This is a message/     ; Message with descriptor.

;-------------------------------------------------------------
; Usage with a message descriptor (position independent).
```

```
pushab  5$                              ; Generate message desc.
pushl   #10$-5$                         ; on stack.
movl    sp,r0                           ; Pass desc. addr. in R0.
jsb     @#CP$MSG_OUT_NOLF_R4            ; Call routine.
clrq    (sp)+                           ; Purge desc. from stack.
.
.

5$:     .ascii  /This is a message/     ; Message.
10$:            ;

;---------------------------------------------------------------
```

4.7.2.3 CP$READ_WTH_PRMPT_R4

This routine outputs a prompt message and then inputs a character string from the console. When the input is accepted, DELETE, CONTROL-U and CONTROL-R functions are supported.

As with CP$MSG_OUT_NOLF_R4, either a message code or the address of a string descriptor is passed in R0 to specify the prompt string. A value of zero results in no prompt.

A descriptor of the input string is returned in R0 and R1. R0 contains the length of the string and R1 contains the address. This routine inputs the string into the console program string buffer and therefore the caller need not provide an input buffer. Successive calls however destroy the previous contents of the input buffer.

Registers R0, R1, R2, R3 and R4 are modified by this routine, all others are preserved.

```
;---------------------------------------------------------------
; Usage with a message descriptor (position independent).
pushab  10$                             ; Generate prompt desc.
pushl   #10$-5$                         ; on stack.
movl    sp,r0                           ; Pass desc. addr. in R0.
jsb     @#CP$READ_WTH_PRMPT_R4          ; Call routine.
clrq    (sp)+                           ; Purge prompt desc.
.                                       ; Input desc in R0 and R1.
.

5$:     .ascii  /Prompt> /              ; Prompt string.
10$:
;---------------------------------------------------------------
```

4.7.3 SSC RAM Layout

The KA640 firmware uses the 1KB of NVRAM on the SSC for storage of firmware specific data structures and other information that must be preserved across power cycles. This NVRAM resides in the SSC starting at address 2014 0400. The NVRAM should not be used by the operating systems except as described in the following sections. This NVRAM is not reflected in the bitmap built by the firmware.

4.7.4 Public Data structures

Following is a template of the data structures used in the public area of SSC NVRAM, which starts at physical address 2014 0400.

Fields that are designated as reserved and/or internal use should not be written to. Note that not all fields may be written to.

4.7.4.1 Console Program Mailbox (CPMBX)

The console program mailbox (CPMBX) is a software data structure located at the beginning of the SSC battery backed-up RAM (2014 0800). The CPMBX is used to pass information between the KA640 firmware and diagnostics, VMB, or an operating system. It consists of three bytes referred to here as NVR0, NVR1, and NVR2. NVR0 resides at address 2014 0800; NVR1 resides at address 2014 0801; NVR2 resides at address 2014 0802. The bit fields of NVR0, NVR1, and NVR2 are described in Tables 4–11, 4–12, and 4–13 respectively.

Table 4-11 NVR0

Field	Acronym	Description
7:4	LANGUAGE	This field specifies the current selected language for displaying halt and error messages on terminals which support MCS.
3	RIP	If set, a restart attempt is in progress. This flag must be cleared by the operating system if the restart succeeds.
2	BIP	If set, a bootstrap attempt is in progress. This flag must be cleared by the operating system if the bootstrap succeeds.
1:0	HLT_ACT	Processor halt action - this field in conjunction with the conditions specified in Table 4-1 is used to control the automatic restart/bootstrap procedure. HLT_ACT is normally written by the operating system. 0 : Restart; if that fails, reboot; if that fails, halt. 1 : Restart; if that fails, halt. 2 : Reboot; if that fails, halt. 3 : Halt.

Table 4-12 NVR1

Field	Acronym	Description
2	MCS	If set, indicates that the attached terminal supports multinational character set (MCS). If clear, MCS is not supported.
1	CRT	If set, indicates that the attached terminal is a CRT. If clear, indicates that the terminal is hardcopy.

Table 4-13 NVR2

Field	Acronym	Description
7:0	KEYBOARD	This field indicates the national keyboard type in use.

4.7.4.2 Firmware Stack

This section contains the stack that is used by all of the firmware, with the exception of VMB, which has its own built in stack.

4.7.4.3 Diagnostic State

This area is used by the firmware resident diagnostics. This section is not documented here.

4.7.4.4 USER Area

The KA640 console reserves the last longword (address 2014 07FC) of the NVRAM for customer use. This location is not tested by the console firmware. Its value is undefined.

4.8 Error Messages

The error messages issued by the KA640 firmware fall into three categories: halt messages, console error messages, and VMB error messages.

4.8.1 Halt Messages

Table 4–14 lists messages that may appear on the console terminal when a system error occurs.

Table 4–14 Halt Messages

Code	Message	Explanation
?02	EXT HLT	External halt, caused by either console BREAK condition, or Q22-bus BHALT_L.
?04	ISP ERR	Caused by attempt to push interrupt or exception state onto the interrupt stack when the interrupt stack was mapped NO ACCESS or NOT VALID.
?05	DBL ERR	A second machine check occurred while the processor was attempting to service a normal exception.
?06	HLT INST	The processor executed a HALT instruction in kernel mode.
?07	SCB ERR3	The vector had bits <1:0> = 3.
?08	SCB ERR2	The vector had bits <1:0> = 2.
?0A	CHM FR ISTK	A change mode instruction was executed when PSL<IS> was set.
?0B	CHM TO ISTK	The SCB vector for a changemode had bit <0> set.
?0C	SCB RD ERR	A hard memory error occurred during a processor read of an exception or interrupt vector.

Table 4–14 (Cont.) Halt Messages

Code	Message	Explanation
?10	MCHK AV	An access violation or an invalid translation occurred during machine check exception processing.
?11	KSP AV	An access violation or an invalid translation occurred during invalid kernel stack pointer exception processing.
?12	DBL ERR2	Double machine check error. A machine check occurred while trying to service a machine check.
?13	DBL ERR3	Double machine check error. A machine check occurred while trying to service a kernel stack not valid exception.
?19	PSL EXC5	PSL <26:24> = 5 on interrupt or exception.
?1A	PSL EXC6	PSL <26:24> = 6 on interrupt or exception.
?1B	PSL EXC5	PSL <26:24> = 7 on interrupt or exception.
?1D	PSL EXC5	PSL <26:24> = 5 on rei instruction.
?1E	PSL EXC5	PSL <26:24> = 6 on rei instruction.
?1F	PSL EXC5	PSL <26:24> = 7 on rei instruction.

4.8.2 Console Error Messages

Table 4–15 lists messages issued in response to a console command that has errors.

Table 4–15 Console Error Messages

Code	Message	Explanation
?20	CORRPTN	The console data base was corrupted. The console simulates a power-up sequence and rebuilds its data base.
?21	ILL REF	The requested reference would violate virtual memory protection, address is not mapped, or is invalid in the specified address space, or value is invalid in the specified destination.
?22	ILL CMD	The command string cannot be parsed.
?23	INV DGT	A number has an invalid digit.
?24	LTL	The command was too large for the console to buffer. The message is sent only after the console receives the Return at the end of the command.
?25	ILL ADR	The specified address is not in the address space.
?26	VAL TOO LRG	The specified value does not fit in the destination.

Table 4–15 (Cont.) Console Error Messages

Code	Message	Explanation
?27	SW CONF	Switch conflict. For example, an EXAMINE command specifies two different data sizes.
?28	UNK SW	The switch is not recognized.
?29	UNK SYM	The EXAMINE or DEPOSIT symbolic address is not recognized.
?2A	CHKSM	An X command has an incorrect command or data checksum. If the data checksum is incorrect, this message is issued, and is not abbreviated to "Illegal command."
?2B	HLTED	The operator entered a HALT command.
?2C	FND ERR	A FIND command failed either to find the RPB or 64 Kbytes of good memory.
?2D	TMOUT	Data failed to arrive in the expected time during an X command.
?2E	MEM ERR	Memory error.
?2F	UNXINT	An unexpected interrupt or exception occurred.
?30	UNIMPLEMENTED	Unimplemented function.
?31	QUAL NOVAL	Qualifier does not take a value.
?32	QUAL AMBG	Ambiguous qualifier.
?33	QUAL REQ VAL	Qualifier requires a value.
?34	QUAL OVERF	Too many qualifiers.
?35	ARG OVERF	Too many arguments.
?36	AMBG CMD	Ambiguous command.
?37	INSUF ARG	Insufficient arguments.

4.8.3 VMB Error Messages

If VMB is unable to boot, it returns an error message to the console. Table 4–16 lists the error messages and their descriptions.

Table 4–16 VMB Error Messages

Message Number	Mnemonic	Interpretation
?40	NOSUCHDEV	No bootable devices found
?41	DEVASSIGN	Device is not present
?42	NOSUCHFILE	Program image not found
?43	FILESTRUCT	Invalid boot device file structure
?44	BADCHKSUM	Bad checksum on header file
?45	BADFILEHDR	Bad file header
?46	BADDIRECTORY	Bad directory file
?47	FILNOTCNTG	Invalid program image format
?48	ENDOFFILE	Premature end of file encountered
?49	BADFILENAME	Bad file name given
?4A	BUFFEROVF	Program image does not fit in available memory
?4B	CTRLERR	Boot device I/O error
?4C	DEVINACT	Failed to initialize boot device
?4D	DEVOFFLINE	Device is offline
?4E	MEMERR	Memory initialization error
?4F	SCBINT	Unexpected SCB exception or machine check
?50	SCB2NDINT	Unexpected exception after starting program image
?51	NOROM	No valid ROM image found
?52	NOSUCHNODE	No response from load server
?53	INSFMAPREG	Invalid memory configuration
?54	RETRY	No devices bootable, retrying

A
Specifications

This appendix contains the physical, electrical and environmental specifications for the KA640 CPU module.

A.1 Physical Specifications

The KA640 and MS650-AA are quad-height modules with the following dimensions:

Dimension	Measurement
Height	10.457 (+0.015/-0.020) inches
Length	8.430 (+0.010/-0.010) inches
Width	0.375 inches maximum (nonconductive) 0.343 inches maximum (conductive)

NOTE

Width, as defined for Digital modules, is the height of components above the surface of the module.

A.2 Electrical Specifications

The power requirements for the KA640 CPU module are as follows:

+5 V ±5%	+12 V ±5%
6.0 A maximum	0.14 A maximum

Typical currents are 10% less than the specified maximum.

The bus loads for the KA640 CPU module are as follows:

- 3.5 ac loads
- 1.0 dc loads

A.3 Environmental Specifications

The environmental specifications for the KA640 CPU module are as follows:

Operating Conditions

Temperature	5 °C (41 °F) to 60 °C (140 °F) with a rate of change no greater than 20 ±2 °C (36 ±4 °F) per hour at sea level. Derate maximum temperature by 1.8 °C for each 1000 meters (1 °F for each 1000 ft) of altitude above sea level.
Humidity	0% to 95% noncondensing, with a maximum wet bulb temperature of 32 °C (90 °F) and a minimum dew point temperature of 2 °C (36 °F).
Altitude	Up to 2,400 meters (8,000 feet) with a rate of change no greater than 300 meters per minute (1000 feet per minute).

Nonoperating Conditions Less than 60 Days

Temperature	-40 °C to +66 °C (-40 °F to +151 °F) with a rate of change no greater than 11 ±2 °C (20 ±4 °F) per hour at sea level. Derate the maximum temperature by 1.8 °C for each 1000 meters (1 °F for each 1000 ft) of altitude above sea level.
Humidity	Up to 95% noncondensing.
Altitude	Up to 4,900 meters (16,000 feet) with a rate of change no greater than 600 meters per minute (2000 feet per minute).

Nonoperating Conditions Greater Than 60 Days

Temperature	+5 °C to +60 °C (+40 °F to +140 °F) with a rate of change no greater than 20 ±2 °C (36 ±4 °F) per hour at sea level. Derate the maximum temperature by 1.8 °C for each 1000 meters (1 °F for each 1000 ft) of altitude above sea level.
Humidity	10% to 95% noncondensing, with a maximum wet bulb temperature of 32 °C (90 °F) and a minimum dew point temperature of 2 °C (36 °F).
Altitude	Up to 2,400 meters (8,000 feet) with a rate of change no greater than 300 meters per minute (1000 feet per minute).

B

Address Assignments

B.1 General Local Address Space Map

Table B-1 lists the VAX memory space.

Table B-1 VAX Memory Space

Address Range	Contents
0000 0000 - 033F FFFF	Local memory space (52 Mbytes)
0340 0000 - 1FFF FFFF	Reserved memory space (460 Mbytes)

Table B-2 lists the VAX input/output memory space.

Table B-2 VAX Input/Output Space

Address Range	Contents
2000 0000 - 2000 1FFF	Local Q22-bus I/O space (8 Kbytes)
2000 2000 - 2003 FFFF	Reserved local I/O space (248 Kbytes)
2004 0000 - 2005 FFFF	Local ROM space - halt protected space (128 Kbytes)
2006 0000 - 2007 FFFF	Local ROM space - halt unprotected space (128 Kbytes)
2008 0000 - 201F FFFF	Local register I/O space (1.5 Mbytes)
2020 0000 - 23FF FFFF	Reserved local I/O space (62.5 Mbytes)
2400 0000 - 27FF FFFF	Reserved local I/O space (64 Mbytes)
2800 0000 - 2BFF FFFF	Reserved local I/O space (64 Mbytes)
2C08 0000 - 2FFF FFFF	Reserved local I/O space (64 Mbytes)
3000 0000 - 303F FFFF	Local Q22-bus memory space (4 Mbytes)
3040 0000 - 33FF FFFF	Reserved local I/O space (60 Mbytes)
3400 0000 - 37FF FFFF	Reserved local I/O space (64 Mbytes)

Table B–2 (Cont.) VAX Input/Output Space

Address Range	Contents
3800 0000 - 3BFF FFFF	Reserved local I/O space (64 Mbytes)
3C00 0000 - 3FFF FFFF	Reserved local I/O space (64 Mbytes)

B.2 Detailed Local Address Space Map

Table B–3 describes the contents of the VAX memory space.

Table B–3 VAX Memory Space

Contents	Address Range
Local memory space (up to 52 Mbytes)	0000 0000 - 033F FFFF
Q22-bus map - top 32 Kbytes of main memory	
Reserved memory space	0340 0000 - 1FFF FFFF

Table B–4 describes the contents of the VAX input/output memory space.

Table B–4 VAX Input/Output Space

Contents	Address Range
Local Q22-bus I/O Space	**2000 0000 - 2000 1FFF**
Reserved Q22-bus I/O space	2000 0000 - 2000 0007
Q22-bus floating address space	2000 0008 - 2000 07FF
User reserved Q22-bus I/O space	2000 0800 - 2000 0FFF
Reserved Q22-bus I/O space	2000 1000 - 2000 1F3F
Interprocessor communication register (normal operation)	2000 1F40
Interprocessor communication register (reserved)	2000 1F42
Interprocessor communication register (reserved)	2000 1F44
Interprocessor communication register (reserved)	2000 1F46
Reserved Q22-bus I/O space	2000 1F48 - 2000 1FFF
Reserved Local I/O Space	**2000 2000 - 2003 FFFF**

Table B–4 (Cont.) VAX Input/Output Space

Contents	Address Range
Local ROM Space	**2004 0000 - 2007 FFFF**
Local ROM protected space	2004 0000 - 2005 FFFF
MicroVAX system type register (in ROM)	2004 0004
Local ROM unprotected space	2006 0000 - 2007 FFFF
Local Register I/O Space	**2008 0000 - 201F FFFF**
DMA system configuration register	2008 0000
DMA system error register	2008 0004
Q22-bus error address register	2008 0008
DMA error address register	2008 000C
Q22-bus map base register	2008 0010
Reserved local register I/O space	2008 0014 - 2008 013C
Main memory error status register	2008 0140
Main memory control/diagnostic status register	2008 0144
Reserved local register I/O space	2008 0018 - 2008 3FFF
Reserved (1 copy of BDR)	2008 4000
Boot and diagnostic register	2008 4004
Reserved (126 copies of BDR)	2008 4008 - 2008 41FF
NI station address ROM	2008 4200 - 2008 427C
Reserved (4 copies of NISA ROM)	2008 4280 - 2008 43FF
NI register data port	2008 4400
NI register address port	2008 4404
64 copies of NIRDP, NIRAP	2008 4408 - 2008 45FF
MSI diagnostic register 0	2008 4600
MSI diagnostic register 1	2008 4604
MSI diagnostic register 2	2008 4608
MSI control and status register	2008 460C
MSI ID register	2008 4610
Reserved MSI register	2008 4614
Reserved MSI register	2008 4618
MSI timeout register	2008 461C
Reserved MSI register	2008 4620
Reserved MSI register	2008 4624
Reserved MSI register	2008 4628

Table B–4 (Cont.) VAX Input/Output Space

Contents	Address Range
Reserved MSI register	2008 462C
Reserved MSI register	2008 4630
Reserved MSI register	2008 4634
Reserved MSI register	2008 4638
MSI long target list pointer	2008 463C
MSI initiator list pointer	2008 4640
MSI DSSI control register	2008 4644
MSI DSSI status register	2008 4648
Reserved MSI register	2008 464C
Reserved MSI register	2008 4650
MSI diagnostic control register	2008 4654
MSI clock control register	2008 4658
MSI internal state register 0	2008 465C
MSI internal state register 1	2008 4660
MSI internal state register 2	2008 4664
MSI internal state register 3	2008 4668
Reserved MSI register	2008 466C
Reserved MSI register	2008 4670
Reserved MSI register	2008 4674
Reserved MSI register	2008 4678
Reserved MSI register	2008 467C
Reserved (4 copies of MSI reg block)	2008 4680 - 2008 47FF
NI buffer address extension register	2008 4800
Reserved (127 copies of NIBAER)	2008 4804 - 2008 49FF
Reserved local register I/O space	2008 4A00 - 2008 7FFF
Q22-bus map registers	2008 8000 - 2008 FFFF
Reserved local register I/O space	2009 0000 - 200F FFFF
MSI buffer RAM	2010 0000 - 2011 FFFF
Reserved local Register I/O space	2012 0000 - 2014 0020
Diagnostic LED register	2014 0030
Reserved local register I/O space	2014 0034 - 2014 0068
Diagnostic registers	2014 006C - 2014 00FF

Table B–4 (Cont.) VAX Input/Output Space

Contents	Address Range
Local Register I/O Space (Continued)	
Timer 0 control register	2014 0100
Timer 0 interval register	2014 0104
Timer 0 next interval register	2014 0108
Timer 0 interrupt vector	2014 010C
Timer 1 control register	2014 0110
Timer 1 interval register	2014 0114
Timer 1 next interval register	2014 0118
Timer 1 interrupt vector	2014 011C
Reserved local register I/O space	2014 0120 - 2014 03FF
Battery backed-up RAM	2014 0400 - 2014 07FF
Reserved local register I/O space	2014 0800 - 201F FFFF
Reserved local I/O space	2020 0000 - 2FFF FFFF
Local Q22-bus memory space	3000 0000 - 303F FFFF
Reserved local register I/O space	3040 0000 - 3FFF FFFF

B.3 External IPRs

Several of the internal processor registers (IPRs) on the KA640 are implemented in the SSC rather than in the CVAX chip. These registers are referred to as external IPRs, and are listed in Table B–5.

Table B–5 External IPRs

IPR Number	Register Name	Abbreviation
27	Time of year register	TOY
28	Console storage receiver status	CSRS*
29	Console storage receiver data	CSRD*
30	Console storage transmitter status	CSTS*
31	Console storage transmitter data	CSDB*
32	Console receiver control/status	RXCS
33	Console receiver data buffer	RXDB
34	Console transmitter control/status	TXCS
35	Console transmitter data buffer	TXDB
55	I/O system reset register	IORESET

*These registers are not fully implemented. Accesses yield *unpredictable* results.

B.4 Global Q22-bus Address Space Map

The addresses and memory contents of the Q22-bus memory space is listed in Table B–6.

Table B–6 Q22-bus Memory Space

Contents	Address
Q22-bus memory space (octal)	0000 0000 - 1777 7777

The contents and addresses of the Q22-bus I/O space with BBS7 asserted is listed Table B–7.

Table B–7 Q22-bus I/O Space with BBS7 Asserted

Contents	Address
Q22-bus I/O space (Octal)	1776 0000 - 1777 7777
Reserved Q22-bus I/O space	1776 0000
Q22-bus floating address space	1776 0010 - 1776 3777
User reserved Q22-bus I/O space	1776 4000 - 1776 7777
Reserved Q22-bus I/O space	1777 0000 - 1777 7477
Interprocessor communication register (normal operation)	1777 7500
Interprocessor communication register (reserved)	1777 7502
Interprocessor communication register (reserved)	1777 7504
Interprocessor communication register (reserved)	1777 7506
Reserved Q22-bus I/O space	1777 7510 - 1777 7777

C
Q22-bus Specification

C.1 General Description

The Q22-bus, also known as the extended LSI-11 bus, is the low-end member of Digital's bus family. All of Digital's microcomputers, such as the MicroVAX I, MicroVAX II, MicroVAX 3500, MicroVAX 3600, and MicroPDP-11 use the Q22-bus.

The Q22-bus consists of 42 bidirectional and 2 unidirectional signal lines. These form the lines along which the processor, memory, and I/O devices communicate with each other.

Addresses, data, and control information are sent along these signal lines, some of which contain time-multiplexed information. The lines are divided as follows:

- Sixteen multiplexed data/address lines—BDAL<15:00>
- Two multiplexed address/parity lines—BDAL<17:16>
- Four extended address lines—BDAL<21:18>
- Six data transfer control lines—BBS7, BDIN, BDOUT, BRPLY, BSYNC, BWTBT
- Six system control lines—BHALT, BREF, BEVNT, BINIT, BDCOK, BPOK
- Ten interrupt control and direct memory access control lines—BIAKO, BIAKI, BIRQ4, BIRQ5, BIRQ6, BIRQ7, BDMGO, BDMR, BSACK, BDMGI

In addition, a number of power, ground, and space lines are defined for the bus. Refer to Table C-1 for a detailed description of these lines.

The discussion in this appendix applies to the general 22-bit physical address capability. All modules used with the KA640 CPU module must use 22-bit addressing.

Most Q22-bus signals are bidirectional and use terminations for a negated (high) signal level. Devices connect to these lines by way of a high-impedance bus receivers and open collector drivers. The asserted state is produced when a bus driver asserts the line low.

Although bidirectional lines are electrically bidirectional (any point along the line can be driven or received), certain lines are functionally unidirectional. These lines communicate to or from a bus master (or signal source), but not both. Interrupt acknowledge (BIAK) and direct memory access grant (BDMG) signals are physically unidirectional in a daisy-chain fashion. These signals originate at the processor output signal pins. Each is received on device input pins (BIAKI or BDMGI) and is conditionally retransmitted via device output pins (BIAKO or BDMGO). These signals are received from higher-priority devices and are retransmitted to lower-priority devices along the bus, establishing the position-dependent priority scheme.

C.1.1 Master/Slave Relationship

Communication between devices on the bus is asynchronous. A master/slave relationship exists throughout each bus transaction. Only one device has control of the bus at any one time. This controlling device is termed the bus master, or arbiter. The master device controls the bus when communicating with another device on the bus, termed the slave.

The bus master (typically the processor or a DMA device) initiates a bus transaction. The slave device responds by acknowledging the transaction in progress and by receiving data from, or transmitting data to, the bus master. Q22-bus control signals transmitted or received by the bus master or bus slave device must complete the sequence according to bus protocol.

The processor controls bus arbitration, that is, which device becomes bus master at any given time. A typical example of this relationship is a disk drive, as master, transferring data to memory as slave. Communication on the Q22-bus is interlocked so that, for certain control signals issued by the master device, there must be a response from the slave in order to complete the transfer. It is the master/slave signal protocol that makes the Q22-bus asynchronous. The asynchronous operation precludes the need for synchronizing with, and waiting for, clock pulses.

Since bus cycle completion by the bus master requires response from the slave device, each bus master must include a timeout error circuit that aborts the bus cycle if the slave does not respond to the bus transaction within 10 μs. The actual time before a timeout error occurs must be longer than the reply time of the slowest peripheral or memory device on the bus.

C.2 Q22-bus Signal Assignments

Table C-1 lists the data and address signal assignments. Table C-2 lists the control signal assignments. Table C-3 lists the power and ground signal assignments. Table C-4 lists the spare signal assignments.

Table C-1 Data and Address Signal Assignments

Data and Address Signal	Pin Assignment
BDAL0	AU2
BDAL1	AV2
BDAL2	BE2
BDAL3	BF2
BDAL4	BH2
BDAL5	BJ2
BDAL6	BK2
BDAL7	BL2
BDAL8	BM2
BDAL9	BN2
BDAL10	BP2
BDAL11	BR2
BDAL12	BS2
BDAL13	BT2
BDAL14	BU2
BDAL15	BV2
BDAL16	AC1
BDAL17	AD1
BDAL18	BC1
BDAL19	BD1
BDAL20	BE1
BDAL21	BF1

Table C–2 Control Signal Assignments

Control Signal	Pin Assignment
Data Control	
BDOUT	AE2
BRPLY	AF2
BDIN	AH2
BSYNC	AJ2
BWTBT	AK2
BBS7	AP2
Interrupt Control	
BIRQ7	BP1
BIRQ6	AB1
BIRQ5	AA1
BIRQ4	AL2
BIAKO	AN2
BIAKI	AM2
DMA Control	
BDMR	AN1
BSACK	BN1
BDMGO	AS2
BDMGI	AR2
System Control	
BHALT	AP1
BREF	AR1
BEVNT	BR1
BINIT	AT2
BDCOK	BA1
BPOK	BB1

Table C-3 Power and Ground Signal Assignments

Power and Ground	Pin Assignment
+5 B (battery) or +12 B (battery)	AS1
+12 B	BS1
+5 B	AV1
+5	AA2
+5	BA2
+5	BV1
+12	AD2
+12	BD2
+12	AB2
-12	AB2
-12	BB2
GND	AC2
GND	AJ1
GND	AM1
GND	AT1
GND	BC2
GND	BJ1
GND	BM1
GND	BT1

Table C-4 Spare Signal Assignments

Spares	Pin Assignment
SSpare1	AE1
SSpare3	AH1
SSpare8	BH1
SSpare2	AF1
MSpareA	AK1
MSpareB	AL1
MSpareB	BK1
MSpareB	BL1
PSpare1	AU1
ASpare2	BU1

C.3 Data Transfer Bus Cycles

Data transfer bus cycles, executed by bus master devices, transfer 32-bit words or 8-bit bytes to or from slave devices. In block mode, multiple words can be transferred to sequential word addresses, starting from a single bus address. Data transfer bus cycles are listed and defined in Table C-5.

The bus signals listed in Table C-6 are used in the data transfer operations described in Table C-5.

Table C-5 Data Transfer Operations

Bus Cycle Mnemonic	Description	Function (with respect to the bus master)
DATI	Data word input	Read
DATO	Data word output	Write
DATOB	Data byte output	Write-byte
DATIO	Data word input/output	Read-modify-write
DATIOB	Data word input/byte output	Read-modify-write byte
DATBI	Data block input	Read block
DATBO	Data block output	Write block

Data transfer bus cycles can be reduced to five basic types: DATI, DATO(B), DATIO(B), DATBI, and DATBO. These transactions occur between the bus master and one slave device selected during the addressing part of the bus cycle.

C.3.1 Bus Cycle Protocol

Before initiating a bus cycle, the previous bus transaction must have been completed (BSYNC L negated) and the device must become bus master. The bus cycle can be divided into two parts: addressing and data transfer. During addressing, the bus master outputs the address for the desired slave device, memory location, or device register. The selected slave device responds by latching the address bits and holding this condition for the duration of the bus cycle until BSYNC L becomes negated. During data transfer the actual data transfer occurs.

Table C-6 Bus Signals for Data Transfers

Mnemonic	Description	Function
BDAL<21:00> L	22 Data/address lines	BDAL<15:00> L are used for word and byte transfers. BDAL<17:16> L are used for extended addressing, memory parity error (16), and memory parity error enable (17), functions. BDAL<21:18> L are used for extended addressing beyond 256 Kbytes.
BSYNC L	Bus cycle control	Indicates bus transaction in progress.
BDIN L	Data input indicator	Strobe signals.
BDOUT L	Data output indicator	Strobe signals.
BRPLY L	Slave's acknowledge of bus cycle	Strobe signals.
BWTBT L	Write/byte control	Control signals.
BBS7	I/O device select	Indicates address is in the I/O page.

C.3.2 Device Addressing

Device addressing of a data transfer bus cycle comprises an address setup and deskew time, and an address hold and deskew time. During address setup and deskew time, the bus master does the following operations:

- Asserts BDAL<21:00> L with the desired slave device address bits.
- Asserts BBS7 L if a device in the I/O page is being addressed.
- Asserts BWTBT L if the cycle is a DATO(B) or DATBO bus cycle.

During this time, the address, BBS7 L, and BWTBT L signals are asserted at the slave bus receiver for at least 75 ns before BSYNC goes active. Devices in the I/O page ignore the nine high-order address bits BDAL<21:13>, and instead, decode BBS7 L along with the 13 low-order address bits. An active BWTBT L signal during address setup time indicates that a DATO(B) or DATBO operation follows, while an inactive BWTBT L indicates a DATI, DATBI, or DATIO(B) operation.

The address hold and deskew time begins after BSYNC L is asserted.

The slave device uses the active BSYNC L bus received output to clock BDAL address bits, BBS7 L, and BWTBT L into its internal logic. BDAL<21:00> L, BBS7 L, and BWTBT L remain active for 25 ns minimum after the BSYNC L bus receiver goes active. BSYNC L remains active for the duration of the bus cycle.

Memory and peripheral devices are addressed similarly, except for the way the slave device responds to BBS7 L. Addressed peripheral devices must not decode address bits on BDAL<21:13> L. Addressed peripheral device can respond to a bus cycle when BBS7 L is asserted (low) during the addressing of the cycle. When asserted, BBS7 L indicates that the device address resides in the I/O page (the upper 4 Kbytes address space). Memory devices generally do not respond to addresses in the I/O page; however, some system applications may permit memory to reside in the I/O page for use as DMA buffers, read-only memory bootstraps, and diagnostics.

DATI

The DATI bus cycle, shown in Figure C-1, is a read operation. During DATI, data is input to the bus master. Data consists of 16-bit word transfers over the bus. During data transfer of the DATI bus cycle, the bus master asserts BDIN L 100 ns minimum after BSYNC L is asserted. The slave device responds to BDIN L active as follows:

- Asserts BRPLY L 0 ns minimum (8 ns maximum to avoid bus timeout) after receiving BDIN L, and 125 ns maximum before BDAL bus driver data bits are valid.
- Asserts BDAL<21:00> L with the addressed data and error information 0 ns (minimum) after receiving BDIN, and 125 ns (maximum) after assertion of BRPLY.

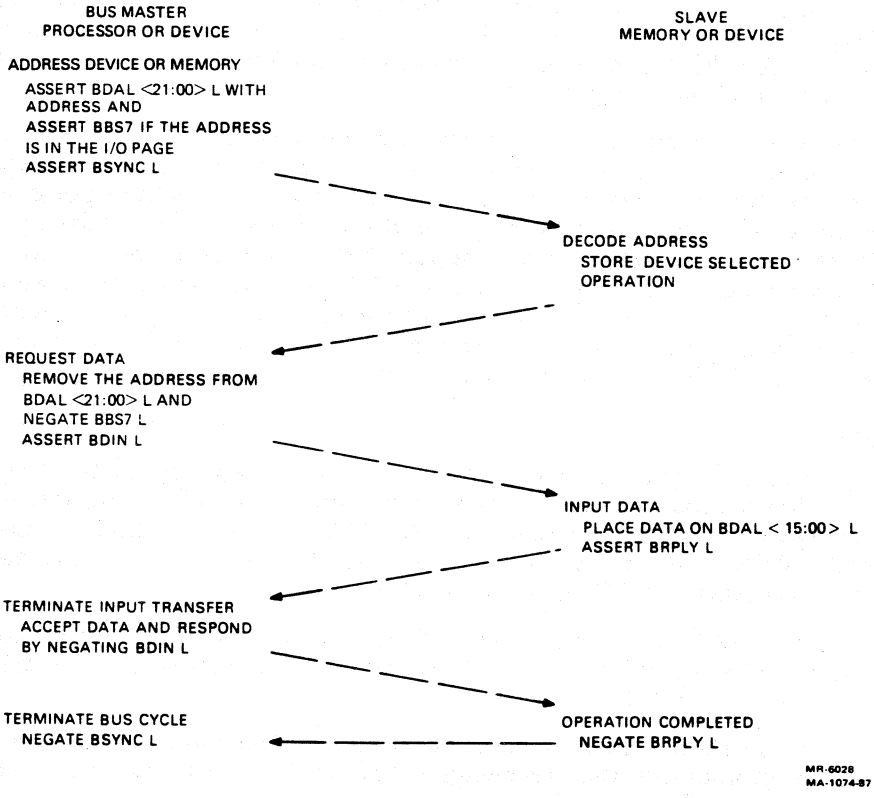


Figure C–1 DATI Bus Cycle

When the bus master receives BRPLY L, it does the following:

- Waits at least 200 ns deskew time and then accepts input data at BDAL<17:00> L bus receivers. BDAL <17:16> L are used for transmitting parity errors to the master.
- Negates BDIN L 200 ns minimum to 2 μs maximum after BRPLY L goes active.

The slave device responds to BDIN L negation by negating BRPLY L and removing read data from BDAL bus drivers. BRPLY L must be negated 100 ns maximum prior to removal of read data. The bus master responds to the negated BRPLY L by negating BSYNC L.

Conditions for the next BSYNC L assertion are as follows:

- BSYNC L must remain negated for 200 ns minimum.
- BSYNC L must not become asserted within 300 ns of previous BRPLY L negation.

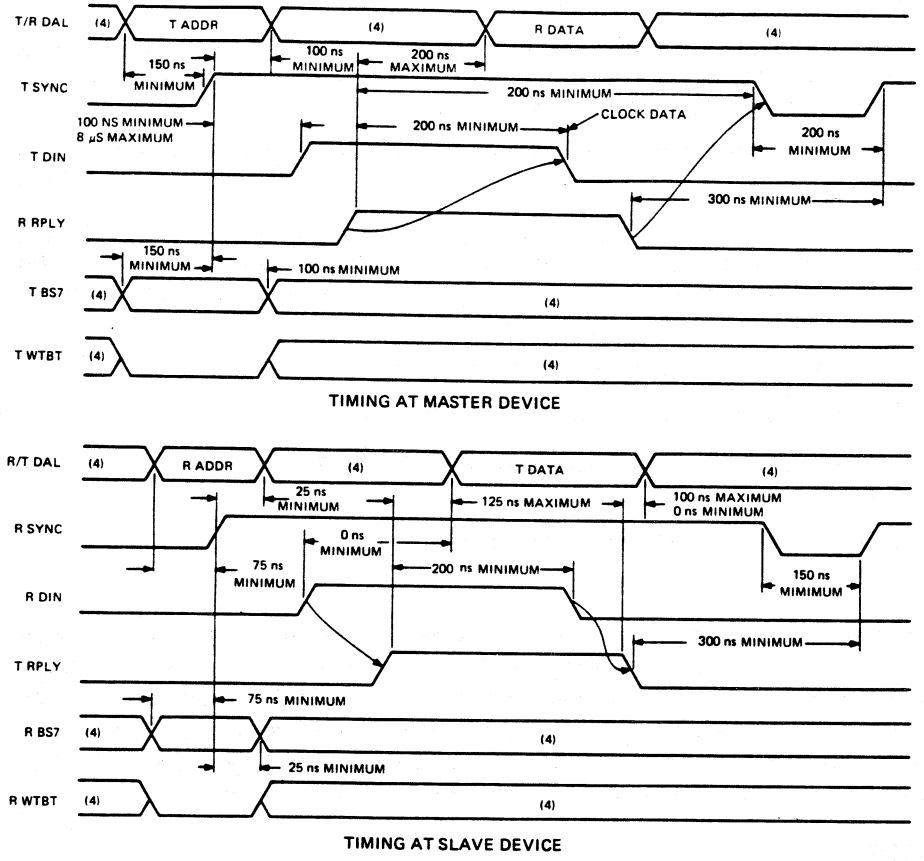
Figure C-2 shows DATI bus cycle timing.

NOTE

Continuous assertion of BSYNC L retains control of the bus by the bus master, and the previously addressed slave device remains selected. This is done for DATIO(B) bus cycles where DATO or DATOB follows a DATI without BSYNC L negation and a second device addressing operation. Also, a slow slave device can hold off data transfers to itself by keeping BRPLY L asserted, which causes the master to keep BSYNC L asserted.

DATOB

DATO(B), shown in Figure C-3, is a write operation. Data is transferred in 32-bit words (DATO) or 8-bit bytes (DATOB) from the bus master to the slave device. The data transfer output can occur after the addressing portion of a bus cycle when BWTBT L has been asserted by the bus master, or immediately following an input transfer part of a DATIO(B) bus cycle.



NOTES:

1. TIMING SHOWN AT MASTER AND SLAVE DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS.
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
   T = BUS DRIVER INPUT
   R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.
4. DON'T CARE CONDITION.

MR-6037
MA-1084-87

Figure C–2 DATI Bus Cycle Timing

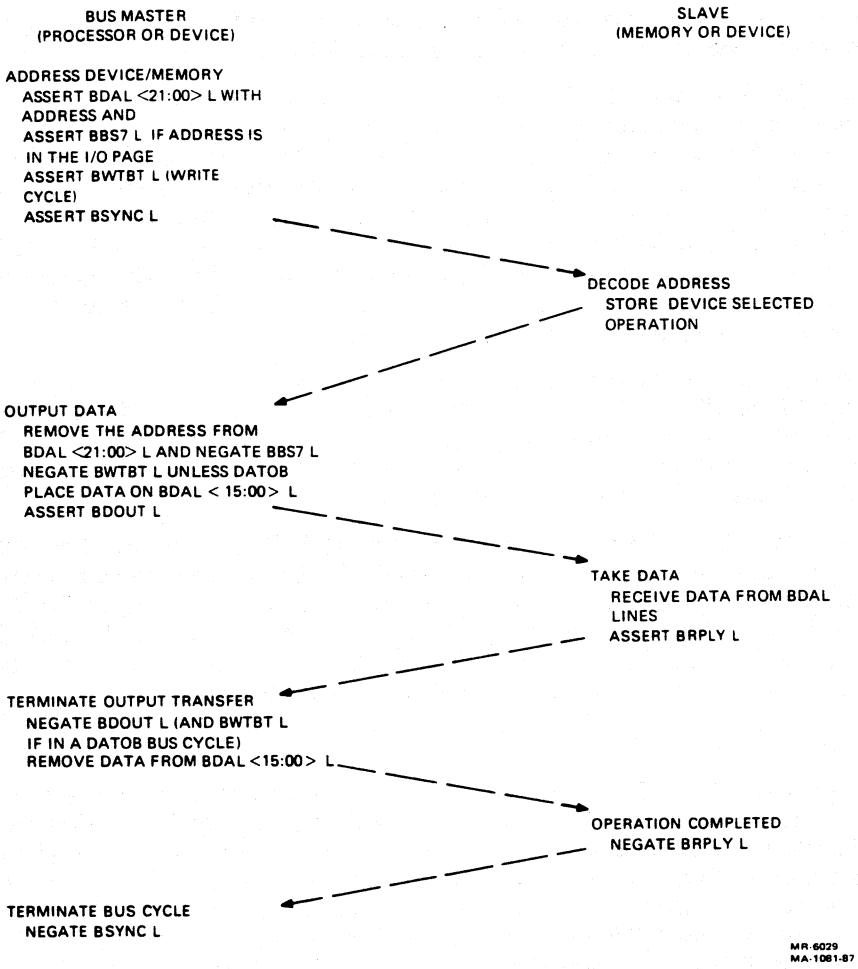


Figure C–3 DATO or DATOB Bus Cycle

The data transfer portion of a DATO(B) bus cycle comprises a data setup and deskew time and a data hold and deskew time.

During the data setup and deskew time, the bus master outputs the data on BDAL<15:00> L at least 100 ns after BSYNC L assertion. BWTBT L remains negated for the length of the bus cycle. If the transfer is a byte transfer, BWTBT L remains asserted. If it is the output of a DATIOB, BWTBT L becomes asserted and lasts the duration of the bus cycle.

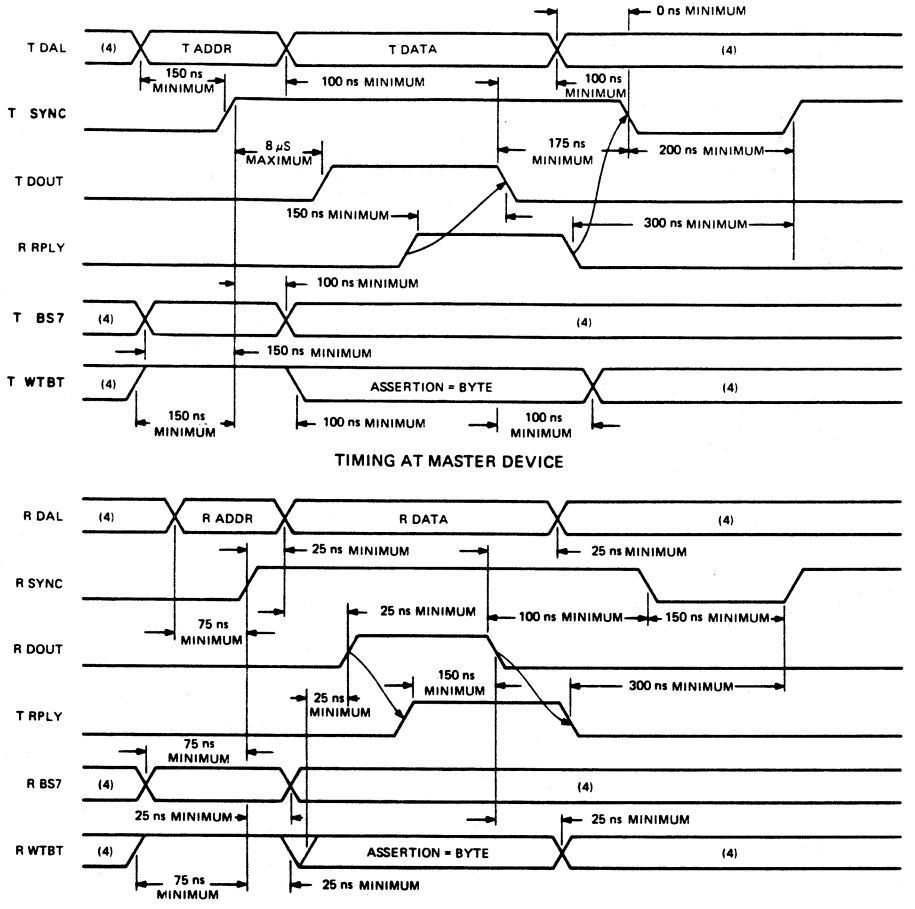
During a byte transfer, BDAL<00> L selects the high or low byte. This occurs in the addressing part of the cycle. If asserted, the high byte (BDAL<15:08> L) is selected; otherwise, the low byte (BDAL<07:00> L) is selected. An asserted BDAL 16 L at this time forces a parity error to be written into memory if the memory is a parity-type memory. BDAL 17 L is not used for write operations. The bus master asserts BDOUT L at least 100 ns after BDAL and BDWTBT L bus drivers are stable. The slave device responds by asserting BRPLY L within 10 μs to avoid bus timeout. This completes the data setup and deskew time.

During the data hold and deskew time, the bus master receives BRPLY L and negates BDOUT L, which must remain asserted for at least 150 ns from the receipt of BRPLY L before being negated by the bus master. BDAL<17:00> L bus drivers remain asserted for at least 100 ns after BDOUT L negation. The bus master then negates BDAL inputs.

During this time, the slave device senses BDOUT L negation. The data is accepted and the slave device negates BRPLY L. The bus master responds by negating BSYNC L. However, the processor does not negate BSYNC L for at least 175 ns after negating BDOUT L. This completes the DATO(B) bus cycle. Before the next cycle, BSYNC L must remain unasserted for at least 200 ns. Figure C-4 shows DATO(B) bus cycle timing.

DATIOB

The protocol for a DATIO(B) bus cycle is identical to the addressing and data transfer portions of the DATI and DATO(B) bus cycles, and is shown in Figure C-5. After addressing the device, a DATI cycle is performed as explained earlier; however, BSYNC L is not negated. BSYNC L remains active for an output word or byte transfer (DATO(B)). The bus master maintains at least 200 ns between BRPLY L negation during the DATI cycle and BDOUT L assertion. The cycle is terminated when the bus master negates BSYNC L, as described for DATO(B). Figure C-6 illustrates DATIO(B) bus cycle timing.



NOTES:

1. TIMING SHOWN AT MASTER AND SLAVE DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS.
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
   T = BUS DRIVER INPUT
   R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.
4. DON'T CARE CONDITION.

MR-1179
MA-1060-87

Figure C–4 DATO or DATOB Bus Cycle Timing

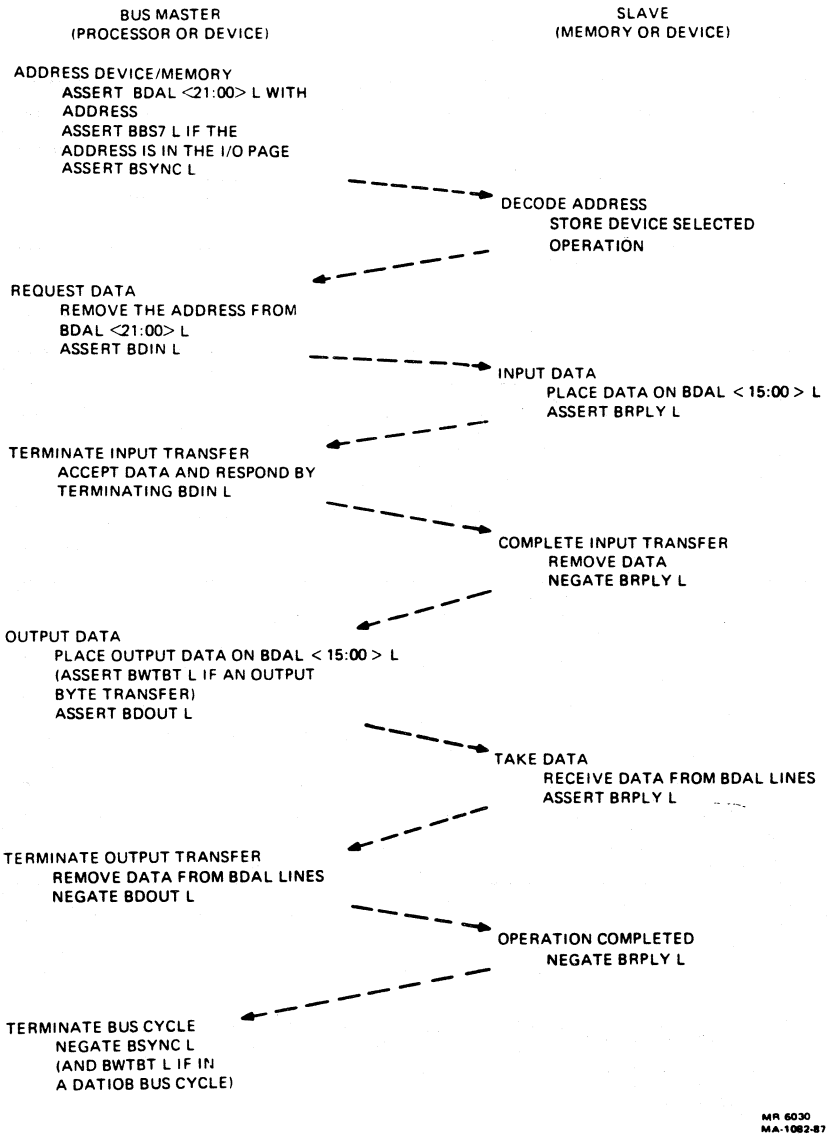
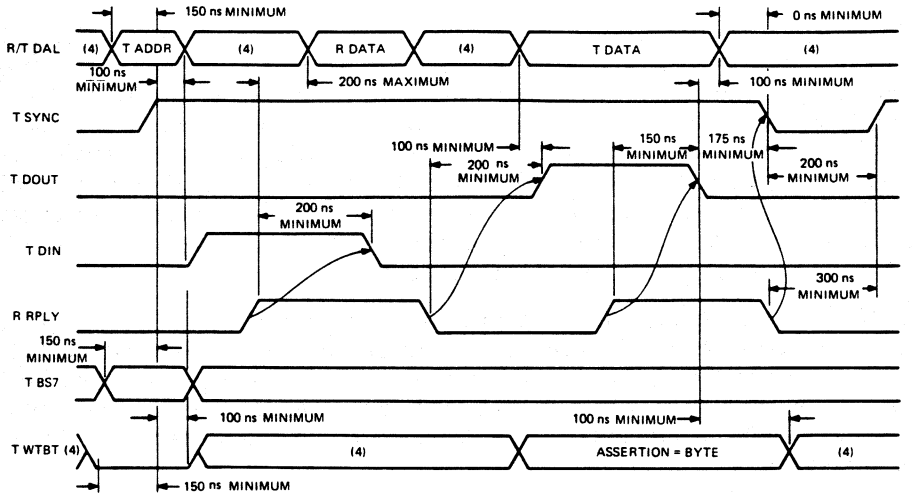
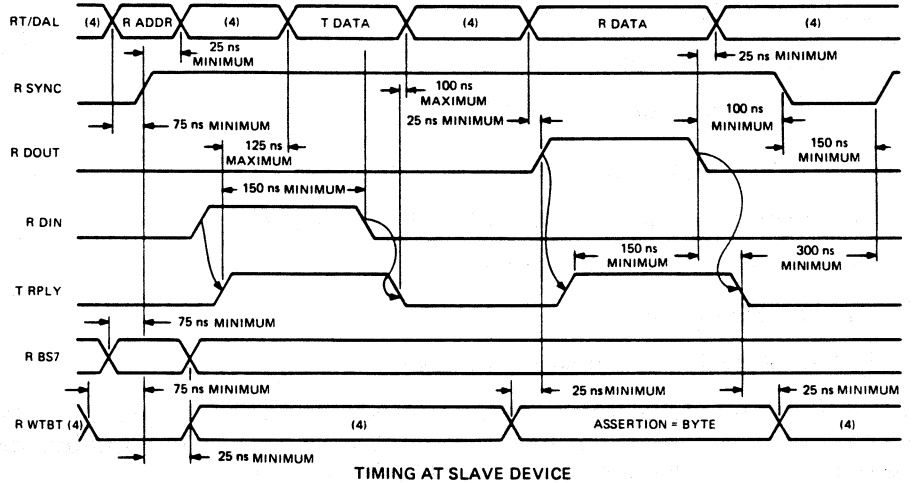


Figure C–5 DATIO or DATIOB Bus Cycle



TIMING AT MASTER DEVICE



TIMING AT SLAVE DEVICE

NOTES:

1. TIMING SHOWN AT REQUESTING DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
   T = BUS DRIVER INPUT
   R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.
4. DON'T CARE CONDITION.

MR-6036
MA-1060-87

Figure C–6 DATIO or DATIOB Bus Cycle Timing

C.4 Direct Memory Access

The direct memory access (DMA) capability allows direct data transfer between I/O devices and memory. This is useful when using mass storage devices (for example, disks) that move large blocks of data to and from memory. A DMA device needs to know only the starting address in memory, the starting address in mass storage, the length of the transfer, and whether the operation is read or write. When this information is available, the DMA device can transfer data directly to or from memory. Since most DMA devices must perform data transfers in rapid succession or lose data, DMA devices are given the highest priority.

DMA is accomplished after the processor (normally bus master) has passed bus mastership to the highest-priority DMA device that is requesting the bus. The processor arbitrates all requests and grants the bus to the DMA device electrically closest to it. A DMA device remains bus master until it relinquishes its mastership. The following control signals are used during bus arbitration:

- BDMGI L DMA grant input
- BDMGO L DMA grant output
- BDMR L DMA request line
- BSACK L bus grant acknowledge

C.4.1 DMA Protocol

A DMA transaction can be divided into three phases.

- Bus mastership acquisition phase
- Data transfer phase
- Bus mastership relinquishment phase

During the bus mastership acquisition phase, a DMA device requests the bus by asserting BDMR L. The processor arbitrates the request and initiates the transfer of bus mastership by asserting BDMGO L.

The maximum time between BDMR L assertion and BDMGO L assertion is DMA latency. This time is processor-dependent. BDMGO L/BDMGI L is one signal that is daisy-chained through each module in the backplane. It is driven out of the processor on the BDMGO L pin, enters each module on the BDMGI L pin, and exits on the BDMGO L pin. This signal passes through the modules in descending order of priority until it is stopped by the requesting device. The requesting device blocks the output of BMDGO L and asserts BSACK L. If BDMR L is continuously asserted, the bus hangs.

During the data transfer phase, the DMA device continues asserting BSACK L. The actual data transfer is performed as described earlier.

The DMA device can assert BSYNC L for a data transfer 250 ns minimum after it received BDMGI L and its BSYNC L bus receiver becomes negated.

During the bus mastership relinquishment phase, the DMA device gives up the bus by negating BSACK L. This occurs after completing (or aborting) the last data transfer cycle (BRPLY L negated). BSACK L can be negated up to a maximum of 300 ns before negating BSYNC L.

NOTE

If multiple data transfers are performed during this phase, consideration must be given to the use of the bus for other system functions, such as memory refresh (if required).

Figure C-7 shows the DMA protocol, and Figure C-8 shows DMA request/grant timing.

C.4.2 Block Mode DMA

For increased throughput, block mode DMA can be implemented on a device for use with memories that support this type of transfer. In a block mode transaction, the starting memory address is asserted, followed by data for that address, and data for consecutive addresses.

By eliminating the assertion of the address for each data word, the transfer rate is almost doubled.

There are two types of block mode transfers, DATBI (input) and DATBO (output). The DATBI bus cycle is described in Section C.4.2.1 and illustrated in Figure C-9.

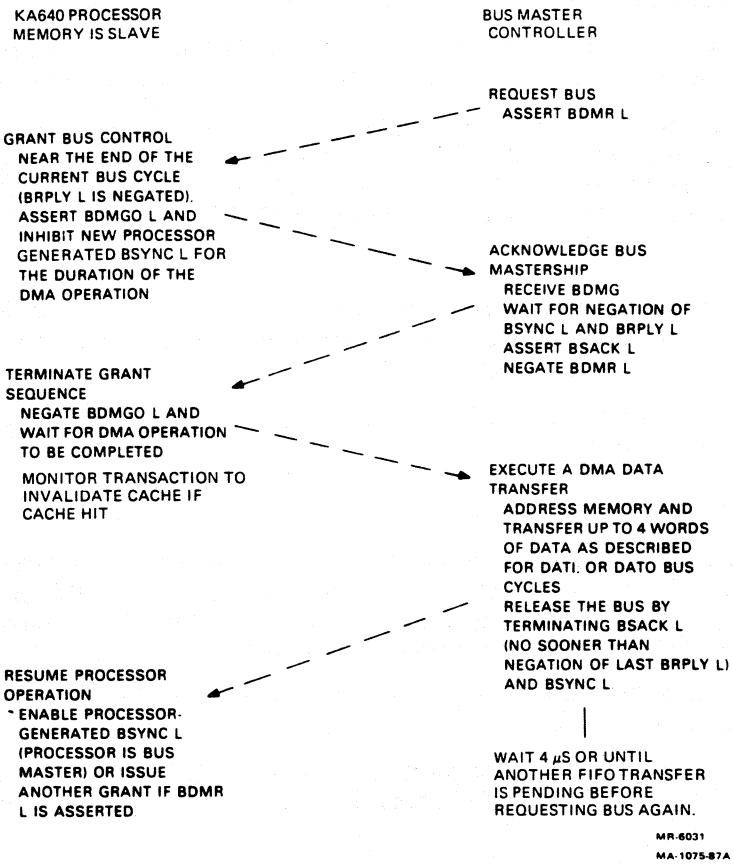
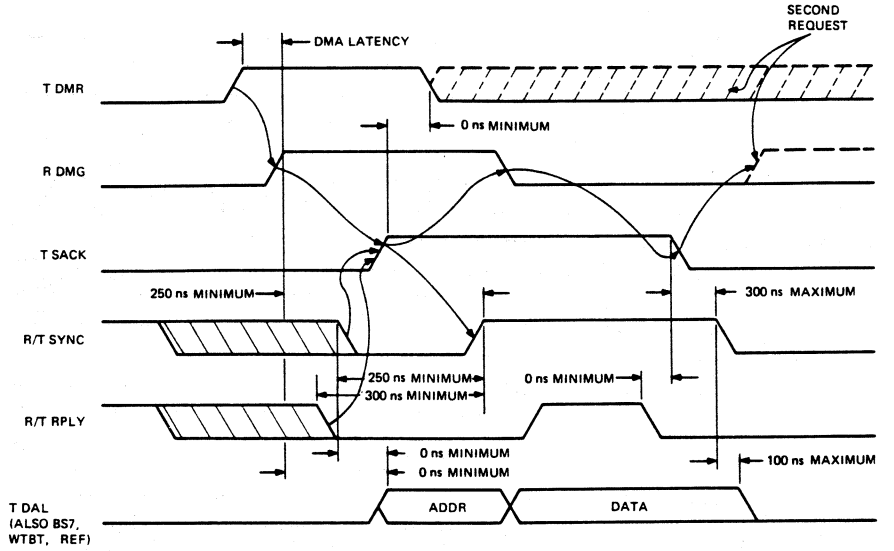


Figure C–7 DMA Protocol



NOTES:

1. TIMING SHOWN AT REQUESTING DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS.
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
   T = BUS DRIVER INPUT
   R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.

MR-3680
MA-1078-87

Figure C–8 DMA Request/Grant Timing

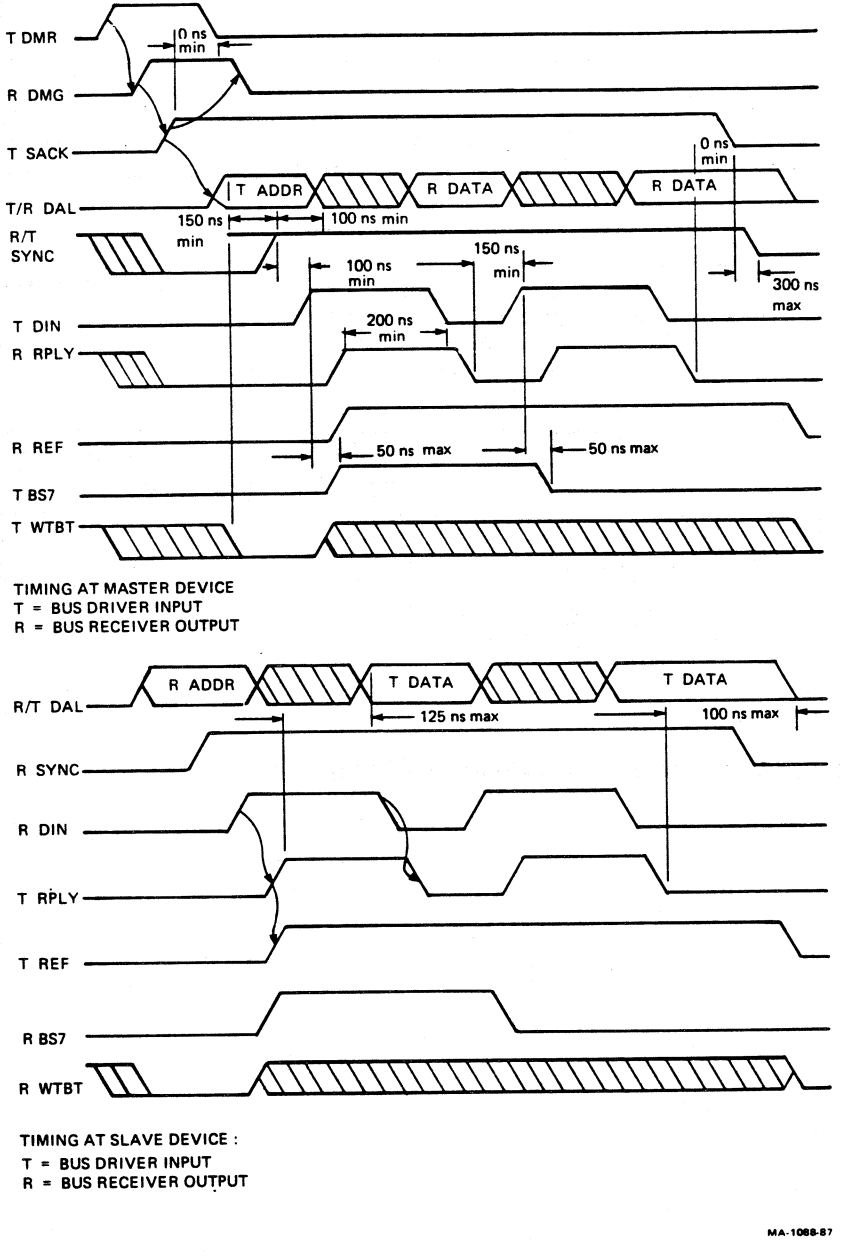


Figure C-9 DATBI Bus Cycle Timing

The DATBO bus cycle is described in Section C.4.2.2 and illustrated in Figure C-10.

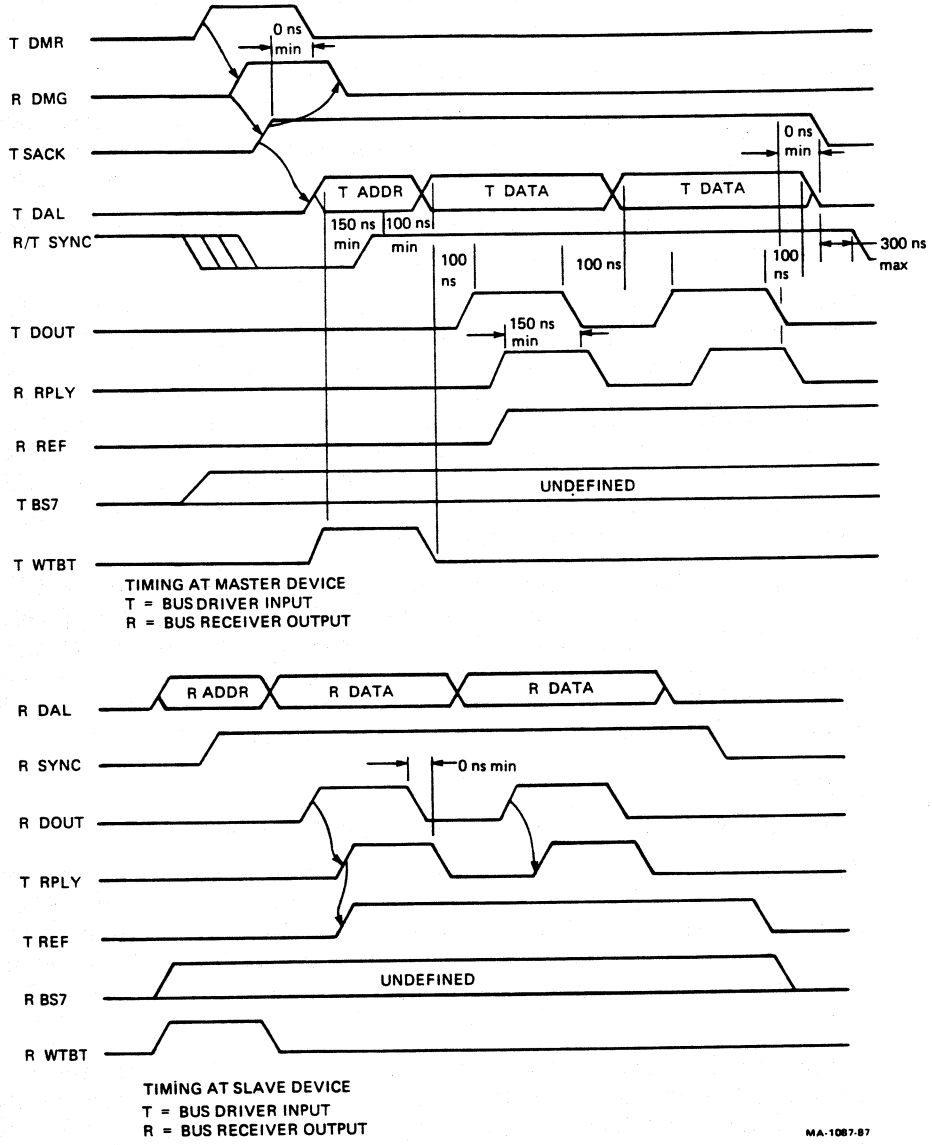


Figure C-10 DATBO Bus Cycle Timing

C.4.2.1 DATBI Bus Cycle

Before a DATBI block mode transfer can occur, the DMA bus master device must request control of the bus. This occurs under conventional Q22-bus protocol.

A block mode DATBI transfer is executed as follows:

- **Address Device Memory**—The address is asserted by the bus master on TADDR<21:00> along with the negation of TWTBT. The bus master asserts TSYNC 150 ns minimum after gating the address onto the bus.
- **Decode Address**—The appropriate memory device recognizes that it must respond to the address on the bus.
- **Request Data**—The address is removed by the bus master from TADDR<21:00> 100 ns minimum after the assertion of TSYNC. The bus master asserts the first TDIN 100 ns minimum after asserting TSYNC. The bus master asserts TBS7 50 ns maximum after asserting TDIN for the first time. TBS7 remains asserted until 50 ns maximum after the assertion of TDIN for the last time. In each case, TBS7 can be asserted or negated as soon as the conditions for asserting TDIN are met. The assertion of TBS7 indicates the bus master is requesting another read cycle after the current read cycle.
- **Send Data**—The bus slave asserts TRPLY 0 ns minimum (8000 ns maximum to avoid a bus timeout) after receiving RDIN. The bus slave asserts TREF concurrent with TRPLY if, and only if, it is a block mode device which can support another RDIN after the current RDIN. The bus slave gates TDATA<15:00> onto the bus 0 ns minimum after receiving RDIN and 125 ns maximum after the assertion of TRPLY.

  NOTE

  Block mode transfers must not cross 16 word boundaries.

- **Terminate Input Transfer**—The bus master receives stable RDATA<15:00> from 200 ns maximum after receiving RRPLY until 20 ns minimum after the negation of RDIN. (The 20 ns minimum represents total minimum receiver delays for RDIN at the slave and RDATA<15:00> at the master.) The bus master negates TDIN 200 ns minimum after receiving RRPLY.
- **Operation Completed**—The bus slave negates TRPLY 0 ns minimum after receiving the negation of RDIN. If RBS7 and TREF are both asserted when TRPLY negates, the bus slave prepares for another DIN cycle. RBS7 is stable from 125 ns after RDIN is received until 150 ns after TRPLY negates. If TBS7 and RREF were both asserted when TDIN negated, the bus master asserts TDIN 150 ns minimum after receiving the negation of RRPLY and continues with timing relationship *Send*

Data above. RREF is stable from 75 ns after RRPLY asserts until 20 ns minimum after TDIN negates. (The 0 ns minimum represents total minimum receiver delays for RDIN at the slave and RREF at the master.)

NOTE

The bus master must limit itself to not more than eight transfers unless it monitors RDMR. If it monitors RDMR, it may perform up to 16 transfers as long as RDMR is not asserted at the end of the seventh transfer.

- **Terminate Bus Cycle**—If RBS7 and TREF were not both asserted when TRPLY negated, the bus slave removes TDATA<15:00> from the bus 0 ns minimum and 100 ns maximum after negating TRPLY. If TBS7 and RREF were not both asserted when TDIN negated, the bus master negates TSYNC 250 ns minimum after receiving the last assertion of RRPLY and 0 ns minimum after the negation of that RRPLY.
- **Release The Bus**—The DMA bus master negates TSACK 0 ns after negation of the last RRPLY. The DMA bus master negates TSYNC 300 ns maximum after it negates TSACK. The DMA bus master must remove RDATA<15:00>, TBS7, and TWTBT from the bus 100 ns maximum after clearing TSYNC.

At this point the block mode transfer is complete, and the bus arbitration logic in the CPU enables processor-generated TSYNC or issues another bus grant (TDMGO) if RDMR is asserted.

C.4.2.2 DATBO Bus Cycle

Before a block mode transfer can occur, the DMA bus master device must request control of the bus. This occurs under conventional Q22-bus protocol.

A block mode DATBO transfer is executed as follows:

- **Address Device Memory**—The address is asserted by the bus master on TADDR<21:00> along with the aasertion of TWTBT. The bus master asserts TSYNC 150 ns minimum after gating the address onto the bus.
- **Decode Address**—The appropriate memory device recognizes that it must respond to the address on the bus.
- **Send Data**—The bus master gates TDATA<15:00> along with TWTBT 100 ns minimum after the assertion of TSYNC. TWTBT is negated. The bus master asserts the first TDOUT 100 ns minimum after gating TDATA<15:00>.

NOTE

During DATBO cycles, TBS7 is undefined.

- **Receive Data**—The bus slave receives stable data on RDATA<15:00> from 25 ns minimum before receiving RDOUT until 25 ns minimum after receiving the negation of RDOUT. The bus slave asserts TRPLY 0 ns minimum after receiving RDOUT. The bus slave asserts TREF concurrent with TRPLY if, and only if, it is a block mode device which can support another RDOUT after the current RDOUT.

NOTE

Block mode transfers must not cross 16 word boundaries.

- **Terminate Output Transfer**—The bus master negates TDOUT 150 ns minimum after receiving RRPLY.

- **Operation Completed**—The bus slave negates TRPLY 0 ns minimum after receiving the negation of RDOUT. If RREF was asserted when TDOUT negated and if the bus master wants to transfer another word, the bus master gates the new data on TDATA<15:00> 100 ns minimum after negating TDOUT. RREF is stable from 75 ns maximum after RRPLY asserts until 20 ns minimum after RDOUT negates. (The 20 ns minimum represents minimum receiver delays for RDOUT at the slave and RREF at the master). The bus master asserts TDOUT 100 ns minimum after gating new data on TDATA<15:00> and 150 ns minimum after receiving the negation of RRPLY. The cycle continues with the timing relationship in *Receive Data* above.

NOTE

The bus master must limit itself to not more than eight transfers unless it monitors RDMR. If it monitors RDMR, it may perform up to 16 transfers as long as RDMR is not asserted at the end of the seventh transfer.

- **Terminate Bus Cycle**—If RREF was not asserted when RRPLY negated or if the bus master has no additional data to transfer, the bus master removes data on TDATA<15:00> from the bus 100 ns minimum after negating TDOUT. If RREF was not asserted when TDOUT negated, the bus master negates TSYNC 275 ns minimum after receiving the last RRPLY and 0 ns minimum after the negation of the last RRPLY.

- **Release The Bus**—The DMA bus master negates TSACK 0 ns after negation of the last RRPLY. The DMA bus master negates TSYNC 300 ns maximum after it negates TSACK. The DMA bus master must remove TDATA, TBS7, and TWTBT from the bus 100 ns maximum after clearing TSYNC.

At this point the block mode transfer is complete, and the bus arbitration logic in the CPU enables processor-generated TSYNC or issues another bus grant (TDMGO) if RDMR is asserted.

C.4.3 DMA Guidelines

The following are DMA guidelines:

- Systems with memory refresh over the bus must not include devices that perform more than one transfer per acquisition.
- Bus masters that do not use block mode are limited to four DATI, four DATO, or two DATIO transfers per acquisition.
- Block mode bus masters that do not monitor BDMR are limited to eight transfers per acquisition.
- If BDMR is not asserted after the seventh transfer, block mode bus masters that do monitor BDMR may continue making transfers until the bus slave fails to assert BREF, or until they reach the total maximum of 16 transfers. Otherwise, they stop after eight transfers.

C.5 Interrupts

The interrupt capability of the Q22-bus allows an I/O device to temporarily suspend (interrupt) current program execution and divert processor operation to service the requesting device. The processor inputs a vector from the device to start the service routine (handler). Like the device register address, hardware fixes the device vector at locations within a designated range below location 001000. The vector indicates the first of a pair of addresses. The processor reads the contents of the first address, the starting address of the interrupt handler. The contents of the second address is a new processor status word (PS).

The new PS can raise the interrupt priority level, thereby preventing lower-level interrupts from breaking into the current interrupt service routine. Control is returned to the interrupted program when the interrupt handler is ended. The original interrupted program's address (PC) and its associated PS are stored on a stack. The original PC and PS are restored by a return from interrupt (RTI or RTT) instruction at the end of the handler. The use of the stack and the Q22-bus interrupt scheme can allow interrupts to occur within interrupts (nested interrupts), depending on the PS.

Interrupts can be caused by Q22-bus options or the MicroVAX CPU. Those interrupts that originate from within the processor are called traps. Traps are caused by programming errors, hardware errors, special instructions, and maintenance features.

The following Q22-bus signals are used in interrupt transactions:

BIRQ4 L	Interrupt request priority level 4
BIRQ5 L	Interrupt request priority level 5
BIRQ6 L	Interrupt request priority level 6
BIRQ7 L	Interrupt request priority level 7
BIAKI L	Interrupt acknowledge input
BIAKO L	Interrupt acknowledge output
BDAL<21:00>	Data/address lines
BDIN L	Data input strobe
BRPLY L	Reply

C.5.1 Device Priority

The Q22-bus supports the following two methods of device priority:

- Distributed Arbitration—priority levels are implemented on the hardware. When devices of equal priority level request an interrupt, priority is given to the device electrically closest to the processor.
- Position-Defined Arbitration—priority is determined solely by electrical position on the bus. The closer a device is to the processor, the higher its priority is.

C.5.2 Interrupt Protocol

Interrupt protocol on the Q22-bus has three phases.

- Interrupt request
- Interrupt acknowledge and priority arbitration
- Interrupt vector transfer phase

The interrupt request phase begins when a device meets its specific conditions for interrupt requests. For example, the device is ready, done, or an error has occurred. The interrupt enable bit in a device status register must be set. The device then initiates the interrupt by asserting the interrupt request line(s). BIRQ4 L is the lowest hardware priority level and is asserted for all interrupt requests for compatibility with previous Q22-bus processors. The level at which a device is configured must also be asserted. A special case exists for level 7 devices that must also assert level 6. For an explanation, refer to the discussion on arbitration involving the 4-level scheme.

Interrupt Level	Lines Asserted by Device
4	BIRQ4 L
5	BIRQ4 L, BIRQ5 L
6	BIRQ4 L, BIRQ6 L
7	BIRQ4 L, BIRQ6 L, BIRQ7 L

Figure C-11 shows the interrupt request/acknowledge sequence.

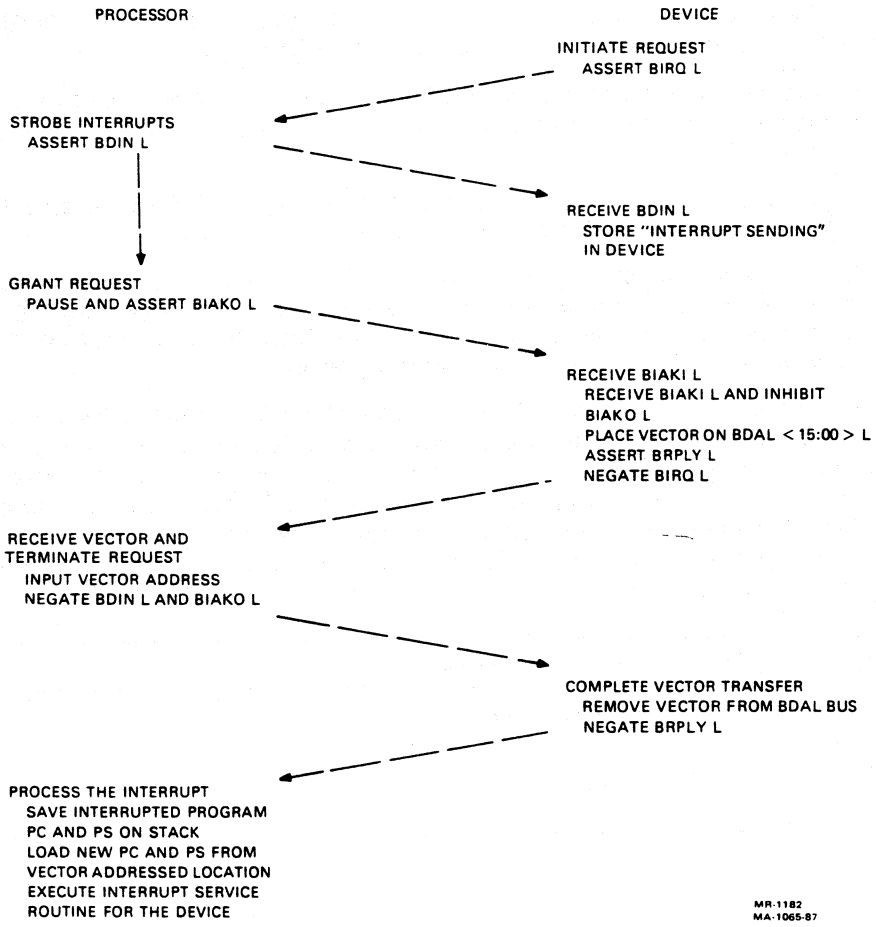


Figure C-11 Interrupt Request/Acknowledge Sequence

The interrupt request line remains asserted until the request is acknowledged.

During the interrupt acknowledge and priority arbitration phase, the processor acknowledges interrupts under the following conditions:

- The device interrupt priority is higher than the current PS<7:5>.
- The processor has completed instruction execution and no additional bus cycles are pending.

The processor acknowledges the interrupt request by asserting BDIN L, and 150 ns minimum later asserting BIAKO L. The device electrically closest to the processor receives the acknowledge on its BIAKI L bus receiver.

At this point, the two types of arbitration must be discussed separately. If the device that receives the acknowledge uses the 4-level interrupt scheme, it reacts as follows:

- If not requesting an interrupt, the device asserts BIAKO L and the acknowledge propagates to the next device on the bus.
- If the device is requesting an interrupt, it must check that no higher-level device is currently requesting an interrupt. This is done by monitoring higher-level request lines. The table below lists the lines that need to be monitored by devices at each priority level.

In addition to asserting levels 7 and 4, level 7 devices must drive level 6. This is done to simplify the monitoring and arbitration by level 4 and 5 devices. In this protocol, level 4 and 5 devices need not monitor level 7 because level 7 devices assert level 6. Level 4 and 5 devices become aware of a level 7 request because they monitor the level 6 request. This protocol has been optimized for level 4, 5, and 6 devices, since level 7 devices are very seldom necessary.

Device Priority Level	Line(s) Monitored
4	BIRQ5, BIRQ6
5	BIRQ6
6	BIRQ7
7	-

- If no higher-level device is requesting an interrupt, the acknowledge is blocked by the device. (BIAKO L is not asserted.) Arbitration logic within the device uses the leading edge of BDIN L to clock a flip-flop that blocks BIAKO L. Arbitration is won, and the interrupt vector transfer phase begins.
- If a higher-level request line is active, the device disqualifies itself and asserts BIAKO L to propagate the acknowledge to the next device along the bus.

Signal timing must be considered carefully when implementing 4-level interrupts (Figure C–12).

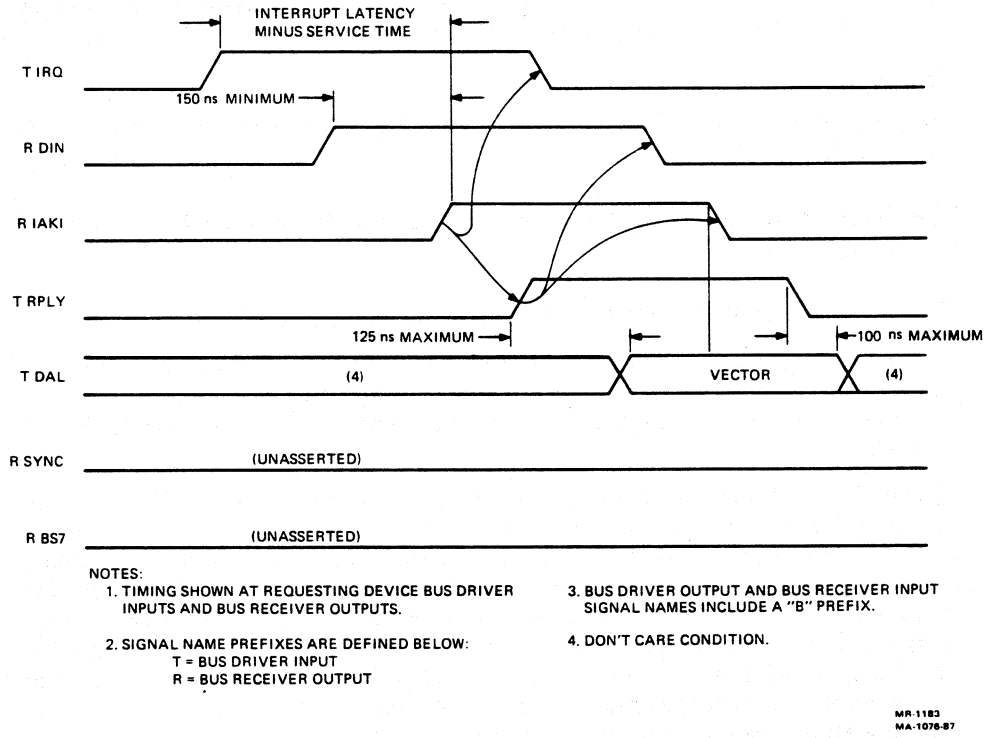


Figure C–12 Interrupt Protocol Timing

If a single-level interrupt device receives the acknowledge, it reacts as follows:

- If not requesting an interrupt, the device asserts BIAKO L and the acknowledge propagates to the next device on the bus.
- If the device was requesting an interrupt, the acknowledge is blocked using the leading edge of BDIN L, and arbitration is won. The interrupt vector transfer phase begins.

The interrupt vector transfer phase is enabled by BDIN L and BIAKI L. The device responds by asserting BRPLY L and its BDAL<15:00> L bus driver inputs with the vector address bits. The BDAL bus driver inputs must be stable within 125 ns maximum after BRPLY L is asserted. The processor then inputs the vector address and negates BDIN L and BIAKO L. The device then negates BRPLY L and 100 ns maximum later removes the vector address bits. The processor then enters the device's service routine.

NOTES

Propagation delay from BIAKI L to BIAKO L must not be greater than 500 ns per Q22-bus slot.

The device must assert BRPLY L within 10 μs maximum after the processor asserts BIAKI L.

C.5.3 Q22-bus 4-Level Interrupt Configurations

If you have high-speed peripherals and desire better software performance, you can use the 4-level interrupt scheme. Both position-independent and position-dependent configurations can be used with the 4-level interrupt scheme.

Figure C-13 shows the position-independent configuration. This allows peripheral devices that use the 4-level interrupt scheme to be placed in the backplane in any order. These devices must send out interrupt requests and monitor higher-level request lines as described. The level 4 request is always asserted from a requesting device regardless of priority. If two or more devices of equally high priority request an interrupt, the device physically closest to the processor wins arbitration. Devices that use the single-level interrupt scheme must be modified, or placed at the end of the bus, for arbitration to function properly.

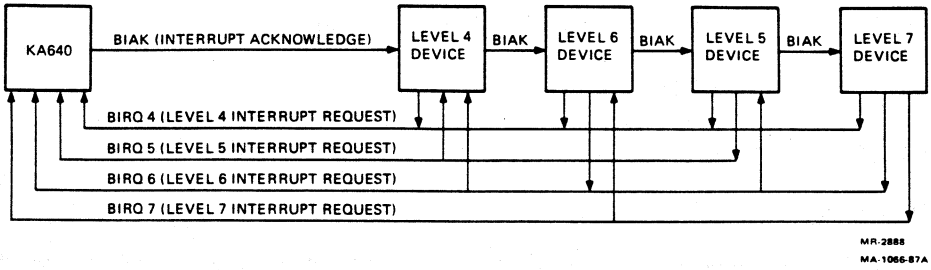


Figure C–13 Position-Independent Configuration

Figure C–14 shows the position-dependent configuration. This configuration is simpler to implement. A constraint is that peripheral devices must be inserted with the highest-priority device located closest to the processor, and the remaining devices placed in the backplane in decreasing order of priority (with the lowest-priority devices farthest from the processor). With this configuration, each device has to assert only its own level and level 4. Monitoring higher-level request lines is unnecessary. Arbitration is achieved through the physical positioning of each device on the bus. Single-level interrupt devices on level 4 should be positioned last on the bus.

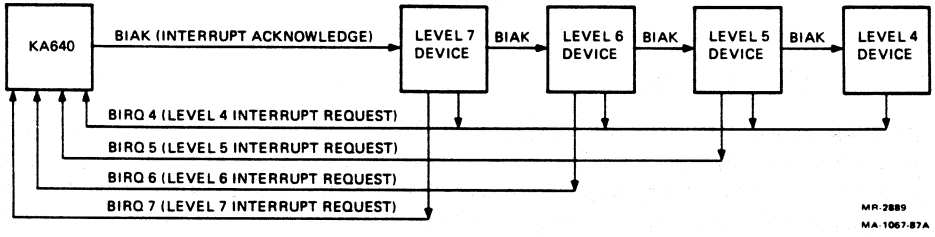


Figure C–14 Position-Dependent Configuration

C.6 Control Functions

The following Q22-bus signals provide control functions:

BREF L	Memory refresh (also block mode DMA)
BHALT L	Processor halt
BINIT L	Initialize
BPOK H	Power OK
BDCOK H	DC power OK

C.6.1 Memory Refresh

If BREF is asserted during the address part of a bus data transfer cycle, it causes all dynamic MOS memories to be addressed simultaneously. The sequence of addresses required for refreshing the memories is determined by the specific requirements for each memory. The complete memory refresh cycle consists of a series of refresh bus transactions. A new address is used for each transaction. A complete memory refresh cycle must be completed within 1 or 2 ms. Multiple data transfers by DMA devices must be avoided since they could delay memory refresh cycles. This type of refresh is done only for memories that do not perform on-board refresh.

C.6.2 Halt

Assertion of BHALT L for at least 25 ns interrupts the processor, which stops program execution and forces the processor unconditionally into console I/O mode.

C.6.3 Initialization

Devices along the bus are initialized when BINIT L is asserted. The processor can assert BINIT L as a result of executing a reset instruction as part of a power-up or power-down sequence. BINIT L is asserted for approximately 10 μs when reset is executed.

C.6.4 Power Status

Power status protocol is controlled by two signals, BPOK H and BDCOK H. These signals are driven by an external device (usually the power supply).

C.6.5 BDCOK H

When asserted, this control indicates that dc power has been stable for at least 3 ms. Once asserted, this line remains asserted until the power fails. It indicates that only 5 μs of dc power reserve remains.

C.6.6 BPOK H

When asserted, this control indicates there is at least an 8 ms reserve of dc power, and that BDCOK H has been asserted for at least 70 ms. Once BPOK has been asserted, it must remain asserted for at least 3 ms. The negation of this line, the first event in the power-fail sequence, indicates that power is failing and that only 4 ms of dc power reserve remains.

C.6.7 Power-Up/Power-Down Protocol

Power-up protocol begins when the power supply applies power with BDCOK H negated. This forces the processor to assert BINIT L. When the dc voltages are stable, the power supply or other external device asserts BDCOK H. The processor responds by clearing the PS, floating point status register (FPS), and floating point exception register (FEC). BINIT L is asserted for 12.6 μs, and then negated for 110 μs. The processor continues to test for BPOK H until it is asserted. The power supply asserts BPIK H 70 ms minimum after BDCOK H is asserted. The processor then performs its power-up sequence. Normal power must be maintained at least 3.0 ms before a power-down sequence can begin.

A power-down sequence begins when the power supply negates BPOK H. When the current instruction is completed, the processor traps to a power-down routine at location 24. The end of the routine is terminated with a halt instruction to avoid any possible memory corruption as the dc voltages decay.

When the processor executes the halt instruction, it tests the BPOK H signal. If BPOK H is negated, the processor enters the power-up sequence. It clears internal registers, generates BINIT L, and continues to check for the assertion of BPOK H. If it is asserted and dc voltages are still stable, the processor performs the rest of the power-up sequence. Figure C-15 shows power-up/power-down timing.

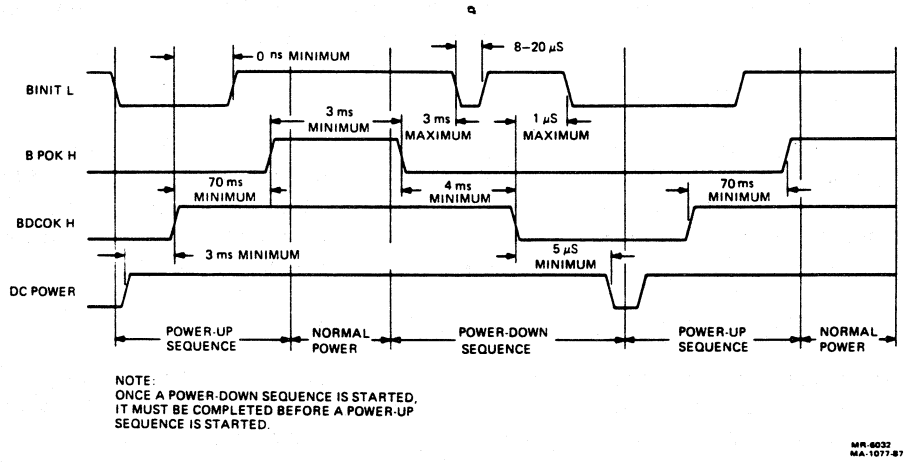


Figure C–15 Power-Up/Power-Down Timing

C.7 Q22-bus Electrical Characteristics

The input and output logic levels for Q22-bus signals are given in Section C.7.1.

C.7.1 Signal Level Specifications

The signal level specifications for the Q22-bus are as follows:

Input Logic Level

TTL logical low	0.8 Vdc maximum
TTL logical high	2.0 Vdc minimum

Output Logic Level

TTL logical low	0.4 Vdc maximum
TTL logical high	2.4 Vdc minimum

C.7.2 Load Definition

AC loads make up the maximum capacitance allowed per signal line to ground. A unit load is defined as 9.35 pF of capacitance. DC loads are defined as maximum current allowed with a signal line driver asserted or unasserted. A unit load is defined as 210 μA in the unasserted state.

C.7.3 120-Ohm Q22-bus

The electrical conductors interconnecting the bus device slots are treated as transmission lines. A uniform transmission line, terminated in its characteristic impedance, propagates an electrical signal without reflections. Since bus drivers, receivers, and wiring connected to the bus have finite resistance and nonzero reactance, the transmission line impedance is not uniform, and introduces distortions into pulses propagated along it. Passive components of the Q22-bus (such as wiring, cabling, and etched signal conductors) are designed to have a nominal characteristic impedance of 120 ohms.

The maximum length of interconnecting cable, excluding wiring within the backplane, is limited to 4.88 m (16 feet).

C.7.4 Bus Drivers

Devices driving the 120-ohm Q22-bus must have open collector outputs and meet the following specifications:

DC Specifications

- Output low voltage when sinking 70 mA of current is 0.7 V maximum.
- Output high leakage current when connected to 3.8 Vdc is 25 μA (even if no power is applied, except for BDCOK H and BPOK H).
- These conditions must be met at worst-case supply temperature, and input signal levels.

AC Specifications

- Bus driver output pin capacitance load should not exceed 10 pF.
- Propagation delay should not exceed 35 ns.
- Skew (difference in propagation time between slowest and fastest gate) should not exceed 25 ns.
- Transition time (from 10% to 90% for positive transition—rise time, from 90% to 10% for negative transition—fall time) must be no faster than 10 ns.

C.7.5 Bus Receivers

Devices that receive signals from the 120-ohm Q22-bus must meet the following requirements:

DC Specifications

- Input low voltage maximum is 1.3 V.
- Input high voltage minimum is 1.7 V.
- Maximum input current when connected to 3.8 Vdc is 80 μA (even if no power is applied).

These specifications must be met at worst-case supply voltage, temperature, and output signal conditions.

AC Specifications

- Bus receiver input pin capacitance load should not exceed 10 pF.
- Propagation delay should not exceed 35 ns.
- Skew (difference in propagation time between slowest and fastest gate) should not exceed 25 ns.

C.7.6 Bus Termination

The 120-ohm Q22-bus must be terminated at each end by an appropriate terminator, as shown in Figure C–16. This is to be done as a voltage divider with its Thevenin equivalent equal to 120 ohms and 3.4 V (nominal). This type of termination is provided by an REV11-A refresh/boot/terminator, BDV11-AA, KPV11-B, TEV11, or by certain backplanes and expansion cards.

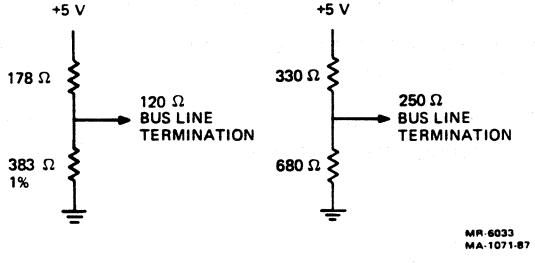


Figure C–16 Bus Line Terminations

Each of the several Q22-bus lines (all signals whose mnemonics start with the letter B) must see an equivalent network with the following characteristics at each end of the bus.

Input impedance (with respect to ground)	120 ohm +5%, -15%
Open circuit voltage	3.4 Vdc +5%
Capacitance load	Not to exceed 30 pF

NOTE

The resistive termination can be provided by the combination of two modules. (The processor module supplies 220 ohms to ground. This, in parallel with another 220-ohm card, provides 120 ohms.) Both terminators must reside physically within the same backplane.

C.7.7 Bus Interconnecting Wiring

This section gives specific information about bus interconnecting wiring.

C.7.7.1 Backplane Wiring

The wiring that connects all device interface slots on the Q22-bus must meet the following specifications:

- The conductors must be arranged so that each line exhibits a characteristic impedance of 120 ohms (measured with respect to the bus common return).
- Crosstalk between any two lines must be no greater than 5%. Note that worst-case crosstalk is manifested by simultaneously driving all but one signal line and measuring the effect on the undriven line.
- DC resistance of the signal path, as measured between the near-end terminator and the far-end terminator module (including all intervening connectors, cables, backplane wiring, and connector-module etch) must not exceed 20 ohms.
- DC resistance of the common return path, as measured between the near-end terminator and the far-end terminator module (including all intervening connectors, cables, backplane wiring, connector-module etch, etc.) must not exceed an equivalent of 2 ohms per signal path. Thus, the composite signal return path dc resistance must not exceed 2 ohms divided by 40 bus lines, or 50 milliohms. Note that although this common return path is nominally at ground potential, the conductance must be part of the bus wiring. The specified low impedance return path must be provided by the bus wiring as distinguished from the common system or power ground path.

C.7.7.2 IntraBackplane Bus Wiring

The wiring that connects the bus connector slots within one contiguous backplane is part of the overall bus transmission line. Owing to implementation constraints, the nominal characteristic impedance of 120 ohms may not be achievable. Distributed wiring capacitance in excess of the amount required to achieve the nominal 120-ohm impedance may not exceed 60 pF per signal line per backplane.

C.7.7.3 Power and Ground

Each bus interface slot has connector pins assigned for the following dc voltages. The maximum allowable current per pin is 1.5 A. +5 Vdc must be regulated to 5% with a maximum ripple of 100 mV pp. +12 Vdc must be regulated to 3% with a maximum ripple of 200 mV pp.

- +5 Vdc—three pins (4.5 A maximum per bus device slot)
- +12 Vdc—two pins (3.0 A maximum per bus device slot)
- Ground—eight pins (shared by power return and signal return)

NOTE

Power is not bussed between backplanes on any interconnecting bus cables.

C.8 System Configurations

Q22-bus systems can be divided into two types.

- Systems containing one backplane
- Systems containing multiple backplanes

Before configuring any system, three characteristics for each module in the system must be identified.

- Power consumption—+5 Vdc and +12 Vdc are the current requirements.
- AC bus loading—The amount of capacitance a module presents to a bus signal line. AC loading is expressed in terms of ac loads, where one ac load equals 9.35 pF of capacitance.
- DC bus loading—The amount of dc leakage current a module presents to a bus signal when the line is high (undriven). DC loading is expressed in terms of dc loads, where one dc load equals 210 μA (nominal).

Power consumption, ac loading, and dc loading specifications for each module are included in the *Microcomputer Interfaces Handbook*.

NOTE

The ac and dc loads and the power consumption of the processor module, terminator module, and backplane must be included in determining the total loading of a backplane.

Rules for configuring single-backplane systems are as follows:

- When using a processor with 220-ohm termination, the bus can accommodate modules that have up to 20 ac loads before additional termination is required (Figure C–17). If more than 20 ac loads are included, the other end of the bus must be terminated with 120 ohms, and then up to 35 ac loads may be present.
- With 120-ohm processor termination, up to 35 ac loads can be used without additional termination. If 120-ohm bus termination is added, up to 45 ac loads can be configured in the backplane.
- The bus can accommodate modules up to 20 dc loads (total).
- The bus signal lines on the backplane can be up to 35.6 cm (14 inches) long.

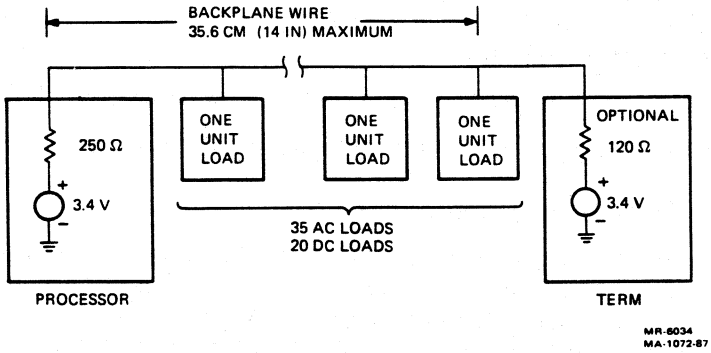


Figure C–17 Single-Backplane Configuration

Rules for configuring multiple-backplane systems are as follows:

- Figure C–18 shows that up to three backplanes can make up the system.
- The signal lines on each backplane can be up to 25.4 cm (10 inches) long.

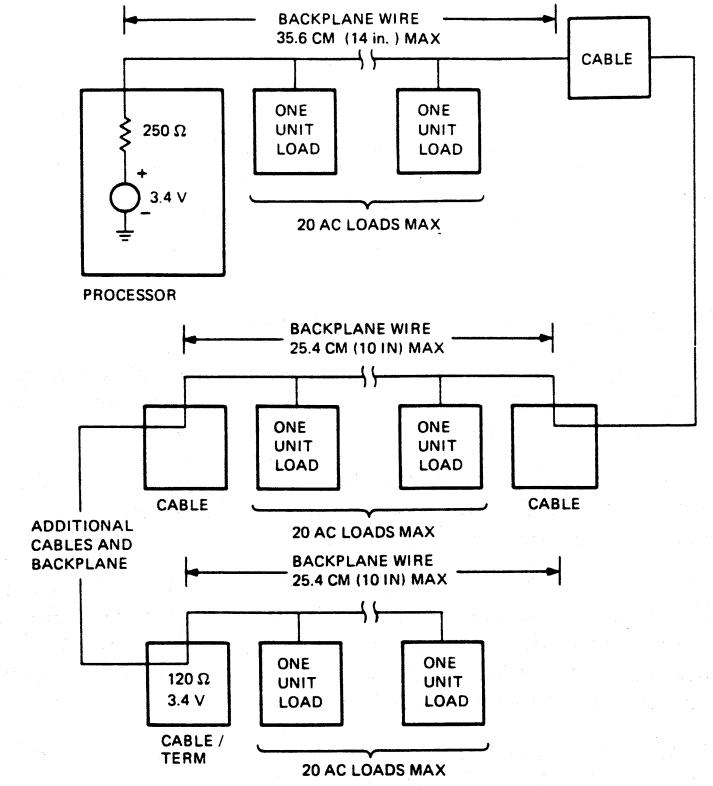
- Each backplane can accommodate modules that have up to 22 ac loads. Unused ac loads from one backplane may not be added to another backplane if the second backplane loading exceeds 22 ac loads. It is desirable to load backplanes equally, or with the highest ac loads in the first and second backplanes.
- DC loading of all modules in all backplanes cannot exceed 20 loads.
- Both ends of the bus must be terminated with 120 ohms. This means the first and last backplanes must have an impedance of 120 ohms. To achieve this, each backplane can be lumped together as a single point. The resistive termination can be provided by a combination of two modules in the backplane with the processor providing 220 ohms to ground in parallel with an expansion paddle card providing 250 ohms to give the needed 120-ohm termination.

  Alternately, a processor with 120-ohm termination would need no additional termination on the paddle card to attain 120 ohms in the first box. The 120-ohm termination in the last box can be provided in two ways: the termination resistors may reside either on the expansion paddle card, or on a bus termination card (such as the BDV11).
- The cable(s) connecting the first two backplanes is 61 cm (2 feet) or more in length.
- The cable(s) connecting the second backplane to the third backplane is 122 cm (4 feet) longer or shorter than the cable(s) connecting the first and second backplanes.
- The combined length of both cables cannot exceed 4.88 m (16 feet).
- The cables used must have a characteristic impedance of 120 ohms.

C.8.1 Power Supply Loading

Total power requirements for each backplane can be determined by obtaining the total power requirements for each module in the backplane. Obtain separate totals for +5 V and +12 V power. Power requirements for each module are specified in the *Microcomputer Interfaces Handbook*.

When distributing power in multiple-backplane systems, do not attempt to distribute power via the Q22-bus cables. Provide separate, appropriate power wiring from each power supply to each backplane. Each power supply should be capable of asserting BPOK H and BDCOK H signals according to bus protocol; this is required if automatic power-fail/restart programs are implemented, or if specific peripherals require an orderly power-down halt sequence. The proper use of BPOK H and BDCOK H signals is strongly recommended.



NOTES:
1. TWO CABLES (MAX) 4.88 M (16 FT) (MAX) TOTAL LENGTH.
2. 20 DC LOADS TOTAL (MAX).

MR-6035
MA-1073-87

Figure C–18 Multiple-Backplane Configuration

C.9 Module Contact Finger Identification

Digital's plug-in modules all use the same contact finger (pin) identification system. A typical pin is shown in Figure C–19.

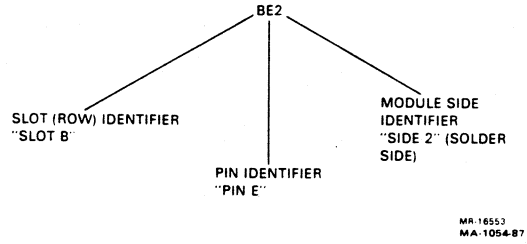


Figure C–19 Typical Pin Identification System

The Q22-bus is based on the use of quad-height modules that plug into a 2-slot bus connector. Each slot contains 36 lines (18 lines on both the component side and the solder side of the circuit board).

Slots, row A, and row B include a numeric identifier for the side of the module. The component side is designated side 1, the solder side is designated side 2, as shown in Figure C–20.

Letters ranging from A through V (excluding G, I, O, and Q) identify a particular pin on a side of a slot. Table C–7 lists and identifies the bus pins of the quad-height module. A bus pin identifier ending with a 1 is found on the component side of the board, while a bus pin identifier ending with a 2 is found on the solder side of the board.

The positioning notch between the two rows of pins mates with a protrusion on the connector block for correct module positioning.

The dimensions for a typical Q22-bus module are represented in Figure C–21.

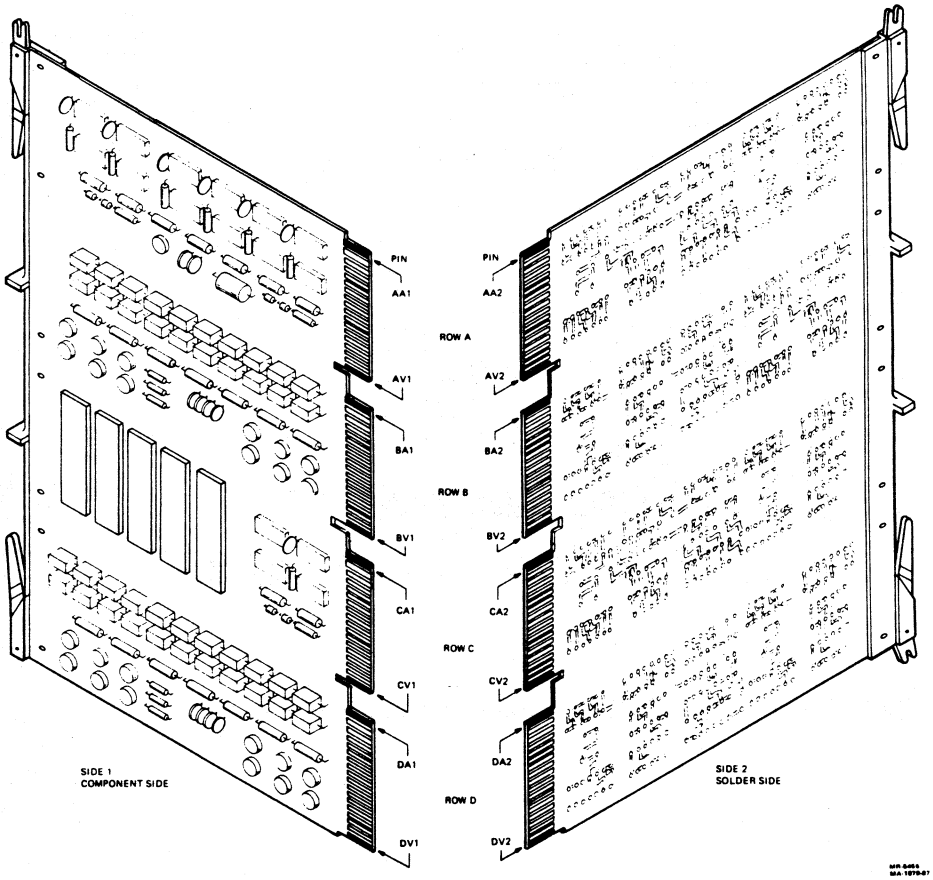


Figure C–20 Quad-Height Module Contact Finger Identification

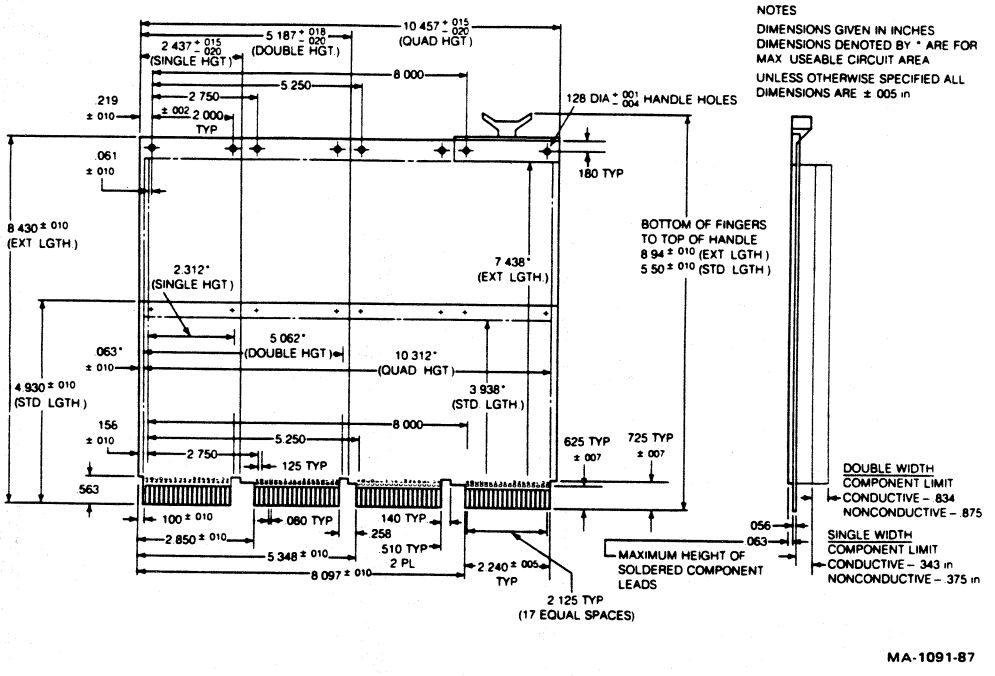


Figure C–21 Typical Q22-bus Module Dimensions

Table C-7 Bus Pin Identifiers

Bus Pin	Mnemonic(s)	Description
AA1	BIRQ5 L	Interrupt request priority level 5.
AB1	BIRQ6 L	Interrupt request priority level 6.
AC1	BDAL16 L	Extended address bit during addressing protocol; memory error data line during data transfer protocol.
AD1	BDAL17 L	Extended address bit during addressing protocol; memory error logic enable during data transfer protocol.
AE1	SSPARE1 (alternate +5B)	Special spare—not assigned or bussed in Digital's cable or backplane assemblies; available for user connection. Optionally, this pin can be used for +5 V battery (+5 B) back-up power to keep critical circuits alive during power failures. A jumper is required on Q22-bus options to open (disconnect) the +5 B circuit in systems that use this line as SSPARE1.
AF1	SSPARE2	Special spare—not assigned or bussed in Digital's cable or backplane assemblies; available for user interconnection. In the highest-priority device slot, the processor can use this pin for a signal to indicate its RUN state.
AH1	SSPARE3 SRUN	Special spare—not assigned or bussed simultaneously in Digital's cable or backplane assemblies; available for user interconnection. An alternate SRUN signal can be connected in the highest-priority set.
AJ1	GND	Ground—system signal ground and dc return.
AK1	MSPAREA	Maintenance spare—normally connected together on the backplane at each option location (not a bussed connection).
AL1	MSPAREB	Maintenance spare—normally connected together on the backplane at each option location (not a bussed connection).
AM1	GND	Ground—system signal ground and dc return.

Table C-7 (Cont.) Bus Pin Identifiers

Bus Pin	Mnemonic(s)	Description
AN1	BDMR L	Direct memory access (DMA) request—a device asserts this signal to request bus mastership. The processor arbitrates bus mastership between itself and all DMA devices on the bus. If the processor is not bus master (it has completed a bus cycle and BSYNC L is not being asserted by the processor), it grants bus mastership to the requesting device by asserting BDMGO L. The device responds by negating BDMR L and asserting BSACK L.
AP1	BHALT L	Processor halt—when BHALT L is asserted for at least 25 μs, the processor services the halt interrupt and responds by halting normal program execution. External interrupts are ignored but memory refresh interrupts in Q22-bus operations are enabled if W4 on the M7264 and M7264-YA processor modules is removed and DMA request/grant sequences are enabled. The processor executes the ODT microcode, and the console device operation is invoked.
AR1	BREF L	Memory refresh—asserted by a DMA device. This signal forces all dynamic MOS memory units requiring bus refresh signals to be activated for each BSYNC L/BDIN L bus transaction. It is also used as a control signal for block mode DMA. **CAUTION** **The user must avoid multiple DMA data transfers (burst or hot mode) that could delay refresh operation if using DMA refresh. Complete refresh cycles must occur once every 1.6 ms if required.**
AS1	+12 B or +5 B	+12 Vdc or +5 V battery back-up power to keep critical circuits alive during power failures. This signal is not bussed to BS1 in all of Digital's backplanes. A jumper is required on all Q22-bus options to open (disconnect) the backup circuit from the bus in systems that use this line at the alternate voltage.

Table C-7 (Cont.) Bus Pin Identifiers

Bus Pin	Mnemonic(s)	Description
AT1	GND	Ground—system signal ground and dc return.
AU1	PSPARE 1	Spare—not assigned; customer usage not recommended. Prevents damage when modules are inserted upside down.
AV1	+5 B	+5 V battery power—secondary +5 V power connection. Battery power can be used with certain devices.
BA1	BDCOK H	DC power OK—a power supply generated signal that is asserted when the available dc voltage is sufficient to sustain reliable system operation.
BB1	BPOK H	Power OK—asserted by the power supply 70 ms after BDCOK is negated when ac power drops below the value required to sustain power (approximately 75% of nominal). When negated during processor operation, a power-fail trap sequence is initiated.
BC1	SSPARE4 BDAL18 L (22-bit only)	Special spare in the Q22-bus—not assigned. Bussed in 22-bit cable and backplane assemblies; available for user interconnection.
BD1	SSPARE5 BDAL19 L (22-bit only)	**CAUTION** **These pins may be used by manufacturing as test points in some options.**
BE1	SSPARE6 BDAL20 L	In the Q22-bus, these bussed address lines are address lines <21:18>; currently not used during data time.
BF1	SSPARE7 BDAL21 L	In the Q22-bus, these bussed address lines are address lines <21:18>; currently not used during data time.
BH1	SSPARE8	Special spare—not assigned or bussed in Digital's cable and backplane assemblies; available for user interconnection.
BJ1	GND	Ground—system signal ground and dc return.

Table C-7 (Cont.) Bus Pin Identifiers

Bus Pin	Mnemonic(s)	Description
BK1 BL1	MSPAREB MSPAREB	Maintenance spare—normally connected together on the backplane at each option location (not a bussed connection).
BM1	GND	Ground—system signal ground and dc return.
BN1	BSACK L	This signal is asserted by a DMA device in response to the processor's BDMGO L signal, indicating that the DMA device is bus master.
BP1	BIRQ7 L	Interrupt request priority level 7.
BR1	BEVNT L	External event interrupt request—when asserted, the processor responds by entering a service routine via vector address 1008. A typical use of this signal is as a line time clock (LTC) interrupt.
BS1	+12 B	+12 Vdc battery back-up power (not bussed to AS1 in all of Digital's backplanes).
BT1	GND	Ground—system signal ground and dc return.
BU1	PSPARE2	Power spare 2—not assigned a function; not recommended for use. If a module is using -12 V (on pin AB2), and if the module is accidentally inserted upside down in the backplane, -12 Vdc appears on pin BU1.
BV1	+5	+5 V power—normal +5 Vdc system power.
AA2	+5	+5 V power—normal +5 Vdc system power.
AB2	-12	-12 V power—-12 Vdc power for (optional) devices requiring this voltage. **NOTE** **Each Q22-bus module that requires negative voltages contains an inverter circuit that generates the required voltage(s). Therefore, -12 V power is not required with Digital's options.**
AC2	GND	Ground—system signal ground and dc return.

Table C–7 (Cont.) Bus Pin Identifiers

Bus Pin	Mnemonic(s)	Description
AD2	+12	+12 V power—+12 Vdc system power.
AE2	BDOUT L	Data output—when asserted, BDOUT implies that valid data is available on BDAL<0:15> L and that an output transfer, with respect to the bus master device, is taking place. BDOUT L is deskewed with respect to data on the bus. The slave device responding to the BDOUT L signal must assert BRPLY L to complete the transfer.
AF2	BRPLY L	Reply—BRPLY L is asserted in response to BDIN L or BDOUT L and during IAK transactions. It is generated by a slave device to indicate that it has placed its data on the BDAL bus or that it has accepted output data from the bus.
AH2	BDIN L	Data input—BDIN L is used for two types of bus operations. • When asserted during BSYNC L time, BDIN L implies an input transfer with respect to the current bus master, and requires a response (BRPLY L). BDIN L is asserted when the master device is ready to accept data from the slave device. • When asserted without BSYNC L, it indicates that an interrupt operation is occurring. The master device must deskew input data from BRPLY L.
AJ2	BSYNC L	Synchronize—BSYNC L is asserted by the bus master device to indicate that it has placed an address on BDAL<0:17> L. The transfer is in process until BSYNC L is negated.

Table C–7 (Cont.) Bus Pin Identifiers

Bus Pin	Mnemonic(s)	Description
AK2	BWTBT L	Write/byte—BWTBT L is used in two ways to control a bus cycle. • It is asserted at the leading edge of BSYNC L to indicate that an output sequence (DATO or DATOB), rather than an input sequence, is to follow. • It is asserted during BDOUT L, in a DATOB bus cycle, for byte addressing.
AL2	BIRQ4 L	Interrupt request priority level 4—a level 4 device asserts this signal when its interrupt enable and interrupt request flip-flops are set. If the PS word bit 7 is 0, the processor responds by acknowledging the request by asserting BDIN L and BIAKO L.
AM2 AN2	BIAKI L BIAKO L	Interrupt acknowledge—in accordance with interrupt protocol, the processor asserts BIAKO L to acknowledge receipt of an interrupt. The bus transmits this to BIAKI L of the device electrically closest to the processor. This device accepts the interrupt acknowledge under two conditions. • The device requested the bus by asserting BIRQn L (where **n**= 4, 5, 6 or 7) • The device has the highest-priority interrupt request on the bus at that time. If these conditions are not met, the device asserts BIAKO L to the next device on the bus. This process continues in a daisy-chain fashion until the device with the highest-interrupt priority receives the interrupt acknowledge signal.
AP2	BBS7 L	Bank 7 select—the bus master asserts this signal to reference the I/O page (including that part of the page reserved for nonexistent memory). The address in BDAL<0:12> L when BBS7 L is asserted is the address within the I/O page.

Table C-7 (Cont.) Bus Pin Identifiers

Bus Pin	Mnemonic(s)	Description
AR2 AS2	BDMGI L BDMGO L	Direct memory access grant— the bus arbitrator asserts this signal to grant bus mastership to a requesting device, according to bus mastership protocol. The signal is passed in a daisy-chain from the arbitrator (as BDMGO L) through the bus to BDMGI L of the next priority device (the device electrically closest on the bus). This device accepts the grant only if it requested to be the bus master (by a BDMR L). If not, the device passes the grant (asserts BDMGO L) to the next device on the bus. This process continues until the requesting device acknowledged the grant. **CAUTION** **DMA device transfers must not interfere with the memory refresh cycle.**
AT2	BINIT L	Initialize—this signal is used for system reset. All devices on the bus are to return to a known, initial state; that is, registers are reset to zero, and logic is reset to state 0. Exceptions should be completely documented in programming and engineering specifications for the device.
AU2 AV2	BDAL0 L BDAL1 L	Data/address lines—these two lines are part of the 16-line data/address bus over which address and data information are communicated. Address information is first placed on the bus by the bus master device. The same device then either receives input data from, or outputs data to, the addressed slave device or memory over the same bus lines.
BA2	+5	+5 V power—normal +5 Vdc system power.
BB2	-12	-12 V power (voltage not supplied)—-12 Vdc power for (optional) devices requiring this voltage.
BC2	GND	Ground—system signal ground and dc return.
BD2	+12	+12 V power—+12 V system power.

Table C–7 (Cont.) Bus Pin Identifiers

Bus Pin	Mnemonic(s)	Description
		Data/address lines— these 14 lines are part of the 16-line data/address bus.
BE2	BDAL2 L	
BF2	BDAL3 L	
BH2	BDAL4 L	
BJ2	BDAL5 L	
BK2	BDAL6 L	
BL2	BDAL7 L	
BM2	BDAL8 L	
BN2	BDAL9 L	
BP2	BDAL10 L	
BR2	BDAL11 L	
BS2	BDAL12 L	
BT2	BDAL13 L	
BU2	BDAL14 L	
BV2	BDAL15 L	

D
Acronyms

This appendix lists and defines the acronyms that are most frequently used in this manual.

ACRONYM	DEFINITION
ACV	Access control violation
AIE	Alarm interrupt enable
ANSI	American National Standards Institute
AP	Argument pointer
ASTLVL	Asynchronous system trap level
BBU	Battery back-up unit
BCD	Binary coded decimal
BDR	Boot and diagnostic register
BM	Byte mask
BRS	Baud rate select signals
CADR	Cache disable register
CMCTL	CVAX memory controller chip
CPMBX	Console program mailbox
CQBIC	CVAX Q22-bus interface chip
CRC	Cyclic redundancy check
CSR	Control and status register
CSTD	Console storage transmit data
CSTS	Console storage transmit status
DEAR	DMA Error address register
DIP	Dual in-line package
DM	Data mode
DMA	Direct memory access
DSE	Daylight saving enable
DSSI	Digital small storage interconnect

ACRONYM	DEFINITION
EDITPC	EDIT packed to character string
EIA	Electronic Industries Association
EPROM	Erasable programmable read-only memory
ERR	Error signal
ESP	Executive stack pointer
FP	Frame pointer
FPA	Floating-point accelerator
FPU	Floating point unit
GPR	General purpose register
ICCS	Interval clock control and status register
ICR	Interval count register
IORESET	I/O bus reset register
IPCR	Interprocessor communication register
IPL	Interrupt priority level
IPR	Internal processor register
ISP	Interrupt stack pointer
KSP	Kernel stack pointer
LSI	Large scale integration
MAPEN	Memory management mapping enable register
MBRK	Microprogram break register
MBZ	Must be zero
MCESR	Machine check error summary register
MCS	Multinational character set
MFPR	Move from process register
MMU	Memory management unit
MOP	Maintenance operation protocol
MOS	Metal oxide semiconductor
MSER	Memory system error register
MSI	Mass Storage Interface
MTPR	Move to process register
NI	Network interface
NICR	Next interval count register
NPA	Network physical address
NXM	Nonexistent memory
P0BR	P0 base register
P1BR	P1 base register
PC	Program counter
PCB	Process control block

ACRONYM	DEFINITION
PCBB	Process control block base
PIE	Periodic interrupt enable
P0LR	P0 length register
P1LR	P1 length register
PMR	Performance monitor enable register
P0PT	P0 page table
P1PT	P1 page table
PROM	Programmable read only memory
PSL	Processor status longword
PSW	Processor status word
PTE	Page table entry
QBEAR	Q22-bus error address register
RAM	Random-access memory
RBD	Receive Buffer Descriptor
RPB	Restart parameter block
RXCS	Console receiver control/status register
RXDB	Console receiver data buffer
SAVPC	Console saved PC register
SAVPSL	Console saved PSL register
SBR	System base register
SCA	System communications architecture
SCB	System control block
SCBB	System control block base
SID	System identification register
SIE	System identification extension
SIRR	Software interrupt request register
SISR	Software interrupt summary register
SLR	System length register
SLU	Serial line unit
SP	Stack pointer
SPT	System page table
SQWE	Square-wave enable
SSC	System support chip
SSP	Supervisor stack pointer
TBCHK	Translation buffer check register
TBD	Transmit Buffer Descriptor
TBDATA	Translation buffer data
TBDR	Translation buffer disable register

ACRONYM	DEFINITION
TBIA	Translation buffer invalidate all
TBIS	Translation buffer invalidate single
TNV	Translation not valid
TODR	Time of year register
TOY	Time-of-year
TXCS	Console transmit control/status register
TXDB	Console transmit data buffer
UIE	Update interrupt enable
UIP	Update in progress bit
USP	User stack pointer
VLSI	Very large scale integration
VPN	Virtual page number
VRT	Valid RAM and time bit
VMB	Virtual memory bootstrap
XFC	Extended Function Call
ZIP	Zig-zag in-line package

Index

A

Abort, 31
Accessing the Q22-bus map registers, 89
Adding to a buffer list, 148

B

Backplane wiring, 297
Battery backed-up RAM, 84
Baud rate, 72
BDCOK H, 293
Block mode DMA, 277
Boot and diagnostic facility, 79
Boot and diagnostic register, 79
Boot block format, 233
Boot devices
  names, 228
  supported, 228
Boot flags, 194,229
Bootstrap
  conditions, 226
  disk and tape, 232
  initialization, 227
  MOP listening, 235
  network, 234
  PROM, 233
  sample output, 232
Bootstrap
  device names, 194
BPOK H, 293
Break response, 72
Buffer management, 123
Bus cycle protocol, 265
Bus drivers, 295
Bus interconnecting wiring, 297
Bus receivers, 296
Bus termination, 296

C

Cache, 48
Cacheable references, 47
Cache address translation, 49
Cache behavior on writes, 51
Cache data block allocation, 50
Cache disable register, 51
Cache error detection, 55
Cache memory, 5,47
Cache organization, 48
CDAL bus to Q22-bus address translation, 91
Central processing unit, 3
Central processor, 18
Clock functions, 8
Collision detect routine, 139
Compatible system enclosures, 17
Configuring the KA640, 11
Configuring the Q22-bus map, 94
Console commands, 183
Console control characters, 181
Console emulation, 181
Console error messages, 248
Console I/O mode, 183
Console interrupt specifications, 73
Console receiver control/status register, 68
Console receiver data buffer, 69
Console registers, 68
Console serial line, 68
Console transmitter control/status register, 70
Console transmitter data buffer, 72
Control functions, 292
CPU references, 45

D

Data-stream read references, 46
Data transfer bus cycles, 265
Data types, 25
DATBI bus cycle, 282
DATBO bus cycle, 283
Detailed local address space map, 254
Determination of the console device, 181
Device addressing, 266
Device priority, 286
Diagnostic and test registers, 163
Diagnostic executive, 177
Diagnostic LED register, 81
Direct memory access, 276
DMA error address register, 99
DMA guidelines, 285
DMA protocol, 276
DMA system error register, 96
DSSI bus, 143

E

Electrical specifications, 251
Entry/dispatch code, 172
Environmental specifications, 252
Error displays, 175
Error handling, 100
Error message, 175,247
Ethernet, 102
Exceptions, 30
Exceptions and interrupts, 28
External halts, 180
External IPRs, 257

F

Fault, 31
Firmware, 8
Floating point accelerator, 4, 46
Floating point accelerator data types, 47
Floating point accelerator instructions, 47
Floating point errors, 34

G

General local address space map, 253
General purpose registers, 19
Global Q22-bus address space map, 258

H

H3602-SA CPU cover panel, 15
Halt, 180,292
Halt messages, 247
Hardware detected errors, 41
Hardware halt procedure, 42
Hardware reset, 84

I

I/O bus initialization, 85
Information saved on a machine check, 33
Initialization, 292
Initialization routine, 136
Initiator operation, 146
Instruction set, 25
Instruction-stream read references, 45
Internal processor registers, 21
Interprocessor communication register, 92
Interrupt errors, 35
Interrupt protocol, 286
Interrupts, 28, 285
Interval timer, 74
IntraBackplane bus wiring, 298

K

KA640 connectors, 11
KA640 firmware, 171
KA640 initialization, 84
KA640 resident firmware operation, 83

L

LANCE chip, 104
LANCE operation, 135

LANCE programming notes, 140
List pointer registers, 162
Load definition, 294
Look-for-work routine, 137

M

Main memory addressing, 59
Main memory behavior on writes, 60
Main memory control and diagnostic status register, 64
Main memory error detection and correction, 66
Main memory error status register, 60
Main memory organization, 59
Main memory system, 56
Mass storage interface, 142
Memory controller, 5
Memory management, 26
Memory management control registers, 27
Memory management errors, 34
Memory refresh, 292
Memory system error register, 54
Microcode errors, 35
MicroVAX system support functions, 7
Module contact finger identification, 301
MOP functions, 235
MS650 memory modules, 5
MSI clock control register, 170
MSI command block, 148
MSI command block word 0, 149
MSI command block word 1, 149
MSI command block word 2, 151
MSI command block words 3-5, 152
MSI control/status register, 152
MSI diagnostic control register, 164
MSI diagnostic register 0, 165
MSI diagnostic register 1, 166
MSI diagnostic register 2, 169
MSI DSSI connection register, 156
MSI DSSI control register, 153
MSI DSSI timeout register, 160
MSI ID register, 159
MSI initiator list pointer register, 163
MSI internal state registers, 170
MSI link word 0, 146
MSI link word 1, 146
MSI registers, 152
MSI target list pointer register, 162

N

Network interface, 101
Network interface control and status register, 107
Network interface control and status register 1, 112
Network interface control and status register 2, 112
Network interface control and status register 3, 113
Network interface initialization block word 0, 115
Network interface initialization block words 10,11, 121
Network interface initialization block words 1-3, 118
Network interface initialization block words 4-7, 118
Network interface receive descriptor ring, 124
Network interface register address port, 105
Network interface register data port, 106
Network interface transmit descriptor ring, 129
NI initialization block, 114
NI initialization block words 8,9, 120
NISA ROM, 103
Nonoperating conditions greater than 60 days, 252
Nonoperating conditions less than 60 days, 252

O

120-Ohm Q22-bus, 295
Operating conditions, 252

P

Physical specifications, 251
Power status, 292
Power supply loading, 300
Power-up/power-down protocol, 293
Power-up initialization, 84
Power-up processing, 173
Processor initialization, 85
Processor state, 19
Processor status longword, 20
Programmable timers, 75

Q

Q22-bus 4-Level interrupt configurations, 290
Q22-bus electrical characteristics, 294
Q22-bus error address register, 98
Q22-bus interface, 85
Q22-bus interface, 7
Q22-bus interrupt handling, 93
Q22-bus map base address register, 94
Q22-bus map cache, 90
Q22-bus map registers, 88
Q22-bus signal assignments, 262
Q22-bus to main memory address translation, 86

R

Read errors, 36
Receive buffer descriptor n word 0, 125
Receive buffer descriptor n word 1, 125
Receive buffer descriptor n word 2, 127
Receive buffer descriptor n word 3, 128
Receive buffer descriptors, 124
Receive buffers, 128
Receive DMA routine, 138
Receive poll routine, 137
Receive routine, 137
ROM address space, 82
ROM-based diagnostics, 177
ROM memory, 82
ROM socket, 82

S

Scripts, 177
Signal level specifications, 294
Switch routine, 136
System configuration register, 95
System configurations, 298
System control block, 39
System identification, 44

T

Target operation, 145
Time of year clock, 74
Time of year clock and timers, 73
Timer control registers, 75
Timer interrupt vector registers, 78
Timer interval registers, 77
Timer next interval registers, 78
Translation buffer, 26
Transmit buffer descriptor n word 0, 131
Transmit buffer descriptor n word 1, 131
Transmit buffer descriptor n word 2, 133
Transmit buffer descriptor n word 3, 133
Transmit buffer descriptors, 130
Transmit buffers, 134
Transmit data segment links, 146
Transmit DMA routine, 139
Transmit poll routine, 138
Transmit routine, 139
Trap, 30

V

VMB

VMB (cont'd.)
boot flags, 194
description, 230
procedure, 230
VMB error messages, 249

W

Write errors, 36
Write references, 46