4000-200

KA660 CPU Module Technical Manual

Order Number EK-KA660-TM-001

Digital Equipment Corporation

March 1991

This document was prepared and published by Educational Services Development and Publishing, Digital Equipment Corporation.

Contents

4 KA660 Cache Memory

5 KA660 Main Memory System

6 KA660 Console Serial Line

7 KA660 Clock and Timer Registers

8 KA660 Boot and Diagnostic Facility

9 KA660 Q22-bus Interface

10 KA660 Network Interface

11 KA660 Mass Storage Interface

12 KA660 Firmware

A Q22-bus Specification

B Specifications

C Address Assignments

D VAX Instruction Set

E Machine State on Power-Up

F Maintenance Operations Protocol (MOP) Support

G ROM Partitioning

H Battery Backed-up RAM Partitioning

I Data Structures

J Error Messages

Glossary

Index

Examples

Figures

Tables

About This Manual

The *KA660-AA CPU Module Technical Manual* contains the functional, physical, and environmental characteristics of both variations of the KA660 CPU module, and includes information on the MS650 memory expansion modules. The KA660-AA is for a multiuser environment. The KA660-BA is for a single user environment and does not support multiuser VMS or ULTRIX operating system licenses.

This manual 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 twelve chapters, ten appendices, and a glossary:

Chapter 1, **Overview,** describes the KA660 Subsystem including the KA660-AA CPU module, the MS650 memory module, and the H3602 console module.

Chapter 2, **Installation and Configuration,** describes procedures for installing and configuring the CPU, the memory, and the console modules in the Q22-bus backplanes and system enclosures.

Chapter 3, **Central Processor**, provides information about the Systom-on-a-Chip central processor and basic VAX computer architecture. This chapter lists all the internal processor registers used in the KA660 processor design. Some information on error handling is given also.

Chapter 4, **KA660 Cache Memory**, describes the organization of the KA660 cache. It shows the format of cache entries and provides information on cache address translation.

Chapter 5, **KA660 Main Memory System**, provides information about the main memory organization, including the registers associated with main memory error checking and status and also includes information on cycle access times.

Chapter 6, **KA660 Console Serial Line**, describes the serial line interface and its associated registers.

Chapter 7, **KA660 Clock and Timer Registers**, provides information about the VAX standard Time-of-Year clock and timers.

Chapter 8, **KA660 Boot and Diagnostic Facility**, describes the KA660 initialization process and provides information about boot and diagnostic registers and the EPROM memory resident firmware.

Chapter 9, **KA660 Q22-bus Interface**, describes the Q22-bus interface which is implemented with the CQBIC. Information on Q22-bus address translation is provided along with descriptions of all Q22-bus interface registers. Some information on CQBIC error handling is also included.

Chapter 10, **KA660 Network Interface**, provides information on the second generation Ethernet Controller chip and the logic that supports the Ethernet network interface. An overview of Ethernet principles with descriptions of packet format and programming instructions is provided. The transmission and reception processes are described also.

Chapter 11, **KA660 Mass Storage Interface**, describes how the Single Host Adapter Chip provides a Digital storage system interconnect mass storage interface for the KA660. An overview is provided as well as descriptions of all the registers associated with the DSSI interface.

Chapter 12, **KA660 Firmware**, describes the functional firmware located in the EPROMS. The services provided by the firmware are described.

Appendix A, **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 3500, MicroVAX 3600, MicroPDP–11, use the Q22-bus.

Appendix B, **Specifications,** describe the physical, electrical, and environmental characteristics of the KA660-AA CPU module.

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

Appendix D, **VAX Instruction Set,** is a list of the VAX computer instructions.

Appendix E, **Machine State on Power-Up,** describes the state of the KA660 after a power-up halt.

Appendix F, **Maintenance Operations Protocol (MOP) Support,** describes the Maintenance Operation Protocol (MOP) support features in the KA660 firmware.

Appendix G, **ROM Partitioning,** describes the ROM partitioning and subroutine entry points that are public and guaranteed to be compatible with future versions of the KA660 firmware.

Appendix H, **Battery Backed-up RAM Partitioning,** describes how the KA660 firmware partitions the 1 Kbyte of battery-backed-up RAM.

Appendix I, **Data Structures,** describes the global data structures that are used by the KA660 firmware.

Appendix J, **Error Messages** , lists the firmware error messages detected by the KA660.

The **Glossary** provides a list of the acronyms and terms used in this manual.

Conventions

The following conventions are used in this manual:

Table 1 Conventions

Convention	Meaning
Note	Contains general information.
Caution	Contains information to prevent damage to equipment.
<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 a general purpose register R0.
[x:y]	Represents a range of bits, from y through x.
Return	A label enclosed in a box represents a key (usually a control or a special character key) on the keyboard (in this case, the return key).
CONFIGURE	Words in all capital letters are system commands you must enter to initiate a desired function.
n	A small n in bold type indicates a variable.
{}	Represents a console command element.
[ ]	Represents an optional console command element.
...	Represents a list of command elements.
PN	Part number.

Related Documents

The following documents are related to the KA660 CPU:

- *Microcomputer Interfaces Handbook* (EB-20175-20)
- *Microcomputers and Memories Handbook* (EB-18451-20)
- *VAX Architecture Handbook* (EB-19580-20)
- *VAX–11 Architecture Reference Manual* (EK-VAXAR-RM)

You can order these documents from Digital Equipment Corporation at the following address:

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

Attention: Documentation Products

1 Overview

This chapter provides a brief description of the KA660 CPU/Memory Subsystem.

1.1 The KA660 Subsystem

The KA660 processor module combines with the MS650 memory module and the H3602 console module to form the CPU/Memory subsystem for the VAX 4000-200 product. The subsystem is available in two enclosures: The BA430 or the BA215. It uses the Digital storage system interconnect bus to communicate with mass storage devices and the Q22-bus to communicate with I/O devices. A single KA660 CPU module can support a maximum of four MS650 memory modules.

Figure 1–1 is a block diagram of the CPU/Memory subsystem's major functions.

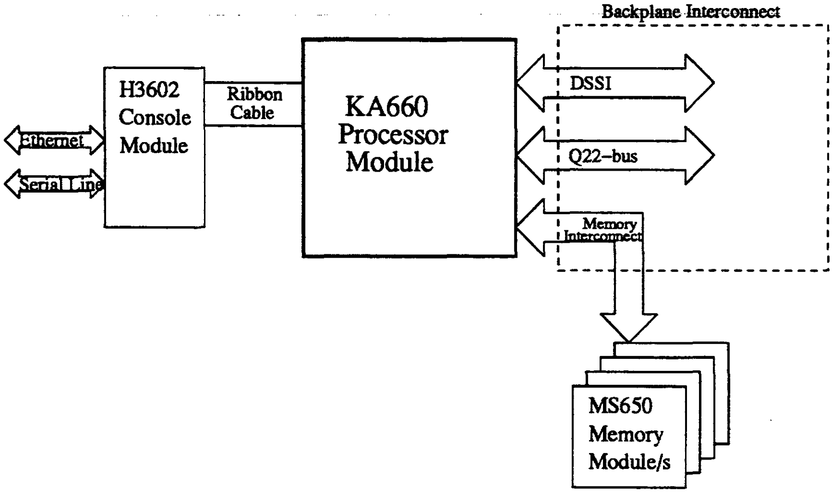


Figure 1–1 KA660 Module in a System

The KA660 and the MS650 designs are implemented in standard quad-height sized modules. Both modules mount in standard Q22-bus backplane slots which implement the Q22-bus in the AB rows and the CD interconnect in the CD rows.

The KA660 Processor Module communicates with the memory modules across a memory interconnect routed through a 50-pin ribbon cable and the CD interconnect on the backplane. The DSSI connects through a 50-pin ribbon cable located on top of the memory interconnect cable. The backplane connector also connects the subsystem with the Q22-bus. There are no jumpers or switches to configure on the processor module. The

KA660 connects to the H3602 console module and the Ethernet controller with a 40-pin ribbon cable. The console module contains configuration switches, Ethernet and DSSI connectors, fuses, and an LED display.

1.2 KA660 Processor Module

The KA660 processor can be configured *only* as an arbiter on the Q22-bus. An arbiter is the single entity responsible for controlling the Q22-bus. It must reside in the first backplane slot where it arbitrates bus mastership and fields bus interrupt requests and any on-board interrupt requests. This processor module is designed for use in high-speed, real-time applications and for multiuser, multitasking environments. There are two variants: The KA660-AA, which runs multiuser software; and the KA660-BA, which runs single-user software.

Figure 1–2 is a photograph of the KA660 Processor Module.

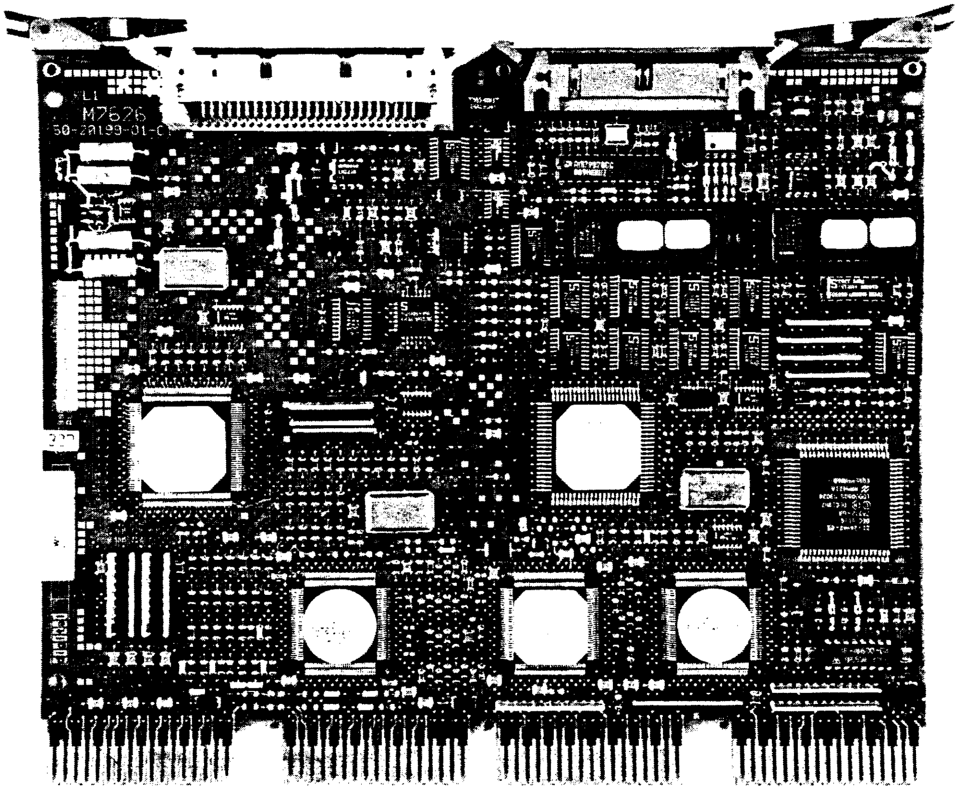


Figure 1–2 The KA660 Processor Module

The major hardware components of the KA660 CPU module are listed below. The chip identification numbers are shown in Figure 1–3:

• System-on-a-Chip (SOC) CPU	DC222
• A main memory controller (CMCTL)	DC557
• Q22-Bus interface (CQBIC)	DC527
• System support chip (SSC)	DC511
• Second generation Ethernet controller (SGEC)	DC541
• Single host adapter chip to interface DSSI (SHAC)	DC542

- Two firmware EPROMs
- A boot and diagnostic facility
- Console connection
- VAX compatible console port
- Backplane connection

Figure 1–3 shows the positions of the major chips on the KA660.

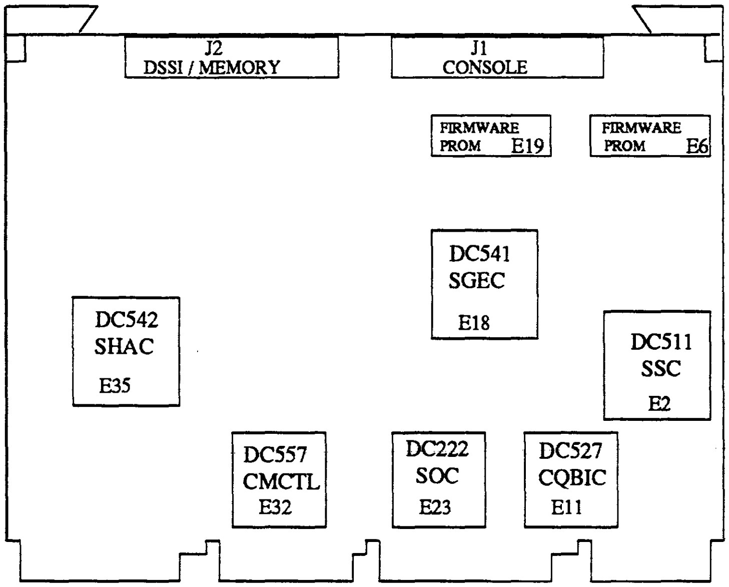


Figure 1–3 KA660 CPU Module Component Side

The KA660 Processor Module is divided into several major functional subsystems as listed next and shown in Figure 1–4.

- The central processing subsystem
- The memory control subsystem
- The Q22-bus subsystem
- The DSSI subsystem
- The system support subsystem
- The Ethernet subsystem

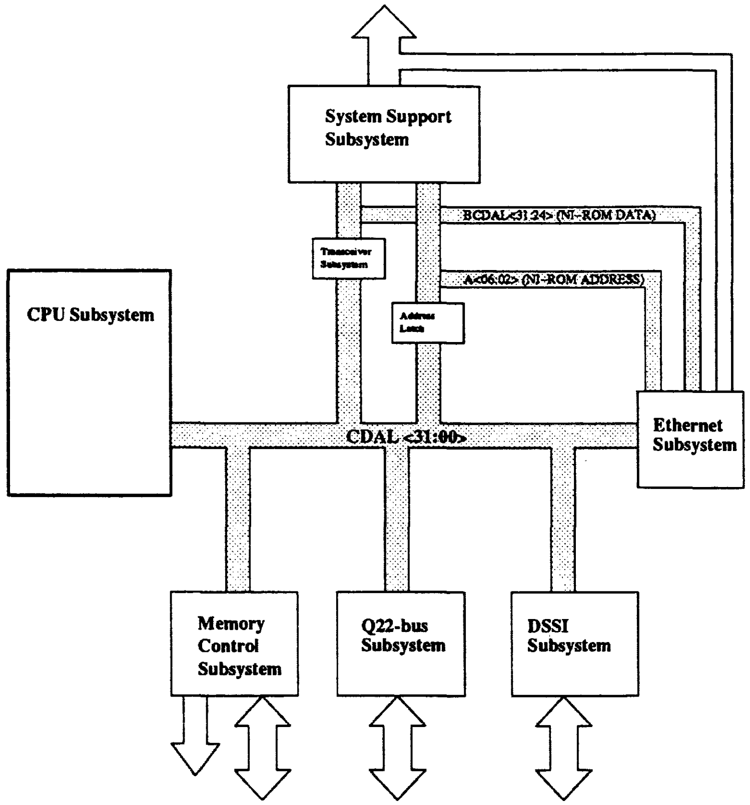


Figure 1–4 KA660 Processor Module Major Functional Blocks

The rest of Section 1.1 describes the subsystems.

The Central Processing Subsystem

The central processing subsystem features the system-on-a-Chip (SOC) CPU and its accompanying support logic. The SOC chip is a unique design that contains several system components on a single substrate contained in a 132-pin surface mount, CERQUAD chip package. The SOC contains the central processing unit (CPU), the floating point accelerator unit (FPA), and 8 Kbytes of cache to optimize system performance.

The central processor in the SOC supports the following MicroVAX computer instruction set with the following string instructions:

- CMPC3 (compare character - 3 operand)
- CMPC5 (compare character - 5 operand)
- LOCC (locate character)
- SKPC (skip character)
- SPANC (span character)
- SCANC (scan character)

The following subset of the VAX data types are provided:

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

Support for the remaining VAX data types can be provided through macrocode emulation. The processor also supports full VAX memory management with demand paging and a 4 Gbyte virtual address space.

The floating point accelerator unit (FPU) in the SOC executes the VAX f_, d_, and g_ floating point instructions. It executes 61 floating point instructions and 2 longword-length integer multiply instructions in the VAX base instruction group. It supports the MicroVAX chip subset of the VAX floating point instruction set and data types.

The Memory Control Subsystem

The memory control subsystem contains the memory controller chip (CMCTL) and its associated termination logic. This subsystem provides an interface between the Data and Address Lines (CDAL) lines from the KA660 CDAL bus and the data and address lines on the MS650 MDATA bus.

The CMCTL chip contains approximately 25,000 transistors in a 132-pin CERQUAD surface mount package. It supports up to 64 Kbytes of ECC memory, with a 450 nanosecond cycle time for longword transfers and a 720 nanosecond cycle time for quadword transfers,

The memory resides on one to four MS650 memory modules, depending on the system configuration. The MS650 communicates with the KA660 through the memory interconnect, which uses the CD interconnect and a 50-pin ribbon cable.

The Q22-bus Subsystem

The Q22-bus subsystem contains the Q22-bus interface and asociated termination logic. This subsystem provides an interface between the Q22-bus and the central processor's CDAL bus. The interface is implemented with the CQBIC. The CQBIC contains approximatley 40,870 transistors in a 132-pin CERQUAD surface mount package. The CQBIC is a 32-bit to 16 bit adapter which provides physical memory address translation for direct memory access (DMA) devices on the Q22-bus. It supports as many as 16-word, block mode transfers between a Q22-bus DMA device and main memory, and as many as 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 8192-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)

The DSSI Subsystem

This subsystem provides an interface between the DSSI bus and the KA660 CDAL bus. It contains the single host adapter chip (SHAC), the DSSI jumpers, 16 MHz Oscillator, and associated termination and control logic. The SHAC is in a 164-pin CERQUAD package. It facilitates scatter and gather mapping along with internal FIFO buffering.

The DSSI interface allows the DSSI bus on the KA660 to transmit packets of data to, and receive packets from, as many as seven other DSSI devices. These devices include the RF-series integrated storage elements (ISEs), a KFQSA module, a second KA660 module, or a KA640 module.

The DSSI bus improves system performance with its higher transfer rate than the Q22-bus and relieves the Q22-bus of disk traffic. The DSSI bus has eight data lines, one parity line, and eight control lines. Controllers are built into the ISEs, enabling many functions to be handled without host or adapter intervention.

The System Support Subsystem

The system support subsystem handles the basic functions required to support the console in a system environment. This subsystem contains the system support chip (SSC), the firmware ROMs, the boot and diagnostic register, and the station address ROM.

The SSC chip is implemented in an 84-pin CERQUAD surface mount package. It provides console and boot code support functions, operating system support functions, timers, and the following features:

- Word-wide ROM unpacking
- Halt-arbitration logic
- Interval timer with 10ms interrupts
- IORESET register
- Two programmable timers
- 1 Kbyte battery backed-up RAM
- Console serial line
- VAX-standard time-of-year clock with battery backup
- Programmable CDAL bus timeout
- Register controlling the diagnostic LEDs

Resident firmware Read Only Memory is located on two chips, each 128 KByte by 8-bit EPROMS. The firmware gains control when the CPU halts. This code contains programs that provide the following services:

- Board initialization
- Power-up self-testing of the KA660 and MS650 modules
- Emulation of a subset of the VAX standard console (auto or manual bootstrap, auto or manual restart, and a simple command language for examining or altering the state of the processor)
- Booting from supported Q22-bus devices
- Multilingual translation of key system messages

  The boot and diagnostic register (BDR) allows the firmware and the operating system to read KA660 configuration bits. The station address ROM contains the network address of the system. It is implemented in a 32-byte by 8-bit ROM (6331).

The Ethernet Subsystem

The Ethernet subsystem handles communications between the CPU module and other nodes on the Ethernet. It is implemented with the second generation Ethernet controller chip (SGEC, DC541) on-board network interface. Used in connection with the H3602 console module, the SGEC allows the KA660 to connect to either a thinwire or standard Ethernet. It supports the Ethernet data link layer and the CP bus parity protection. The SGEC chip is in an 84-pin package. The chip facilitates scatter and gather mapping along with dual internal FIFO buffering.

1.3 MS650 Memory Module

The MS650 memory module for the KA660 CPU is available in two variations. The MS650-BA contains 16 Mbytes of memory and the MS650-BB contains 8 Mbytes. The memory is arranged in 39-bit wide arrays implemented with 1 Mbyte, 120 nanosecond, dynamic RAMs in surface mount packages. Of the 39 bits, 32 bits are data and 7 bits are error checking and correction (ECC) bits. The MS650 modules are single, quad-height, Q22-bus modules as shown in Figure 1–5 and Figure 1–6.

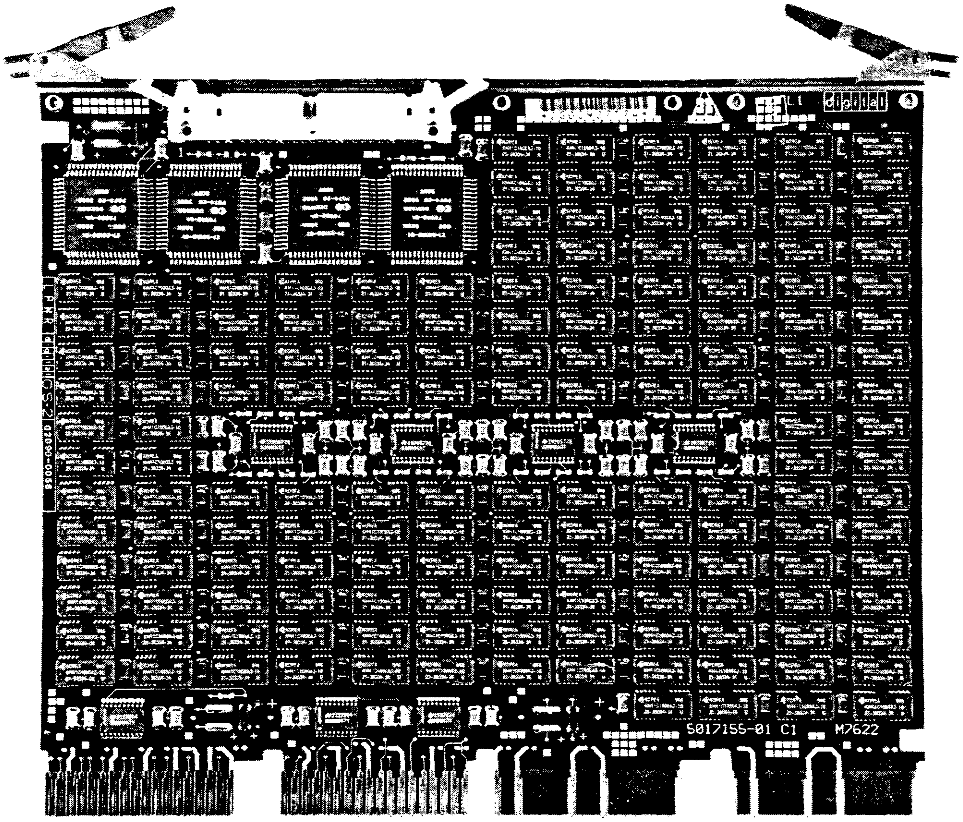


Figure 1–5 MS650 Memory Module (16 MB)

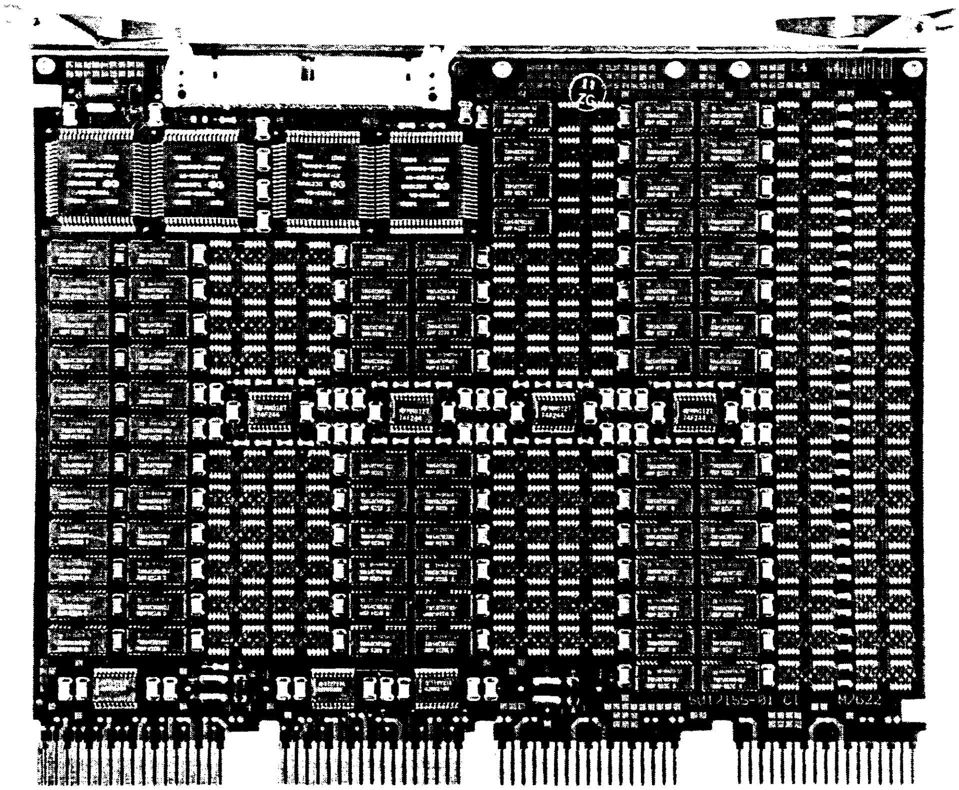


Figure 1–6 MS650 Memory Module (8 MB)

1.4 H3602 Console Module

The H3602 console module (Figure 1–7) is a unique I/O panel that is used in BA213 and BA215 enclosures. A one-piece ribbon cable on the H3602 plugs into J1 (system support connector) on the KA660. The H3602 fits over backplane slots one and two, covering both the KA660 processor module and the first of four possible MS650 memory modules. The H3602 allows the KA660 CPU module to interface to a serial line console device, a DSSI bus, and to the Ethernet. Adhesive tags are included for the user to name the modules in the respective slots.

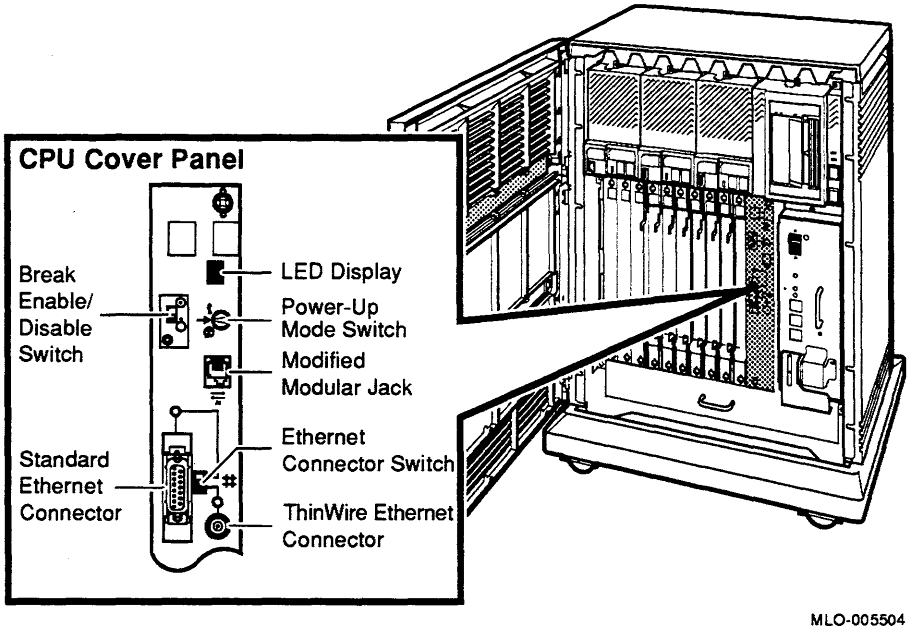

CPU Cover Panel
Break Enable/ Disable Switch
LED Display
Power-Up Mode Switch
Modified Modular Jack
Standard Ethernet Connector
Ethernet Connector Switch
ThinWire Ethernet Connector
MLO-005504

Figure 1–7 H3602 Console Module

The exterior H3602 console panel has the following features:

- Modified modular jack (MMU) SLU connector
- Power-up mode switch
- Hexadecimal LED display
- Break enable switch
- Standard/ThinWire Ethernet connectors
- Standard/ThinWire Ethernet selector switch
- Indicator LEDs

The interior console panel has the following features:

- Baud rate rotary switch
- Battery backup unit (BBU) for TOY clock
- 40-pin cable connector
- List of baud rate switch settings

2
Installation and Configuration

2.1 Introduction

This chapter describes how to install the KA660 in a system. It incudes the following topics:

- Installing the KA660 and MS650 modules
- Configuring the KA660
- The KA660 connectors

2.2 Installing the KA660 and MS650 Memory Modules

The KA660 and MS650 (BB or BA models only) 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 KA660 CPU module and the MS650 memory modules must be installed in the five right-most backplane slots. The KA660 CPU module must be installed in slot 1 of the Q22/CD backplane. MS650 memory modules must be installed in slots immediately adjacent to the CPU module. Figure 2–1 shows the positions of the module slots in the backplane.

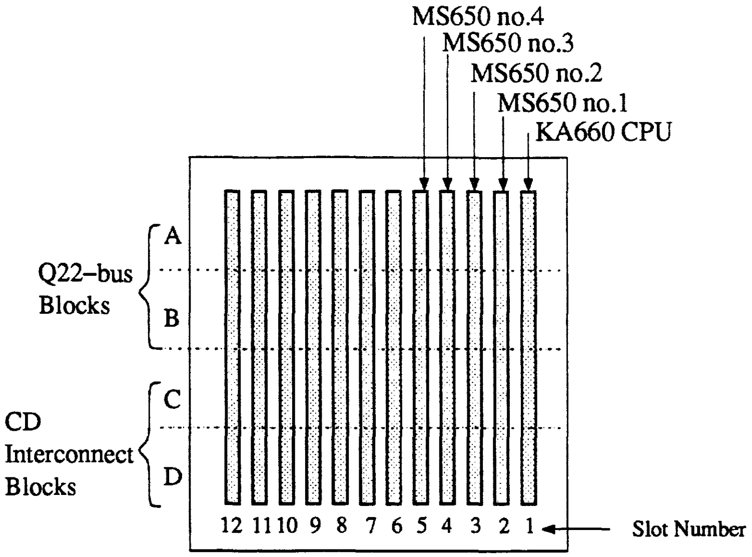


Figure 2–1 Backplane Slots

As many as four MS650 memory modules can be installed, occupying slots 2, 3, 4, and 5 respectively. A 50-pin ribbon cable is used to connect the KA660 processor module and the MS650 memory modules as shown in Figure 2–2.

The KA660 module is installed in backplane slot 1 and the memory modules are installed in slots 2 through 5. Use the following procedure to install the KA660 and MS650 modules:

1. Install the KA660 CPU in slot 1 of the Q22-bus/CD backplane.
2. Install the MS650 memory module in slots 2, immediately adjacent to the KA660 CPU. When installing additional memory use slots 3 through 5. Do not leave a gap between memory modules.
3. Install a 50-pin ribbon cable between the KA660 CPU and the MS650 memory modules (see Figure 2–2.)

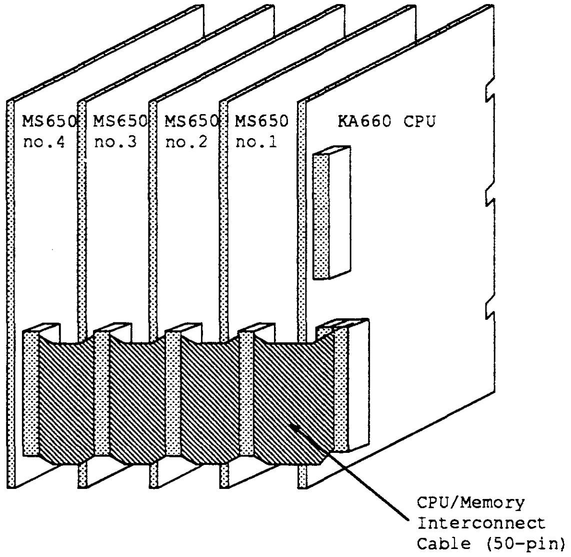


Figure 2–2 Processor and Memory Module Connection

2.3 Module Configuration and Naming

Each module in a system must use a unique device address and interrupt vector. The device address is also known as the control and status register (CSR) address. Most modules have switches or jumpers for setting the CSR address and interrupt vector values. The value of a floating address depends on what other modules are housed in the system.

Set CSR addresses and interrupt vectors for a module as follows:

1. Determine the correct values for the module with the CONFIGURE command at the console I/O prompt (>>>). The CONFIG utility eliminates the need to boot the VMS operating system to determine CSRs and interrupt vectors. Enter the CONFIGURE command, then enter HELP for the list of supported devices:

```
>>> 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       KXXXX-DISK  TQK50   TQK70   TU81E   RV20
KXXXX-TAPE  KMV11       IEQ11   DHQ11   DHV11   CXA16
CXB16       CXY08       VCB02   QDSS    DRV11J  DRQ3B
VSV21       IBQ01       IDV11A  IDV11B  IDV11C  IDV11D
IAV11A      IAV11B      MIRA    ADQ32   DTC04   DESQA
IGQ11
```

The LPV11–SA has two sets of CSR address and interrupt vectors. To determine the correct values for an LPV11–SA, enter LPV11,2 at the DEVICE prompt for one LPV11–SA, or enter LPV11,4 for two LPV11–SA modules.

2. See the *KA660 CPU System Maintenance Manual* for switch and CSR and interrupt vector jumper settings for supported options.

2.4 Mass Storage Configuration

In a BA213 enclosure there is space for four mass storage devices, and three integrated storage elements (ISE) and one TK70 (or four ISEs). The ISEs are part of the Digital Storage System Interconnect (DSSI) bus.

The DSSI bus is part of the backplane. The ISEs are of the RF-series and plug into the backplane to become part of the bus. Each ISE must have its own unique DSSI node ID. The ISE receives its node ID from a plug on the operator control panel (OCP) on the front panel.

The VMS operating system creates DSSI disk device names according to the following scheme:

```
(nodename $ DIA unit number.)
```

For example,

```
(SUSAN$DIA3)
```

You can use the device name for booting, as follows:

```
>>> BOOT SUSAN$DIA3
```

You can access local programs in the RF-series ISE through the MicroVAX Diagnostic Monitor (MDM) or through the VMS operating system (version 5.0) and console I/O mode SET HOST/DUP command. This command creates a virtual terminal connection to the storage device and the designated local program using the diagnostic and utilities protocol (DUP) standard dialog. Section 2.4.3 describes the procedure for accessing DUP through the VMS operating system.

2.4.1 Changing the Node Name

Each ISE has a node name that is maintained in EPROM on board the controller module. This node name is determined in manufacturing from an algorithm based on the drive serial number. You can change the node name of the DSSI device to something more meaningful by following the procedure in Example 2–1. In the example, the node name for the ISE at DSSI node address 1 is changed from R3YBNE to DATADISK.

```
>>> sho dssi
DSSI Node 0 (MDC)
-DIA0 (RF71)

DSSI Node 1 (R3YBNE)        !The node name for this drive will be
-DIA1 (RF71)                !changed from R3YBNE to DATADISK.

DSSI Node 7 (*)
>>>
>>> set host/dup/dssi 1
Starting DUP server...
Copyright  1988  Digital Equipment Corporation
DRVEXR V1.0  D  5-NOV-1988 15:33:06
DRVTST V1.0  D  5-NOV-1988 15:33:06
HISTRY V1.0  D  5-NOV-1988 15:33:06
ERASE  V1.0  D  5-NOV-1988 15:33:06
PARAMS V1.0  D  5-NOV-1988 15:33:06
DIRECT V1.0  D  5-NOV-1988 15:33:06
End of directory
Task Name? params
Copyright  1988  Digital Equipment Corporation

PARAMS> sho nodename

Parameter     Current           Default        Type     Radix
--------- ----------------  ----------------  --------  -----
NODENAME          R3YBNE            RF71        String  Ascii  B

PARAMS> set nodename datadisk

PARAMS> write                    !This command writes the change
                                 !to EPROM.
Changes require controller initialization, ok? [Y/(N)] y

Stopping DUP server...
>>> sho dssi
DSSI Node 0 (MDC)
-DIA0 (RF71)

DSSI Node 1 (DATADISK)      !The node name has changed from
-DIA1 (RF71)                !R3YBNE to DATADISK.

DSSI Node 7 (*)
```

Example 2–1 Changing a DSSI Node Name

2.4.2 Changing the DSSI Unit Number

By default, the ISE drive assigns the disk's unit number to the same value as the DSSI node address for that drive.

Example 2–2 shows how to change the unit number of a DSSI device. This example changes the unit number for the RF71 drive at DSSI node address 2 from 1 to 50 (decimal). You must change two parameters: UNITNUM and FORCEUNI. Changing these parameters overrides the default, which assigns the unit number the same value as the node address.

```
>>> sho dssi
DSSI Node 0 (MDC)
-DIA0 (RF71)

DSSI Node 1 (R3QJNE)     !The unit number for this drive will be
-DIA1 (RF71)             !changed from 1 to 50 (DIA1 to DIA50).

DSSI Node 7 (*)
>>>
>>> set host/dup/dssi 1
Starting DUP server...
Copyright  1988  Digital Equipment Corporation
DRVEXR V1.0  D  5-NOV-1988 15:33:06
DRVTST V1.0  D  5-NOV-1988 15:33:06
HISTRY V1.0  D  5-NOV-1988 15:33:06
ERASE  V1.0  D  5-NOV-1988 15:33:06
PARAMS V1.0  D  5-NOV-1988 15:33:06
DIRECT V1.0  D  5-NOV-1988 15:33:06
End of directory

Task Name? params
Copyright 1988  Digital Equipment Corporation

PARAMS> sho unitnum

Parameter     Current           Default        Type     Radix
--------- ----------------  ----------------  --------  -----
UNITNUM               0                 0      Word     Dec    U

PARAMS> sho forceuni

Parameter     Current           Default        Type     Radix
--------- ----------------  ----------------  --------  -----
FORCEUNI              1                 1   Boolean     0/1    U

PARAMS> set unitnum 50

PARAMS> set forceuni 0

PARAMS> write       !This command writes the changes to EPROM.

PARAMS> ex
Exiting...

Task Name?

Stopping DUP server...
>>>
>>>sho dssi
DSSI Node 0 (MDC)
-DIA0 (RF71)

DSSI Node 1 (R3QJNE)          !The unit number has changed
-DIA50 (RF71)                 !and the node ID remains at 1.

DSSI Node 7 (*)
```

Example 2–2 Changing a DSSI Unit Number

2.4.3 Access to RF-series Firmware in VMS Through DUP

You can also access the RF-series ISE firmware utilities from the VMS operating system as well as through the console commands.

Access the ISE firmware through the VMS operating system to find or to view parameter settings, but not to change them. To change ISE parameter settings, enter the ISE firmware through the console I/O mode SET HOST/DUP command.

Load the FYDRIVER using the following commands in SYSGEN:

```
$ MCR SYSGEN
SYSGEN> LOAD FYDRIVER/NOADAPTER
SYSGEN> CONNECT FYA0/NOADAPTER
SYSGEN> EXIT
$
```

You can then access the ISE firmware utilities using the following VMS command:

```
$ SET HOST/DUP/SERVER=MSCP$DUP/TASK=PARAMS nodename
```

2.4.3.1 Allocation Class

When a KA660 system containing ISEs is configured in a cluster, either as a boot node or a satellite node, you must assign the allocation class in VMS SYSGEN and for the ISE matching non-zero values. To change the allocation class of the ISE, use the following commands:

```
>>> SET HOST/DUP/DSSI <DSSI node number> PARAMS
Starting DUP server..

PARAMS> SET ALLCLASS <allocation class value>

PARAMS> WRITE
Changes require controller initialization, ok? [Y/N] Y

Stopping DUP server..
>>>
```

2.5 DSSI Cabling, Device Identity, and Bus Termination

The ISEs in one particular BA430 enclosure are connected to the system backplane and communicate internally over the backplane. There are no internal DSSI cables. Externally, a 50-pin ribbon cable connects the DSSI bus to other devices, either hosts or expanders.

All DSSI devices on the same bus must have unique identifiers.

The ID plug provides an identity for the DSSI bus.

2.6 KA660 Connectors

The KA660 uses two connectors, J1 and J2. J1 (system support connector) is the connector for the 40-pin ribbon cable that goes to the console module. Users configure the KA660 through the H3602 console module. Figure 1–3 shows the location of the connectors on the KA660 module. J2 is a dual connector. The upper half contains 50 pins for the DSSI connection and the lower half contains 50 pins for the memory module connection.

3 Central Processor

This section provides an overview of the user-visible features of the SOC/C (CPU) chip and MicroVAX computer architecture.

The central processor of the KA660 supports the MicroVAX chip subset (plus six additional string instructions) of the VAX instruction set and data types plus full VAX memory management. It is implemented as part of the SOC/C chip.

3.1 Processor State

The processor state is the part of a process which is stored in process 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 non-privileged 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 which can be executed only while running in kernel mode.

3.2 General Purpose Registers (GPRs)

The KA660 implements 16 general purpose registers as implemented per the *VAX Architecture Reference Manual*. These registers are used for temporary storage, accumulators, and base and index registers for addressing. These registers are denoted R0 - R15. The bits of a register are numbered from right to left, <0> through <31>. Figure 3–1 shows the general purpose register format. Table 3–1 describes the registers.

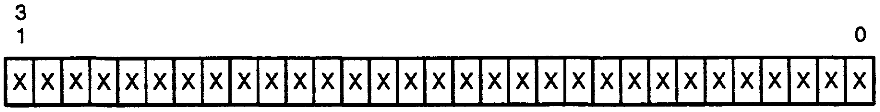

31
0
X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X

Figure 3–1 GPR Format

Table 3–1 lists certain registers that have been assigned special meaning by the VAX–11 computer architecture standard.

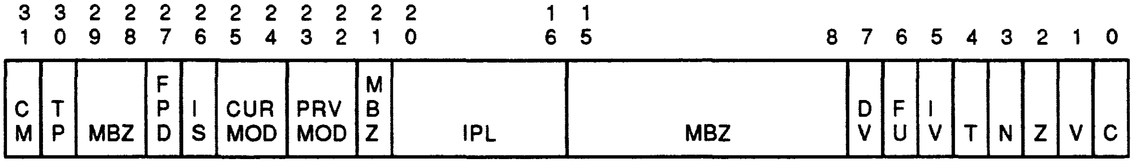
Table 3–1 Special GPRs

Register	Register Name	Mnemonic	Description
R15	Program counter	PC	The PC contains the address of the next instruction byte of the program.
R14	Stack pointer	SP	The SP contains the address of the top of the processor defined stack.
R13	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	Argument pointer	AP	The VAX–11 procedure call convention uses a data structure termed 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.

3.3 Processor Status Longword (PSL)

The KA660 processor status longword (PSL) is implemented per the *VAX Architecture Reference Manual.* 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 PSL format. Table 3–2 lists the bits and definitions.



ESB90P0002

Figure 3–2 PSL Format

Table 3–2 Processor Status Longword Format

PSL Data Bit	Name	Definition
<31>	CM	Compatibility mode. This bit always reads as zero. Loading a 1 into this bit is an NOP.
<30>	TP	Trace pending.
<29:28>	MBZ	Must be written as zero.
<27>	FPD	First part done.

Table 3–2 (Cont.) Processor Status Longword Format

PSL Data Bit	Name	Definition
<26>	IS	Interrupt stack.
<25:24>	CUR	Current mode.
<23:22>	PRV	Previous mode.
<21>	MBZ	Must be written as zero.
<20:16>	IPL	Interrupt priority level.
<15:8>	MBZ	Must be written as zero.
<7>	DV	Decimal overflow trap enable. This read/write bit has no effect on KA660 hardware; it can be used by macrocode which emulates VAX decimal instructions.
<6>	FU	Floating underflow fault enable.
<5>	IV	Integer overflow trap enable.
<4>	T	Trace trap enable.
<3>	N	Negative condition code.
<2>	Z	Zero condition code.
<1>	V	Overflow condition code.
<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.

3.4 Internal Processor Registers (IPRs)

The privileged internal processor register (IPR) space provides access to many types of CPU control and status registers such as the memory management base registers, parts of the PSL, and the multiple stack pointers. These registers are explicitly accessible only by the move-to-processor register (MTPR) and the move-from-processor register (MFPR) instructions which require kernel privileges. The addresses of the KA660 internal processor registers are given in Table C–2. Internal processor registers are longword size, as shown in Figure 3–3.

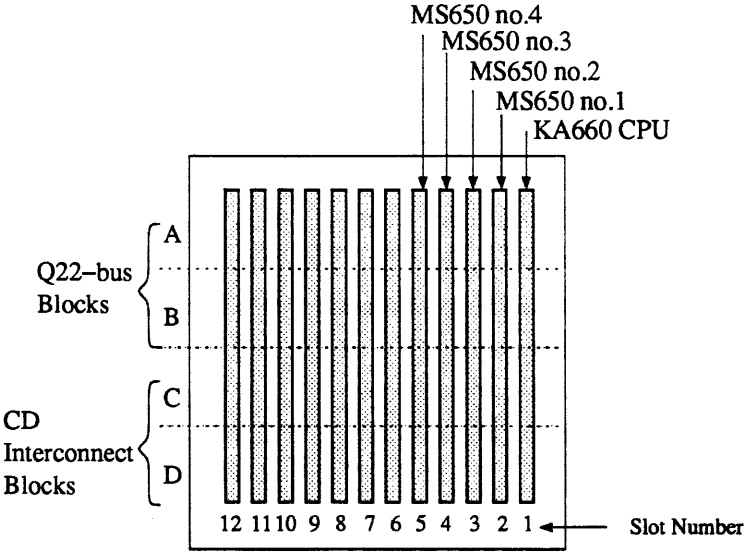


Figure 3–3 Internal Processor Register (IPR) Format

IPR Categories

Each IPR falls into one of the following categories:

(1) VAX standard IPRs implemented by KA660 in the *SOC/C* chip.
(2) VAX standard IPRs implemented by KA660 in the *SSC* chip.
(3) Unique KA660 IPRs implemented by all designs that use the *SOC/C* chip.
(4) Unique KA660 IPRs implemented by all designs that use the *SSC* chip.
(5) Not implemented, timed out by the CDAL Bus Timer (in the SSC chip) after 4 microseconds. 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.

Table 3–3 explains each IPR.

Table 3–3 KA660 Internal Processor Registers

IPR Number						Implemented		
Decimal	Hex	Register Name	Mnemonic	Type	Scope	Where	Init?	Category
0	0	Kernel stack pointer	KSP	RW	PROC	SOC/C		1

Table Heading Key

IPR Number *Decimal:* The decimal number of the processor register; *Hex:* The hex number of the processor register.
Type *R:* read-only register; *W:* write-only register; *RW:* read/write register.
Scope *CPU:* CPU wide register; *PROC:* per the processor register.
Implemented Where? *SOC/C:* Implemented in the SOC CPU Chip; *SSC:* Implemented in the MicroVAX system support chip
Init?: Is this register initialized on Module RESET? (Power-up, Negation of DCOK) *yes* or *no*.
Category: The processor register category as previously defined in Table 3–3.

Table 3–3 (Cont.) KA660 Internal Processor Registers

IPR Number						Implemented		
Decimal	**Hex**	**Register Name**	**Mnemonic**	**Type**	**Scope**	**Where**	**Init?**	**Category**
1	1	Executive stack pointer	ESP	RW	PROC	SOC/C		1
2	2	Supervisor stack pointer	SSP	RW	PROC	SOC/C		1
3	3	User stack pointer	USP	RW	PROC	SOC/C		1
4	4	Interrupt stack pointer	ISP	RW	CPU	SOC/C		1
5-7	5-7	Reserved						5
8	8	P0 Base register	P0BR	RW	PROC	SOC/C		1
9	9	P0 Length register	P0LR	RW	PROC	SOC/C		1
10	A	P1 Base register	P1BR	RW	PROC	SOC/C		1
11	B	P1 Length register	P1LR	RW	PROC	SOC/C		1
12	C	System base register	SBR	RW	CPU	SOC/C		1
13	D	System length register	SLR	RW	CPU	SOC/C		1
14-15	E-F	Reserved						5
16	10	Process control block base	PCBB	RW	PROC	SOC/C		1
17	11	System control block base	SCBB	RW	CPU	SOC/C		1
18	12	Interrupt priority level	IPL	RW	CPU	SOC/C	Yes	1
19	13	AST Level	ASTLVL	RW	PROC	SOC/C	Yes	1
20	14	Software interrupt request register	SIRR	W	CPU	SOC/C		1
21	15	Software interrupt summary register	SISR	RW	CPU	SOC/C	Yes	1
22-23	16-17	Reserved						5
24	18	Interval counter control status	ICCS	RW	CPU	SOC/C	Yes	3
25	19	Next interval count	NICR					5
26	1A	Interval count	ICR					5
27	1B	Time-of-Year register	TODR	RW	CPU	SOC/C		2

Table Heading Key

IPR Number *Decimal:* The decimal number of the processor register; *Hex:* The hex number of the processor register.
Type *R:* read-only register; *W:* write-only register; *RW:* read/write register.
Scope *CPU:* CPU wide register; *PROC:* per the processor register.
Implemented Where? *SOC/C:* Implemented in the SOC CPU Chip; *SSC:* Implemented in the MicroVAX system support chip
Init?: Is this register initialized on Module RESET? (Power-up, Negation of DCOK) *yes* or *no*.
Category: The processor register category as previously defined in Table 3–3.

Table 3–3 (Cont.) KA660 Internal Processor Registers

IPR Number						Implemented		
Decimal	**Hex**	**Register Name**	**Mnemonic**	**Type**	**Scope**	**Where**	**Init?**	**Category**
28	1C	Console storage receiver status	CSRS	RW	CPU	SOC/C	Yes	7
29	1D	Console storage receiver data	CSRD	R	CPU	SOC/C	Yes	7
30	1E	Console storage transmitter status	CSTS	RW	CPU	SOC/C	Yes	7
31	1F	Console storage transmitter data	CSTD	W	CPU	SOC/C	Yes	7
32	20	Console receiver control/status	RXCS	RW	CPU	SOC/C	Yes	4
33	21	Console receiver data buffer	RXDB	R	CPU	SOC/C	Yes	4
34	22	Console transr control /status	TXCS	RW	CPU	SOC/C	Yes	4
35	23	Console transr data buffer	TXDB	W	CPU	SOC/C	Yes	4
36	24	Translation buffer disable	TBDR					5
37	25	Cache control	CCR	RW		SOC/C	Yes	3
39	27	Memory system error	MSER	RW	CPU	SOC/C	YES	3
40	28	Reserved						5
41	29	Reserved						5
42	2A	Console saved PC	SAVPC	R	CPU	SOC/C		3
43	2B	Console saved PSL	SAVPSL	R	CPU	SOC/C		3
44-54	2C-36	Reserved						5
55	37	I/O System reset register	IORESET	W	CPU	SOC/C		4
56	38	Memory management enable	MAPEN	RW	CPU	SOC/C	Yes	1
57	39	Translation buffer invalidate all	TBIA	W	CPU	SOC/C		1
58	3A	Translation buffer invalidate single	TBIS	W	CPU	SOC/C		1
59-61	3B-3D	Reserved						5
62	3E	System identification	SID	R	CPU	SOC/C		1

Table Heading Key

IPR Number *Decimal:* The decimal number of the processor register; *Hex:* The hex number of the processor register.
Type *R:* read-only register; *W:* write-only register; *RW:* read/write register.
Scope *CPU:* CPU wide register; *PROC:* per the processor register.
Implemented Where? *SOC/C:* Implemented in the SOC CPU Chip; *SSC:* Implemented in the MicroVAX system support chip
Init?: Is this register initialized on Module RESET? (Power-up, Negation of DCOK) *yes* or *no*.
Category: The processor register category as previously defined in Table 3–3.

Table 3–3 (Cont.) KA660 Internal Processor Registers

IPR Number						Implemented		
Decimal	Hex	Register Name	Mnemonic	Type	Scope	Where	Init?	Category
63	3F	Translation buffer check	TBCHK	W	CPU	SOC/C		1
64-127	40-7F	Reserved						6

Table Heading Key

IPR Number *Decimal:* The decimal number of the processor register; *Hex:* The hex number of the processor register.
Type *R:* read-only register; *W:* write-only register; *RW:* read/write register.
Scope *CPU:* CPU wide register; *PROC:* per the processor register.
Implemented Where? *SOC/C:* Implemented in the SOC CPU Chip; *SSC:* Implemented in the MicroVAX system support chip
Init?: Is this register initialized on Module RESET? (Power-up, Negation of DCOK) *yes* or *no*.
Category: The processor register category as previously defined in Table 3–3.

3.5 Process Structure

A process is a single thread of execution. The context of the current process is contained in the process control block (PCB), pointed to by the process control block base register (PCBB).) The KA660 implements these structures as defined in the *VAX Architecture Reference Manual*, which should be referenced for a description of the PCB and the PCBB.

3.6 Data Types

The central processor provides the following subset of the VAX data types:

- Byte
- Quadword
- F-floating
- Word
- Character string
- G-floating
- Longword
- Variable-length bit field
- D-floating

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

3.7 Instruction Set

The KA660 CPU implements the following subset of the VAX instruction set types in microcode:

- Integer arithmetic and logical
- Variable length bit field
- Procedure call
- Operating system support
- G_floating
- Address
- Control
- Miscellaneous
- F_floating
- D_floating

- Queue
- Character string
  - MOVC3
  - MOVC5
  - CMPC3[1]
  - CMPC5[1]
  - LOCC[1]
  - SCANC[1]
  - SKPC[1]
  - SPANC[1]

The KA660 SOC CPU provides special microcode assistance to aid the macrocode emulation of the following instruction groups:

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

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

- Octaword
- Compatibility mode instructions

Appendix D lists the entire KA660 instruction set, indicating which instructions are implemented in the floating point accelerator (FPA), and which instructions have microcode assists to speed up macrocode emulation.

3.8 Memory Management

The KA660 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).

3.8.1 Translation Buffer

To reduce the overhead associated with translating virtual addresses to physical addresses, the KA660 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 the following actions are performed:

- Memory management is enabled or disabled (for example, by writes to IPR 56).
- Any page table base or length registers are modified (for example, by writes to IPRs 13:8).

[1] These instructions were in the microcode assisted category on the KA630-A (MicroVAX II) and therefore had to be emulated.

- IPR 57 (TBIA) or IPR 58 (TBIS) are written to.

Each entry is divided into two parts: a 23-bit tag register and a 32-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 page frame number 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.

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 are used for the translation.

If there is no match, the translation buffer does not contain the necessary VAX PTE information to translate the address of the reference, 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). This pointer is called the NLU pointer.

3.8.2 Memory Management Control Registers

There are four IPRs that control the memory management unit (MMU):

IPR 56 (MAPEN)
IPR 57 (TBIA)
IPR 58 (TBIS)
IPR 63 (TBCHK)

Memory management can be enabled/disabled using IPR 56 (MAPEN). Writing 0 to this register with an MTPR instruction disables memory management and writing a 1 to this register with an 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 page table entry for the system or current process region, or a system page table entry that maps any part of the current process page table, all process pages mapped by the page table entry 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 an MFPR instruction from any of these registers is UNDEFINED.

3.9 Interrupts and Exceptions

Both interrupts and exceptions divert execution from the normal flow of control.

An interrupt is caused by some activity outside the current process and typically transfers control outside the process (for example, an interrupt from an external hardware device). An exception is caused by the execution of the current instruction and is typically handled by the current process (for example, an arithmetic overflow).

3.9.1 Interrupts

Interrupts can be divided into two classes: Non-Maskable and maskable.

Non-maskable interrupts cause a halt with the hardware halt procedure. The hardware halt procedure does the following:

- Saves the PC, PSL, MAPEN<0> and a halt code in IPRs.
- Raises the processor IPL to 1F.
- 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 because this halt code is generated after a hardware reset).

Maskable interrupts cause the following:

- The PC and PSL is saved.
- The processor IPL is raised to the priority level of the interrupt (except for Q22-bus, mass storage and network interface interrupts where the processor IPL is set to 17 independent of the level at which the interrupt was received.)
- The interrupt is dispatched to the appropriate service routine through the SCB.

The various interrupt conditions for the KA660 are listed in Table 3–4 along with their associated priority levels and SCB offsets.

Table 3–4 Interrupt Priority Levels

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).	**
	BINIT asserted on Q22-bus when configured as an auxiliary	*

* —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.) Interrupt Priority Levels

Priority Level	Interrupt Condition	SCB Offset
	BHALT asserted on Q22-bus	**
	BREAK generated by the console device	**
1F	Unused	
1E	BPOK negated on Q22-bus	0C
1D	CDAL Bus parity error	60
	Q22-bus NXM on a write	60
	Uncorrectable main memory errors	60
	CDAL Bus timeout during DMA	60
	Main memory NXM 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
	Programmable Timers	78,7C
	SHAC Mass Storage Interface (DSSI port 1)(External)	104
	SGEC Network Interface	10C
	Interprocessor Doorbell	204
	BR4 L asserted	Q22-bus Vector plus 200_{16}
13:10	Unused	
0F:01	Software interrupt requests	84-BC

* —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).

NOTE

Because the Q22-bus does not allow differentiation between the four bus grant levels (for example, a level 7 device could respond to a level 4 bus grant), the KA660 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 KA660 maintains the IPL at the priority of the interrupt for all other interrupts.

The interrupt system is controlled by following three IPRs:

- IPR 18, the interrupt priority level register (IPLR) (Figure 3–4), is used for loading the processor priority field in the PSL (bits<20:16>).
- IPR 20, the software interrupt request register (SIRR) (Figure 3–5), is used for creating software interrupt requests.
- IPR 21, the software interrupt summary register (SISR) (Figure 3–6), records pending software interrupt requests at levels 1 through 15.

The format of these registers is presented in Figure 3–4, Figure 3–5, and Figure 3–6. Refer to the *VAX Architecture Reference Manual* for more information on these registers.

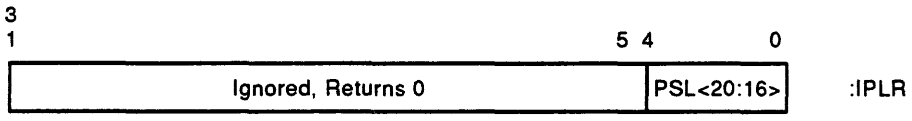


Figure 3–4 Interrupt Priority Level Register (IPLR) - (IPR 18_{10} 12_{16})

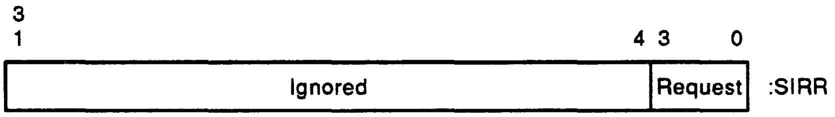


Figure 3–5 Software Interrupt Request Register (SIRR) - (IPL 20_{10} 14_{16})

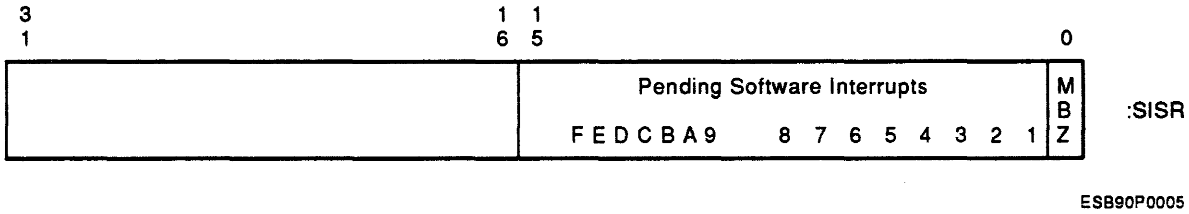


Figure 3–6 Software Interrupt Summary Register (SISR) - (IPL 21_{10} 15_{16})

3.9.2 Exceptions

Exceptions can be divided into three types: trap, fault, and 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. It 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 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 can be divided into six classes. A list of exceptions, grouped by class, is given in the next table. 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 Section 3.9.3.2.

Table 3–5 Exception Classes

Exception Class	Type	SCB Offset
Arithmetic Exceptions		
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		
Access control violation	Fault	20
Translation not valid	Fault	24
Operand Reference Exceptions		
Reserved addressing mode	Fault	1C
Reserved operand		18
Instruction Execution Exceptions		
Reserved/Privileged instruction	Fault	10
Emulated instruction	Fault	C8,CC
Change mode	Trap	40-4C
Breakpoint	Fault	2C
Tracing Exception		
Trace	Fault	28
System Failure Exceptions		
Interrupt stack not valid	Abort	[2]
Kernel stack not valid	Abort	08

[2] Dispatched by resident firmware rather than through the SCB.

Table 3–5 (Cont.) Exception Classes

Exception Class	Type	SCB Offset
Machine check including the following: • 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	Abort	04

3.9.3 Information Saved on a Machine Check Exception

In response to a machine check exception, the following information is pushed onto the stack as shown in Figure 3–7:

- Contents of the processor status longword
- Contents of the program counter
- Four parameters
- A byte count

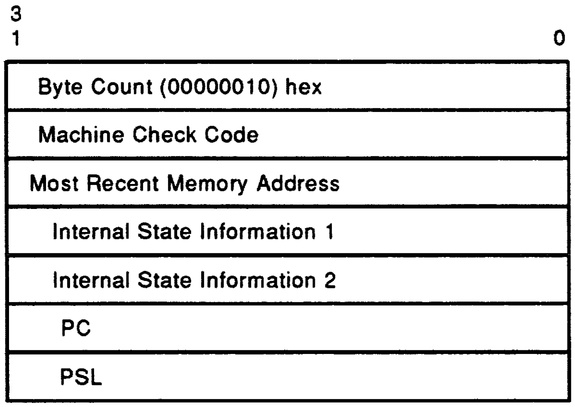

3
1
0
Byte Count (00000010) hex
Machine Check Code
Most Recent Memory Address
Internal State Information 1
Internal State Information 2
PC
PSL

Figure 3–7 The Processor Stack After a Machine Check Exception

The diagram of the stack pointer is explained in the following paragraphs.

3.9.3.1 Byte Count

Byte Count<31:0> 00000010_{16}, 16_{10}. This value indicates the number of bytes of information that follow on the stack (not including the PC and PSL).

3.9.3.2 Machine Check Code Parameter

Machine check code <31:0> is 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 are as follows:

Floating Point Errors - These codes indicate that the floating point accelerator (FPA) detected an error while communicating with the CPU during the execution of a floating point instruction. The most likely cause of these types of machine checks is a problem internal to the SOC CPU chip. 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 on the stack by this type of machine check is from the instruction that caused the machine check.

Table 3–6 Floating Point Error Machine Checks

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

Table 3–7 Memory Management Error Machine Checks

Hex Code	Error Description
05	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."
06	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 after a translation buffer "miss."

Table 3–7 (Cont.) Memory Management Error Machine Checks

Hex Code	Error Description
07	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.
08	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 SOC 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 SOC CPU 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.

Table 3–8 Interrupt Error Machine Checks

Hex Code	Error Description
09	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 SOC CPU microcode will cause random microinstructions to be executed. The most likely cause of this type of machine check is a problem internal to the SOC CPU 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.

Table 3–9 Microcode Error Machine Checks

Hex Code	Error Description
0A	An impossible state was detected during an MOVC3 or MOVC5 instruction (not move forward, move backward, or fill).

Read Errors - These codes indicate that an error was detected while the SOC 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, MEAR, and SEAR. 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 on the stack by this type of machine check is from the instruction that caused the machine check.

Table 3–10 Read Error Machine Checks

Hex Code	Error Description
80	An error occurred while reading an operand, a process page table entry 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 SOC 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, MEAR, SEAR, 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.

Table 3–11 Write Error Machine Checks

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

3.9.3.3 Most Recent Virtual Address Parameter

Most recent virtual address <31:0> 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 a 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; if the machine check occurred on a write operation, the contents of this field cannot be used for error recovery.

3.9.3.4 Internal State Information 1 Parameter

Internal state information 1 is divided into five fields as described in the following table.

Table 3–12 Internal State Information 1 Field Description

Bit Field	Description
<31:24>	This field captures the opcode of the instruction that was being read or executed at the time of the machine check.
<23:20>	This field is read as <1111>.
<19:16>	This field captures the current contents of HSIR<3:0>.
<15:8>	This field captures the state of CCR<7:0> at the time of the machine check. See section 5.2.5 for interpretation of the contents of this register.
<7:0>	This field captures the state of MSER<7:0> at the time of the machine check. See section 5.2.6 for interpretation of the contents of this register.

3.9.3.5 Internal State Information 2 Parameter

Internal state information 2 is divided into seven fields as described in the following table:

Table 3–13 Internal State Information 2 Field Description

Bit Field	Description
<31:24>	This field captures internal state of the SOC CPU chip at the time of the machine check. This field contains the SC register <7:0>.
<23:22>	This field is read as <11>.
<21:16>	This field contains the state flags <5:0>.
<15>	This field captures the state of the VAX CAN'T RESTART flag at the time of the machine check.
<14:12>	This field is read as <111>.
<11:8>	These bits contain the arithmetic logic unit (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 time of the machine check.

3.9.3.6 PC

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

3.9.3.7 PSL

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

3.9.4 System Control Block (SCB)

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). The format of the system control block base register is shown in Figure 3–8.

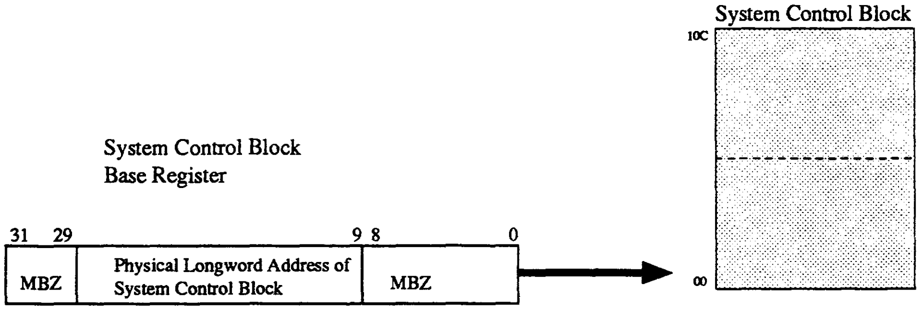


Figure 3–8 System Control Block Base Register (SCBB)

The description of the format is listed in Table 3–14.

Table 3–14 The System Control Block Format

SCB Offset	Interrupt/Exception Name	Type	# Params	Notes
00	Unused	-	-	IRQ passive release on other VAX computers.
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 3–14 (Cont.) The System Control Block Format

SCB Offset	Interrupt/Exception Name	Type	# Params	Notes
SCB Offset	**Interrupt/Exception Name**	**Type**	**# Params**	**Notes**
30	Unused	-	-	Compatibility mode in other VAX computers
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	Correctable main memory errors	Interrupt	0	IPL is 1A (CRD_L).
58-5C	Unused	-	-	
60	Memory error	Interrupt	0	IPL is 1D (MEMERR_L)
64-74	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.

Table 3–14 (Cont.) The System Control Block Format

SCB Offset	Interrupt/Exception Name	Type	# Params	Notes
90-BC	Software levels 4-15	Interrupt	0	
C0	Interval timer	Interrupt	0	IPL is 16(INTTIM_L).
C4	Unused	-	-	
C8	Emulation start	Fault	10	Same mode exception,FPD = 0; parameters are opcode, PC, specifiers.
CC	Emulation continue	Fault	0	Same mode exception,FPD = 1: no parameters.
D0-DC	Unused	-	-	
E0-EC	Unused	-	-	
F0-F4	Unused	-	-	
F8	Console receiver	Interrupt	0	IPL is 14
FC	Console transmitter	Interrupt	0	IPL is 14
104	Mass storage interface (DSSI PORT)	Interrupt	0	IPL is 14
10C	Network interface	Interrupt	0	IPL is 14

3.9.5 Hardware Detected Errors

The KA660 is capable of detecting the following types of error conditions during program execution:

- CDAL bus parity errors indicated by MSER<6> (on a read) or MEMCSR16<7> (on a write) being set.
- Cache tag parity errors indicated by MSER<0> being set.
- Cache data parity errors indicated by MSER<1> being set.
- Q22-bus NXM errors indicated by DSER<7> being set.
- Q22-bus NO SACK errors (no indicator).
- Q22-bus NO GRANT errors indicated by DSER<2> being set.
- Q22-bus device parity errors indicated by DSER<5> being set.
- CDAL-Bus timeout errors indicated by DSER<4>(only on DMA) being set.
- Main memory NXM errors indicated by DSER<0> (only on DMA) being set.
- Main memory correctable errors indicated by MEMCSR16<29> being set.
- 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.

3.9.6 The 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>, a halt code and VALID bit in IPR 43 (SAVPSL). The formats for (SAVPC) and (SAVPSL) are shown in Figure 3–9 and Figure 3–10 respectively.

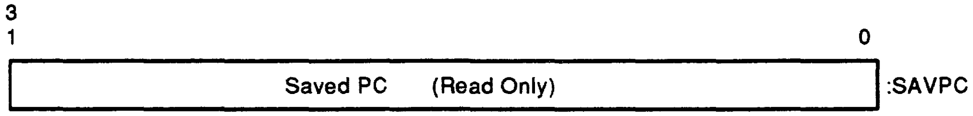


Figure 3–9 Console Saved PC (SAVPC) - (IPR 42_{10} $2A_{16}$)

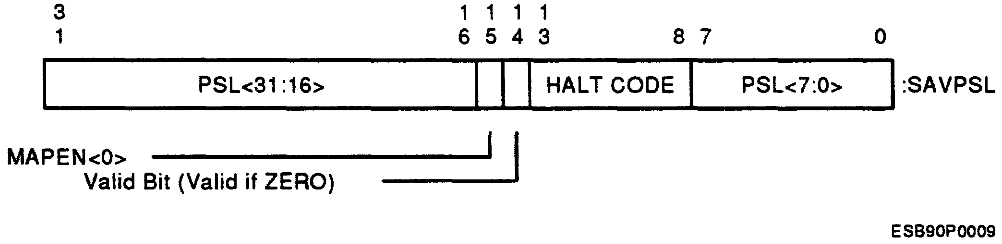


Figure 3–10 Console Saved PSL (SAVPSL) - (IPR 43_{10} $2B_{16}$)

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 $_{16}$. Table 3–15 shows the state of the CPU.

Table 3–15 CPU State After a HALT

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 (IPR 4)
PSL	041F 0000 $_{16}$
PC	2004 0000 $_{16}$
MAPEN	0

Table 3–15 (Cont.) CPU State After a HALT

Register	New Contents
ICCS	0 (for a halt code of 3)
MSER	0 (for a halt code of 3)
CCR	0 (for a halt code of 3)
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 hardware event indicators to dispatch the interrupt or exception that caused the halt, to the appropriate firmware routine (either console emulation, power-up, reboot, or restart). The halt codes and their event indicators are listed in Table 3–16. Complete halt code descriptions are given in Table J–1.

Table 3–16 HALT Codes

Halt Code	Interrupt Condition	Event Indicator
	HALT Codes for unmaskable interrupts	
2	**External halt (SOC/C HALT_L pin asserted)**	
	BHALT asserted on the Q22-bus	DSER<15>
	Remote boot serviced by SGEC	NICSR5<7>
	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 (SOC/C RESET pin asserted)**	
	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.	-
	HALT Codes for Exceptions	
6	HALT instruction executed in Kernel Mode	

Table 3–16 (Cont.) HALT Codes

Halt Code	Interrupt Condition	Event Indicator
HALT Codes for Exceptions that occurred while Serving 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	
10	Access control violation (ACV) or translation not valid (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	
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	

3.10 System Identification

The firmware and operating system software references two registers to determine the processor on which they are running. The first, the system identification register (SID), is an internal processor register. The second, the system identification extension register (SIE), is a firmware register located in the on board EPROM.

3.10.1 System Identification Register

The system identification register (SID), IPR 62, is a read-only register implemented in the SOC CPU chip . This 32-bit, read-only register is used to identify the processor type and its microcode revision level. The SID longword is read from IPR 62 using the MFPR instruction. This longword value is processor specific. The format is shown in Figure 3–11. Bit definitions are listed in Table 3–17.

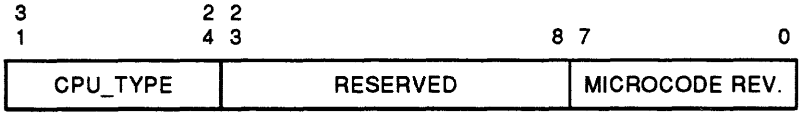


Figure 3–11 System Identification Register (SID) - (IPR 62_{10} $3E_{16}$)

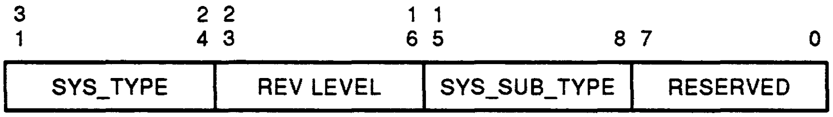
Table 3–17 System Identification Register (SID) Bits

Field	Name	RW	Description
<31:24>	CPU_ TYPE	ro	CPU type is the processor specific identification code. This field always reads as 20_{10} indicating the processor is implemented with an SOC CPU chip.
<23:8>	RESERVED	ro	Reserved for future use.
<7:0>	VERSION	ro	Version of the microcode.

3.10.2 System Identification Extension Register (SIE) (20040004)

The system identification extension register is an extension of the SID register and is used to further differentiate between hardware configurations. The SID register identifies which CPU and microcode is executing, and the SIE register identifies what module and firmware revision are present. Note that the fields in this register are dependent on SID<31:24>(CPU_TYPE).

By convention, all MicroVAX systems implement a longword at physical location 20040004 in the firmware EPROM for the SIE register. This 32-bit Read-Only register is implemented in the KA660 ROM. Figure 3–12 shows the format of this register. Table 3–18 lists the definitions of the register bits.



ESB90P0011

Figure 3–12 System Identification Extension Register (SIE)

Table 3–18 System Identification Extension Register (SIE) Bits

Field	Name	RW	Description
31:24	SYS_TYPE	ro	This field identifies the type of system for a specific processor. *01 : Q22-bus single processor system.*
23:16	VERSION	ro	This eight bit field contains two hexadecimal digits that reflect the version of the resident firmware (EPROM). For example, if the firmware version is V5.0, the banner displays V5.0 and this field encodes as a 50_{16}.
15:8	SYS_SUB_ TYPE	ro	This field reflects the particular system sub-type. *01 : KA650* *02 : KA640* *03 : KA655* *04 : KA670*
7:0	RESERVED		This field is reserved.

3.11 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

3.11.1 Request Instruction-Stream Read References

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

3.11.2 Demand 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 the following references:

- Operand
- Page table entry (PTE)
- System control block (SCB)
- Process control block (PCB)

When interlocked instructions, such as branch on bit set and set interlock (BBSSI) are executed, a demand D-Stream read-lock reference is generated.

Because the CPU does not impose any restrictions on data alignment (other than the aligned operands of the ADAWI and interlocked queue instructions) and because 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.

3.11.3 Write References

Whenever data is stored or moved, a write reference is generated. Because the CPU does not impose any restrictions on data alignment (other than the aligned operands of the ADAWI and interlocked queue instructions), and because 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.

4
KA660 Cache Memory

To maximize CPU performance, the SOC CPU provides a 6 Kbyte, 6-way associative write-through cache with an 8 byte block and fill size.

4.1 Cacheable References

Any reference that is 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 DSSI interface, or the Ethernet interface.

Both I-Stream and D-Stream references are cached in the enabled banks of the cache.

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 READ reference that is not stored in the cache, a quadword READ 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 transfer is a request I-stream read (prefetch) and the second transfer is 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 is a demand D-stream read and the second is a request D-stream read (fill).

If the CPU is retried on the second (fill) longword then the first longword is delivered to the CPU but not cached. The request will not be retried if the need for the data is resolved (for example, it was a prefetch but a branch was taken).

4.2 Cache Organization

The cache is logically organized as 6, 1 Kbyte banks, shown next. It is a read-allocate, no-write-allocate, and write-through cache. A single parity bit is generated, stored and checked for each byte of data and each tag. The cache is enabled/disabled via a bit in the cache control register (CCR).

Other bits in these two registers allow the data and tags to be addressed directly, to check the parity generating/checking logic and to provide the hit/miss status of each bank for the most recent D-stream read or write cycle.

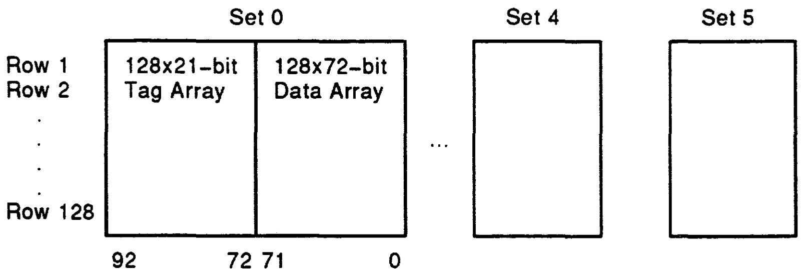


Figure 4–1 Logical Organization of Cache

Each row within a set corresponds to a cache entry, and there are 128 entries in each set. Each entry contains a 21-bit tag block and a 72-bit (eight-byte) data block. A cache entry is logically organized as shown in Figure 4–2:

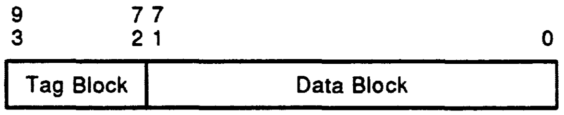


Figure 4–2 Cache Entry

Figure 4–3 shows the format of a cache tag entry.

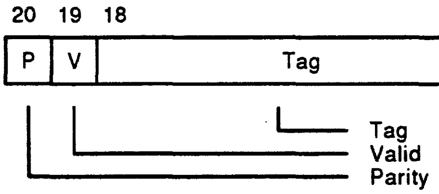


Figure 4–3 Cache Tag Entry

The parity bit stores odd parity over the tag field only. The valid bit is not included in the parity calculation.

The valid bit indicates whether or not the corresponding entry in the cache refers to a useable data/address pair.

The tag consists of bits <28:10> of the physical address.

Figure 4–4 shows the format of a cache data entry.

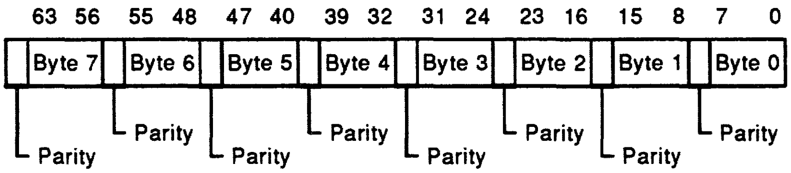


Figure 4–4 Cache Data Entry

Each data entry consists of eight bytes of data, each with an associated parity bit. Odd parity is generated for the odd data bytes and even parity is generated for the even data bytes.

4.2.1 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 follows:

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 <9:3> of the physical address, is used to select one of the 128 rows of the cache, with each row containing a single entry from each set. The label field, bits <28:10> of the physical address, is then compared to the tag block of the entry from all six 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 4–5 shows how cache addresses are translated.

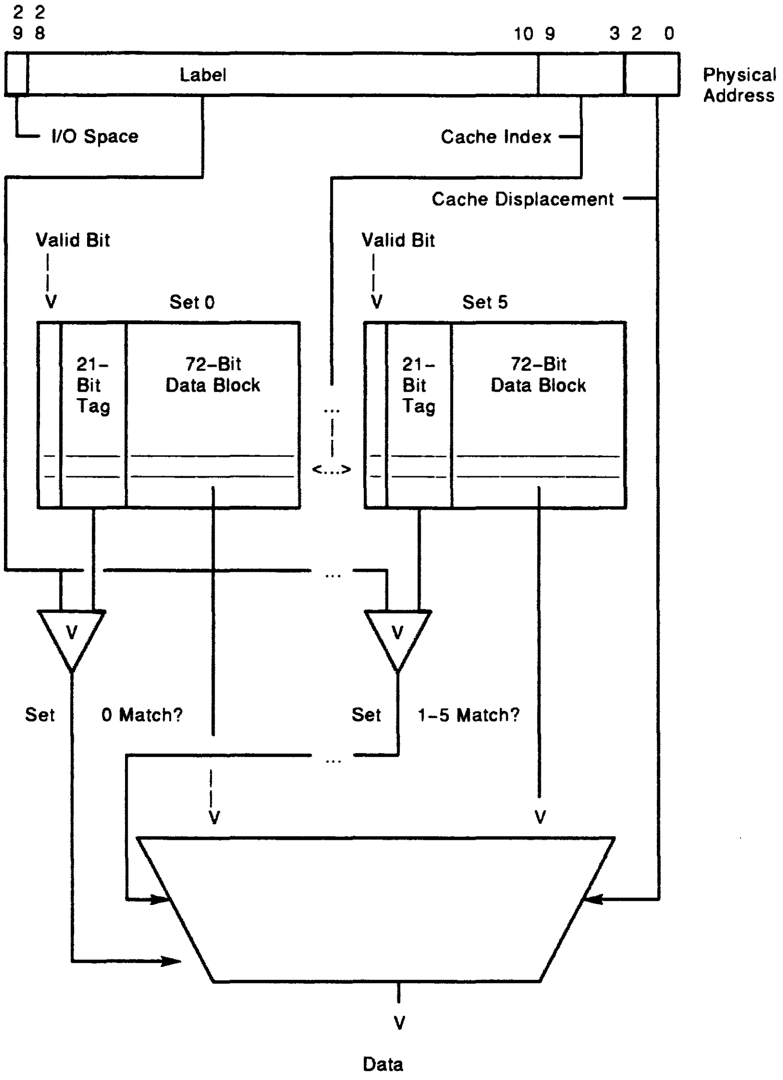


Figure 4–5 Cache Address Translation

4.2.2 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.

Because the cache is six-way associative, there are six data blocks (one in each set) that can be allocated to a given quadword. The cache index field determines which row of 6 data blocks is to be used while set selection is determined by a not most recently used (NMRU) algorithm. The bank to be selected is pointed to by a three bit counter. The counter is set to 000 when the cache is disabled (CCR ENABLE_CACHE=0); it is advanced:

- Every time a block is allocated.

- Every time a read or write hits in the bank to which the counter is currently pointing.

The counter counts modulo 6 with no missing counts regardless of which banks are enabled. The contents of the counter is driven out on pins TEST<2:0> (test mode is in observe miscellaneous state), which enables external logic to track which addresses are cached internally.

4.2.3 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.

When a DMA write references hits the cache, the cache entry containing the copy of the referenced location is invalidated.

4.2.4 Cache Control Register (CCR, IPR 37)

The cache control register (CCR), internal processor register 37, controls cache operating mode and flushes the cache. It is unique to CPU designs that use the SOC CPU chip. Figure 4–6 shows the register format and Table 4–1 describes the bits.

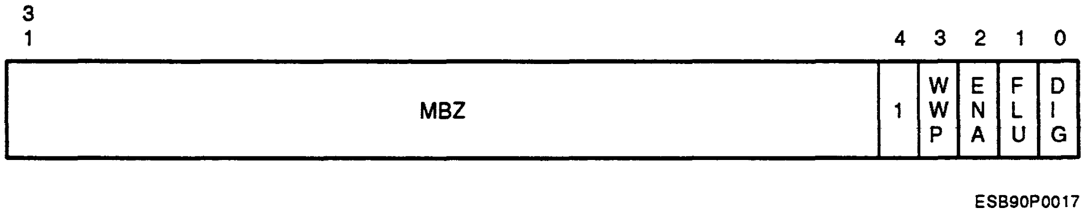


Figure 4–6 Cache Control Register

Table 4–1 Cache Control Register Bits

Data Bit	Name	Description	Type
<31:5>	Unused	These bits always read as zeros. Writes have no effect.	
<4>	Unused	This bit is always read as a one. Writes have no effect.	
<3>	WWP	Write wrong parity. For diagnostic use. When set, it inverts the data parity bits accessed from the cache. This causes a parity error when they are compared to parity computed from the accessed data, and is used to test the parity generation /checking logic. When clear data parity bits are not affected. This bit does not affect tag parity bits. Cleared when RESET_L is asserted.	Read/Write

Table 4–1 (Cont.) Cache Control Register Bits

Data Bit	Name	Description	Type
<2>	ENA	Enable cache. It enables or disables normal operation of the cache. When set, both I-stream and D-stream references are cached in the enabled banks of the cache, and cache tag and data parity errors are reported. When clear, all references (read and write) result in a miss and no cache parity is checked. When enable cache is set the cache should be flushed by writing a one to the flush cache bit. Enable cache is cleared when RESET_L is asserted. **NOTE** **The cache may be operated with both the Diagnostic bit set and the Enable Cache bit set.**	Read/Write
<1>	FLU	Flush cache. Always read as zero. Writing a one to this bit clears all valid bits in the cache tag array. Writing a zero has no effect.	Write only
<0>	DIA	Diagnostic. When this bit is set, the cache entries and BEHR register may be accessed via a region in I/O space. When clear, references to the same region of I/O space result in bus cycles.	Read/Write

Table 4–2 Cache Diagnostic Mode Addresses

Information	Address Range
Cache Tag	20150000 - 201503FF
Cache Data	20150400 - 201507FF
BEHR	20150800 - 20150FFF

Note the BEHR register may be accessed at multiple addresses.

When the diagnostic bit is set, writes to cache data addresses will write to the cache data longword indexed by bits <9:2> of the address. All banks which have the corresponding BEHR ENABLE_BANK bit set will be written, and correct data parity will also be written. Byte and word writes as well as longword are possible.

Reads from cache data addresses read the cache data longword addressed; if more than one ENABLE_BANK bit is set then the highest priority bank enabled will return the data (bank 0 is highest priority, bank 7 is lowest).

Writes to cache tag addresses write to the cache tag indexed by bits <9:3> of the address. All banks which have the corresponding BEHR ENABLE_BANK bit set will be written. The write data format is shown in Figure 4–7.

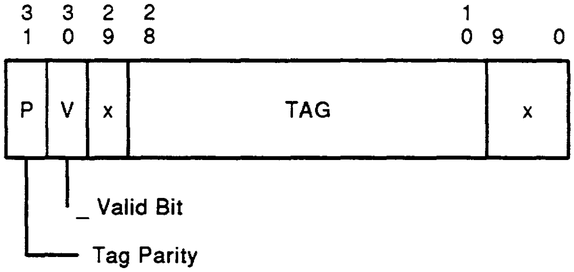


Figure 4–7 Tag Diagnostic Write Data Format

Reads from cache tag addresses read the cache tag addressed, plus the data parity from the data longword addressed by bits <9:2> of the address as shown in the next figure. If more than one ENABLE_BANK bit is set then the highest priority bank enabled will return the tag.

The read data format is shown in Figure 4–8.

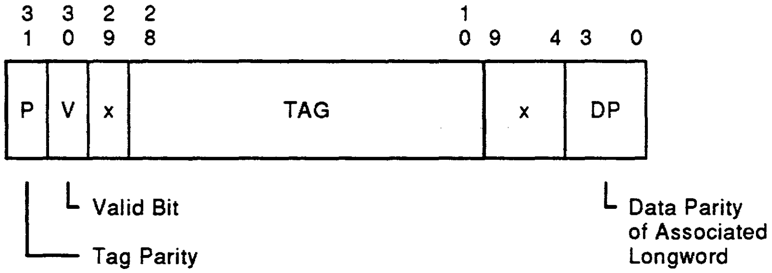


Figure 4–8 Tag Diagnostic Read Data Format

4.2.5 Bank Enable/Hit Miss Register (BEHR)

The bank enable/hit mis (BEHR) register allows individual sets of the cache to be enabled /disabled and also provides bits which indicate the hit/miss status for each set of the cache. The format is shown in Figure 4–9.

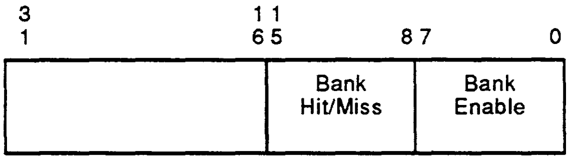


Figure 4–9 Bank Enable/Hit Miss Register

Table 4–3 Bank Enable/Hit Miss Register (BEHR)

Data Bit	Name	Description	Type
<15:8>	Bank hit	These bits are provided for use in testing the cache. They represent the hit/miss status of each bank of the cache for the most recent D-stream read or write cycle. A 1 indicates that there was a hit in the corresponding bank, a 0 indicates a miss. Bank hit<7:0> are cleared when RESET_L is asserted.	Read only
<7:0>	Enable Bank	These bits are written to by the power-up diagnostics. The diagnostics are responsible for determining and enabling good cache banks. When set to one, these bits enable banks <7:0> of the cache, respectively. Note that for any bank to be enabled, CCR enable cache or CCR diagnostic must be set. When cleared to zero, the corresponding banks are disabled. Enable Bank <7:0> are cleared when RESET_L is asserted. Whenever a bank is either enabled or disabled, software should also flush the cache by writing a one to CCR Flush. For any reference there should be a hit in only one bank. If, due to a hardware malfunction (corrupted tag bit), the reference hits in more than one bank, only the bank with the higher priority drives the data. The banks are numbered in decreasing order of priority. Bank 0 has the highest priority and bank 5 has the lowest priority. Normally, the corrupted tag would be reported via the MSER.	Read Only

4.2.6 Memory System Error Register (MSER, IPR 39)

The memory system error register (MSER), internal processor register 39, reports information about DAL bus and cache errors. The two basic classes of errors reported are:

- Errors that don't immediately affect operation of the SOC CPU and therefore post an interrupt but not change instruction flow.
- Errors that do affect operation and will cause a machine check (trap through SCB vector 4.

The format is shown in Figure 4–10.

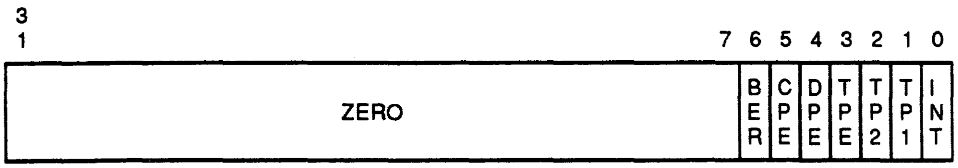


Figure 4–10 Memory System Error Register (MSER, IPR 39)

Table 4–4 Memory System Error Register (MSER, IPR 39)

Data Bit	Name	Description	Type
<31:7>		Always read as zero.	
<6>	DPE	DAL Parity error. This bit is set when a DAL parity error is detected on either a demand or request read cycle which receives a normal termination response (RDY_L asserted, ERR_L deasserted).	Sticky
<5>	MCADPE	Machine check abort. DAL parity error. This bit is set whenever a machine check is caused by a DAL parity Error. A DAL parity error will only cause a machine check on a demand read cycle.	Sticky
<4>	MCACPE	Machine check abort cache parity error. This bit is set whenever a machine check is caused by a cache parity error (tag or data). A cache parity error only causes a machine check on a demand read cycle that hits the cache.	Sticky
<3:2>		Always read as zero.	
<1:0>		These bits are set independandtly to show the scope of a cache parity error on either a demand or request cycle. MSER<0> is set to indicate that the cache parity error was caused by a tag error; MSER<1> by a data error. Note that a simultaneuos cache tag and data parity error will only log the fact that a cache tag parity error occured.	Sticky

NOTE

MSER bits <6:4, 1:0> are "sticky" in the sense that once set, they remain set until MSER is explicitly cleared by writing the MSER (with a MTPR instruction) irrespective of the data. Parity errors occuring while an error condition is posted in MSER can only set an additional bit (for example, MSER <6:4, 1:0> cannot be cleared through subsequent errors).

4.2.7 Cache Error Detection

Both the tag and data arrays in the cache are protected by parity. Each 8-bit byte of data and the 19-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. The following are the different reference types:

- During a demand D-stream read reference, the entire cache is flushed, and the CCR is cleared (which disables the cache.) The cause of the error is logged in MSER<4,1:0> and a machine check abort is initiated.

- During a request I-stream read reference, the entire cache is flushed (unless CCR<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 the cache remains enabled.
- During a masked or unmasked write reference, the entire cache is flushed (unless CCR<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 the cache remains 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.

4.2.7.1 Use of the C-Chip Registers

The 10 registers implemented by the C-chip provide full control over the backup cache tag store and the primary tag store in the C-chip. Access to these registers is with the MTPR and MFPR instructions, which require kernel-mode privilege.

4.2.7.1.1 Control of the Cache

Normal operational control of the backup cache and primary tag store in the C-chip is provided through writes to BCCTL. Bits in this register enable the use of backup cache and primary tag store.

The backup cache and primary tag store may be flushed during normal operation by writing a 0 to BCFBTS and BCFPTS, respectively.

4.2.7.1.2 Error Recovery

When an error is detected by the C-chip, it latches error information. This information is available by reading BCSTS and BCERR. STATUS_LOCK (BCSTS<0>) may be written to tell the C-chip that the error information has been read, and to enable it to detect subsequent errors.

If the error was a tag parity error in one of the tag stores, the error may be corrected by creating a new tag entry with a write to BCIDX, followed by a write to BCBTS, BCP1TS, or BCP2TS.

4.2.7.1.3 Cache Initialization

At power-up, the backup cache tag store and primary tag store must be initialized by writing each entry with an invalid tag with good parity. Each entry may be written with a write to BCIDX, followed by a write to BCBTS, BCP1TS, or BCP2TS.

As part of cache initialization, cache refresh must be enabled, and the cache RAM speed must be specified by writing to BCCTL. The Console macrocode will set the RAM speed for 1 Cycle.

4.2.7.1.4 Diagnostics

The tag stores and the backup cache data RAMs may be tested by reading and writing cache tags with BCIDX, BCBTS, BCP1TS, and BCP2TS. Cache refresh may be tested by reading and writing BCRFR. Error detection may be tested by constructing an error and then reading the state from BCSTS and BCERR.

5
KA660 Main Memory System

The KA660-AA includes a main memory controller implemented by a single VLSI chip called the CMCTL. The KA660 main memory controller communicates with the MS650 memory boards over the MS650 memory interconnect, which uses the CD interconnect for the address and control lines and a 50-pin, ribbon cable for the data lines. It supports up to four MS650 memory boards, for a maximum of 64 Mbytes of ECC memory.

NOTE
The KA660 supports only MS650_Bx variations and does not support MS650-AA variations.

The memory 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.

5.1 KA660 Timing

The system clock for the VAX 4000-200 computer operates at 114.285 MHz creating a cycle time of 70 nanoseconds. Table 5–1 lists the reference times for CPU reads and writes and Q22-bus interface reads and writes. This table also provides information about error handling times.

Table 5–1 KA660 Reference Timing

CPU Read Reference	
Longword	280 ns
Quadword	420 ns
First longword	280 ns
Second longword	140 ns
Aborted reference	280 ns
Longword (locked)	630 ns min
Aborted reference	280 ns
Retry (locked)	350 ns
CPU Write Reference	
Longword	140 ns

Table 5–1 (Cont.) KA660 Reference Timing

Longword (masked)	350 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 Bus Interface.	
Q22-bus Interface Read Reference	
Longword	420 ns
Quadword	560 ns
First longword	350 ns
Second longword	210 ns
Longword (locked)	420 ns
Q22-bus Interface Write Reference	
Longword	280 ns
Longword (masked)	420 ns
Quadword	490 ns
First longword	2800 ns
Second longword	210 ns
Quadword (masked)	770 ns
First longword	280 ns
Second longword	490 ns
Hexword	700 ns
First longword	280 ns
Second longword	210 ns
Third longword	210 ns
Hexword (masked)	980 ns
First longword	280 ns
Second longword	210 ns
Third longword	4900 ns
Octaword	910 ns
First longword	280 ns
Second longword	210 ns
Third longword	210 ns
Fourth longword	2100 ns
Octaword (masked)	1190 ns
First longword	280 ns
Second longword	210 ns
Third longword	210 ns
Fourth longword	490 ns

Table 5–1 (Cont.) KA660 Reference Timing

The above timing 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.

Error Handling

Correctable error	70 ns
Uncorrectable error	140 ns-read
Uncorrectable error	70 ns-write
CDAL parity error	70 ns-write
Refresh collision	280 ns

5.2 Main Memory Organization

Main memory is logically and physically divided into four boards which correspond to the four possible MS650 memory expansion modules that can be attached to a KA660-AA. Each memory board can contain zero (no memory module present) or four (MS650-BA present) memory banks. Each bank contains 1,048,576 (1M) aligned longwords. Each aligned longword is divided into four data bytes and is stored with seven ECC check bits, resulting in a memory array width of 39 bits.

5.3 Main Memory Addressing

The KA660-AA main memory controller is capable of controlling up to 16 banks of RAM, each bank containing 4 Mbytes 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 the bank.

A 4 Mbyte bank is accessed when bit <29> of the physical address is equal to zero, 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.

5.4 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 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.

5.5 Main Memory Error Status Register (MEMCSR16)

The main memory error status register, 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.

Data Bit	Name	Description	Type
<31>	RDS ERROR	When set, an uncorrectable ECC error occurred during a memory read or masked write reference. Cleared by writing a one to it. Writing a zero has no effect. Undefined if MEMCSR16<7> (CDAL BUS ERROR) is set. Cleared on power-up and the negation of DCOK when SCR<7> is clear.	Read/Write to clear
<30>	RDS HIGH ERROR RATE	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 one to it. Writing a zero has no effect. Undefined if MEMCSR16<7> (CDAL BUS ERROR) is set. Cleared on power-up and the negation of DCOK when SCR<7> is clear.	Read/Write to clear
<29>	CRD ERROR	When set, a correctable (single bit) error occurred during a memory read or masked write reference. Cleared by writing a one to it. Writing a zero has no effect. Undefined if MEMCSR16<7> (CDAL BUS ERROR) is set. Cleared by writing one, on power-up and the negation of DCOK when SCR<7> is clear.	Read/Write to clear
<28:9>	PAGE ADDRESS OF ERROR	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 SCR<7> is clear. The types of error conditions follow in order of priority:	Read only

Data Bit	Name	Description	Type
		• CDAL bus parity errors during a CPU write reference, as logged by the CDAL BUS ERROR bit. • Uncorrectable ECC errors during a CPU or DMA read or masked write reference, as logged by the RDS ERROR LOG bit. • Correctable ECC errors during a CPU or DMA read or masked write reference, as logged by CRD ERROR bit.	
<8>	DMA ERROR	When set, an error occured during a DMA read or write reference. Cleared by writing a one to it. Writing a zero has no effect. Cleared on power-up and the negation of DCOK when SCR<7> is clear.	Read/Write to clear
<7>	CDAL BUS ERROR	When set, a CDAL bus parity error occurred on a CPU write reference. Cleared by writing a one to it. Writing a zero has no effect. Cleared on power-up and the negation of DCOK when SCR<7> is clear.	Read/Write to clear
<6:0>	ERROR SYNDROME	This field stores the error syndrome. A non-zero syndrome indicates a detectable error has occured. A unique syndrome is generated for each possible single-bit (correctable) error. A list of the these syndromes and their associated single-bit errors appears on the next page. Any non-zero syndrome that is not contained on this list indicates a multiple bit (uncorrectable) error has occured. This field handles multiple errors in the same manner as MEMCSR16<28:9>. Cleared on power-up and the negation of DCOK when SCR<7> is clear.	Read only

The following is a list of the syndromes and their associated single-bit errors:

SYNDROME<6:0>	BIT POSITION IN ERROR
0000000	no error detected

```
                                     Data Bits (0-31 decimal)
1011000                            0    |
0011100                            1    |
0011010                            2    V
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      |
0000010                           33      |
0000100                           34      V
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
```

5.6 Main Memory Control and Diagnostic Status Register (MEMCSR17)

The main memory control and diagnostic status register, 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 5–1 shows the format of this register.

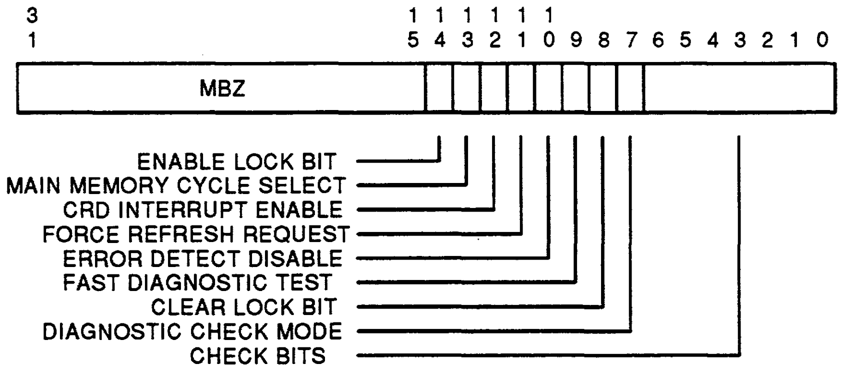


Figure 5–1 Main Memory Control and Diagnostic Status Register (MEMCSR17)

Data Bit	Name	Description	Type
<31:15>	Unused	This field reads as zero, must be written as zero.	
<14>	ENABLE LOCK BIT	When cleared, the main memory locking function (reflected by the LOCK BIT in MEMCSR15-0<6>) is disabled. When set, the main memory locking function (reflected by the LOCK BIT in MEMCSR15-0<6>) is enabled. Writing this bit has no effect on MEMCSR15-0<6>. This bit should always be clear, because the KA660-AA implements the main memory locking function in the Q22-bus Interface chip (CQBIC). Cleared on power-up and the negation of DCOK when SCR<7> is clear.	Read/Write
<13>	MAIN MEMORY CYCLE SELECT	When set, longword reads and the first longword in quadword reads occur in four CPU cycles (320 ns) and the second longword in a quadword read occurs in two CPU cycles (160 ns). When cleared, longword reads and the first longword in quadword reads occur in five CPU cycles (450 ns) and the second longword in a quadword read occurs in three CPU cycles (240 ns). **NOTE** **With the KA660-AA this bit must be cleared by the firmware memory configuration routine thus using the 5/3 timing. Cleared on power-up and the negation of DCOK when SCR<7> is clear.**	Read/Write

Data Bit	Name	Description	Type
<12>	CRD INTERRUPT ENABLE	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 SCR<7> is clear.	Read/Write
<11>	FORCE REFRESH REQUEST	When cleared, the refresh control logic operates in normal mode (refresh every 9.1 µs for 80 ns cycles and 1 wait state). 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 SCR<7> is clear.	Read/Write
<10>	MEMORY ERROR DETECT DISABLE	When set, error detection and correction (ECC) is disabled, so all memory errors go undetected. When cleared, error detection, correction, state capture, and reporting (through MEMCSR16) is enabled. Cleared on power-up and the negation of DCOK when SCR<7> is clear.	Read/Write
<9>	FAST DIAGNOSTIC TEST	This bit provides a mechanism for speeding up the diagnostic testing of main memory. When set, all main memory banks are read and written in parallel. When cleared, this bit has no effect on memory reads or writes. Cleared on power-up and the negation of DCOK when SCR<7> is clear. **NOTE** **Due to excess power consumption do not use MOVC instructions in fast diagnostic test mode.**	Read/Write
<8>	CLEAR LOCK BIT	Writing a one to this bit clears MEMCSR15-0<6> and "unlocks" main memory. Always read as zero. This bit is used to unlock memory that was locked by an interlocked instruction that was aborted, due to an error condition, before it could unlock memory. This bit should never be needed, because the KA660-AA implements the main memory locking function in the Q22-bus Interface chip (CQBIC). Cleared on power-up and the negation of DCOK when SCR<7> is clear.	Write Only

Data Bit	Name	Description	Type
<7>	DIAGNOSTIC CHECK MODE	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 SCR<7> is clear. **NOTE** **DIAGNOSTIC CHECK MODE is restricted to unmasked memory write references. No masked write references are allowed when DIAGNOSTIC CHECK MODE is enabled.**	Read/Write
<6:0>	CHECK BITS	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 SCR<7> is clear.	Read/Write.

5.7 Main Memory Error Detection and Correction

The KA660-AA 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 ribbon 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. Table 5–2 lists the actions performed on uncorrectable errors.

Table 5–2 Uncorrectable Error Actions

Error	Action 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.
	Bits <6:0> are loaded with the error syndrome which indicates that the error was uncorrectable.
	A machine check abort is initiated.
	If the read was a local-miss, global-hit read, or a read of the Q22-bus 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 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.
	Bits <6:0> are loaded with the error syndrome which indicates that the error was uncorrectable.
prefetch or	
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.
	Bits <6:0> are loaded with the error syndrome which indicates that the error was uncorrectable.
	A machine check abort is initiated.

Table 5–2 (Cont.) Uncorrectable Error Actions

Error	Action Performed
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. 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). 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. 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. ICR<15> is set to notify the initiating device. An interrupt is generated at IPL 1D through vector 60_{16}.

6 KA660 Console Serial Line

The console serial line provides the KA660 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 internal processor registers (IPRs) that control the operation of the console serial line are a superset of the VAX console serial line registers described in

6.1 Console Registers

There are four registers associated with the console serial line unit. They are implemented in the SSC chip and are accessed as IPRs 32-35. Refer to Table 6–1.

Table 6–1 Console Registers

IPR Number		Register Name	Mnemonic
Dec	Hex		
32	20	Console receiver control/status	RXCS
33	21	Console receiver data buffer	RXDB
34	22	Console transmit control/status	TXCS
35	23	Console transmit data buffer	TXDB

6.1.1 Console Receiver Control/Status Register - (IPR 32)

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. The format is shown in Figure 6–1. Table 6–2 lists the bit descriptions.

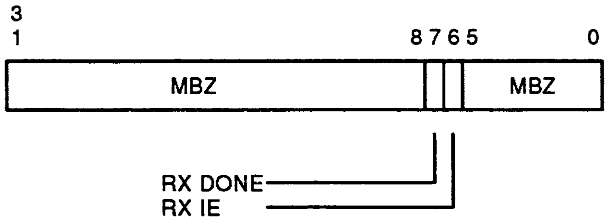


Figure 6–1 Console Receiver Control/Status Register - (IPR 32_{10} 20_{16})

Table 6–2 Console Receiver Control/Status Register

Data Bit	Name	Description
<31:8>	MBZ	These bits read as zero. Writes have no effect.
<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 the RXDB register is read. It is also cleared on power-up and the negation of DCOK.
<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 CSR<7> is clear.
<5:0>	Unused	These bits read as zero. Writes have no effect.

6.1.2 Console Receiver Data Buffer - (IPR 33)

The console receiver data buffer (RXDB), internal processor register 33, is used to buffer incoming data on the serial line and capture error information. The format is shown in Figure 6–2. Bit descriptions are listed in Table 6–3.

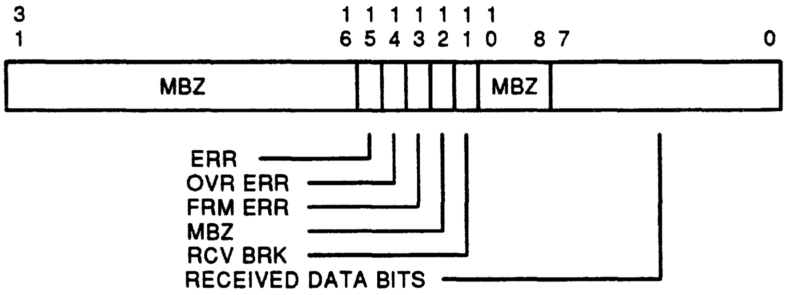


Figure 6–2 Console Receiver Data Buffer - (IPR 33_{10} 21_{16})

Table 6–3 Console Receiver Data Buffer

Data Bit	Name	Description
<31:16>	MBZ	These bits always read as zero. Writes have no effect.
<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.

Table 6–3 (Cont.) Console Receiver Data Buffer

Data Bit	Name	Description
<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.
<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. **Error conditions are updated when the character is received and it remains present until the character is read, at which point, the error bits are cleared.**
<12>	MBZ	This bit always reads as zero. Writes have no effect.
<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.
<10:8>	MBZ	These bits always read as zero. Writes have no effect.
<7:0>	Received Data Bits.	Read only. Writes have no effect. These bits contain the last received character.

6.1.3 Console Transmitter Control/Status Register - (IPR 34)

The console transmitter control/status register (TXCS), IPR 34, is used to control and report the status of outgoing data on the console serial line. The format is shown in Figure 6–3. Bit descriptions are listed in Table 6–4.

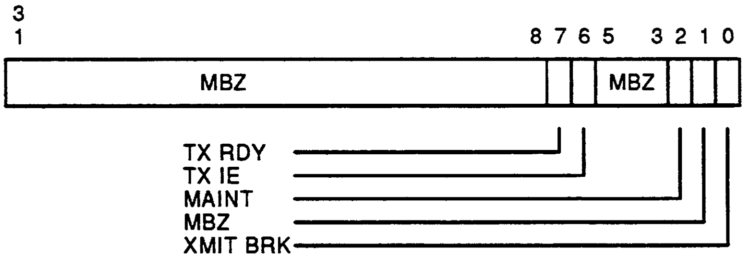


Figure 6–3 Console Transmitter Control/Status Register - (IPR 34_{10} 22_{16})

Table 6–4 Console Transmitter Control/Status Buffer

Data Bit	Name	Description
<31:8>	MBZ	These bits read as zero. Writes have no effect.
<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.
<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.
<5:3>	MBZ	Read as zeros. Writes have no effect.
<2>	MAINT	Maintenance. Read/Write. This bit is used to facilitate a maintenance self-test. When MAINT is set, the external serial output is set to MARK and the serial output is used as the serial input. This bit is cleared on power-up and the negation of DCOK.
<1>	Unused	This bit reads as zero. Writes have no effect.
<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 TXDB<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.

6.1.4 Console Transmitter Data Buffer - (IPR 35)

The console transmitter data buffer (TXDB), IPR 35, is used to buffer outgoing data on the serial line. The format is shown in Figure 6–4. Table 6–5 lists the bit descriptions.

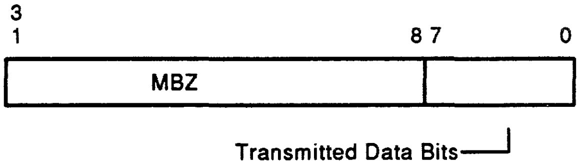


Figure 6–4 Console Transmitter Data Buffer - (IPR 35_{10} 23_{16})

Table 6–5 Console Transmitter Data Buffer

Data Bit	Name	Description
<31:8>	MBZ	Read as zero. Writes have no effect.
<7:0>	Transmitted Data Bits	Write only. These bits are used to load the character to be transmitted on the console serial line.

6.2 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 the KA660 will halt and transfer program control to EPROM location 2004 0000_{16} 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.

6.3 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 the baud rate select signals which are received from an external 8-position switch mounted on the console module (H3602). The KA660 firmware reads this code from boot and diagnostic register bits <6:4> and compliments and then loads it into SSC configuration register bits <14:12>.

Table 6–6 shows the baud rate selection, the corresponding code as read in the boot and diagnostic register bits <6:4>, and the INVERTED code that should be loaded into SSC configuration register bits <14:12>:

Table 6–6 Baud Rate Selection

Baud Rate	BDR<6:4>	SSC<14:12>
300	111	000
600	110	001
1200	101	010
2400	100	011
4800	011	100
9600	010	101
19200	001	110
38400	000	111

6.4 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}.

7
KA660 Clock and Timer Registers

The KA660 clocks include the time-of-year clock (TOY), a subset interval clock (subset ICCS), as defined in and two additional programmable timers modeled after the VAX Standard Interval Clock.

7.1 Time-of-Year Clock (TOY) - EPR 27

The KA660 time-of-year clock (TOY) forms an unsigned 32-bit binary counter that is driven from a 100 Hz oscillator, so that the least significant bit of the clock represents a resolution of 10 milliseconds, with less than .0025% error. The register counts only when it contains a non-zero value. This register is implemented in the SSC chip. The format is shown in Figure 7–1.

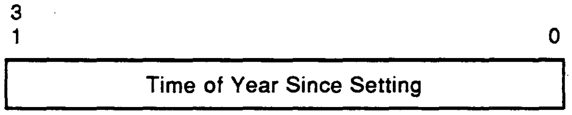


Figure 7–1 Time-of-Year Clock (TOY) - (EPR 27_{10} $1B_{16}$)

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 mounted on the CPU console module. The (TOY) remains valid for greater than 162 hours when using the NiCad battery pack (three batteries in a series) mounted on the H3602 cover panel.

The SSC configuration register contains a battery low (BLO) bit which, if set after initialization, the TOY 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 TOY, software should clear the BLO bit by writing a one to it.

7.2 Interval Timer (ICCS) - EPR 24

The KA660 interval timer (ICCS), IPR 24, is implemented according to The interval clock control/status register (ICCS), is implemented as the standard subset of the Standard VAX ICCS in the CPU chip; while NICR and ICR are not implemented. Figure 7–2 shows the format and Table 7–1 describes the bits.

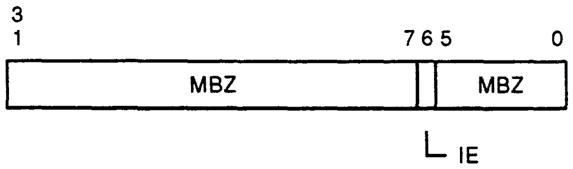


Figure 7–2 Interval Timer (ICCS) - (EPR 24_{10} 18_{16})

Table 7–1 Interval Timer Bit Descriptions

Data Bit	Name	Description
<31:7>	MBZ	Read as zero. Must be written as zero.
<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.
<5:0>	MBZ	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.

7.3 Programmable Timers

The KA660 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 IPRs) and a control bit has been added which stops the timer upon overflow. If so enabled, the timers 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
A timer **n** interrupt vector register

n represents the timer number (0 or 1).

7.3.1 Timer Control Registers (TCR0 and TCR1)

The KA660 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 chip. Figure 7–3 shows the format. Table 7–2 lists the bit descriptions.

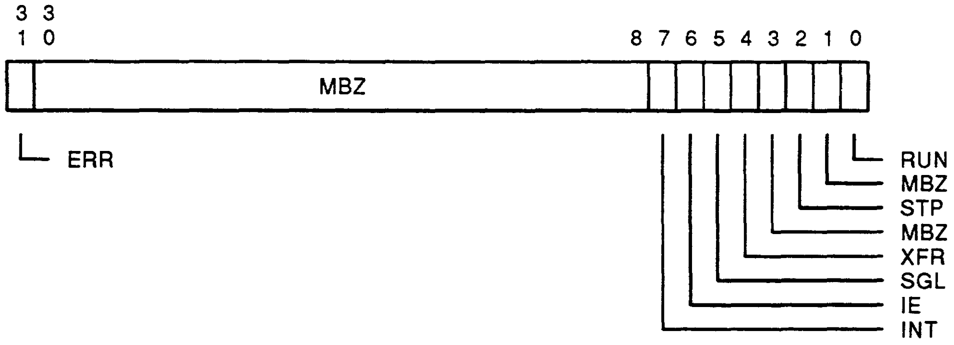


Figure 7–3 Timer Control Registers (TCR0 and TCR1)

Table 7–2 Timer Control Register Bit Descriptions

Date Bit	Name	Description
<31>	ERR	Error. Read/Write to clear. This bit is set whenever the timer interval register overflows and the INT bit is already set. Thus, the ERR bit indicates a missed overflow. Writing a one to this bit clears it. Cleared on power-up.
<30:8>	MBZ	Read as zero. Must be written as zero.
<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 one to this bit clears it. Cleared on power-up.
<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.
<5>	SGL	Read/Write. Setting this bit causes the timer interval register to be incremented by one if the RUN bit is cleared. If the RUN bit is set, then writes to the SGL bit are ignored. This bit is always read as zero. Cleared on power-up.
<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 zero. Cleared on power-up.
<3>	MBZ	Read as zeros, must be written as zeros.
<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.
<1>	MBZ	Read as zero. Must be written as zero.
<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.

7.3.2 Timer Interval Registers (TIR0 and TIR1)

The KA660 has two timer interval registers, one for timer zero (TIR0) and one for timer one (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 IPL14 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. Figure 7–4 shows the format.

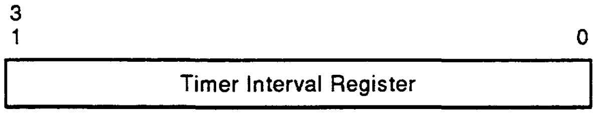


Figure 7–4 Timer Interval Registers (TIR0 and TIR1)

7.3.3 Timer Next Interval Registers (TNIR0 and TNIR1)

The KA660 has two timer next interval registers, one for timer zero (TNIR0), and one for timer one (TNIR1). TNIR0 is accessible at address 2014 0108_{16}, and TNIR1 is accessible at 2014 0118_{16}. These registers are implemented in the SSC chip. The format is shown in Figure 7–5.

The read/write register contains the value which is written into the timer interval register after overflow, or in response to a one written to the XFR bit. This register is cleared on power-up.

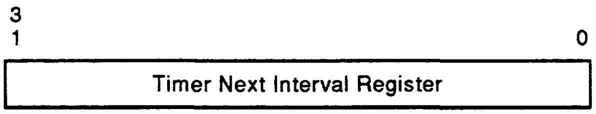


Figure 7–5 Timer Next Interval Registers (TNIR0 and TNIR1)

7.3.4 Timer Interrupt Vector Registers (TIVR0 and TIVR1)

The KA660 has two timer interrupt vector registers, one for timer zero (TIVR0), and one for timer one (TIVR1). TIVR0 is accessible at address 2014 $010C_{16}$, and TIVR1 is accessible at 2014 $011C_{16}$. These registers are implemented in the SSC chip and are set to 78_{16} and $7C_{16}$ respectively by the resident firmware. The format is shown in Figure 7–6.

The read/write register contains the timer's interrupt vector. Bits <31:10> and <1:0> are read as zero and must be written as zero. When TCRn<6> (IE) and TCRn<7> (INT) transition to one, 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 INT bit. This register is cleared on power-up.

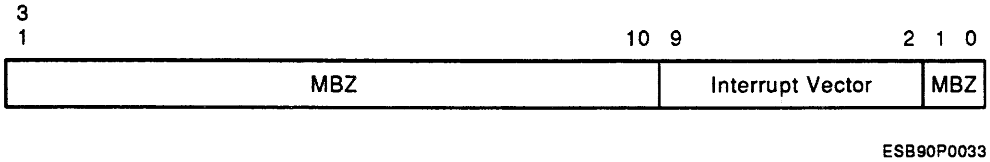


Figure 7–6 Timer Interrupt Vector Registers (TIVR0 and TIVR1)

NOTE

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.

8
KA660 Boot and Diagnostic Facility

The KA660 Boot and Diagnostic Facility features two registers, 256 Kbytes of erasable programmable ROM (EPROM) and 1 Kbyte of battery backed up RAM. The EPROM and battery backed up RAM may be accessed with longword, word, or byte references.

The 256 Kbytes of EPROM contain the resident firmware. If this EPROM is reprogrammed for special applications, the new code must initialize and configure the board, provide HALT and console emulation, as well as provide boot diagnostic functionality.

8.1 Boot and Diagnostic Register (BDR)

The boot and diagnostic register (BDR) is a byte-wide register repeated in the VAX I/O page at physical addresses 2008 4004 - 2008 407C$_{16}$. It is implemented uniquely on the KA660. It can be accessed by KA660 software, but not by external Q22-bus devices. The BDR allows the boot and diagnostic EPROM programs to read various KA660 configuration bits.

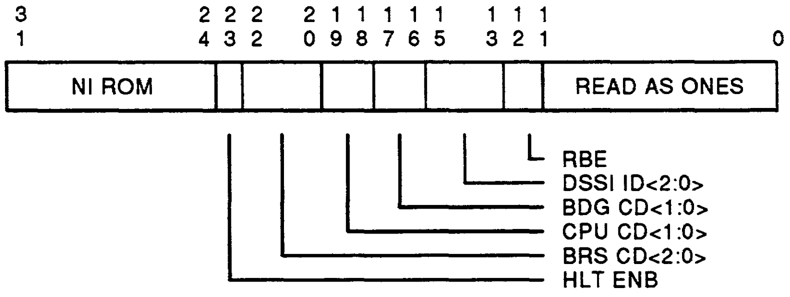


Figure 8–1 Boot and Diagnostic Register

Data Bit	Name	Description	Type
<31:24>	NI ROM	NI Station Address. This byte delivers the NI Station address in the next 32 longwords.	Read only

<table>
<tr><th>Data Bit</th><th>Name</th><th>Description</th><th>Type</th></tr>
<tr><td>&lt;23&gt;</td><td>HLT ENB</td><td>Halt enable. 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 KA660 resident firmware reads the HLT ENB bit to decide whether to enter the console emulation program (HLT ENB set) or to boot the operating system (HLT ENB clear). On the execution of of a HALT instruction while in kernal mode, the KA660 resident firmware reads the HLT ENB bit to decide whether to enter the console emulation program (HLT ENB set) or to restart the operating system (HLT ENB clear).</td><td>Read only. Writes have no effect.</td></tr>
<tr><td>&lt;22:20&gt;</td><td>BRS CD</td><td>Baud rate select. Writes have no effect. These three bits originate from pins <20:28> (BRS<2:0>) of the 40-pin connector. They reflect the setting of the the baud rate select switch on the CPU cover panel.
<table>
<tr><th>BDR&lt;6:4&gt;</th><th>Baud Rate</th></tr>
<tr><td>000</td><td>300</td></tr>
<tr><td>001</td><td>600</td></tr>
<tr><td>010</td><td>1200</td></tr>
<tr><td>011</td><td>2400</td></tr>
<tr><td>100</td><td>4800</td></tr>
<tr><td>101</td><td>9600</td></tr>
<tr><td>110</td><td>19200</td></tr>
<tr><td>111</td><td>38400</td></tr>
</table></td><td>Read only</td></tr>
<tr><td>&lt;19:18&gt;</td><td>CPU CD</td><td>CPU code. Writes have no effect. These two bits always read as zero because the KA660 cannot be configured as an auxiliary CPU.
<table>
<tr><th>CPU CD
&lt;1:0&gt;</th><th>Configuration</th></tr>
<tr><td>00</td><td>KA660-AA Arbiter</td></tr>
</table></td><td>Read Only</td></tr>
</table>

<table>
<tr><th>Data Bit</th><th>Name</th><th>Description</th><th>Type</th></tr>
<tr><td>&lt;17:16&gt;</td><td>BDG CD</td><td>Boot and diagnostic code. This 2-bit code reflects the status of configuration and display connector pins <37:36&gt; (BDG CD&lt;1:0>). The KA660 EPROM programs use BDG CD &lt;1:0> to determine the power up mode as follows:
<table>
<tr><th>BDR&lt;1:0&gt;</th><th>Power-up Mode</th></tr>
<tr><td>00</td><td>Run</td></tr>
<tr><td>01</td><td>Language Inquiry</td></tr>
<tr><td>10</td><td>Test</td></tr>
<tr><td>11</td><td>Manufacturing</td></tr>
</table>
</td><td>Read only. Writes have no effect.</td></tr>
<tr><td>&lt;15:13&gt;</td><td>BDG DSSIID</td><td>DSSI ID. Writes have no effect. This 3-bit code reflects the setting of the DSSI ID jumpers. This code must be decoded and written to the SHAC PPR register.</td><td>Read only</td></tr>
<tr><td>&lt;12&gt;</td><td>RBE</td><td>Remote boot enable.</td><td></td></tr>
<tr><td>&lt;11:0&gt;</td><td>Unused</td><td>Read as one</td><td></td></tr>
</table>

8.2 Diagnostic LED Register (DLEDR)

The diagnostic LED register (DLEDR), address 2014 0020_{16}, is implemented in the SSC chip and contains four read/write bits that control the external LED display. A zero in a bit lights the corresponding LED; all four bits are cleared on power-up and the negation of DCOK when SCR<7> is clear to provide a power-up lamp test. Figure 8–2 shows the register format. Table 8–3 lists the bit descriptions.

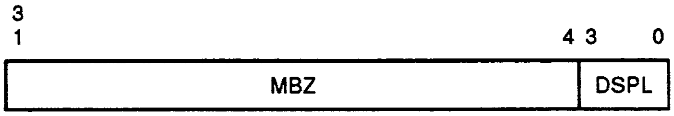


Figure 8–2 Diagnostic LED Register (DLEDR)

Table 8–3 Diagnostic LED Register Bit Descriptions

Data Bit	Name	Description
<31:4>	MBZ	Read as zeros, must be written as zeros.
<3:0>	DSPL	Display. Read/Write. These four bits update an external LED display. Writing a zero to a bit lights the corresponding LED. Writing a one to a bit turns the LED off. The display bits are cleared (all LEDs are lit) on power-up and the negation of DCOK.

8.3 EPROM Memory

The KA660 has 256 Kbyte of EPROM memory for storing code for board initialization, VAX standard console emulation, board self tests, and boot code. The EPROM memory may be accessed with byte, word and longword references. EPROM read accesses take 250 nanoseconds. The EPROM is organized as a 128K x 8-bit array. CDAL bus parity is neither checked nor generated on EPROM references.

NOTE
The EPROM size must be set in the SSC configuration register before attempting to reference outside the first 8 Kbyte block of the Local EPROM Space. (2004 0000 - 2004 1FFF$_{16}$

8.3.1 EPROM Address Space

The entire 256 Kbyte boot and diagnostic EPROM can only be read in the 256 Kbyte halt protect EPROM space (2004 0000 - 2007 FFFF$_{16}$).

Any I-stream read from the EPROM space places the KA660 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.

Writes and D-stream reads to any address space have no effect on run mode/halt mode status. *The KA660 logic that controls halt mode/run mode cannot detect I-stream read references that hit the cache; therefore halt mode/run mode is unaffected by these cache hits.*

8.3.2 KA660 Resident Firmware Operation

The KA660 CPU module's 256 Kbytes of EPROM contain the resident firmware, which can be entered by transferring program control to location 2004 0000$_{16}$.

lists the various halt conditions which cause the KA660 to transfer program control to location 2004 0000$_{16}$.

When running, the 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.

Refer to the KA660 console program specification for a complete description of these features.

8.3.2.1 Power-Up Modes

The boot and diagnostic EPROM programs use boot and diagnostic code <1:0> to determine the power-up modes shown in Table 8–4.)

Table 8–4 Power-Up Modes

Code	Power-up Mode
00	Run (factory setting). If the console terminal supports the Multi-National character set (MCS), the user will be prompted for a language if the time-of-year clock battery backup has failed, or if the SSC RAM is corrupted or uninitialized (1st power-up). Full startup diagnostics are run.
01	Language inquiry. If the console terminal supports MCS, the user is prompted for language on every power up and restart. Full startup diagnostics are run.
10	Test. UVROM programs run wrap-around serial line unit (SLU) tests.
11	Manufacturing. To provide for rapid startup during certain manufacturing test procedures, the EPROM programs omit the power-up memory diagnostics and set up the memory bit map on the assumption that all available memory is functional.

8.4 Battery Backed-Up RAM

The KA660 contains 1 Kbyte of battery backed-up static RAM found in the SSC,for use as a console "scratchpad." The power for the RAM is provided on pins 38 (VBAT H) and 20, 18, 16-13, 10-8, 6-3 (GND) of the 40-pin connector. This RAM supports byte, word, and longword references. Read operations take 700 nanoseconds to complete while write operations require 600 nanoseconds. The RAM is organized as a 256 X 32-bit (one longword) array. The array appears in a 1 Kbyte block of the VAX I/O page at addresses 2014 0400- 2014 $07FF_{16}$. This array is not protected by parity, and CDAL bus parity is neither checked nor generated on reads or writes to this RAM.

8.5 KA660 Initialization

The VAX Architecture defines three kinds of hardware initialization:

- Power-up initialization
- I/O bus initialization
- Processor initialization

8.5.1 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

Hardware Reset

A KA660 hardware reset occurs on power-up or the negation of DCOK. A hardware reset causes the hardware halt procedure (see Section 3.9.6) to be initiated with a halt code of 03. It also initializes some IPR's and most I/O Page registers to a known state. Those IPR's affected by a hardware reset are noted in Section 3.4. The effect a hardware reset has on I/O space registers is documented in the description of the register.

8.5.2 I/O Bus Initialization

An I/O bus initialization occurs on power-up, the negation of DCOK, or as the result of a MTPR to IPR 55 (IORESET). If the KA660 is an arbiter, an I/O bus initialization clears the IPCR and DSER, and causes the Q22-bus interface to acquire both the CDAL bus and Q22-bus, then assert the Q22-bus BINIT signal.

I/O Bus Reset Register (IPR 55)

The I/O bus reset register (IORESET), IPR 55_{10} is implemented in the SSC chip. If the KA660 is configured as an arbiter, an MTPR of any value to IORESET causes an I/O bus initialization.

8.5.3 Processor Initialization

A processor initialization occurs on power-up, the negation of DCOK, and after a halt caused by an error condition.

In addition to initializing those registers defined in the KA660 firmware must also configure main memory, the local I/O page, and the Q22-bus map during a processor initialization.

8.5.3.1 Configuring the Local I/O Page

The following registers control the configuration of the local I/O page. They are unique to CPU designs that use the SSC system support chip and they must be configured by the firmware during a processor initialization:

- SSC base address register
- BDR address decode match register
- BDR address decode mask register
- SSC configuration register
- CDAL bus timeout register

8.5.4 SSC Base Address Register (SSCBR)

The SSC base address register, address 2014 0000_{16}, controls the base addresses of a 2 Kbyte block of the local I/O space which includes the the following:

- Battery backed-up RAM
- The registers for the programmable timers
- The BDR address decode match and mask registers
- The diagnostic LED register
- The CDAL bus timeout register
- A set of diagnostic registers that allow several EPRs to be accessed via I/O page addresses

This read/write register is set to 2014 0000_{16} on power-up and the negation of DCOK when SCR <7> is clear. Bits SSCBR<31:30,10:0> are unused. They read as zero and must be written as zero. SSCBR<29> is read as one and must be written as one. This register should also be set to 2014 0000_{16} by firmware during processor initialization. The SSCBR has the format shown in Figure 8–3.

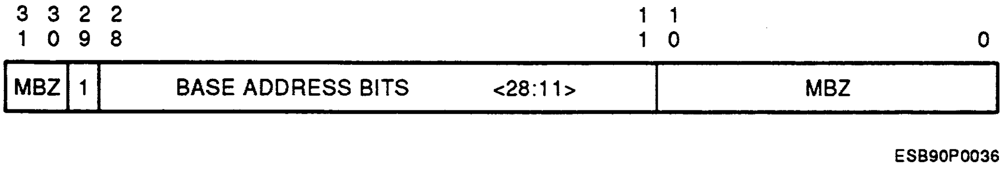


Figure 8–3 SSC Base Address Register (SSCBR)

8.5.5 BDR Address Decode Match Register (BDMTR)

The local I/O device address decode match register, address 2014 0130_{16}, controls the base address of the boot and diagnostic register. This read/write register is cleared on power-up and the negation of DCOK. BDMTR<31:30,1:0> are unused. They read as zero and must be written as zerio. This register should be set to 2008 4000_{16} by firmware during processor initialization. The BDMTR has the format shown in Figure 8–4.

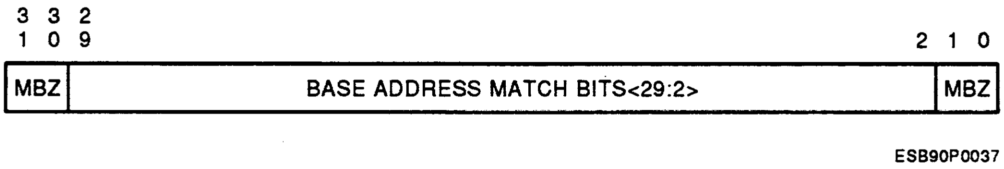


Figure 8–4 BDR Address Decode Match Register (BDMTR)

8.5.6 BDR Address Decode Mask Register (BDMKR)

The BDR address decode mask register, address 2014 0134_{16}, controls the range of addresses to which the BDR responds to. (An example is the number of copies of the BDR that appear in the physical address space.) This read/write register is cleared on power-up and the negation of DCOK. Bits BDMKR<31:30,1:0> are unused. They read as zero and must be written as zero. This register should should be set to 0000 007F $_{16}$ (32 copies of the BDR) by firmware during processor initialization . The BDMKR has the format shown in Figure 8–5.

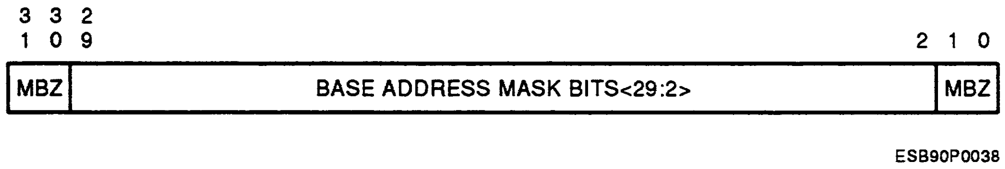


Figure 8–5 BDR Address Decode Mask Register (BDMKR)

NOTE
The KA660 uses only one of the SSC's address strobes. The other strobe's control registers located at 2014 0130_{16} and 2014 0134_{16} are reserved and should not be accessed as they could cause unpredictable behavior.

8.5.7 SSC Configuration Register (SSCCR)

The SSC configuration register, address 2014 0010_{16}, controls the set-up parameters for the console serial line, programmable timers, EPROM, TOY Clock and BDR. The format is shown in Figure 8–6. Table 8–5 contains a list of the bit descriptions.

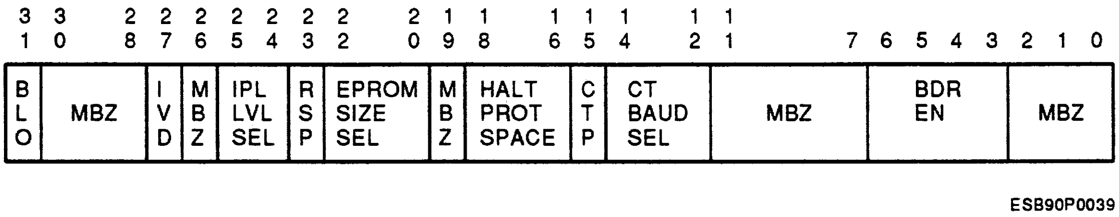


Figure 8–6 SSC Configuration Register (SSCCR)

Table 8–5 SSC Configuration Register Bit Descriptions

Data Bit	Name	Description
<31>	BLO	Battery low. Read/Write. If the battery voltage goes below threshold while the module is powered down, this bit is set on power-up, after the assertion of DCOK after the assertion of POK. Once set, this bit can only be cleared by software writing it as one. If this bit is set, then the TOY clock is cleared by power-up and by the negation of DCOK.
<30:28>	MBZ	Read as zero. must be written as zero.
<27>	IVD	Interrupt vector disable. Read/Write. When set, the console serial line and programmable timers will not respond to interrupt acknowledge cycles. Cleared on power-up, by the negation of DCOK, and by a processor initialization.
<26>	MBZ	Read as zero. Must be written as zero
<25:24>	IPL_ LVL_SEL	IPL Level select. Read/Write. These bits are used to specify the IPL level of interrupt acknowledge cycle that the console serial line and programmable timers respond to. These bits must be cleared (programmed to 00 (binary)) for the console serial line and programmable timers to respond to interrupt acknowledge cycles that they generated (IPL 14). These bits are cleared on power-up, by the negation of DCOK and by a processor initialization.
<23>	RSP	ROM Speed. Read/Write. This bit is used to select the EPROM access time. This bit must be set for the KA660 EPROMs to run at maximum speed. This bit is cleared on power-up and by the negation of DCOK. It must be set to ONE by a processor initialization.

Table 8–5 (Cont.) SSC Configuration Register Bit Descriptions

Data Bit	Name	Description
<22:20>	ROM_ SIZE_ SEL	EPROM Address space size select. Read/Write. These bits control the size of the range of addresses to which the EPROM responds. These bits must be set to 101 (binary) because the KA660 contains 256KB of EPROM, yielding an address range of 256 Kbyte (2004 0000 - 2007 $FFFF_{16}$). These bits are cleared on power-up and by the negation of DCOK, yielding an address range of 8 Kbyte (2004 0000 - 2004 $1FFF_{16}$). These bits must be set to the proper value by a processor initialization.
<18:16>	HALT PROT SPACE	EPROM Halt protect address space size select. Read/Write. These bits control the size of the halt mode address range. These bits must be set to 101 (binary) because the KA660 contains 256 Kbyte of EPROM, yielding a halt mode address range of 256 Kbyte (2004 0000 - 2007 $FFFF_{16}$). These bits are cleared on power-up and by the negation of DCOK yielding a halt mode address range of 8 Kbyte (2004 0000 - 2004 $1FFF_{16}$). These bits must be set to the proper value by a processor initialization. Note, any instruction fetch from the EPROM puts the KA660 in HALT protect mode.
<15>	CTP	Control P enable. Read/Write. When this bit is set, a CTRL/P typed at the console causes the CPU to be halted, if halts are enabled (BDR<7> set). When this bit is cleared, a BREAK typed at the console causes the CPU to be halted, if halts are enabled (BDR<7> set). This bit is cleared on power-up and by the negation of DCOK.
<14:12>	CT BAUD SELECT	Console terminal baud rate select. Read/Write. These bits are used to select the baud rate of the console terminal serial line. They are cleared on power-up and by the negation of DCOK. They should be loaded from compliment of BDR<6:4> by the processor initialization code. The bit encodings correspond to selected baud rates as shown in the following table.

SSCCR<14:12>	Baud Rate
000	300
001	600
010	1200
011	2400
100	4800
101	9600
110	19200
111	38400

Data Bit	Name	Description
<11:7>	MBZ	Read as zero. Must be written as zero.
<6:4>	BDR EN	Read/Write. These bits are used to enable the boot and diagnostic register. They are cleared on power-up and by the negation of DCOK. These bits must be set to 111 (binary) by a processor initialization to enable the BDR.
<3:0>	MBZ	Read as zero. Must be written as zero.

NOTE
The SSC baud clock runs about 1.7% fast, which is within the SRM mandated accuracy. This is due to the accuracy of the crystal oscillator.

8.6 CDAL Bus Timeout Control Register (CBTCR)

The CDAL bus timeout register, address 2014 0020 $_{16}$, controls the amount of time allowed to elapse before a CDAL bus cycle is aborted. The effect of this timer is blocked by the KA660 logic on all Q22-bus references, because the Q22-bus interface has its own timers for Q22-bus references. This timer prevents unanswered CDAL bus cycles (other than those that go to the Q22-bus interface) from hanging the system any longer than the timeout interval. **Even though the effect of the timer is blocked on all Q22-bus references, bits<31:30> are still set on Q22-bus references that take longer than the programmmed value (4us), so these bits are not useful as error indicators.** Figure 8–7 shows the format. Table 8–6 lists the bit descriptions.

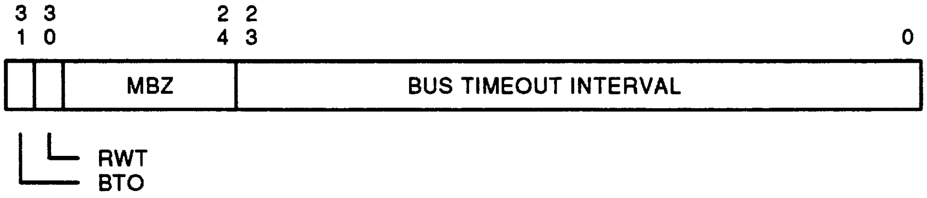


Figure 8–7 CP Bus Timeout Control Register (CBTCR)

Table 8–6 CP Bus Timeout Control Register Bit Descriptions

Data Bit	Name	Description
<31>	BTO	CP Bus timeout. Read/Write to clear. This bit is set when the BUS TIMEOUT INTERVAL set in bits <23:0> has expired during any CP bus cycle. This bit is cleared by writing a one, on power-up, and the negation of DCOK.
<30>	RWT	CP Bus read/write timeout. Read/Write to clear. This bit is set when the BUS TIMEOUT INTERVAL set in bits <23:0> has expired during a CPU or DMA read or write cycle on the CP bus. This bit is cleared by writing a one, on power-up, and the negation of DCOK.
<29:22>	MBZ	Read as zero. Must be written as zero.
<23:0>	BUS TIMEOUT INTERVAL	Read/Write. These bits are used to program the desired timeout period. The available range of 1 to FFFFFF$_{16}$ corresponds to a selectable timeout range of 1µs to 16.77 seconds in 1µs increments. Writing a zero to this field disables the bus timeout function. The BTO bit is used to signify that a bus timeout has occurred. This field is cleared on power-up and the negation of DCOK. This register should be loaded with 0000 4000$_{16}$ on a processor initialization for a timeout value of 15 milliseconds.

9
KA660 Q22-bus Interface

The KA660 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:

- A programmable mapping function (scatter-gather map) for translating 22-bit, Q22-bus addresses into 29-bit CDAL 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 Mbytes per sec. 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 cycles. The maximum read bandwidth for Q22-bus block mode references is 2.4 Mbytes per sec. 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, interprocessor communication, 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, device parity errors.

9.1 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 (Figure 9–1.) 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 1024K pages in main memory. Since local I/O space addresses cannot be mapped to Q22-bus pages, the local I/O page is unaccessible to devices on the Q22-bus. Figure 9–1 shows how Q22-bus addresses are translated into main memory addresses.

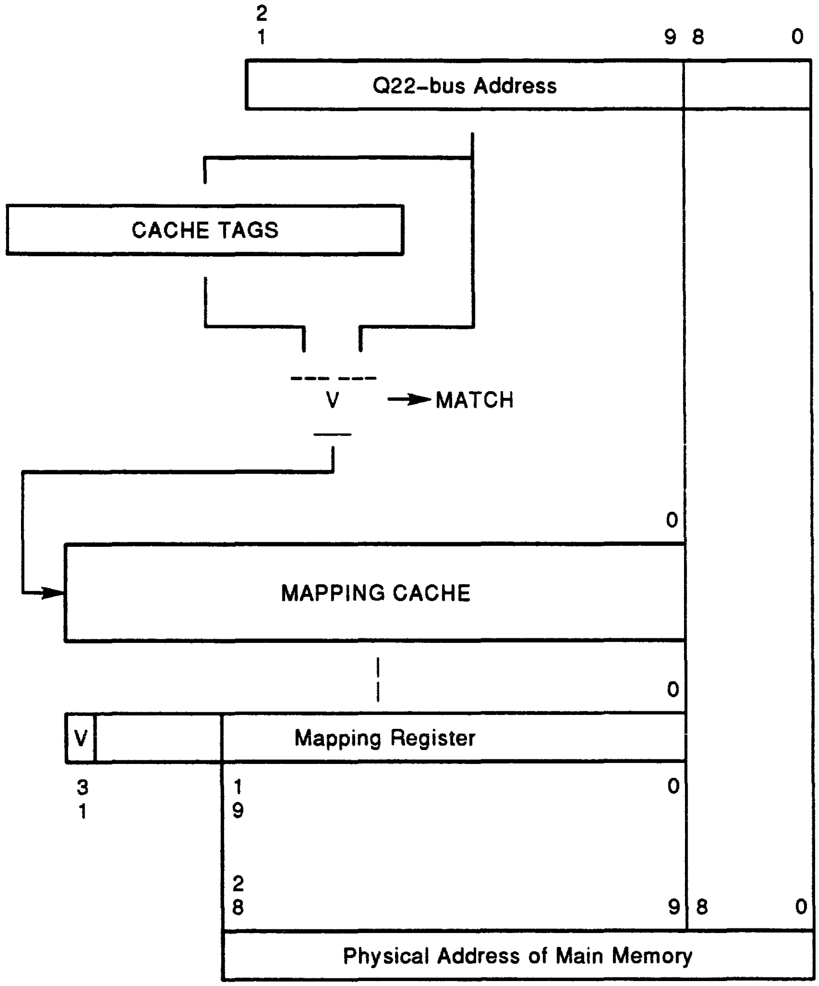


Figure 9–1 Q22-bus Address Translation

At power up time, the Q22-bus map registers, including the valid bits, are undefined. External access to main memory is disabled so 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:

- The interprocessor communication register LM EAE bit is set.
- The valid bit of the selected mapping register is set.
- 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 thhe two previous conditions.

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 asynchronous DMA read of the Q22-bus map register before proceeding with the Q22-bus DMA transfer.

9.1.1 Q22-bus Map Registers (QMRs)

The Q22-bus map contains 8192 registers 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. These registers are implemented in a 32 Kbyte 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. Table 9–1 lists the register addresses.

Table 9–1 Q22-bus Map Register Addresses

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 FA00 - 3F FFFF	17 776 000 - 17 777 777

The Q22-bus map registers (QMR) have the format shown in Figure 9–2.

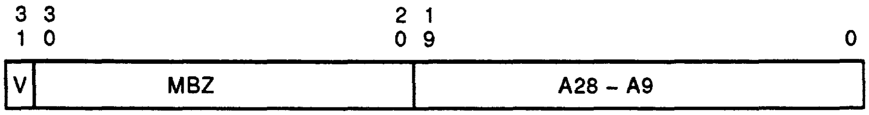


Figure 9–2 Q22-bus Map Register Format

Table 9–2 describes the bits in the Q22-bus map register.

Table 9–2 Q22-bus Map Register Bit Description

Data Bit	Name	Description
<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.
<30:20>	Unused	These bits always read as zero and must be written as zero.
<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. 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.

9.1.2 Accessing the Q22-bus Map Registers

Although the CPU accesses the Q22-bus map registers with aligned longword references to the local I/O page (addresses 2008 8000 - 2008 FFFC $_{16}$), the map actually resides in a 32 KByte 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. Because 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 $FFFC_{16}$). 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 and 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.

9.1.3 The 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. Figure 9–3 shows the format. Table 9–3 contains a description of the Q22-bus map cache entry bits.

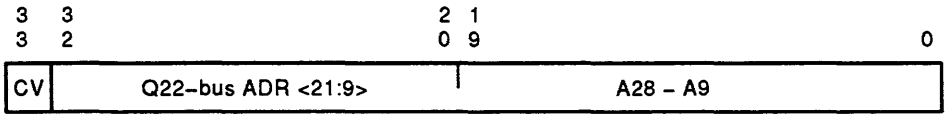


Figure 9–3 Q22-bus Map Cache Entry Format

Table 9–3 Q22-bus Map Cache Entry Bit Description

Data Bit	Name	Description
<33>	CAM Valid	When a mapping register is selected by a Q22-bus address, the CAMValid bit determines whether the cached copy of the mapping register for that address is valid. If the CAMValid bit is set, the mapping register is enabled, and addresses within that page can be mapped. If the CAMValid 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, 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.
<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.
<19:0>	Address bits A28-A9	When a mapping register is selected by a Q22-bus address, and if that register's CAMValid 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.

9.2 CDAL to Q22-bus Address Translation

CDAL bus addresses within the CDAL Translation 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 bus 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 bus address as bits <21:0> of the Q22-bus address.

9.3 Interprocessor Communications Facility

The KA660 can only be configured as a Q22-bus arbiter.

The KA660 interprocessor communication facility allows other processors on the system to request program interrupts from the KA660 without using the Q22-bus interrupt request lines. It also controls external access to local memory (via the Q22-bus map).

9.3.1 Interprocessor Communication Register (IPCR)

The interprocessor communication register (IPCR) is 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 KA660 itself). The IPCR is implemented in the CQBIC chip and is byte accessible, meaning that a write byte instruction can write to either the low or high byte without affecting the other byte. The I/O page address of the IPCR is constant with the KA660 because it only supports arbiter mode and not auxiliary mode. The hex 32-bit address is 2000 1F40 and the octal 22-bit address is 17 777 500. Figure 9–4 shows the format. Table 9–4 describes the bits.

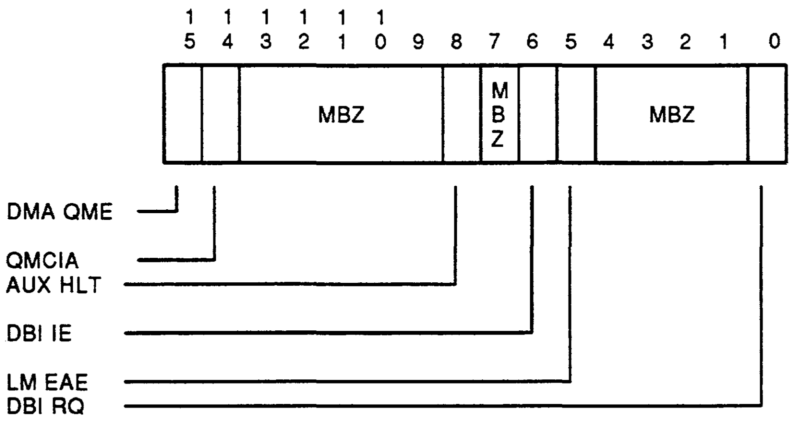


Figure 9–4 Interprocessor Communication Register (IPCR)

Table 9–4 Interprocessor Communication Register Bit Description

Data Bit	Name	Description
<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 timer expires. The MAIN MEMORY ERROR bit indicates that an uncorrectable error occurred when an external device (or CPU) was accessing the KA660 local memory. The CDAL 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 one to it on power-up, by the negation of DCOK, by writes to IPR 55 (IORESET), and whenever DSER<4> is cleared.
<14>	QMCIA	Q22-bus map cache invalidate all. Write only. Writing a one to this bit clears the CAMValid bits in the cached copy of the MAP. This bit always reads as zero. Writing a zero has no effect.
<13:09>	Unused	Read as zeros. Must be written as zeros.
<8>	AUX HLT	Auxiliary halt. Read only. When this bit is set it has no effect on the operation of the on-board CPU. This bit is cleared on power-up, by the negation of DCOK, by writes to IPR 55 (IORESET). **Note: This bit should never be set because the processor does not support auxiliary mode.**
<7>	Unused	Read as zero. Must be written as zero.
<6>	DBI IE	Doorbell interrupt enable. Read/Write when the KA660 is Q22-bus master. Read only when another device is Q22-bus master. When set, this bit enables interprocessor doorbell interrupt requests via IPCR<0>. Cleared on power-up, by the negation of DCOK, and writes to IPR 55 (IORESET).

Table 9–4 (Cont.) Interprocessor Communication Register Bit Description

Data Bit	Name	Description
<5>	LM EAE	Local Memory External Access Enable. Read/Write when the KA660 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.
<4:1>	Unused	Read as zeros. Must be written as zeros.
<0>	DBI RQ	Doorbell interrupt request. Read/Write. If IPCR<6> (DBI IE) is set, setting this bit generates a doorbell interrupt request. If IPCR<6> is clear, setting this bit has no effect. Clearing this bit has no effect. DBI RQ is cleared when the CPU grants the doorbell interrupt request. DBI RQ is held clear whenever DBI IE is clear. This bit is cleared on power-up and the negation of DCOK.

9.3.2 Interprocessor Doorbell Interrupts

If the interprocessor communication register DBI IE bit is set, any Q22-bus master can request an interprocessor doorbell interrupt by writing a one into IPCR bit <0>.

The interrupt vector is 204_{16} and the interrupt priority is 14_{16}. This IPL is the same as BR4 on the Q22-bus. The interprocessor doorbell is the third highest priority IPL 14 device, directly after the console serial line unit and the programmable timers.

NOTE
Following an interprocessor doorbell interrupt, the KA660 CPU sets the IPL to 14. The IPL is set to 17 for external Q22-bus BR4 interrupts.

9.4 Q22-bus Interrupt Handling

The KA660 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 interrupt 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 KA660 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.

9.5 Configuring the Q22-bus Map

The KA660 implements the Q22-bus map in an 8K longword (32 Kbyte) block of main memory. This map must be configured by the KA660 firmware during a processor initialization by writing the base address of the uppermost 32 Kbyte block of good main memory into the Q22-bus map base register. The base of this map must be located on a 32 Kbyte boundary.

NOTE
This 32 Kbyte 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 $_{16}$.

9.5.1 Q22-bus Map Base Address Register (QBMBR)

The Q22-bus map base address register, address 2008 0010 $_{16}$ controls the main memory location of the 32 Kbyte 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. Figure 9–5 shows the format.

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

The contents of this register are undefined on power-up and the negation of DCOK when SCR<7> is clear and is not affected by BINIT being asserted on the Q22-bus.

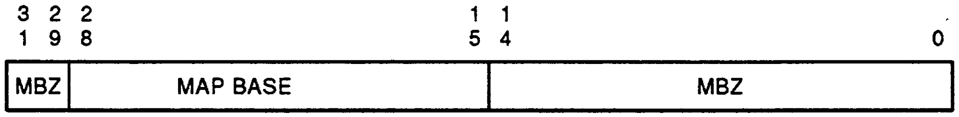


Figure 9–5 Q22-bus Map Base Address Register (QBMBR)

9.6 System Configuration Register (SCR)

The system configuration register (SCR), address 2008 0000$_{16}$, contains the processor number which determines the address of the IPCR register, a BHALT enable bit, a power OK flag and an AUX flag. Figure 9–6 shows the format. Table 9–5 describes the bits in the system configuration register.

The SCR is longword, word, and byte accessible. Programmable option fields are cleared on power-up and by the negation of DCOK when SCR<7> is clear.

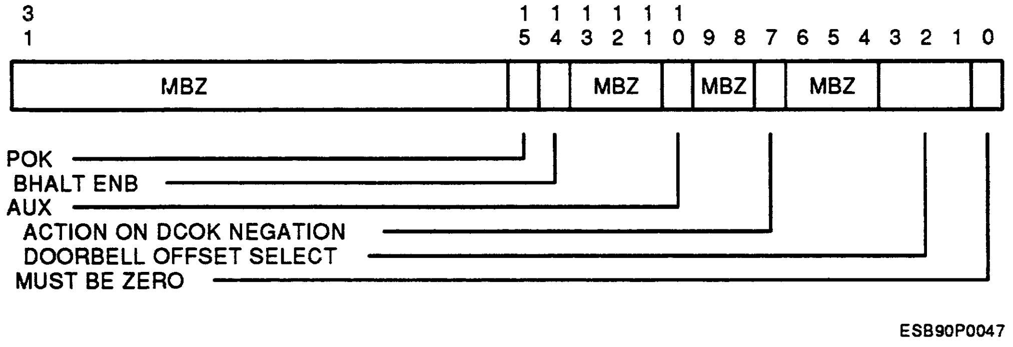


Figure 9–6 System Configuration Register (SCR)

Table 9–5 System Configuration Register Bit Description

Data Bit	Name	Description
<31:16>	Unused	Read as zero. Must be written as zero.
<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.
<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.
<13:11>	Unused	Read as zero. Must be written as zero.
<10>	AUX	Auxiliary. Read only. Writes have no effect. This bit defines auxiliary and arbiter mode of operation of the KA660 When read as a zero, Arbiter mode is selected, and when read as a one, auxiliary mode is selected. Because the KA660 can only be configured as an arbiter this bit should always read as zero.
<9:8>	Unused	Read as zero. Must be written as zero.
<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 HALCYON (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.
<6:4>	Unused	Read as zero. Must be written as zero.
<0>	Unused	Read as zero. Must be written as zero.

9.7 Error Reporting Registers

There are three registers associated with Q22-bus interface error reporting:

- The DMA System error register (DSER)
- The Q22-bus error address register (QBEAR)
- The DMA Error address register (DEAR)

These registers are located in the local VAX I/O address space and can only be accessed by the local processor. The DSER is implemented in the CQBIC chip and it logs main memory errors on DMA transfers, Q22-bus parity errors, Q22-bus non-existent memory errors, and Q22-bus no-grant. The QBEAR contains the address of the page in Q22-bus space which caused a parity error during an access by the local processor. The DEAR 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 provides 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.

9.7.1 DMA System Error Register (DSER)

The DMA System Error Register (DSER) (address 2008 0004_{16}) is a longword, word, or byte accessible read/write register available to the local processor. The bits in this register are cleared to zero on power-up, by the negation of DCOK when SCR <7> is clear, and by writes to IPR 55 (IORESET). All bits are set to one to record the occurrence of an event. They are cleared by writing a 1; writing zeros has no effect.

The format of the DSER is shown in Figure 9–7. Table 9–6 describes the bits in the system error register.

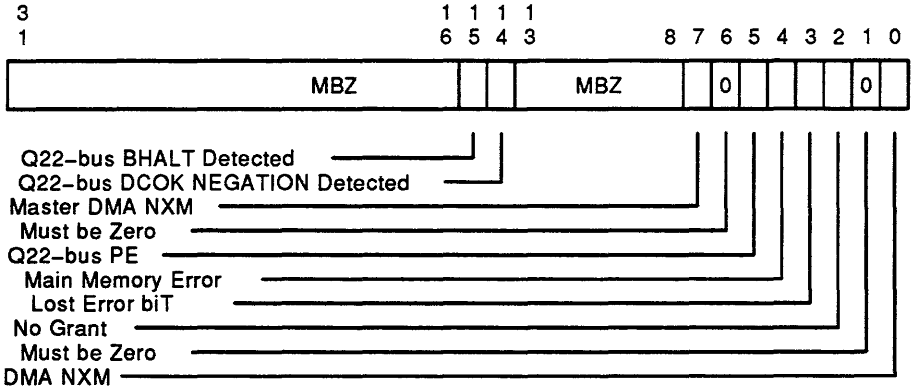


Figure 9–7 DMA System Error Register (DSER)

Table 9–6 DMA System Error Register Bit Description

Data Bit	Name	Description
<31:16>	Unused	Read as zero. Must be written as zero.
<15>	Q22-bus BHALT DETECTED	Read/Write to clear. This bit is 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 one, writes to IPR 55 (IORESET), on power-up, and the negation of DCOK.
<14>	Q22-bus DCOK NEGATION DETECTED	Read/Write to clear. This bit is 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 one, writes to IPR 55 (IORESET), on power-up, and the negation of DCOK.
<13:8>	Unused	Read as zero. Must be written as zero.
<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 10us. During interrupt acknowledge cycles, or request read cycles, this bit is not set. Cleared by writing a one, on power-up, by the negation of DCOK, and by writes to IPR 55 (IORESET).
<6>	Unused	Read as zero. Must be written as zero.

Table 9–6 (Cont.) DMA System Error Register Bit Description

Data Bit	Name	Description
<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. Cleared by writing a one, on power-up, by the negation of DCOK, and by writes to IPR 55 (IORESET).
<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. Cleared by writing a one, on power-up, by the negation of DCOK, and by writes to IPR 55 (IORESET).
<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. Cleared by writing a one, on power-up, by the negation of DCOK, and by writes to IPR 55 (IORESET).
<2>	NO GRANT TIMEOUT	Read/Write to clear. This bit is set if the Q22-bus does not return a bus grant within 10 milliseconds 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. Cleared by writing a one, on power-up, by the negation of DCOK, and by writes to IPR 55 (IORESET).
<1>	Unused	Read as zero. Must be written as zero.
<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. Cleared by writing a one, on power-up, by the negation of DCOK when SCR<7> is clear, and by writes to IPR 55 (IORESET).

9.7.2 Q22-bus Error Address Register (QBEAR)

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> (MASTER DMA NXM) is set.

Reading this register when DSER<5> and DSER<7> are clear returns 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, causes, 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.

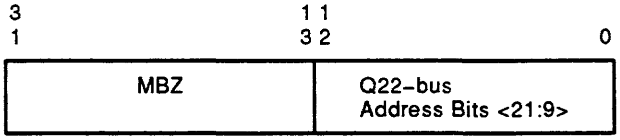


Figure 9–8 Q22-bus Error Address Register (QBEAR)

NOTE
This is a read only register. If a write is attempted a hard error (IPL 1D) is generated.

9.7.3 DMA Error Address Register (DEAR)

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. Figure 9–9 shows the format.

The DBEAR 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 > are set. Additional main memory errors or non-existent memory errors have no effect on the DBEAR until software clears DSER<4> and DSER<0> .

Mapped Q22-bus address bits <28:9> are loaded into DBEAR bits <19:0>. DBEAR bits <31:20> always read as zeros.

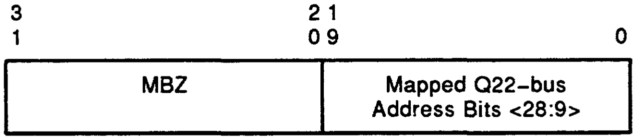


Figure 9–9 DMA Error Address Register (DBEAR)

NOTE
This is a read only register. If a write is attempted a hard error (IPL 1D) is generated.

9.8 Q22-bus Interface Error Handling

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

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

The Q22-bus interface maintains several timers to prevent incomplete accesses from hanging the system indefinitely. They include: a 10 microsecond non-existent memory timer for accesses to the Q22-bus memory and I/O spaces, a 10µs "NO SACK" timer for acknowledgment of Q22-bus DMA grants, and a 10 millisecond "NO GRANT" timer for acquiring the Q22-bus.

If there is a non-existent memory (NXM) error (10 microsecond timeout) while accessing the Q22-bus on a demand read reference, the associated row in the cache is invalidated, bit 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 an 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 an 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 acknowledgment within 10 microseconds after it has granted the Q22-bus, the grant is withdrawn, no errors are reported, and the Q22-bus interface waits 500 nanoseconds 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 the 10 milliseconds, 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 which is being read 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.

10
KA660 Network Interface

The KA660 includes a network interface that is implemented using the second generation ethernet controller chip (SGEC). When used in conjunction with the H3602 cover panel, this interface allows the KA660 to be connected to either a ThinWire or standard Ethernet network. It supports the Ethernet data link layer as specified in the *VAX Architecture Reference Manual.* The SGEC also supports CP bus parity protection.

10.1 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 10–1.

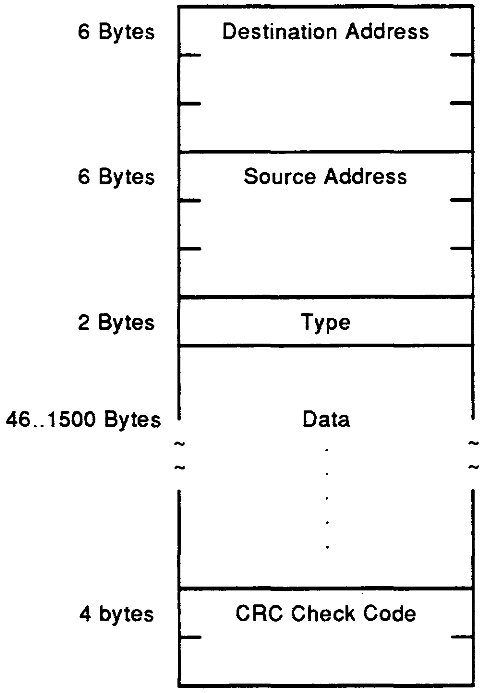


Figure 10–1 Ethernet Packet Format

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 next.

- *Physical address:*

  This is a unique address associated with a particular station on the Ethernet. It should be distinct from the physical address of any other station on any other Ethernet.

- *Multicast address:*

  This is a multi-destination address associated with one or more stations on a given Ethernet (sometimes called a logical address). There are two kinds of multicast addresses:

— *Multicast-group address:*

  An address associated by higher-level convention with a group of logically related stations.

— *Broadcast address:*

  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 KA660 module is determined at the time of manufacture and is stored in the network interface station address ROM. Because every device that is intended to connect to an Ethernet network must have a unique physical address, the bit pattern blasted into the network interface station address ROM must be unique for each KA660. The multicast addresses to which the KA660 responds are determined by the multicast address filter Mask in the network interface initialization block.

10.2 NI Station Address ROM (NISA ROM)

The network interface includes a byte-wide, 32-byte, socketted ROM called the Network Interface Station Address ROM (NISA ROM). One byte of this ROM appears in the second byte of each of 32 consecutive longwords in the address range 2008 4000 - 2008 $407C_{16}$. Bytes one, three and four of each longword are defined in the boot diagnostic register section 9.1. The second byte of the first six longwords contain the 48-bit network physical address (NPA) of the KA660 . The low-order byte in the remaining 26 longwords are used for testing. This address range is read only. Writes to this address range results in a CP bus timeout and a machine check.

10.3 Programming the SGEC

The operation of the SGEC is controlled by a program in host memory called the port driver. The SGEC and the port driver communicate through two data structures: Network Interface Command and Status Registers (NICSRs) located in the SGEC and mapped in the host I/O address space, and through descriptor lists and data buffers, collectively called the host communication area , in host memory.

The NICSRs are used for initialization, global pointers, commands, and global error reporting, while the host memory resident structures handle the actions and statuses related to buffer management.

The SGEC can be viewed as two independent, concurrently executing processes: Reception and transmission. After the SGEC completes its initialization sequence, these two processes alternate between three states: stopped, running or suspended. State transitions occur as a result of port driver commands (writing to a NICSR) or various external event occurrences. Some of the port driver commands require the referenced process to be in a specific state.

A simple programming sequence of the chip may be summarized as follows:

1. After power on (or reset), verifying that the self-test completed successfully.
2. Writing NICSRs to set major parameters such as system base register, interrupt vector, address filtering mode and so on.
3. Creating the transmit and receive lists in memory and writing the NICSRs to identify them to the SGEC.
4. Placing a setup frame in the transmit list to load the internal reception address filtering table.
5. Starting the reception and transmission processes placing them in the running state.
6. Waiting for SGEC interrupts. NICSR5 contains all the global interrupt status bits.
7. Remedying the suspension cause, if the reception or transmission processes enter the suspended state.
8. Issuing a Tx Poll Demand command, to return the transmission process to the running state. In addition to remedy the reception process suspension cause, a Rx poll demand could be issued to return the Reception process to the running state.

   If the Rx poll demand is not issued, the reception process returns to the running state when the SGEC receives the next recognized incoming frame.

The following sections contain detailed programming and state transitions information.

10.3.1 Command and Status Registers

The SGEC contains 16 command and status registers which may be accessed by the host.

10.3.2 Host Access to NICSRs

The SGEC's NICSRs are located in VAX I/O address space.

The NICSRs must be longword aligned and can only be accessed using longword instructions. The address of NICSRx is the base address plus 4x bytes. For example, if the base address is 2000 8000, then the address of NICSR2 is 2000 8008. In the following paragraphs, NICSRs bits are specified with several access modes. The different access modes for bits are as follows:

Table 10–1 Bit Access Modes

Bit Marked	Meaning
0	Reserved for future expansion. Ignored on write. Read as 0.
1	Reserved for future expansion. Ignored on write. Read as 1.
R	Read only. Ignored on write.
R/W	Read or write.
W	Write only. Unpredictable on read.
R/W1	Read or clear by writing a 1. Writing a 0 has no effect.

In order to save chip real estate yet not tie up the host bus for extended periods of time, the 16 NICSRs are subdivided into two groups:

1. Physical NICSRs - 0 through 7, 15.
2. Virtual NICSRs - 8 through 14.

The group, of which the NICSR is part determines the way the host accesses NILSR.

10.3.2.1 Physical NICSRs

These registers are physically present in the chip. Host access to these NICSRs is by a single instruction (for example, MOVL). There is no host perceivable delay and the instruction completes immediately. Most commonly used SGEC features are contained in the physical NICSRs.

10.3.2.2 Virtual NICSRs

These registers are not physically present in the SGEC and are incarnated by the on-chip processor. Accesses to SGEC functions implied by these registers can take up to 20 µseconds. To avoid tying up the host bus, virtual NICSR access requires several steps by the host.

NICSR5<DN> is used to synchronize access to the virtual NICSRs; after the first virtual NICSR access, the SGEC de-asserts NICSR5<DN> until it completes the action.

NOTE

Accessing the virtual NICSRs without polling first on the NICSR5<DN> reassertion causes unpredictable results.

10.3.2.2.1 NICSR Write

To write to a virtual NICSR the host takes the following actions:

1. Issue a write NICSR instruction. The instruction completes immediately, but the data is not yet copied by the SGEC.
2. Wait for NICSR5<DN>. *No SGEC virtual NICSR may be accessed before NICSR5<DN> asserts.*

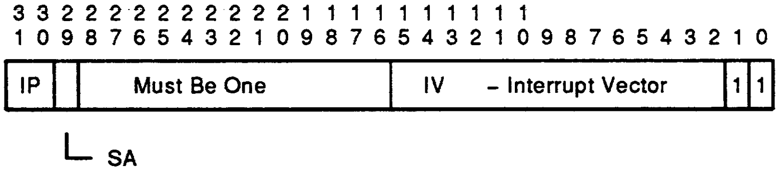
10.3.2.2.2 NICSR Read

To read a virtual NICSR the host takes the following actions:

1. Issue a read NICSR instruction. The instruction completes immediately, but no valid data is sent to the host.
2. Wait for NICSR5<DN>. *No SGEC virtual NICSR may be accessed before NICSR5<DN> asserts.*
3. Reissue a read NICSR instruction, to the *same* NICSR as in step 1. The host receives valid data.

10.3.3 Vector Address, IPL, Sync/Asynch (NICSR0)

Because the SGEC could generate an interrupt on parity errors, during host writes to NICSR's this register must be the first one written by the host. The format is shown in Figure 10–2 and the bit description is given in Table 10–2.



I/O Address: 2000 8000 $_{(16)}$

Longword Read/Write Access

Figure 10–2 Vector Address, IPL, Sync/Asynch (NICSR0)

NOTE

A parity error during NICSR0 host write may cause a host system crash due to an erroneous interrupt vector. To protect against this NICSR0 must be written as follows while the IPL to which the SGEC is assigned is disabled:

1. **Write NICSR0.**
2. **Read NICSR0.**
3. **Compare value read to value written. If the values mismatch, repeat from step 1.**
4. **Read NICSR5 and examine NICSR5<ME> for pending parity interrupt. If an interrupt be pending, write NICSR5 to clear it.**

Table 10–2 NICSR0 Bits

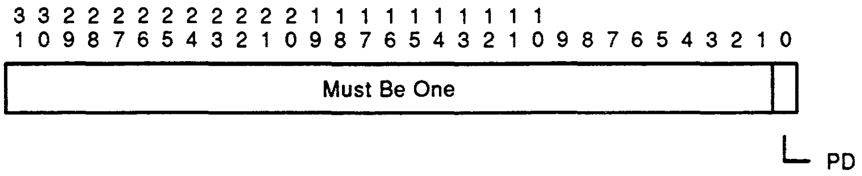
<table>
<tr><th>Bit</th><th>Name</th><th>Access</th><th>Description</th></tr>
<tr><td>&lt;31:30&gt;</td><td>IP</td><td>R/W</td><td>Interrupt priority is the VAX interrupt priority level that the SGEC respond to.
<table><tr><th>IP</th><th>IPL (hex)</th></tr><tr><td>00</td><td>14</td></tr><tr><td>01</td><td>15</td></tr><tr><td>10</td><td>16</td></tr><tr><td>11</td><td>17</td></tr></table>
Although the SGEC has only one interrupt request pin, that pin might be wired to any of the four IRQ pins on the host. The value in IP should be 14_{16} for the KA690.</td></tr>
<tr><td>&lt;29&gt;</td><td>SA</td><td>R/W</td><td>Sync/Asynch - This bit determines the SGEC operating mode when it is the bus master. When set, the SGEC operates as a synchronous device and when clear, the SGEC operates as an asynchronous device.</td></tr>
<tr><td>&lt;15:00&gt;</td><td>IV</td><td>R/W</td><td>Interrupt Vector. During an interrupt acknowledge cycle for an SGEC interrupt, this is the value that the SGEC drives on the host bus CDAL<31:0> pins (CDAL pins <1:0> and <31:16> are set to 0). Bits <1:0> are ignored when NICSR0 is written, and set to 1 when read.</td></tr>
</table>

Table 10–3 NICSR0 Access

Value after **RESET**:	1FFF0003 hex
Read access rules:	None
Write access rules:	The IPL to which the SGEC is assigned must be DISABLED

10.3.4 Transmit Polling Demand (NICSR1)

The polling demand NICSR (NICSR1) is used by the port driver to tell the SGEC that it put a packet on the transmit or receive list. The format is shown in Figure 10–3 and the bit description is in Table 10–4.



I/O Address: $2000\ 8004_{(16)}$

Longword Write Only Access

Figure 10–3 Polling Demand (NICSR1)

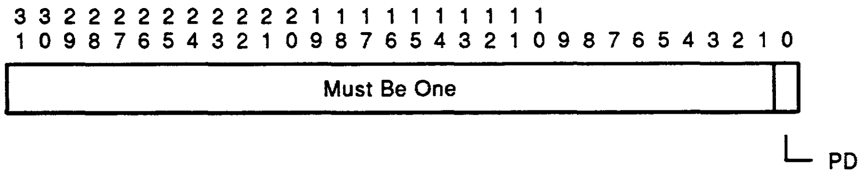
Table 10–4 NICSR1 Bits

Bit	Name	Access	Description
<31:01>	MBZ		Must be 1. This field is reserved for future expansion. Write as 1.
<00>	PD	W	Tx polling demand. Checks the transmit list for frames to be transmitted. The PD value is meaningless.

Table 10–5 NICSR1 Access

Value after **RESET**:	Not applicable
Read access rules:	None
Write access rules:	Tx process SUSPENDED

10.3.5 Receive Polling Demand (NICSR2)



I/O Address: $2000\ 8008_{(16)}$

Figure 10–4 NICSR2 Format

Table 10–6 NICSR2 Bits

Bit	Name	Access	Description
<31:01>	MBO		Must be 1. This field is reserved for future expansion. Write as 1.
<00>	PD	W	Rx polling demand. Checks the receive list for receive descriptors to be acquired. The PD value is meaningless.

Table 10–7 NICSR2 Access

Value after **RESET**:	Not applicable
Read access rules:	None
Write access rules:	Rx process SUSPENDED

10.3.6 Descriptor List Addresses (NICSR3, NICSR4)

The two descriptor list address registers are identical in function, one being used for the transmit buffer descriptors and one being used for the receive buffer descriptors. In both cases, the registers are used to point the SGEC to the start of the appropriate buffer descriptor list.

The descriptor lists reside in VAX physical memory space and must be longword aligned.

NOTE

For best performance, the descriptor lists be OCTAWORD should be OCTAWORD aligned.

TRANSMIT LIST

If the Transmit descriptor list is built as a ring (the chain descriptor points at the first descriptor of the list), the ring must contain at least two descriptors in addition to the chain descriptor.

Initially, these registers **must be written before the respective start command is given** (see Section 10.3.8), else the respective process remains in the stopped state. New list head addresses are only acceptable while the respective process is in the stopped or suspended states. Addresses written while the respective process is in the running state, are ignored and discarded.

If the host attempts to read any of these registers before ever writing to them, the SGEC responds with unpredictable values.

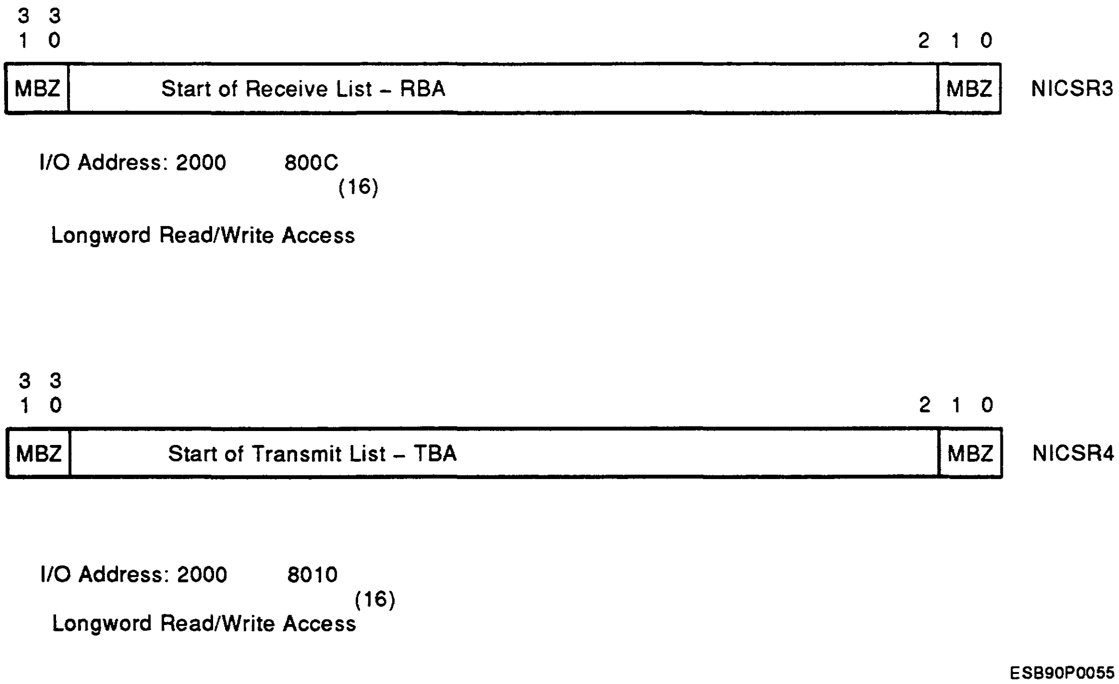

3 3
1 0
2 1 0
MBZ
Start of Receive List - RBA
MBZ
NICSR3
I/O Address: 2000 800C (16)
Longword Read/Write Access
3 3
1 0
2 1 0
MBZ
Start of Transmit List - TBA
MBZ
NICSR4
I/O Address: 2000 8010 (16)
Longword Read/Write Access
ESB90P0055

Figure 10–5 Descriptor List Addresses Format

Table 10–8 Descriptor Lists Addresses Bits

Bit	Name	Access	Description
<31:30>	MBZ		Must be zero. Ignored on writes. Read as zero
<29:00>	RBA or TBA	R/W	Address of the start of the receive list (NICSR3) or transmit list (NICSR4). This is a 30-bit VAX physical address.

NOTE
The descriptor lists must be longword aligned.

Table 10–9 NICSR3 Access

Value after **RESET**:	Unpredictable
Read access rules:	None
Write access rules:	Rx process stopped or suspended

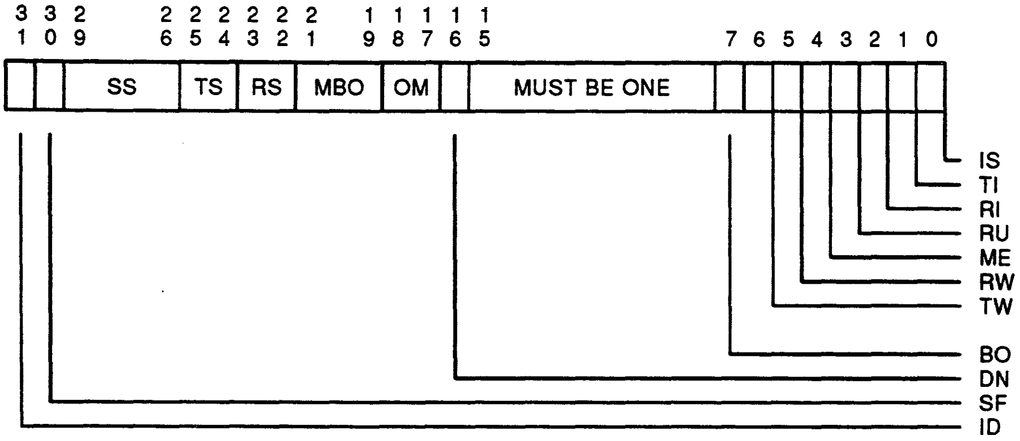
Table 10–10 NICSR4 Access

Value after **RESET**:	unpredictable
Read access rules:	None
Write access rules:	Tx process STOPPED or SUSPENDED

After either of NICSR3 or NICSR4 are written, the new address is readable from the written NICSR. However, if the SGEC status did not match the related write access rules, the new address does not take effect and the written information is lost, **EVEN if the SGEC matches later the right condition.**

10.3.7 Status Register (NICSR5)

This register contains all the status bits the SGEC reports to the host. Figure 10–6 shows the register format and Table 10–11 describes the register bits.



I/O Address: 2000 8014
(16)

Longword Access with:
Bits <31:16> Read Only
Bits <16:0> Read/Write One to Clear

ESB90P0056

Figure 10–6 NICSR5 Bits

Table 10–11 NICSR5 Bits

Bit	Name	Access	Description
<31>	ID	R	Initialization done. When set, indicates the SGEC has completed the initialization (reset and self test) sequences, and is ready for further commands. When clear, indicates the SGEC is performing the initialization sequence and ignoring all commands. After the initialization sequence completes, the transmission and reception processes are in the stopped state.
<30>	SF	R	Self test failed - When set, indicates the SGEC self test has failed. The self test completion code bits indicate the failure type.

Table 10–11 (Cont.) NICSR5 Bits

Bit	Name	Access	Description
<29:26>	SS	R	Self test status. The self test completion code is listed in the following table. Only valid if SF is set. **Value** — **Meaning** 0001 — ROM error 0010 — RAM error 0011 — Address filter RAM error 0100 — Transmit FIFO error 0101 — Receive FIFO error 0110 — Self_test loopback error **INFO** **Self test takes 25ms to complete after hardware or software reset.**
<25:24>	TS	R	Transmission process state. Indicates the current state of the transmission process as follows: **Value** — **Meaning** 00 — STOPPED 01 — RUNNING 10 — SUSPENDED Section 10.3.19.5 explains the transmission process operation and state transitions.
<23:22>	RS	R	Reception process state. Indicates the current state of the reception process, as follows: **Value** — **Meaning** 00 — STOPPED 01 — RUNNING 10 — SUSPENDED Section 10.3.19.4 explains the reception process operation and state transitions.

Table 10–11 (Cont.) NICSR5 Bits

<table>
<tr><th>Bit</th><th>Name</th><th>Access</th><th>Description</th></tr>
<tr><td>&lt;18:17&gt;</td><td>OM</td><td>R</td><td>Operating mode. Ethernet Controller operating mode as in the following table:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>00</td><td>Normal operating mode.</td></tr>
<tr><td>01</td><td>Internal loopback. Indicates the SGEC is disengaged from the Ethernet wire. Frames from the transmit list are looped back to the receive list, subject to address filtering. Section 10.3.19.6 explains this mode of operation.</td></tr>
<tr><td>10</td><td>External loopback. Indicates the SGEC is working in full duplex mode. Frames from the transmit list are transmitted on the Ethernet wire and also looped back to the receive list, subject to address filtering. Section 10.3.19.6 explains this mode of operation.</td></tr>
<tr><td>11</td><td>Reserved for diagnostics.</td></tr>
</table>
</td></tr>
<tr><td>These bits indicate the current SGEC</td><td></td><td></td><td></td></tr>
<tr><td>&lt;16&gt;</td><td>DN</td><td>R</td><td>Done. When set, indicates the SGEC has completed a requested virtual NICSR access. After a reset, this bit is set.</td></tr>
<tr><td>&lt;15:8&gt;</td><td>MBO</td><td></td><td>Must be one. This field is reserved. Writes are ignored, read as 1.</td></tr>
<tr><td>&lt;7&gt;</td><td>BO</td><td>R/W1</td><td>Boot_Message. When set, indicates that the SGEC has detected a boot_message on the serial line and has set the external pin BOOT_L.</td></tr>
<tr><td>&lt;6&gt;</td><td>TW</td><td>R/W1</td><td>Transmit watchdog timer interrupt. When set, indicates the transmit watchdog timer has timed out, indicating the SGEC transmitter was babbling. The Transmission process is aborted and placed in the stopped state. (Also reported into the Tx descriptor status TDES0&lt;TO&gt; flag.)</td></tr>
</table>

Table 10–11 (Cont.) NICSR5 Bits

Bit	Name	Access	Description
<5>	RW	R/W1	Receive watchdog timer interrupt. When set, indicates the receive watchdog timer has timed out, indicating that some other node is babbling on the network. Current frame reception is aborted and RDES0<LE> and RDES0<LS> are set. Bit NICSR5<RI> also set. The reception process remains in the running state.
<4>	ME	R/W1	Memory error. Is set when any of the following occur: • SGEC is the CP-BUS master and the ERR_L pin is asserted by external logic (generally indicative of a memory problem). • Parity error detected on an host to SGEC NICSR write or SGEC read from memory. When a memory error is set, the reception and transmission processes are aborted and placed in the STOPPED state. **NOTE** **At this point, it is mandatory that the port driver issue a Reset command and rewrite all NICSRs.**
<3>	RU	R/W1	Receive buffer unavailable. When set, indicates that the next descriptor on the receive list is owned by the host and could not be acquired by the SGEC. The reception process is placed in the suspended state. Section 10.3.19.4 explains the reception process state transitions. Once set by the SGEC, this bit is not set again until the SGEC encounters a descriptor it can not acquire. To resume processing receive descriptors, the host must flip the ownership bit of the descriptor and can issue the Rx poll demand command. If no Rx poll demand is issued, the Reception process resumes when the next recognized incoming frame is received.
<2>	RI	R/W1	Receive interrupt. When set, indicates that a frame has been placed on the receive list. Frame specific status information was posted in the descriptor. The Reception process remains in the RUNNING state.

Table 10–11 (Cont.) NICSR5 Bits

Bit	Name	Access	Description
<1>	TI	R/W1	Transmit Interrupt - When set, indicates one of the following: • Either all the frames in the transmit list have been transmitted (next descriptor owned by the host), or a frame transmission was aborted due to a locally induced error. The port driver must scan down the list of descriptors to determine the exact cause. The transmission process is placed in the SUSPENDED state. Section 10.3.19.5 explains the transmission process state transitions. To resume processing transmit descriptors, the port driver must issue the poll demand command. • A frame transmission completed and TDES1<IC> was set. The transmission process remains in the RUNNING state, unless the next descriptor is owned by the host or the frame transmission aborted due to an error. In the latter cases, the transmission process is placed in the SUSPENDED state.
<0>	IS	R/W1	Interrupt summary. The logical OR of NICSR5 bits 1 through 6.

Table 10–13 NICSR5 Access

Value after **RESET**:	0039FF00 hex
Read access rules:	None
Write access rules:	NICSR5<07:01> bits cleared by 1, others bits not writeable

10.3.7.1 NICSR5 Status Report

The status register NICSR5 is split into two words as follows:

The high word which contains the global status of the SGEC, as the initialization status, the DMA and operation mode and the receive and transmit process states.

The low word which contains the status related to the receive and transmit frames.

Any change of the NICSR5 bits <ID>, <SF>, <OM> or <DN> which is always the result of a host command is reported without an interrupt.

Any process state change initiated by a host command) NICSR6<ST> or NICSR6<SR>, is reported without an interrupt.

In the previous two cases, the driver must poll on NICSR5 to get the acknowledge of its command (For example, polling on <ID, SF> after reset or polling on <TS> after a START_TX command).

Any process state change initiated by the SGEC activity is immediately followed by at least one of the NICSR5<6:1> interrupts and the interrupt_summary NICSR5<IS>.

The SGEC 16 bit internal processor updates the 32 bits NICSR5 register in two phases: the high word is modified first, then the low word is written, which generates an interrupt to the host. In this case, the driver must scan first the NICSR5 low word to get the interrupt status, then the NICSR high word to get the related process state. (For example, <TI> interrupt with <TS>= SUSPENDED reports an end of transmission due to a Tx descriptor unavailable.)

If the host polls on the process state change, it may detect a change without interrupt due to the small time window separating the NICSR5 high word and low word updates.

Maximum time window is 4*Tcycles of the host clock.)

10.3.8 Command and Mode Register (NICSR6)

This register is used to establish operating modes and for port driver commands.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RE	IE	r	BL				Must be ONE				BE	SE	r	r	r	Must be ONE				ST	SR	OM		DC	FC	r	r	PB	AF		r

I/O Address: 2000 8018 (16)

Longword Read/Write Access

r = reserved

ESB90P0057

Figure 10–7 NICSR6 Format

Table 10–14 NICSR6 Bits

Bit	Name	Access	Description
<31>	RE	R/W	Reset command. Upon being set, the SGEC aborts all processes and starts the reset sequence. After completing the reset and self test sequence, the SGEC sets bit NICSR5<ID>. Clearing this bit has no effect. **NOTE** **The NICSR6<RE> value is unpredictable on read after HARDWARE reset.**
<30>	IE	R/W	Interrupt enable mode. When set, setting of NICSR5 bits 1 through 6 will cause an interrupt to be generated.
<29>	r		Reserved

Table 10–14 (Cont.) NICSR6 Bits

Bit	Name	Access	Description
<28:25>	BL	R/W	Burst limit mode. Specifies the maximum number of longwords to be transferred in a single DMA burst on the host bus. When NICSR6<SE> is cleared, permissible values are 1,2,4,8 . When SE is set, the only permissible values are 1 and 4: a value of 2 or 8 is respectively forced to 1 or 4. After initialization, the burst limit is set to 1.
<24:21>	MBO		This field is reserved. Writes are ignored. Read as one.
<20>	BE	R/W	Boot_message enable mode. When set, enables the boot_ message recognition. When the SGEC recognizes an incoming boot message on the serial line, NICSR5<BO> is set and the external pin BOOT_L is asserted for a duration of 6*Tcycles (of the host clock).
<19>	SE	R/W	Single_cycle enable mode. When set, the SGEC transfers only a single longword or an octaword in a single DMA burst on the host bus.
<18:12>	MBO		Must Be One. This field is reserved. Writes are ignored. Read as ONE.
<11>	ST	R/W	Start/Stop transmission command. When set, the transmission process is placed in the RUNNING state, the SGEC checks the transmit list at the *current* position for a frame to transmit the address set by *NICSR4* or the position retained when the Tx process was previously stopped. If it does not find a frame to transmit, the transmission process enters the SUSPENDED state. The start transmission command is honored only when the transmission process is in the STOPPED state. *The first time this command is issued, an additional requirement is that NICSR4 has already been written to, or else the Transmission process remains in the STOPPED state.* When cleared, the transmission process is placed in the STOPPED state after completing transmission of the current frame. The next descriptor position in the transmit list is saved and becomes the current position after transmission is restarted. The stop transmission command is honored only when the transmission process is in the RUNNING or SUSPENDED states. Refer to Section 10.3.19.5 for more information.

Table 10–14 (Cont.) NICSR6 Bits

<table>
<tr><th>Bit</th><th>Name</th><th>Access</th><th>Description</th></tr>
<tr><td>&lt;10&gt;</td><td>SR</td><td>R/W</td><td>Start/Stop reception command. When set, the reception process is placed in the RUNNING state, the SGEC attempts to acquire a descriptor from the receive list and process incoming frames. Descriptor acquisition is attempted from the <i>current</i> position in the list - the address set by <i>NICSR3</i> or the position retained when the Rx process was previously stopped. If no descriptor can be acquired, the reception process enters the SUSPENDED state.

The start reception command is honored only when the reception process is in the STOPPED state. <i>The first time this command is issued, an additional requirement is that NICSR3 has already been written to, or else the reception process remains in the STOPPED state.</i>

When cleared, the reception process is placed in the STOPPED state after completing reception of the current frame. The next descriptor position in the receive list is saved, and becomes the <i>current</i> position after reception is restarted. The stop reception command is honored only when the Reception process is in the RUNNING or SUSPENDED states.

Refer to Section 10.3.19.4 for more information.</td></tr>
<tr><td>&lt;9:8&gt;</td><td>OM</td><td>R/W</td><td>Operating Mode - These bits determine the SGEC main operating mode.

<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>00</td><td>Normal operating mode.</td></tr>
<tr><td>01</td><td>Internal Loopback. The SGEC loopbacks buffers from the transmit list. The data is passed from the transmit logic back to the receive logic. The receive logic treats the looped frame as it would any other frame, and subjects it to the address filtering and validity check process.</td></tr>
<tr><td>10</td><td>External Loopback. The SGEC transmits normally and, enables its receive logic to its own transmissions. The receive logic treats the looped frame as it would any other frame, and subjects it to the address filtering and validity check process.</td></tr>
<tr><td>11</td><td>Reserved for diagnostic.</td></tr>
</table>
</td></tr>
<tr><td>&lt;7&gt;</td><td>DC</td><td>R/W</td><td>Disable data chaining mode. When set, no data chaining occurs in reception; frames, longer than the current receive buffer, truncated. RDES0&lt;FS,LS> is always set. The frame length returned in RDES0&lt;FL&gt;is the <i>true</i> length of the non-truncated frame while RDES0&lt;BO&gt; indicates that the frame has been truncated due to buffer overflow.

When clear, frames too long for the current receive buffers are transferred to the next buffer(s) in the receive list.</td></tr>
</table>

Table 10–14 (Cont.) NICSR6 Bits

<table>
<tr><th>Bit</th><th>Name</th><th>Access</th><th>Description</th></tr>
<tr><td>&lt;6&gt;</td><td>FC</td><td>R/W</td><td>Force Collision mode. This bit allows the collision logic to be tested. The chip must be in <b>internal loopback</b> mode for FC to be valid. If FC is set, a collision is forced during the next transmission attempt. This results in 16 transmission attempts with excessive collision reported in the transmit descriptor.</td></tr>
<tr><td>&lt;5:4&gt;</td><td>MBO</td><td></td><td>Must Be 1. This field is reserved. Writes are ignored. Read as 1.</td></tr>
<tr><td>&lt;3&gt;</td><td>PB</td><td>R/W</td><td>Pass bad frames mode. When this bit is set, the SGEC pass frames that have been damaged by collisions or are too short due to premature reception termination. Both events should have occurred within the collision window (64 bytes), or else other errors are reported.
When clear, these frames will be discarded and never show up in the host receive buffers.
<b>NOTE</b>
<b>Pass bad frames is subject to the address filtering mode. For example, to monitor the network, this mode must be set together with the promiscuous address filtering mode.</b></td></tr>
<tr><td>&lt;2:1&gt;</td><td>AF</td><td>R/W</td><td>Address filtering mode. These bits define the way incoming frames are address filtered:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>00</td><td>Normal. Incoming frames are filtered according to the values of the <HP> and <IF> bits of the setup frame descriptor.</td></tr>
<tr><td>01</td><td>Promiscuous. All incoming frames are passed to the host, regardless of the &lt;HP> bit value.</td></tr>
<tr><td>10</td><td>All multicast. All incoming frames with Multicast address destinations are passed to the host. Incoming frames with physical address destinations are filtered according to the <HP> bit value.</td></tr>
<tr><td>11</td><td>Unused. Reserved.</td></tr>
</table></td></tr>
<tr><td>&lt;0&gt;</td><td>MBO</td><td></td><td>Must Be 1. This field is reserved. Writes are ignored. Read as 1.</td></tr>
</table>

Table 10–15 NICSR6 Access

Value after **RESET**:	83E0F000 hex or 03E0F000 hex
Read access rules:	None
Write access rules:	
* <RE, IE, BE>	Unconditional
* <BL, SE, OM>	Rx and Tx processes STOPPED

Table 10–15 (Cont.) NICSR6 Access

* <FC>	Rx and Tx processes STOPPED, Internal_Loopback mode
* <DC, PB, AF>	Rx STOPPED
* Start_Receive <SR>=1	Rx STOPPED and NICSR3 Initialized
* Start_Transmit <ST>=1	Tx STOPPED and NICSR4 Initialized
* Stop_Receive <SR>=0	Rx RUNNING or SUSPENDED
* Stop_Transmit <ST>=0	Tx RUNNING or SUSPENDED

After NICSR6 is written, the new value is readable from NICSR6. However, if the SGEC status does not match the related write access rules, the new mode setting and command do not take effect and the written information is *lost*, **EVEN if the SGEC matches later the right condition**.

10.3.9 System Base Register (NICSR7)

This NICSR contains the physical starting address of the VAX system page table. This register must be loaded by host software before any address translation occurs so that memory is not be corrupted.

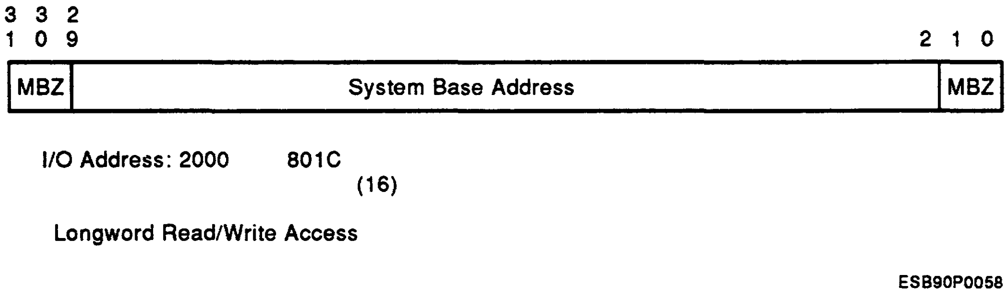


Figure 10–8 NICSR7 Format

Table 10–16 NICSR7 Bits

Bit	Name	Access	Description
<31:30>	MBZ		Must Be zero. Read as zero. Writes are ignored.
<29:00>	SB	R/W	System base address. The physical starting address of the VAX system page table. Not used if VA (virtual addressing) is cleared in all descriptors. **This register should be loaded only once after a reset. Subsequent modifications of this register at any other time can cause unpredictable results.**

Table 10–17 NICSR7 Access

Value after **RESET**:	Unpredictable
Read access rules:	None
Write access rules:	Writing once after initialization

10.3.10 Reserved Register (NICSR8)

This entire register is reserved.

10.3.11 Watchdog Timers (NICSR9)

The SGEC has two timers that restrict the length of time in which the chip can receive or transmit.

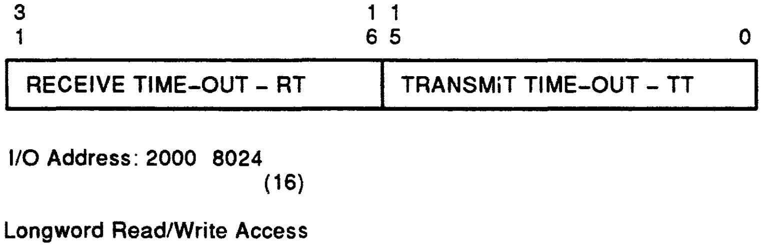


Figure 10–9 NICSR9 Format

Table 10–18 NICSR9 Bits

Bit	Name	Access	Description
<31:16>	RT	R/W	Receive watchdog time-out. The receive watchdog timer protects the host CPU against babbling transmitters on the network. If the receiver stays on for *RT* * 16 cycles of the serial clock, the SGEC cuts off reception and sets the NICSR5<RW> bit. If the timer is set to zero, never times-out. The value of RT is an unsigned integer. With a 10 MHz serial clock, this provides a range of 72µs to 100ms. The default value is 1250 corresponding to 2ms. The Rx watchdog timer is programmed only while the reception process is in the STOPPED state. **NOTE** **A Rx watchdog value between 1 and 44 is forced to the minimum time_out value of 45 (72µs).**

Table 10–18 (Cont.) NICSR9 Bits

Bit	Name	Access	Description
<15:00>	TT	R/W	Transmit watchdog time-out. The transmit watchdog timer protects the network against babbling SGEC transmissions, on top of any such circuitry present in tranceivers. If the transmitter stays on for *TT* * 16 cycles of the serial clock, the SGEC cuts off the transmitter and sets the NICSR5<TW> bit. If the timer is set to zero, never times-out. The value of TT is an unsigned integer. With a 10 MHz serial clock, this provides a range of 72µs to 100ms. The default value is 1250 corresponding to 2ms. The Tx watchdog timer is programmed only while the transmission process is in the STOPPED state. **NOTE** **A Tx watchdog value between 1 and 44 is forced to the minimum time_out value of 45 (72µs).**

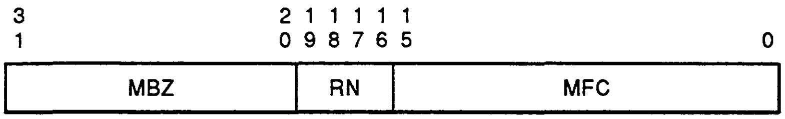
Table 10–19 NICSR9 Access

Value after **RESET**:	00000000 hex
Read access rules:	None
Write access rules:	
* Rx Watchdog timer	Rx process STOPPED
* Tx Watchdog timer	Tx process STOPPED

The watchdog timers are enabled by default. The timers assume the default values after hardware or software resets.

10.3.12 Revision Number and Missed Frame Count (NICSR10)

This register contains a missed frame counter and SGEC identification information.



I/O Address: 2000 802C
(16)

Longword Read Only Access

Figure 10–10 Revision Number and Missed Frame Count (VIRTUAL NICSR10)

Table 10–20 NICSR10 Bits

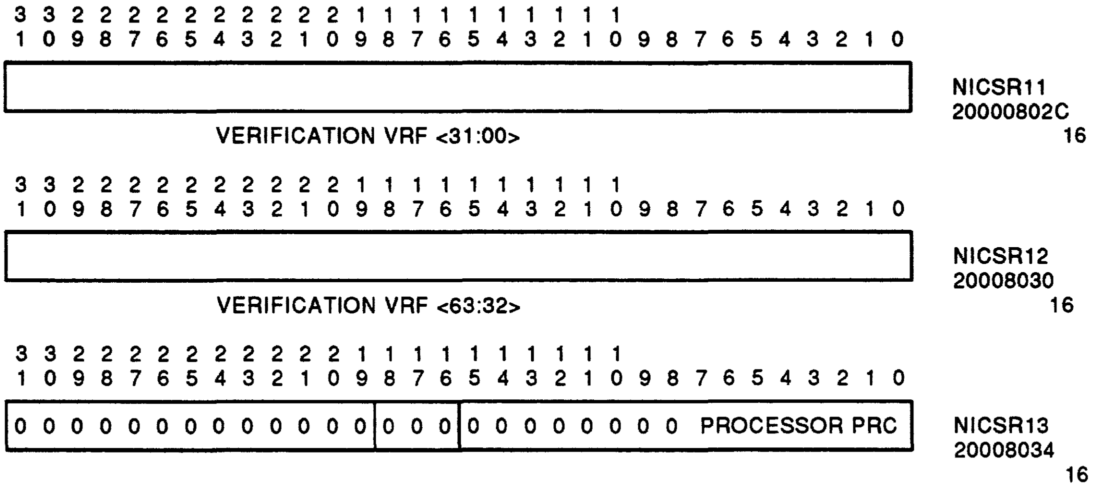
Bit	Name	Access	Description
<31:21>	MBZ		Must BE zero. Read as zero. Writes are ignored.
<20:16>	RN	R	Chip Revision Number. This stores the revision number.
<15:00>	MFC	R	Missed frame count. This is the counter for the number of frames that were discarded and lost because host receive buffers were unavailable. The counter is cleared when read by the host.

Table 10–21 NICSR10 Access

Value after **RESET**:	00030000 hex
Read access rules:	Missed_frame counter cleared by read
Write access rules:	Not applicable

10.3.13 Boot Message (NICSR11, 12, 13)

These registers contain the boot message VERIFICATION and PROCESSOR fields. The format is shown in Figure 10–11 and the bit descriptions are in Table 10–22.



Longword Read/Write Access

Figure 10–11 Boot Message

Table 10–22 NICSR11, NICSR12, NICSR13 Bits

Bit	Name	Access	Description
NICSR11 <31:00>	VRF<31:00>	R/W	Boot message VERIFICATION field <31:00>
NICSR12 <31:00>	VRF<63:32>	R/W	Boot message VERIFICATION field <63:32>
NICSR13 <07:00>	PRC	R/W	Boot message PROCESSOR field

NOTE
The least significant bit of the verification field (VRF<0>) corresponds to the first incoming bit of the verification field in the serial boot message.

Table 10–23 NICSR11, NICSR12, NICSR13 Access

Value after **RESET**:	00000000 hex for each of NICSR11,NICSR12,NICSR13
Read access rules:	None
Write access rules:	Boot message DISABLED (<NICSR6<BE> = 0)

10.3.14 Diagnostic Registers (NICSR14, 15)

These registers are reserved for diagnostic features.

10.3.14.1 Diagnostic Breakpoint Address Register (NICSR14)

This register is virtual CSR. It contains the breakpoint address that causes the internal CPU to jump to a patch address. Figure 10–12 shows the format of the register. Table 10–24 lists the bits and descriptions. This register can be loaded only in diagnostic mode (NICSR6 <OM>=<11>).

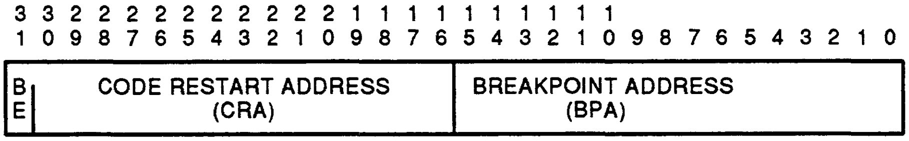


Figure 10–12 NICSR14 Format

Table 10–24 NICSR14 Bits

Bit	Name	Type	Description
<31>	BE	R/W	When set, breakpoint enabled.
<30:16>	CRA	R/W	Code restart address. This is the first address in the internal RAM to where the internal processor will jump after a breakpoint occurred.

Table 10–24 (Cont.) NICSR14 Bits

Bit	Name	Type	Description
<15:0>	BPA	R/W	Breakpoint address. This is the internal processor address at which the program will halt and jump to the RAM loaded code.

NOTE
This registers in conjunction with the diagnostic descriptors's allows software patches.

Table 10–25 NICSR14 Access

Value after **RESET**:	00000000$_{16}$
Read access rules:	None
Write access rules:	DIAGNOSTIC mode
Violation:	Addressing NICSR14 while NICSR5<DN> is deasserted

10.3.14.2 Monitor Command Register (NICSR15)

This register is a physical CSR. It contains the bits which select the internal test block operation mode. Figure 10–13 shows the format of the register. Table 10–26 lists the bits and descriptions.

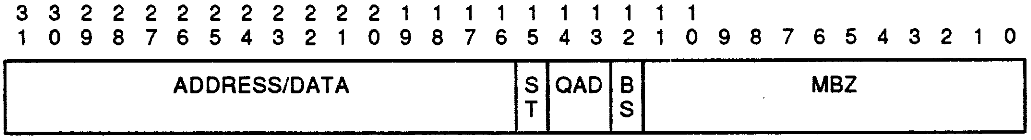


Figure 10–13 NICSR15 Format

Table 10–26 NICSR15 Bits

Bit	Name	Type	Description
<31:16>	ADDR/DATA	R/W	Before the "Examine" cycle, it points to the location to be read. Three cycles after the assertion of <ST>, it contains the READ data.
<15>	ST	W	Start read. When set, this starts the "Examine" cycle. The data addressed by CSR<31:16> is fetched and stored into the same register field. Reset by hardware at the end of operation.

Table 10–26 (Cont.) NICSR15 Bits

<table>
<tr><th>Bit</th><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>&lt;14:13&gt;</td><td>QAD</td><td>W</td><td>Quad selects bits. These bits define the specific four bits of the internal Data_bus or Address_bus which are monitored on the external test pins BM_L/TEST&lt;3:0>. Meaningful only in test mode (TSM=1).
The 2 bit code is interpreted as follows:
<table>
<tr><th>QAD</th><th>Data</th><th>Address</th></tr>
<tr><td>00</td><td>&lt;03:00&gt;</td><td>&lt;03:00&gt;</td></tr>
<tr><td>01</td><td>&lt;07:04&gt;</td><td>&lt;07:04&gt;</td></tr>
<tr><td>10</td><td>&lt;11:08&gt;</td><td>&lt;11:08&gt;</td></tr>
<tr><td>11</td><td>&lt;15:12&gt;</td><td>0, IOP_WR_L,<13:12&gt;</td></tr>
</table>
</td></tr>
<tr><td>&lt;12&gt;</td><td>BS</td><td>W</td><td>Bus select. When resets the internal Data_bus is monitored on the external test pins BM_L/TEST&lt;3:0>. When set, the monitoring is applied on the internal Address_bus. Meaningful only in test mode (TSM=1).</td></tr>
<tr><td>&lt;11:0&gt;</td><td>MBZ</td><td>–</td><td>Must be zero.</td></tr>
</table>

Table 10–27 NICSR15 Access

Value after **RESET**:	$00000FFF_{16}$
Read access rules:	None
Write access rules:	Reserved for DEBUGGING
Violation:	Setting <ST> with "random" SGEC internal address

10.3.15 Descriptors and Buffers Format

The SGEC transfers frame data to and from the receive and transmit buffers in host memory. These buffers are pointed to by descriptors which are also resident in host memory.

There are two descriptor lists: One for receive and one for transmit. The starting address of each list is written into NICSRs 3 and 4 respectively. A descriptor list is a forward-linked (either implicitly or explicitly) list of descriptors, the last of which may point back to the first entry, thus creating a ring structure. Explicit chaining of descriptors, through setting xDES1<CA>s is called Descriptor Chaining. The descriptor lists reside in VAX *physical* memory address space.

NOTE
The SGEC first reads the descriptors, ignoring all unused bits regardless of their state. The only word the SGEC writes back is the first word (xDES0) of

each descriptor. Unused bits in xDES0 are written as zero. Unused bits in xDES1 - xDES3 can be used by the port driver and the SGEC never disturbs them.

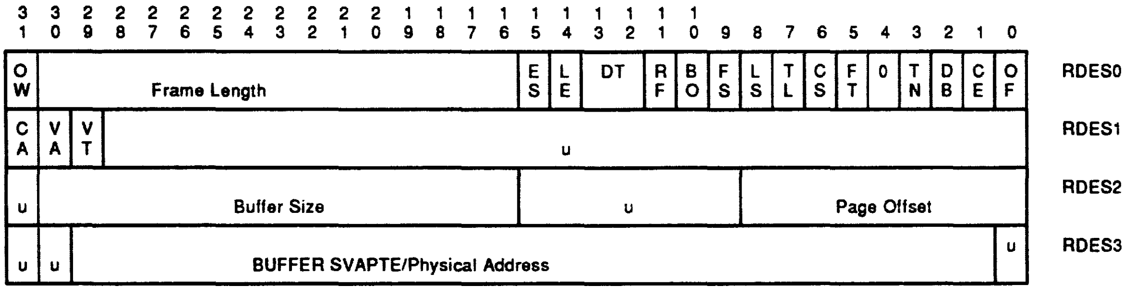
A data buffer can contain an entire frame or part of a frame, but it cannot contain more than a single frame. Buffers contain only data; buffer status is contained in the descriptor. The term Data Chaining is used to refer to frames spanning multiple data buffers. Data Chaining can be enabled or disabled, in reception, through NICSR6<DC>. Data buffers reside in either VAX physical or virtual memory space.

NOTE
The virtual-to-physical address translation is based on the assumption that PTEs are locked in the host memory at the time the SGEC owns the related buffer.

NOTE
For best performance in virtual addressing mode, PPTE vectors must not cross a page of the PPTE table.

10.3.16 Receive Descriptors

The receive descriptor format is shown in Figure 10–14, and described in the following paragraphs.



0 - SGEC writes as 0
u - Ignored by the SGEC on read, never written

Figure 10–14 Receive Descriptor Format

10.3.16.1 RDES0 Word

RDES0 word contains received frame status, lengths and descriptor ownership information.

Table 10–28 RDES0 Bits

Bit	Name	Description
<31>	OW	Own bit. When set, this bit indicates the descriptor is owned by the SGEC. When cleared, indicates the descriptor is owned by the host. The SGEC clears this bit when processing of the descriptor and its associated buffer is complete.
<30:16>	FL	Frame Length. The length in bytes of the received frame. Meaningless if RDES0<LE> is set.

Table 10–28 (Cont.) RDESO Bits

<table>
<tr><th>Bit</th><th>Name</th><th>Description</th></tr>
<tr><td>&lt;15&gt;</td><td>ES</td><td>Error Summary. The logical "OR" of RDES0 bits OF,CE,TN,CS,TL,LE,RF.</td></tr>
<tr><td>&lt;14&gt;</td><td>LE</td><td>Length Error. When set, indicates a frame truncation caused by one of the following:
• The frame segment does not fit within the current buffer and the SGEC does not own the next descriptor. The frame is truncated.
• The receive watchdog timer expired. NICSR5<RW> is also set.</td></tr>
<tr><td>&lt;13:12&gt;</td><td>DT</td><td>Data Type. Indicates the type of frame the buffer contains, according to the following table:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>00</td><td>Serial received frame</td></tr>
<tr><td>01</td><td>Internally looped back frame</td></tr>
<tr><td>10</td><td>Externally looped back frame, serial received frame [2]</td></tr>
</table></td></tr>
<tr><td>&lt;11&gt;</td><td>RF</td><td>Runt Frame. When set, indicates this frame was damaged by a collision or premature termination before the collision window had passed. Runt frames are only passed on to the host if (NICSR6<PB>) is set. Meaningless if (RDES0&lt;OF&gt;) is set.</td></tr>
<tr><td>&lt;10&gt;</td><td>BO</td><td>Buffer overflow. When set, indicates that the frame has been truncated due to a buffer too small to fit the frame size. This bit may only be set if data chaining is disabled (NICSR6<DC&gt; = 1).</td></tr>
<tr><td>&lt;09&gt;</td><td>FS</td><td>First segment. When set, indicates this buffer contains the first segment of a frame.</td></tr>
<tr><td>&lt;08&gt;</td><td>LS</td><td>Last segment. When set, indicates this buffer contains the last segment of a frame and status information is valid.</td></tr>
<tr><td>&lt;07&gt;</td><td>TL</td><td>Frame too long. When set, indicates the frame length exceeds the maximum Ethernet specified size of 1518 bytes.
NOTE
<b>Frame too long is only a frame length indication and does not cause any frame truncation.</b></td></tr>
<tr><td>&lt;06&gt;</td><td>CS</td><td>Collision seen. When set, indicates the frame was damaged by a collision that occurred after the 64 bytes following the SFD.</td></tr>
</table>

Table 10–28 (Cont.) RDES0 Bits

<table>
<tr><th>Bit</th><th>Name</th><th>Description</th></tr>
<tr><td>&lt;05&gt;</td><td>FT</td><td>Frame type. When set, indicates the frame is an Ethernet type frame (frame length_field > 1500). When clear, indicates the frame is an IEEE 802.3 type frame. Meaningless for runt frames < 14 bytes.</td></tr>
<tr><td>&lt;04&gt;</td><td>0</td><td>Zero. SGEC writes as zero.</td></tr>
<tr><td>&lt;03&gt;</td><td>TN</td><td>Translation not valid. When set, indicates that a translation error occurred when the SGEC was translating a VAX virtual buffer address. It will only set if RDES1<VA> was set. The reception process remains in the RUNNING state and attempts to acquire the next descriptor.</td></tr>
<tr><td>&lt;02&gt;</td><td>DB</td><td>Dribbling Bits. When set, indicates the frame contained a non-integer multiple of eight bits. This error is reported only if the number of dribbling bits in the last byte is greater than two. Meaningless if RDES0&lt;CS&gt; or RDES0&lt;RF&gt; are set.

The CRC check is performed independent of this error, however, only whole bytes are run through the CRC logic. <b>Consequently, received frames with up to six dribbling bits have this bit set, but if <CE> (or another error indicator) is not set, these frames should be considered valid:</b>
<table>
<tr><th>CE</th><th>DB</th><th>Error</th></tr>
<tr><td>0</td><td>0</td><td>None</td></tr>
<tr><td>0</td><td>1</td><td>None</td></tr>
<tr><td>1</td><td>0</td><td>CRC error</td></tr>
<tr><td>1</td><td>1</td><td>Alignment error</td></tr>
</table>
</td></tr>
<tr><td>&lt;01&gt;</td><td>CE</td><td>CRC Error. When set, indicates that a CRC error has occurred on the received frame.</td></tr>
<tr><td>&lt;00&gt;</td><td>OF</td><td>Overflow. When set, indicates received data in this descriptor's buffer was corrupted due to internal FIFO overflow. This generally occurs if SGEC DMA requests are not granted before the internal receive FIFO fills up.</td></tr>
</table>

10.3.16.2 RDES1 Word

Table 10–29 RDES1 Bits

<table>
<tr><th>Bit</th><th>Name</th><th>Descriptor</th></tr>
<tr><td>&lt;31&gt;</td><td>CA</td><td>Chain address. When set, RDES3 is interpreted as another descriptor's VAX physical address. This allows the SGEC to process multiple, non-contiguous descriptor lists and explicitly "chain" the lists. Note that contiguous descriptors are implicitly chained.
In contrast to what is done for a Rx buffer descriptor, the SGEC clears neither the ownership bit RDES0&lt;OW&gt; nor one of the other bits of RDES0 of the chain descriptor after processing.
To protect against an infinite loop, a chain descriptor pointing back to itself is seen as <b>owned by the host</b>, regardless of the ownership bit state.</td></tr>
<tr><td>&lt;30&gt;</td><td>VA</td><td>Virtual addressing. When set, RDES3 is interpreted as a virtual address. The type of virtual address translation is determined by the RDES1<VT> bit. The SGEC uses RDES3 and RDES2<Page Offset> to perform a VAX virtual address translation process to obtain the physical address of the buffer. When clear, RDES3 is interpreted as the actual physical address of the buffer:
<table>
<tr><th>VA</th><th>VT</th><th>Addressing Mode</th></tr>
<tr><td>0</td><td>x</td><td>Physical</td></tr>
<tr><td>1</td><td>0</td><td>Virtual - SVAPTE</td></tr>
<tr><td>1</td><td>1</td><td>Virtual - PAPTE</td></tr>
</table>
</td></tr>
<tr><td>&lt;29&gt;</td><td>VT</td><td>Virtual type. In case of virtual addressing (RDES1<VA> = 1), indicates the type of virtual address translation. When set, the buffer address RDES3 is interpreted as a SVAPTE (system virtual address of the page table entry). When clear, the buffer address is interpreted as a PAPTE (Physical Address of the Page Table Entry). Meaningful only if RDES1<VA> is set.</td></tr>
<tr><td>&lt;28:0&gt;</td><td>u</td><td>Unused. Ignored by the SGEC on reads. Never written.</td></tr>
</table>

10.3.16.3 RDES2 Word

Table 10–30 RDES2 Bits

Bit	Name	Descriptor
<31>	u	Unused. Ignored by the SGEC on reads. Never written.
<30:16>	BS	Buffer size. The size, in bytes, of the data buffer. **NOTE** **Receive buffer size must be an EVEN number of bytes.**
<15:9>	u	Unused. Ignored by the SGEC on reads. Never written.

Table 10–30 (Cont.) RDES2 Bits

Bit	Name	Descriptor
<08:00>	PO	Page offset. The byte offset of the buffer within the page. Only meaningful if RDES1<VA> is set. **NOTE** **Receive buffers must be word aligned.**

10.3.16.4 RDES3 Word

Table 10–31 RDES3 Bits

Bit	Name	Descriptor
<31:00>	SV/PV/PA	SVAPTE/PAPTE/Physical address. When RDES1<VA> is set, RDES3 is interpreted as the address of the page table entry and used in the virtual address translation process. The type of the address system virtual address (SVAPTE) or physical address (PAPTE) is determined by RDES1<VT>. When RDES1<VA> is clear, RDES3 is interpreted as the physical address of the buffer. When RDES1<CA> is set, RDES3 is interpreted as the VAX physical address of another descriptor. **NOTE** **Receive buffers must be word aligned.**

10.3.16.5 Receive Descriptor Status Validity

The following table summarizes the validity of the receive descriptor status bits regarding the reception completion status:

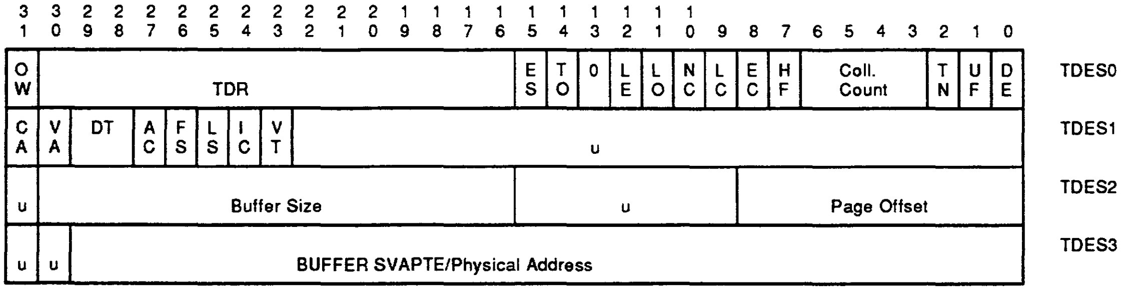
Table 10–32 Receive Descriptor Status Validity

Reception	Rx Status Report						
Status	RF	TL	CS	FT	DB	CE	(ES,LE,BO,DT,FS,LS,FL,TN, OF)
Overflow	X	V	X	V	X	X	V
Collision after 512 bits	V	V	V	V	X	X	V
Runt frame	V	V	V	V	X	X	V
Runt frame < 14 bytes	V	V	V	X	X	X	V
Watchdog timeout	V	V	X	V	X	X	V

V - Valid
X - Meaningless

10.3.17 Transmit Descriptors

The transmit descriptor format is shown in Figure 10–15 and described in the following paragraphs.



0 - SGEC writes as 0
u - Ignored by the SGEC on read, never written
ESB90P0065

Figure 10–15 Transmit Descriptor Format

10.3.17.1 TDES0 Word

TDES0 word contains transmitted frame status and descriptor ownership information.

Table 10–33 TDES0 Bits

Bit	Name	Description
<31>	OW	Own bit. When set, indicates the descriptor is owned by the SGEC. When cleared, indicates the descriptor is owned by the host. The SGEC clears this bit when processing of the descriptor and its associated buffer is complete.
<29:16>	TDR	Time domain reflectometer. This is a count of bit time and is useful for locating a fault on the cable using the velocity of propagation on the cable. Only valid if TDES0<EC> is also set. Two excessive collisions in a row and with the same or similar (within 20) TDR values indicate a possible cable open.
<15>	ES	Error summary. The logical "OR" of UF, TN, EC, LC, NC, LO, LE and TO.
<14>	TO	Transmit watchdog timeout. When set, indicates the transmit watchdog timer has timed out, indicating the SGEC transmitter was babbling. The interrupt NICSR5<TW> is set and the transmission process is aborted and placed in the STOPPED state.
<13>	MBZ	Must be ZERO. SGEC writes as zero.

Table 10–33 (Cont.) TDES0 Bits

Bit	Name	Description
<12>	LE	Length error. When set, indicates one of the following: • Descriptor unavailable (owned by the host) in the middle of data chained descriptors. • Zero length buffer in the middle of data chained descriptors. • Setup or diagnostic descriptors (data type TDES1<DT><> 0) in the middle of data chained descriptors. • Incorrect order of first_segment TDES1<FS> and last_segment TDES1<LS> descriptors in the descriptor list. The transmission process enters the SUSPENDED state and sets NICSR5<TI>.
<11>	LO	Loss of carrier. When set, indicates loss of carrier during transmission (possible short circuit in the Ethernet cable). Meaningless in internal loopback mode (NICSR5<OM>=1).
<10>	NC	No carrier. When set, indicates the carrier signal from the transceiver was not present during transmission (possible problem in the transceiver or transceiver cable). Meaningless in internal loopback mode (NICSR5<OM>=1).
<09>	LC	Late collision. When set, indicates frame transmission was aborted due to a late collision. Meaningless if TDES0<UF>.
<08>	EC	Excessive collisions. When set, indicates that the transmission was aborted because 16 successive collisions occurred while attempting to transmit the current frame.
<07>	HF	Heartbeat fail. When set, indicates heartbeat collision check failure (the transceiver failed to return a collision pulse as a check after the transmission. Some tranceivers do not generate heartbeat, and so will always have this bit set. If the transceiver does support it, it indicates transceiver failure.) Meaningless if TDES0<UF>.
<06:03>	CC	Collision count. A four bit counter indicating the number of collisions that occurred before the transmission attempt succeeded or failed. Meaningless when TDES0<EC> is also set.
<02>	TN	Translation not valid. When set, indicates that a translation error occurred when the SGEC was translating a VAX virtual buffer address. It may only set if TDES1<VA> was set. The transmission process enters the SUSPENDED state and sets NICSR5<TI>.
<01>	UF	Underflow error. When set, indicates that the transmitter has truncated a message due to data late from memory. UF indicates that the SGEC encountered an empty transmit FIFO while in the midst of transmitting a frame. The transmission process enters the SUSPENDED state and sets NICSR5<TI>.
<00>	DE	Deferred. When set, indicates that the SGEC had to defer while trying to transmit a frame. This condition occurs if the channel is busy when the SGEC is ready to transmit.

10.3.17.2 TDES1 Word

Table 10–34 TDES1 Bits

<table>
<tr><th>Bit</th><th>Name</th><th>Descriptor</th></tr>
<tr><td>&lt;31&gt;</td><td>CA</td><td>Chain address. When set, TDES3 is interpreted as another descriptor's VAX physical address. This allows the SGEC to process multiple, non-contiguous descriptor lists and explicitly "chain" the lists. Note that contiguous descriptors are implicitly chained.

In contrast to what is done for a Rx buffer descriptor, the SGEC clears neither the ownership bit TDES0&lt;OW&gt; nor one of the other bits of TDES0 of the chain descriptor after processing.

To protect against infinite loop, a chain descriptor pointing back to itself, is seen as <b>owned by the an host</b>, regardless of the ownership bit state.</td></tr>
<tr><td>&lt;30&gt;</td><td>VA</td><td>Virtual addressing. When set, TDES3 is interpreted as a virtual address. The type of virtual address translation is determined by the TDES1<VT> bit. The SGEC uses TDES3 and TDES2<Page Offset> to perform a VAX virtual address translation process to obtain the physical address of the buffer. When clear, TDES3 is interpreted as the actual physical address of the buffer:

<table>
<tr><th>VA</th><th>VT</th><th>Addressing Mode</th></tr>
<tr><td>0</td><td>x</td><td>Physical</td></tr>
<tr><td>1</td><td>0</td><td>Virtual - SVAPTE</td></tr>
<tr><td>1</td><td>1</td><td>Virtual - PAPTE</td></tr>
</table></td></tr>
<tr><td>&lt;29:28&gt;</td><td>DT</td><td>Data type. Indicates the type of data the buffer contains, according to the following table:

<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>00</td><td>Normal transmit frame data</td></tr>
<tr><td>10</td><td>Setup frame explained in Section 10.3.18</td></tr>
<tr><td>11</td><td>Diagnostic frame</td></tr>
</table></td></tr>
<tr><td>&lt;27&gt;</td><td>AC</td><td>Add CRC disable. When set, the SGEC does not append the CRC to the end of the transmitted frame. To take effect, this bit must be set in the descriptor where FS is set.

<b>NOTE</b>
<b>If the transmitted frame is shorter than 64 bytes, the SGEC adds the padding field and the CRC regardless of the <AC&gt; flag.</b></td></tr>
<tr><td>&lt;26&gt;</td><td>FS</td><td>First segment. When set, indicates the buffer contains the first segment of a frame.</td></tr>
</table>

Table 10–34 (Cont.) TDES1 Bits

Bit	Name	Descriptor
<25>	LS	Last segment. When set, indicates the buffer contains the last segment of a frame.
<24>	IC	Interrupt on completion. When set, the SGEC sets NICSR5<TI> after this frame has been transmitted. To take effect, this bit must be set in the descriptor where LS is set.
<23>	VT	Virtual type. In case of virtual addressing (TDES1<VA> = 1), indicates the type of virtual address translation. When set, the buffer address TDES3 is interpreted as a SVAPTE (system virtual address of the page table entry). When clear, the buffer address is interpreted as a PAPTE (physical address of the page table entry). Meaningful only if TDES1<VA> is set.
<22:0>	u	Unused. Ignored by the SGEC on reads. Never written.

10.3.17.3 TDES2 Word

Table 10–35 TDES2 Bits

Bit	Name	Descriptor
<31>	u	Unused. Ignored by the SGEC on reads. Never written.
<30:16>	BS	Buffer size. The size, in bytes, of the data buffer. If this field is 0, the SGEC skips over this buffer and ignores it. The frame size is the sum of all BS fields of the frame segments (between and including the descriptors having TDES1<FS> and TDES1<LS> set). **NOTE** **If the port driver wishes to suppress transmission of a frame, this field must be set to zero in all descriptors comprising the frame and prior to the SGEC acquiring them. If this rule is not adhered to, corrupted frames might be transmitted.**
<08:00>	PO	Page offset. The byte offset of the buffer within the page. Only meaningful if TDES1<VA> is set. **NOTE** **Transmit buffers can start on arbitrary byte boundaries.**

10.3.17.4 TDES3 Word

Table 10–36 TDES3 Bits

Bit	Name	Descriptor
<31:00>	SV/PV/PA	SVAPTE/PAPTE/Physical address. When DES1<VA> is set, TDES3 is interpreted as the address of the page table entry and used in the virtual address translation process. The type of the address system Vivtual address (SVAPTE) or physical address (PAPTE) is determined by TDES1<VT>. When TDES1<VA> is clear, TDES3 is interpreted as the physical address of the buffer. When TDES1<CA> is set, TDES3 is interpreted as the VAX physical address of another descriptor. NOTE **Transmit buffers can start on arbitrary byte boundaries.**

10.3.17.5 Transmit Descriptor Status Validity

Table 10–37 summarizes the validity of the transmit descriptor status bits regarding the transmission completion status:

Table 10–37 Transmit Descriptor Status Validity

Transmission	Tx Status Report						
Status	**LO**	**NC**	**LC**	**EC**	**HF**	**CC**	**(ES,TO,LE,TN,UF,DE)**
Underflow	X	X	V	V	X	V	V
Excessive collisions	V	V	V	V	V	X	V
Watchdog timeout	X	V	X	X	X	V	V
Internal loopback	X	X	V	V	X	V	V

V - Valid
X - Meaningless

10.3.18 Setup Frame

A setup frame defines SGEC Ethernet destination addresses. These addresses are used to filter all incoming frames. The setup frame is *never* transmitted over the Ethernet, nor looped back to the receive list. While the setup frame is being processed, the receiver logic temporarily disengages from the Ethernet wire. The setup frame size is *always* 128 bytes and must be wholly contained in a single transmit buffer. There are two types of setup frames:

1. Perfect filtering addresses (16) list
2. Imperfect filtering hash bucket (512) heads plus one physical address

10.3.18.1 First Setup Frame

A setup frame must be *queued* (placed in the transmit list with SGEC ownership) to the SGEC before the reception process is started, except for when the SGEC operates in promiscuous reception mode.

NOTE

The self test completes with the SGEC address filtering table fully set to zero. A reception process which starts without loading a setup frame rejects all the incoming frames except those with a destination physical address of 000000h .

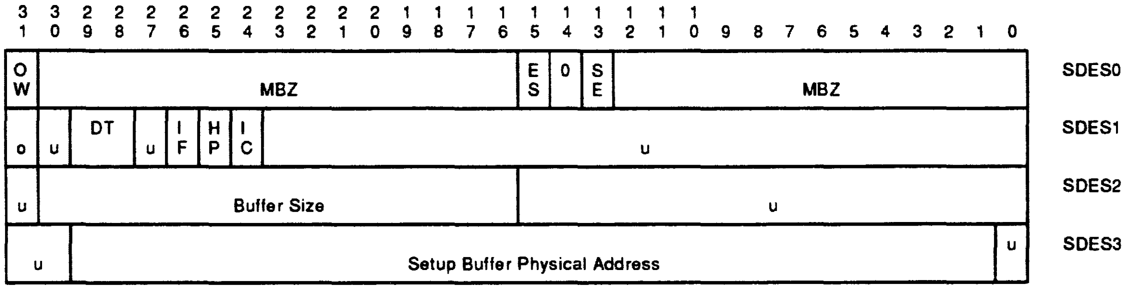
10.3.18.2 Subsequent Setup Frame

Subsequent setup frames may be queued to the SGEC regardless of the reception process state. The only requirement for the setup frame to be processed is that the transmission process be in the RUNNING state. The setup frame is processed after all preceding frames have been transmitted and after the current frame reception, if any, is completed.

The setup frame does not affect the reception process state but during the setup frame processing, the SGEC is disengaged from the Ethernet wire.

10.3.18.3 Setup Frame Descriptor

The setup frame descriptor format is shown in Figure 10–16 and described in the following paragraphs.



0 - SGEC writes as 0
u - Ignored by the SGEC on read, never written

Figure 10–16 Setup Frame Descriptor Format

Table 10–38 Setup Frame Descriptor Bits

Word	Bit	Name	Description
SDES0	<13>	SE	Setup error. When set, indicates the setup frame buffer size in not 128 bytes.
	<15>	ES	Error summary. Set when SE is set.
	<31>	OW	Own bit. When set, indicates the descriptor is owned by the SGEC. When cleared, indicates the descriptor is owned by the host. The SGEC clears this bit when processing of the descriptor and its associated buffer is complete.

Table 10–38 (Cont.) Setup Frame Descriptor Bits

Word	Bit	Name	Description
SDES1	<24>	IC	Interrupt on completion. When set, the SGEC sets NICSR5<TI> after this setup frame has been processed.
	<25>	HP	Hash/Perfect filtering mode. When set, the SGEC interprets the setup frame as a hash table, and does an imperfect address filtering. The imperfect mode is useful when there are more than 16 multicast addresses to listen to. When clear, the SGEC does a perfect address filter of incoming frames according to the addresses specified in the setup frame.
	<26>	IF	Inverse filtering. When set, the SGEC does an inverse filtering: The SGEC receives the incoming frames with destination address not matching the perfect addresses and will reject the frames with a destination address matching one of the perfect addresses. Meaningful only for perfect_filtering (SDES1<HP>=0), while promiscuous and all_multicast modes are not selected (NICSR6<AF>=0).
	<29:28>	DT	Data type. Must be 2 to indicate setup frame.
SDES2	<30:16>	BS	Buffer size. Must be 128.
SDES3	<29:1>	PA	Physical address. Physical address of setup buffer. **NOTE** **The setup buffer must be word aligned.**

10.3.18.4 Perfect Filtering Setup Frame Buffer

This section describes how the SGEC interprets a setup frame buffer when SDES1<HP> is clear.

The SGEC can store 16 (full 48 bits Ethernet) destination addresses. It compares the addresses of any incoming frame to these, and regarding the status of Inverse_Filtering flag SDES1<IF>, rejects the following:

- Those which do not match, if (SDES1<IF> = 0)
- Those which match, if (SDES1<IF> = 1)

The setup frame *must always* supply all 16 addresses. Any mix of physical and multicast addresses can be used. Unused addresses should be duplicates of one of the valid addresses. The addresses are formatted as shown in Figure 10–17.

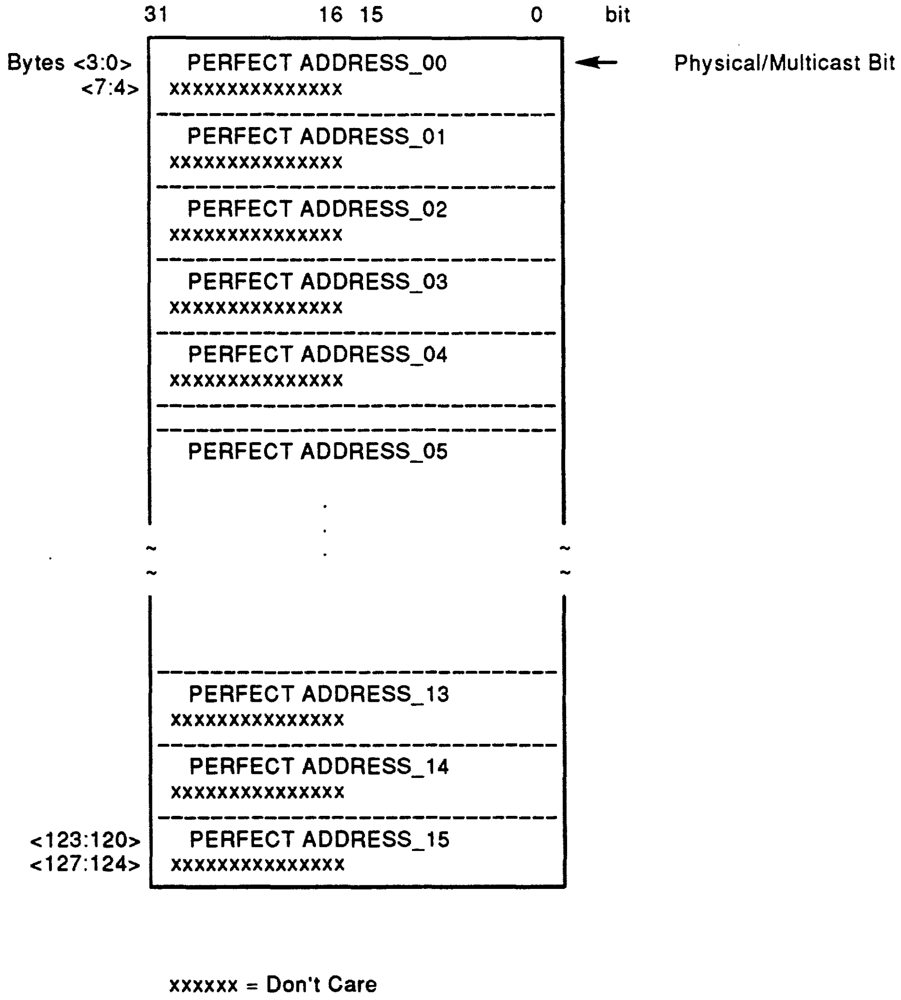


Figure 10–17 Perfect Filtering Setup Frame Buffer format

The low-order bit of the low-order bytes is the address's multicast bit.

Example 10–1 illustrates a perfect filtering setup buffer (fragment).

```
 Ethernet addresses to be filtered:
❶  A8-09-65-12-34-76
   09-BC-87-DE-03-15
    .
    .
    .

 Setup frame buffer fragment:
❷  126509A8
   00007634
   DE87BC09
   00001503
    .
    .
    .
```

❶ Two Ethernet addresses written according to the DEC STD 134 specification for address display.

❷ Those two addresses as they would appear in the buffer.

Example 10–1 Perfect Filtering Buffer

10.3.18.5 Imperfect Filtering Setup Frame Buffer

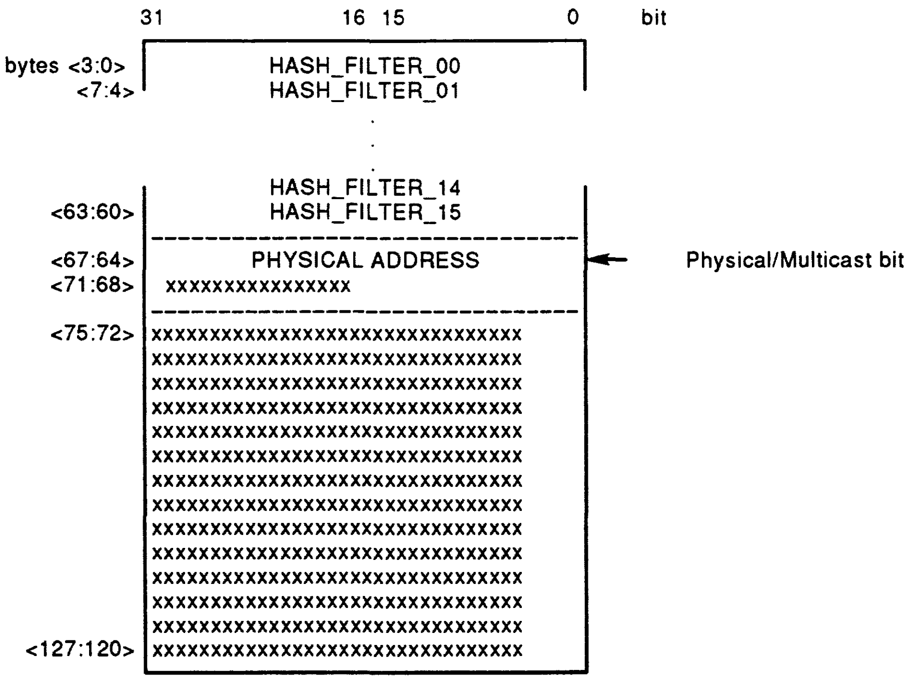
This section describes how the SGEC interprets a setup frame buffer when SDES1<HP> is set.

The SGEC can store 512 bits, serving as hash bucket heads, and one *physical* 48 bit Ethernet address. Incoming frames with multicast destination addresses are subjected to the imperfect filtering. Frames with physical destination addresses are checked against the single physical address.

For any incoming frame with a multicast destination address, the SGEC applies the standard Ethernet CRC function to the first six bytes containing the destination address, then uses the most significant nine bits of the result, as a bit index into the table. If the indexed bit is set, the frame is accepted. If it is cleared, the frame is rejected.

This filtering mode is called imperfect because multicast frames not addressed to this station may slip through, but it still decreases the number of frames the host is presented with.

The format for the hash table and the physical address is shown in Figure 10–18.



xxxxxx = don't care

Figure 10–18 Imperfect Filtering Setup Frame Format

Bits are sequentially numbered from right to left and down the table. For example, if CRC(destination address)<8:0> = 33, the SGEC examines bit #1 in the second longword.

Example 10–2 illustrates an imperfect filtering setup frame buffer.

```
  Ethernet addresses to be filtered:
❶   25-00-25-00-27-00
    A3-C5-62-3F-25-87
    D9-C2-C0-99-0B-82
    7D-48-4D-FD-CC-0A
    E7-C1-96-36-89-DD
    61-CC-28-55-D3-C7
    6B-46-0A-55-2D-7E

❷   A8-12-34-35-76-08

  Setup frame buffer:
❸   00000000
    10000000
    00000000
    00000000
    00000000
    40000000
    00000080
    00100000
```

Example 10–2 (Cont.) Imperfect Filtering Buffer

```
   00000000
   10000000
   00000000
   00000000
   00000000
   00010000
   00000000
   00400000
❹  353412A8
   00000876
```

❶ Ethernet multicast addresses written for address display

❷ An Ethernet physical address

❸ The first part of an imperfect filter setup frame buffer with set bits for the ❶ multicast addresses

❹ The second part of the buffer with the ❷ physical address

Example 10–2 Imperfect Filtering Buffer

Example 10–3 shows a C program to compute the hash bucket heads and create the resultant setup frame buffer.

```
#include <stdio>

unsigned int imperfect_setup_frame[128/4],    /* The setup buffer - 128   */
      /* bytes      */
  address[2],
  crc[33];   /* CRC residue vector     */

main()
{
    int i, hash;
/*          */
/* This program accepts 48 bits Ethernet addresses and builds a Setup frame */
/* buffer for imperfect filtering.         */
/*          */
/* Addresses must be entered in hexadecimal. The multicast bit is the least */
/* significant bit of the least significant digit of the first 32 bits.    */
/* Non-multicast addresses are ignored.         */
/*           */
/* Input is terminated by keying CTRL/Z after which the program prints out  */
/* the buffer.            */
/*          */
main_loop:

/* Prompt user for the Ethernet address         */
    printf("\n\n Enter the first 32 bits (HEX) - ");
    if (scanf("%x", &address[0]) == EOF)
 {

     printf("\n\n Imperfect Setup buffer printout\n");
     for (i=0;  i < 128/4;  i++)
  printf("%08X\n", imperfect_setup_frame[i]);
     exit(1);
 }
```

Example 10–3 (Cont.) Imperfect Filtering Setup Frame Buffer Creation C Program

```
    printf("\n Enter the remaining 16 bits (HEX) - ");
    scanf("%x",&address[1]);

/* Ignore non multicast addresses         */
     if ((address[0] & 1) == 0)
  goto main_loop;

/* Compute the hash function          */
    hash = address_crc(address[0],address[1]);

/* Set the appropriate bit in the Setup buffer        */
    imperfect_setup_frame[hash/32] =
 imperfect_setup_frame[hash/32] | 1 << hash%32;

    goto main_loop;
}

int address_crc( unsigned int lsb32 , unsigned int msb16)
{
    int j,hash = 0;

/* Set CRC to all 1's           */

    for (j=0;  j < 33;  j++)
 crc[j] = 1;

/* Compute the address CRC by running the CRC 48 steps       */

    for (j=0;  j < 32;  j++)
 nextstate(lsb32 & 1<<j ? 1 : 0);
    for (j=0;  j < 16;  j++)
 nextstate(msb16 & 1<<j ? 1 : 0);

/* Extract 9 most significant bits from the CRC residue      */

    for (j=24;  j < 33;  j++)
 hash = hash<<1 | crc[j];

    return hash;
}

nextstate(dat)
int dat;
{
  int i,mean;
  mean = crc[32] ^ dat;
  for(i=32;i>=2;i--) crc[i]=crc[i-1];
  crc[27] = crc[27] ^ mean;
  crc[24] = crc[24] ^ mean;
  crc[23] = crc[23] ^ mean;
  crc[17] = crc[17] ^ mean;
  crc[13] = crc[13] ^ mean;
  crc[12] = crc[12] ^ mean;
  crc[11] = crc[11] ^ mean;
  crc[9] = crc[9] ^ mean;
  crc[8] = crc[8] ^ mean;
  crc[6] = crc[6] ^ mean;
  crc[5] = crc[5] ^ mean;
  crc[3] = crc[3] ^ mean;
  crc[2] = crc[2] ^ mean;
  crc[1] = mean;
```

Example 10–3 Imperfect Filtering Setup Frame Buffer Creation C Program

10.3.19 SGEC Operation

10.3.19.1 Hardware and Software Reset

The SGEC responds to two types of reset commands: A hardware reset through the RESET_L pin, and a software reset command triggered by setting NICSR6<RE>. In **both cases**, the SGEC **aborts** all ongoing processing and starts the reset sequence. The SGEC restarts and reinitializes all internal states and registers. *No internal states are retained, no descriptors are owned, and all the host visible registers are set to zero, except where otherwise noted.*

NOTE
The SGEC does not explicitly disown any owned descriptor; so descriptors OWNED bits can be left in a state indicating SGEC ownership.

Table 10–39 indicates the NICSR fields which are not set to zero after reset:

Table 10–39 NICSR Fields Not Set to Zero After Reset

Field	Value
NICSR3	Unpredictable
NICSR4	Unpredictable
NICSR5<DN>	1
NICSR6<BL>	1
NICSR6<RE>	Unpredictable after HARDWARE reset
	1 after SOFTWARE reset
NICSR7	Unpredictable
NICSR9	RT = TT = 1250

After the reset sequence completes, the SGEC executes the self test procedure to do basic checking.

If the self test completes successfully, the SGEC initializes the SGEC, and sets the initialization done flag NICSR5<ID>.

At the first failure detected in one of the basic tests executed in the self_test routine, the test is aborted and the self_test failure NICSR5<SF> is set together with the self_test error status NICSR5<SS>which indicates the failure reason.

INFO
The self test takes 25 milliseconds to complete after a hardware or software RESET.

If the initialization completes successfully, the SGEC is ready to accept further host commands. Both the reception and transmission processes are placed in the STOPPED state.

Successive reset commands (either hardware or software) could be issued. The only restriction is that SGEC NICSRs should not be accessed during a one millisecond period following the reset. Access during this period results in a CP-BUS timeout error. Access to SGEC NICSRs during the self test is permitted; however, only NICSR5 reads should be performed.

10.3.19.2 Interrupts

Interrupts are generated as a result of various events. NICSR5 contains all the status bits which could cause an interrupt, provided NICSR6<IE> is set. The port driver must clear the interrupt bits (by writing a one to the bit position), to enable further interrupts from the same source.

Interrupts are *not queued*, and if the interrupting event reoccurs *before* the port driver has responded to it, no additional interrupts are generated. For example, NICSR5<RI> indicates *one or more* frames were delivered to host memory. The port driver should scan *all* descriptors, from its last recorded position up to the first SGEC owned one.

An interrupt is only generated *once* for simultaneous, multiple interrupting events. It is the port driver responsibility to scan NICSR5 for the interrupt cause(s). The interrupt is not *regenerated*, unless a *new* interrupting event occurs *after* the host acknowledged the previous one, and provided the port driver *cleared* the appropriate NICSR5 bit(s). For example, NICSR5<TI> and NICSR5<RI> could both set, the host acknowledges the interrupt and the port driver begins executing by reading NICSR5. Now NICSR5<RU> sets. The port driver writes back its copy of NICSR5, clearing NICSR5<TI> and NICSR5<RI>. After the host IPL is lowered below the SGEC level, another interrupt is delivered with the NICSR5<RU> bit set.

If the port driver clear *all* NICSR5 set interrupt bits before the interrupt has been acknowledged, the interrupt is suppressed.

10.3.19.3 Startup Procedure

The following sequence of checks and commands must be performed by the port driver to prepare the SGEC for operation:

1. Wait for the SGEC to complete its initialization sequence by polling on NICSR5<ID> and NICSR5<SF> (refer to Section 10.3.7 for details).
2. Examine NICSR5<SF> to find out whether the SGEC passes its self test. If it does not, it should be replaced (refer to Section 10.3.7 for details).
3. Write NICSR0 to establish system configuration dependent parameters (refer to Section 10.3.3 for details).
4. If the port driver intends to use VAX virtual addresses, NICSR7 must be written to identify the system page table to the SGEC (refer to Section 10.3.9 for details).
5. If the port driver changes the default settings of the watchdog timers, it must write to NICSR9 (refer to Section 10.3.11 for details).
6. The port driver must create the transmit and receive descriptor lists, then write to NICSR3 and NICSR4 to provide the SGEC with the starting address of each list. The first descriptor on the transmit list usually contains a setup frame (refer to Section 10.3.6 for details).
7. Write NICSR6 to set global operating parameters and start the transmission and reception processes. Theses processes enter the RUNNING state and attempt to acquire descriptors from the respective descriptor lists and begin processing incoming and outgoing frames (refer to Section 10.3.8 for details). The processes are independent of each other and can be started and stopped separately.

   CAUTION

   If address filtering (either perfect or imperfect) is desired, the reception process should only be started after the setup frame has been processed.

8. The port driver now waits for any SGEC interrupts. If either the reception or transmission processes were SUSPENDED, the port driver must issue the poll demand command after it has rectified the suspension cause.

10.3.19.4 Reception Process

While in the RUNNING state, the reception process polls the receive descriptor list, attempting to acquire free descriptors. Incoming frames are processed and placed in acquired descriptors' data buffers, while status information is written to the descriptor RDES0 words. The SGEC always tries to acquire an extra descriptor in anticipation of incoming frames. Descriptor acquisition is attempted under the following conditions:

- Immediately after being placed in the RUNNING state through setting of NICSR6<SR>
- The SGEC begins writing frame data to a data buffer pointed to by the current descriptor.
- The last acquired descriptor chained (RDES1<CA> = 1) to another descriptor.
- A virtual translation error was encountered RDES0<TN> while the SGEC was translating the buffer base address of the acquired descriptor.

As incoming frames arrive, the SGEC strips the preamble bits and stores the frame data in the receive FIFO. Concurrently, it performs address filtering according to NICSR6 fields AF, HP, and its internal filtering table. If the frame fails the address filtering, it is ignored and purged from the FIFO. Frames which are shorter than 64 bytes, due to collision or premature termination, are also ignored and purged from the FIFO, unless NICSR6<PB> is set.

After 64 bytes have been received, the SGEC begins transferring the frame data to the buffer pointed to by the current descriptor. If data chaining is enabled (NICSR6<DC> clear(, the SGEC writes frame data overflowing the current data buffer into successive buffer(s). The SGEC sets the (RDES0<FS> and RDES0<LS>) in the first and last descriptors, respectively, to delimit the frame. Descriptors are released (RDES0OW>) bit cleared) as their data buffers fill up or the last segment of a frame has been transferred to a buffer.

The SGEC sets RDES0<LS> and the RDES0 status bits in the last descriptor it releases for a frame. After the last descriptor of a frame is released, the SGEC sets NICSR5<RI>.

This process is repeated until the SGEC encounters a descriptor flagged as owned by the host. After filling up all previously acquired buffers, the reception sets NICSR5<RU> and enters the SUSPENDED state. The position in the receive list is retained.

Any incoming frames while in this state causes the SGEC to fetch the current descriptor in the host memory. If the descriptor is now owned by the SGEC, the reception re-enters the RUNNING state and starts the frame reception.

If the descriptor is still owned by the host, the SGEC increments the Missed Frames Counter (NICSR10<MFC>) and discards the frame.

Table 10–40 summarizes the reception process state transitions and resulting actions:

Table 10–40 Reception Process State Transitions

From State	Event	To State	Action
STOPPED	Start reception command	RUNNING	Receive polling begins from last list position or from the list head if this is the first start command issued, or if the receive descriptor list address (NICSR3) was modified by the port driver.
RUNNING	SGEC attempts acquisition of a descriptor owned by the host	SUSPENDED	NICSR5<RU> is set when the last acquired descriptor buffer is consumed. Position in list is retained.
RUNNING	Stop reception command	STOPPED	Reception process is STOPPED after the current frame, if any, is completely transferred to data buffer(s). Position in list is retained.
RUNNING	Memory or host bus parity error encountered	STOPPED	Reception is cut off and NICSR5<ME> is set.
RUNNING	Reset command	STOPPED	Reception is cut off.
SUSPENDED	Rx Poll demand or Incoming frame and available descriptor	RUNNING	Receive polling resumes from last list position or from the list head if NICSR3 was modified by the port driver.
SUSPENDED	Stop reception command	STOPPED	None.
SUSPENDED	Reset command	STOPPED	None.

10.3.19.5 Transmission Process

While in the RUNNING state, the transmission process polls the transmit descriptor list for any frames to transmit. Frames are built and transmitted on the Ethernet wire. Upon completing frame transmission (or giving up), status information is written to the TDES0 words. Once polling starts, it continues (in sequential or descriptor chained order) until the SGEC encounters a descriptor flagged as owned by the host, or an error condition. At this point, the transmission process is placed in the SUSPENDED state and NICSR5<TI> is set.

NICSR5<TI> is also set after completing transmission of a frame which has TDES1<IC> set in its last descriptor. In this case, the transmission process remains in the RUNNING state.

Frames may be data chained and span several buffers. Frames must be delimited by TDES1<FS> and TDES1<LS> in the first and last descriptors, respectively, containing the frame. While in the RUNNING state, as the transmission process starts, it first expects a descriptor with TDES1<FS> set. Frame data transfer from the host buffer to the internal FIFO is initiated. Concurrently, if the current frame had TDES1<LS> clear, the transmission process attempts to acquire the next descriptor, expecting TDES1<FS> and TDES1<LS> to be clear indicating an intermediary buffer, or TDES1<LS> to be set, indicating the end of the frame. After the last buffer of the frame has been transmitted, the SGEC writes back final status information to the TDES0 word of the descriptor having TDES1<LS> set, optionally sets NICSR5<TI> if TDES1<IC> was set, and repeats the process with the next descriptor(s). Actual frame transmission begins *after at least 72 bytes* have been transferred to the internal FIFO, or *a full frame is contained* in the

FIFO. Descriptors are released (TDES0<OW> bit cleared) as soon as the SGEC is through processing a descriptor.

Transmit polling suspends under the following conditions:

- The SGEC reaches a descriptor with TDES0<OW> clear. To resume, the port driver must give descriptor ownership to the SGEC and issue a poll demand command.
- The TDES1<FS> and TDES1<LS> are incorrectly paired or out of order. TDES0<LE> is set.
- A frame transmission is given up due to a locally induced error. The appropriate TDES0 bit is set.

The transmission process enters the SUSPENDED state and sets NICSR5<TI>. Status information is written to the TDES0 word of the descriptor causing the suspension. The position in the transmit list, in all of these cases, is retained. The retained position is that of the *descriptor following the last descriptor closed (set to host ownership) by the SGEC.*

NOTE
The SGEC *does not* automatically poll the Tx descriptor list and the port driver *must* explicitly issue a Tx poll demand command after rectifying the suspension cause.

The following table summarizes the transmission process state transitions:

Table 10–41 Transmission Process State Transitions

From State	Event	To State	Action
STOPPED	Start transmission command.	RUNNING	Transmit polling begins from the last list position or from the head of the list if this is the first start command issued, or if the transmit descriptor list address (NICSR4) was modified by the port driver.
RUNNING	SGEC attempts acquisition of a descriptor owned by the host.	SUSPENDED	NICSR5<TI> is set. Position in list is retained.
RUNNING	Out-of-order delimiting flag (TDES0 <FS> or TDES0<LS>) encountered.	SUSPENDED	TDES0<LE> and NICSR5<TI> are set. Position in list is retained.
RUNNING	Frame transmission aborts due to a locally induced error.	SUSPENDED	Appropriate TDES0 and NICSR5<TI> bits are set. Position in list retained.
RUNNING	Stop transmission command.	STOPPED	Transmission process is STOPPED after the current frame, if any, is transmitted. Position in list is retained.

Table 10–41 (Cont.) Transmission Process State Transitions

From State	Event	To State	Action
RUNNING	Transmit watchdog expires.	STOPPED	Transmission is cut off and NICSR5<TW> , TDES0<TO> are set. Position in list is retained.
RUNNING	Memory or host bus parity error encountered.	STOPPED	Transmission is cut off and NICSR5<ME> is set.
RUNNING	Reset command.	STOPPED	Transmission is cut off.
SUSPENDED	Tx Poll demand command.	RUNNING	Transmit polling resumes from last list position or from the list head if NICSR4 was modified by the port driver.
SUSPENDED	Stop transmission command.	STOPPED	None.
SUSPENDED	Reset command.	STOPPED	None.

10.3.19.6 Loopback Operations

The SGEC supports two loopback modes:

- Internal loopback

  This mode is generally used to verify correct operations of the SGEC internal logic. While in this mode, the SGEC takes frames from the transmit list and loops them back, internally, to the receive list. The SGEC is disengaged from the Ethernet wire while in this mode.

- External loopback

  This mode is generally used to verify correct operations up to the Ethernet cable. While in this mode, the SGEC takes frames from the transmit list and transmits them on the Ethernet wire. Concurrently, the SGEC listens to the line which carries its own transmissions and places incoming frames in the receive list.

  NOTE
  Caution should be exercised in this mode as transmitted frames are placed on the Ethernet wire. Furthermore, the SGEC does not check the origin of any incoming frames, consequently , frames not necessarily originating from the SGEC might make it to the receive buffers.

In either of these modes, all the address filtering and validity checking rules apply. The port driver needs to take the following actions:

1. Place the reception and transmission processes in the STOPPED state. The port driver must wait for any previously scheduled frame activity to cease. This is done by polling the TS and RS fields in NICSR5.
2. Prepare appropriate transmit and receive descriptor lists in host memory. These may follow the existing lists at the point of suspension, or may be new lists which will have to be identified to the SGEC by appropriately writing NICSR3 and NICSR4.

3. Write to NICSR6<OM> according to the desired loopback mode and place the transmission and reception processes in the RUNNING state through Start commands.
4. Respond and process any SGEC interrupts, as in normal processing.

To restore normal operations, the port driver must execute above step one, then write the OM field in NICSR6 with 00.

10.3.19.7 DNA CSMA/CD Counters and Events Support

This section describes the SGEC features that support the port driver in implementing and reporting the specified counters and events.

Table 10–42 CSMA/CD Counters

Counter	SGEC feature
Time since counter creation	Supported by the host driver.
Bytes received	Port driver must add up the RDES0<FL> fields of all successfully received frames.
Bytes sent	Port driver must add up the TDES2<BS> fields of all successfully transmitted buffers.
Frames received	Port driver must count the successfully received frames in the receive descriptor list.
Frames sent	Port driver must count the successfully transmitted frames in the transmit descriptor list.
Multicast bytes received	Port driver must add up the RDES0<FL> fields of all successfully received frames with multicast address destinations.
Multicast frames received	Port driver must count the successfully received frames with multicast address destinations.
Frames sent, initially deferred	Port driver must count the successfully transmitted frames with TDES0<DE> set.
Frames sent, single collision	Port driver must count the successfully transmitted frames with TDES0<CC> equal to 1.
Frames sent, multiple collisions	Port driver must count the successfully transmitted frames with TDES0<CC> greater than 1.
Send failure - excessive collisions	Port driver must count the transmit descriptors having TDES0<EC> set.
Send failure - carrier check failed	Port driver must count the transmit descriptors having TDES0<LC> set.
Send failure - short circuit	Two successive transmit descriptors with the No_ carrier flag TDES0<NC> set indicate a short circuit.
Send failure - open circuit	Two successive transmit descriptors with the Excessive_collisions flag TDES0<EC> set with the same time domain reflectometer value TDES0<TDR> indicate an open circuit.
Send failure - Remote Failure to Defer	Flagged as a late collision TDES0<LC> in the transmit descriptors.
Receive failure - Block Check Error	Port driver must count the receive descriptors having RDES0<CE> set with RDES0<DB> cleared.

Table 10–42 (Cont.) CSMA/CD Counters

Counter	SGEC feature
Receive failure - Framing Error	Port driver must count the receive descriptors having both RDES0<CE> and RDES0<DB> set.
Receive failure - Frame too long	Port driver must count the receive descriptors having RDES0<TL> set.
Unrecognized frame destination	Not applicable.
Data overrun	Port driver must count the receive descriptors having RDES0<OF> set.
System buffer unavailable	Reported in the Missed_frame counter NICSR10<MFC> (refer to Table 10–20).
User buffer unavailable	Not applicable.
Collision detect check failed	Port driver must count the transmit descriptors having TDES0<HF> set.

CSMA/CD specified events can be reported by the port driver based on the above table. The initialization failed event is reported through NICSR5<SF>.

11
KA660 Mass Storage Interface

The KA660 contains a DSSI bus interface which is implemented with the single host adapter chip (SHAC). This interface allows the KA660 to transmit packets of data to, and receive packets of data from, as many as seven other DSSI devices (typically RF type disk drives and TF type streaming tape drives). It should also be noted that the SHAC supports CP bus parity protection.

11.1 Single Host Adapter Chip Introduction

A single host adapter chip (SHAC) is a single-chip, VLSI version of an SCA port that uses a DSSI bus as the physical interconnect. Another SCA realization, CI, has defined a port-driver/port interface which has been used to connect VAX computers in clusters. DSSI has adopted the same interface, so the same VMS operating system port driver can drive either a CI-port or SHAC. The SHAC can be used to connect a host to any other device that can communicate through the CI-DSSI protocol. In particular, it provides a solution to the following:

- The problem of interfacing a group of mass-storage device controllers (MSDCs) to a VAX.
- The problem of interfacing several VAX computers to a common group of MSDCs and, if higher level protocols support this option, to one another.

Where two or more VAX computers connect to a group of MSDCs (or to one another) through DSSI, each has a SHAC or another DSSI port. When a group of MSDCs connect to the DSSI bus, the controllers provide both the bus interface and the intelligent control required to respond to the CI commands received over the DSSI.

On the 1-byte wide DSSI bus both the MSDCs and the several VAX computers communicate at high-speed, with a 4 to 5 Megabyte burst transfer rate. The SHAC handles the problem of providing the interface between this DSSI bus and the SOC CPU having direct host memory access (DMA) over the host's 32-bit wide, 16 Megabyte CP bus. All communications between those connected to the DSSI follow the CI protocol with the DSSI protocols providing handshaking in the transactions.

Structural parameters limit the number of possible combinations that can be realized with DSSI and SHAC.

- A single DSSI bus has room for eight nodes which may be partitioned among host adapters (for example, SHACs) and MSDCs.
- As many as four SHACs can be installed on a single host bus.
- Because there must be a host, there can be as many as enen MSDCs on a single DSSI.

The SHAC provides a small amount of buffering (1.2 KByte) on chip to improve bus utilization on both sides, but the SHAC is designed to pass data from one bus to the other as rapidly as the two busses permit. DMA services to and from the main memory reside in the SHAC, which responds to requests for transfers between the host and the remote nodes.

The SHAC is operated by an on-chip RISC that obtains its code and internal data from on-chip RAM and ROM. The RAM is loaded from main memory both during initialization and as circumstances require during normal run time. With this capability, it can read in new code and data from the main memory and thus adapt its behavior to new circumstances. This permits inexpensive upgrades of SHACs after they are installed in the field. Furthermore, it allows the SHAC to store infrequently accessed code in main memory, providing more capability than could be included in on-chip ROM.

The overall communication architecture under which the SHAC works is Digital's Systems Communications Architecture (SCA). In this general architecture, four layers are defined, as shown in Figure 11–1. The architecture can be realized in a variety of ways. Two particularizations of the lowest two levels in the diagram are Computer Interconnect (CI) and DSSI. They share the same lowest host layer (CI port driver) but have distinctly different physical interconnects. The layers between the port driver and the DSSI bus itself can be realized at both board and chip level. The SHAC is a chip-level product which connects the host-bus to the DSSI bus, controlled by the SOC CPU through a CI port driver and accepts and delivers CI-defined packets over the DSSI bus. Layers above the port driver are invisible to SHAC.

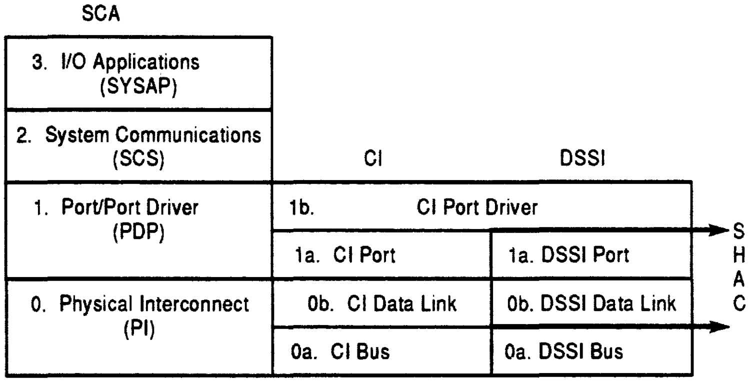

SCA
3. I/O Applications (SYSAP)
2. System Communications (SCS)
1. Port/Port Driver (PDP)
0. Physical Interconnect (PI)
CI
DSSI
1b. CI Port Driver
1a. CI Port
1a. DSSI Port
0b. CI Data Link
0b. DSSI Data Link
0a. CI Bus
0a. DSSI Bus
S
H
A
C

Figure 11–1 Relationship of the DSSI to SCA and CI

The port driver maintains a set of seven queues in its system space. Four of these contain commands for the SHAC to execute. The priority of the command is determined by the queue it is on; order is determined by the position in the queue. Another queue contains all of the responses for the host (from the SHAC or the remote nodes). Finally, there are two queues of "empty envelopes" for the host and the SHAC to use to stuff with commands and responses and then to queue them on the other queues.

These envelopes are simply standard-sized "queuable" blocks of host memory. All commands and responses are copied into one of these standard-sized blocks. Included in the header on each block are a pair of queue pointers (for a doubly linked queue) and various standard identifiers which specify what is contained in the block and how much of the block represents the actual command or response. To be visible, a block must be on a queue, where pointers from other elements or the queue header show its presence. Once a block is removed from a queue, it is visible only to the entity which removed it.

The SHAC's principal task is in accepting and delivering "mail" to other nodes. Externally (for example, on DSSI) the SHAC deals only in standard CI formats. Internally, the SHAC deals with the envelopes just described and with blocks of data. Because DSSI deals with bytes and the CP bus deals in longwords, the SHAC must frequently do byte alignment tasks during transcription.

The SHAC deals with the port driver in the virtual-address mode, unloading from the SOC CPU the obligation to do virtual-to-physical address translation and to be aware of page crossings in virtually-contiguous blocks of information. The SHAC supports full virtual address translation including the use of global I/O pages (to a depth of one).

The rest of this SHAC overview section describes a typical set of steps that the SHAC goes through in serving its role as the CI Port, with mail in both directions.

11.2 CI-DSSI Overview

At start-up, the host provides the SHAC with a number of pointers to internal host structures. One of the structures, the port queue block (PQB), contains pointers and data on all the queues that the host maintains for CI. The SHAC uses this data to carry on its normal business in the following way:

If traffic is not coming in on the DSSI bus, the SHAC goes to the highest command queue which has something enqueued. Choices are CMDQ0..CMDQ3, with 3 being most urgent. It dequeues an element from the queue and examines its header to see what it must do with the queue entry. It could be a command for the SHAC or an item to be delivered to one of the nodes on the DSSI. A command might be an order to deliver a block of data to a remote node. An item to be delivered would be either a datagram or a message.

A *datagram* is a "one-sided" communication—that is, one which will be sent without any assurance of either receipt or reply. An obvious application for such a communication is a request for the party at the other node to identify itself. If the host does not know if anything at all is out there, it must transmit its request without expectation. For this or any similar purpose, it employs a datagram. Datagrams lengths are guaranteed to fit in a datagram envelope.

A *message* is a "two-sided" communication used when a virtual circuit (an established formal relationship) between members of the bus exist. Once such a virtual circuit is established, the host(s) understand how to make requests of the other side. Such a request could be an order for a data transfer in either direction. The message itself (*move data*) is contained in a command (*deliver this message to* ...). Message lengths are guaranteed to fit in a message envelope. Messages are always delivered sequentially to a given node—that is, in the order in which they were enqueued on a particular queue. While the SHAC supports retries if a message fails to get through, once the retry limit is reached without successful delivery, SHAC returns the command to the host, marking it as *undeliverable*, and then breaks the virtual circuit to that node.

An example of full transaction would occur as follows:

1. The host queues a message for node 3 (for example, a disk controller) to copy a block of 16 Kbyte from host memory, starting at location X and to be stored in location Y on disk. The queues are doubly-linked, so at the top of every envelope there is a forward link **FLINK** and a backward link **BLINK**. Enqueuing involves putting link values into the new element's FLINK and BLINK and making the previous last-element's FLINK and the queue header's BLINK point to the new element.

2. When this message gets to the head of the queue, the SHAC dequeues it [1], reads the header and finds that it should "dial up" node 3. To do this, the SHAC goes through the DSSI protocols, contending for the DSSI bus and then, if successful in getting the bus, specifying node 3 as the target. These steps are called *arbitration* and *selection*.

3. Node 3 responds by asking for the DSSI-command (*command-out phase*). In this phase, the SHAC tells node 3 how many bytes are coming and repeats the identification information to confirm a proper selection. Node 3 then tells the SHAC to switch to the *data-out phase*. The SHAC sends a pair of CI header bytes to identify what type of message this is, and then proceeds to transmit the actual message read from the message block in host memory. The step-by-step details of the transfer are handled by hardware in the SHAC which permits simultaneous, buffered reading and writing on the two busses to which the SHAC is connected. Upon proper completion of the transmission, node 3 responds with a 1-byte acknowledgment of success (parity and check-sum proper and no other errors).

4. The SHAC is still holding the only pointer to the message block in host memory. It returns this to the host in one of two ways. If the host has requested a "return receipt," the SHAC puts the block on the response queue **RSPQ** to indicate proper delivery. This is where the port-driver software in the host looks for responses.

   Alternatively, the SHAC simply puts it back on the MFREEQ which holds the standard envelopes for messages. At this point the single message has been delivered and the message envelope is back in circulation.

5. After whatever delay node 3 needed to process the message, it contends for the bus and upon winning it, selects the SHAC as its target. It then sends a standard CI message as described telling the SHAC to transmit the required data. In general, the SHAC does not do this immediately, because it is obliged to handle traffic according to position in the queue and according to queue priority. Instead, it takes an empty envelope from MFREEQ, writes into it the message it is receiving, and puts it on the proper CMDQ as specified in the message it just received.

6. When that message gets to the head of its queue, the SHAC dequeues it once more, carries out its command (in transmissions of 4 Kbyte whenever possible (a 4 Kbyte transmission takes about 1 millisecond on the DSSI), possibly interleaving other transmissions of higher priority to this node or any priority to other nodes, until the last byte is sent. Once the SHAC has completed this operation, it returns the message block to the MFREEQ.

7. Node 3 has put its data on the disk and must report to the host the successful completion of the transaction. Again it contends for the bus and upon winning specifies the SHAC as its target. Then it sends a message to the port-driver through the SHAC confirming the successful transaction. The SHAC dequeues another free envelope and writes this message into that block. Then it queues it on the host's RSPQ. Except for higher level responses in the host, that concludes a whole transaction.

The enqueue/dequeue operations represent a considerable fraction of the effort in delivering a message or datagram. To minimize this effort, the SHAC caches a small number of the envelopes (that is, it hangs onto the pointers to the memory blocks) as they become free in its normal activity. It only fetches an envelope from the free queues when its own supply has disappeared, and it only returns them to the free queues when it has a full supply (four of a type). By this and other attention to effort reduction and traffic conservation, the SHAC attempts to optimize its rate of doing useful work.

[1] Note that the SHAC ends up holding the only pointer to the dequeued block of memory that constitutes the queue element. The port driver no longer "knows" where it is.

11.3 SHAC Registers

The CPU communicates directly with the SHAC chip through a set of device registers in each SHAC. These registers occupy a one-page (512-byte) region in I/O address-space, aligned on a page boundary.

All of the registers are longword registers. They may be accessed only through longword operations.

In addition to the access restrictions listed for specific registers, no register other than SHAC software chip reset (SSWCR) may be read or written while certain chip intialization functions are being executed. The results of such an access during the 100 milliseconds following a reset (power-up or a write to SSWCR), or during the 50 microseconds following a MIN-bit (PMCSR<0>) reset are UNPREDICTABLE.

The registers can be divided into two categories:

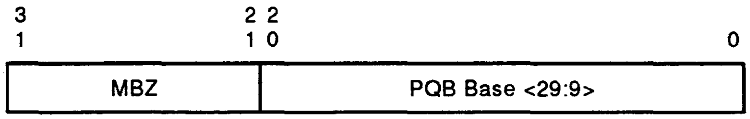
- The CI port registers
- The SHAC specific registers

11.3.1 CI Port Registers

11.3.1.1 Port Queue Block Base Register (PQBBR)
SHAC I/O Address: 2000 4248$_{16}$

This port queue block base register (PQBBR) contains the uppermost bits of the physical address of the base of the port queue block (PQB). After a RESET the PQBBR is loaded by the SHAC with configuration information. This information remains in the PQBBR until the PQBBR is written with the address of the port queue block. Figure 11–2 shows the format. Table 11–1 lists the bit descriptions.

PQBBR is writeable only when the port is in the disabled or disabled/maintenance state and readable anytime (except during chip intialization).



Longword Read/Write Access.

Figure 11–2 Port Queue Block Base Register (PQBBR)

Table 11–1 Port Queue Block Base Address Register (PQBBR)

Data Bit	Name	Description
<31:21>	MBZ	Read as zero. Must be written as zero.
<20:0>	PQB Base <29:9>	This field contains the uppermost bits of the physical address of the base of the PQB. Note, the PQB must be page-aligned, so the remaining bits of the address are assumed to be zero.

Following chip reset, PQBBR contains the configuration shown in Figure 11–3. The bit descriptions are listed in Table 11–2.

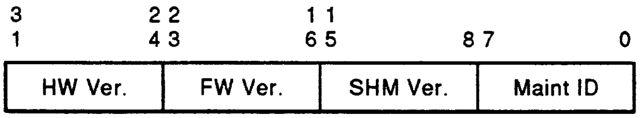


Figure 11–3 Port Queue Block Base Register (PQBBR) After RESET

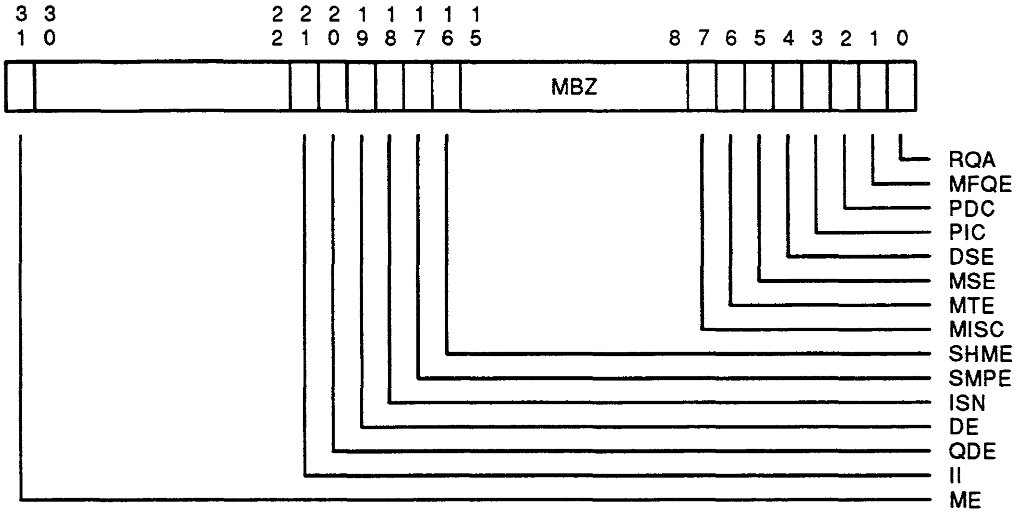
Table 11–2 Port Queue BLock Base Address Register Bits After RESET

Data Bit	Name	Description
<31:24>	HW Ver.	Hardware version. The hardware version of the SHAC which is greater than zero.
<23:16>	FW Ver.	Firmware version. The Firmware version of the SHAC which is greater than zero.
<15:8>	SHW Ver.	Shared host memory version. The shared host memory version of the SHAC which is zero until the shared host memory data area has been read in; thereafter, greater than zero.
<7:0>	Maint ID	CI port maintenance ID. The CI port maintenance ID which should always be 22 $_{16}$.

11.3.1.2 Port Status Register (PSR)
SHAC I/O Address: 2000 424C$_{16}$

The port status register (PSR) contains a status report. If interrupts are enabled, for example (PMCSR<2>) set, the port interrupts the SOC CPU each time that it writes to this register. Once an interrupt is requested by the port, the value of PSR is fixed and is not changed until the SOC CPU releases it by writing the port status release control register (PSRCR). The port status register format is shown in Figure 11–4 and the bit descriptions are in Table 11–3.

PSR is read only and may be read anytime by the port driver, except during chip initialization. Its value following a write to it is UNPREDICTABLE.



Longword Read Only Access.

ESB90P0071

Figure 11–4 Port Status Register (PSR)

Table 11-3 Port Status Register Bit Descriptions

Data Bit	Name	Description
<31>	MTE	Maintenance error. When set, the port has detected an implementation specific error (or hardware status condition). The source of the error may be more accurately determined from the other bits in the upper word of this register (PSR) and the contents of other registers. Also when set the port is in the *uninitialized* state (port is non-functional). Maintenance errors normally indicate a severe SHAC hardware or software failure.
<30:22>	MBZ	Read as zero, writes have no effect.
<21>	II	Illegal interrupt. When set, this bit indicates a SHAC internal error, detected when the SHAC's microprocessor received an interrupt from a invalid source. This causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (port is non-functional).
<20>	QDE	QUIP detected error. When set this bit indicates a SHAC internal error detected when the SHAC's microprocessor (QUIP) was given an invalid instruction. This causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (port is non-functional).
<19>	DE	Diagnostic error. When set an error was detected while the SHAC was running its internal self-test. This causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (port is non-functional).
<18>	ISN	Illegal segment number. When set this indicates a SHAC internal error in which it attempted to load a non-existent external segment from the SHAC shared host memory. This causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (port is non-functional).

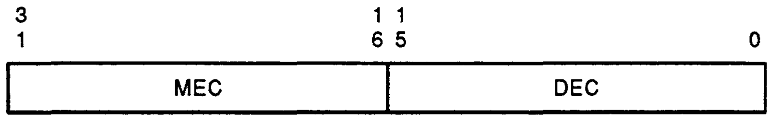
Table 11–3 (Cont.) Port Status Register Bit Descriptions

Data Bit	Name	Description
<17>	SMPE	Slave mode parity error. This bit is set by the occurrence of a parity error during a SOC CPU access of a SHAC device register. The causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (port is non-functional).
<16>	SHME	Share host memory error. This bit is set by the occurrence of an error involving the SHAC shared host memory. The causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (port is non-functional).
<15:8>	MBZ	Read as zero. Writes have on effect.
<7>	MISC	Miscellaneous. When set this bit indicates that the port microcode has detected one of the miscellaneous errors and the port is about to enter the *disabled/maintenance* state. The actual error code is stored in the port error status register.
<6>	ME	Maintenance timer expiration. When set the maintenance timer has expired. The port is in the *uninitialized/maintenance* state.
<5>	MSE	Memory system error. When set the port has encountered an uncorrectable data or non-existent memory error in referencing memory. Port is in the *disabled* or *disabled/maintenance* state. See the port failing address register (PFAR) for further information.
<4>	DSE	Data structure error. When set, the port has encountered an error in a port data structure (for example, queue entry, PQB, BDT, or page table). Port is in the *disabled* or *disabled/maintenance* state. See the port error status register (PESR) and the port failing address register (PFAR) for further information. Note that errors in queue structures leave the queues locked.
<3>	PIC	Port initialization complete. When set, the port has completed internal initialization. The port is in the *disabled* or *disabled/maintenance* state.
<2>	PDC	Port disable complete. When set, the port is in the *disabled* or *disabled /maintenance* state.
<1>	MFQE	Message free queue empty. When set, the port attempted to remove an entry from the message free queue (MFREEQ) and found it empty. Port processing of commands continues and, thus, the MFREEQ may not be empty at the time the port driver gets control.
<0>	RQA	Response queue available. When set this bit indicates port has inserted an entry on an empty Response Queue.

11.3.1.3 Port Error Status Register (PESR)
SHAC I/O Address: 2000 4250$_{16}$

The port error status register (PESR) indicates the type of error which resulted in a DSE (PSR<4>) or an MISC (PSR<7>) error. Figure 11–5 shows the format. Table 11–4 lists the bit descriptions.

PESR is read only by the SOC CPU and valid only after either a DSE or MISC error, or after certain ME (PSR<31>) and DE (PSR<19>) errors. Its value at any other time, or following a write to it, is UNPREDICTABLE.



Longword Read Only Access

Figure 11–5 Port Error Status Register (PESR)

Table 11–4 Port Error Status Register Bit Definitions

Data Bit	Name	Description
<31:16>	MEC	Miscellaneous error code. This code comprises two fields: bits <31:24> define the module within the SHAC code where the error occurred, and bits <23:16> contain the specific error that occurred. These codes are implementation specific.
<15:0>	DEC	Data structure error code.

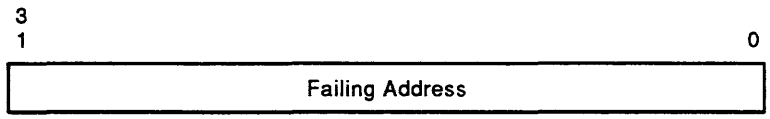
11.3.1.4 Port Failing Address Register (PFAR) SHAC I/O Address: 2000 4254_{16}

The format for the port failing address register is shown in Figure 11–6.

After an DSE, MSE, and ME or DE error (as indicated by PSR), or after a response with buffer memory system error status, the port failing address register (PFAR) contains the memory address at which the failure occurred. The address may be the exact failing address, an address in the same page as the exact failing address or, in the case of DSE, an address in some part of the data structure. For DSE, PFAR contains a virtual address or offset, while for MSE interrupts and buffer memory system errors the PFAR contains a physical address. For ME, the interpretation of the address is error-dependent.

Because the port continues command execution and packet processing after buffer memory system errors, the PFAR is overwritten if subsequent errors occur. For DSE, MSE, and ME errors the PFAR is effectively fixed because the port enters the *disabled*, *disabled/maintenance*, or *uninitialized* state.

PFAR is read only by the SOC CPU and readable after a DSE, MSE, or ME or DE errors, or after a response with buffer memory system error status. Its value at any other time, or following a write to it, is UNPREDICTABLE.

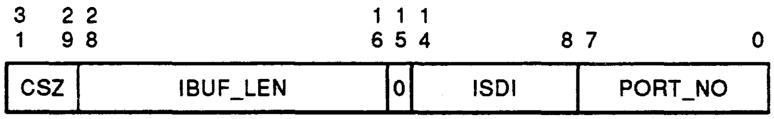


Longword Read Only Access

Figure 11–6 Port Failing Address Register (PFAR)

11.3.1.5 Port Parameter Register (PPR)
SHAC I/O Address: 2000 4258_{16}

The port parameter register (PPR) contains port implementation parameters and the port number. The value of the PPR is set by the port during initialization and valid after a PIC (PSR <3>) interrupt. Its value at any other time, or following a write to it, is UNPREDICTABLE. PPR is read only by the SOC CPU The PPR format is shown in Figure 11–7. The bit descriptions are listed in Table 11–5.



Longword Read Only Access

Figure 11–7 Port Parameter Register (PPR)

Table 11–5 Port Parameter Register Bit Descriptions (PPR)

Data Bit	Name	Description
<31:29>	CSZ	Cluster size. For SHAC, this value always is zero, indicating a maximum of 16 ports on the DSSI bus. (Note that the DSSI architecture only allows up to 8 ports on the bus, but 16 is the smallest size defined for the CSZ field.)
<28:16>	IBUF_LEN	Internal buffer length. This field indicates the size of internal buffers available for message and data transfers. Maximum data packet = IBUF_LEN - 16 bytes. Maximum message or datagram length = IBUF_LEN. For SHAC, the value is 4112 1010_{16}.
<15>	MBZ	Read as zero. Writes have an UNPREDICTABLE effect.
<14:8>	ISDI	Implementation specific diagnostic information. The bits in this field contain information about the local adapter's link layer configuration. For SHAC, the definitions of these bits are read as zero.
<7:0>	Port_NO	Port number. This is the same as the SHAC's DSSI ID.

11.3.1.6 Port Control Registers

The port control registers are 32-bit registers which are write-only by the SOC CPU To invoke the function provided by any of the control registers, the SOC CPU writes a one to the register.

The result of writing any other value to any of these registers is UNPREDICTABLE. The value read from any of them is also UNPREDICTABLE. The format for the port control registers is shown in Figure 11–8.

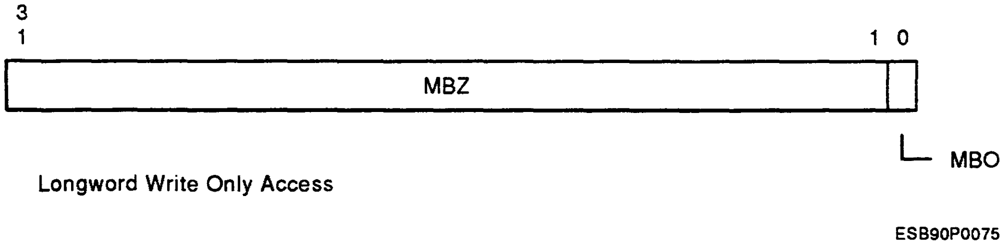


Figure 11–8 Port Control Registers

11.3.1.6.1 Port Command Queue 0 Control Register (PCQ0CR) SHAC I/O Address: 2000 4280_{16}

When the port driver inserts an entry in an empty CMDQ0, the port driver writes PCQ0CR to initiate port execution of the command queue. PCQ0CR can be written only when the port is in the *enabled* or *enabled/maintenance* state. Writing to PCQ0CR when the port is in any other state has no effect. The SHAC I/O address is 2000 4280_{16}.

11.3.1.6.2 Port Command Queue 1 Control Register (PCQ1CR) SHAC I/O Address: 2000 4284_{16}

Same as PCQ0CR except refers to CMDQ1. The SHAC I/O address is 2000 4284_{16}.

11.3.1.6.3 Port Command Queue 2 Control Register (PCQ2CR) SHAC I/O address: 2000 4288_{16}.

Same as PCQ0CR except refers to CMDQ2. The SHAC I/O address is 2000 4288_{16}.

11.3.1.6.4 Port Command Queue 3 Control Register (PCQ3CR) SHAC I/O Address: 2000 $428C_{16}$

Same as PCQ0CR except refers to CMDQ3. The SHAC I/O Address is 2000 $428C_{16}$.

11.3.1.6.5 Port Datagram Free Queue Control Register (PDFQCR) SHAC I/O Address: 2000 4290_{16}

When the port driver inserts an entry on the DFREEQ and the latter was previously empty, the port driver writes PDFQCR to indicate the availability of DFREEQ entries. PDFQCR can be written only if the port is in the *enabled* or *enabled/maintenance* state. Writing to PDFQCR when the port is in any other state has no effect. The SHAC I/O Address is 2000 4290_{16}

11.3.1.6.6 Port Message Free Queue Control Register (PMFQCR) SHAC I/O Address: 2000 4294_{16}

Same as PDFQCR except refers to MFREEQ. The SHAC I/O address is 2000 4294_{16}

11.3.1.6.7 Port Status Release Control Register (PSRCR)

SHAC I/O Address: 2000 4298_{16}

After the port driver has received an interrupt and read the PSR, it returns the PSR to the port by writing PSRCR. The SHAC I/O address is 2000 4298_{16}.

11.3.1.6.8 Port Enable Control Register (PECR)

SHAC I/O Address: 2000 $429C_{16}$

The port driver enables the port by writing PECR. PECR is ignored if the port is in the *uninitialized*, *uninitialized/maintenance*, *enabled*, or *enabled/maintenance* state. The SHAC I/O address is 2000 $429C_{16}$.

11.3.1.6.9 Port Disable Control Register (PDCR)

SHAC I/O Address: 2000 $42A0_{16}$

The port driver disables the port by writing PDCR. When the port is disabled, the port sets PDC (PSR <2>) and if interrupts are enabled requests an interrupt. PDCR is ignored if the port is in the *uninitialized*, *uninitialized/maintenance*, *disabled*, or *disabled/maintenance* state. The SHAC I/O address is 2000 $42A0_{16}$.

11.3.1.6.10 Port Initialize Control Register (PICR)

SHAC I/O Address: 2000 $42A4_{16}$

The port driver initializes the port by writing PICR. When the initialization is complete the port sets PDC (PSR <2>) and requests an interrupt if interrupts are enabled. As part of the initialization, the maintenance timer is set to expire in 100 seconds. The SHAC I/O address is 2000 $42A4_{16}$.

11.3.1.6.11 Port Maintenance Timer Control Register (PMTCR)

SHAC I/O Address: 2000 $42A8_{16}$

The port driver forces the maintenance timer to reset its expiration time by writing the PMTCR. If the PMTCR is not written again before the expiration time, the port will enter the *uninitialized/maintenance* state setting MTE (PSR <6>) and request an interrupt if interrupts are enabled. PMTCR is ignored if the maintenance timer is not running. The SHAC I/O address is 2000 $42A8_{16}$.

11.3.1.6.12 Port Maintenance Timer Expiration Control Register (PMTECR)

SHAC I/O Address: 2000 $42AC_{16}$

The port driver forces a maintenance-timer-expiration interrupt by writing the PMTECR. This register may be written only when the port is in the *enabled*, *enabled/maintenance*, *disabled*, and *disabled/maintenance* states and only while the maintenance timer is not disabled. The SHAC I/O address is 2000 $42AC_{16}$.

11.3.1.6.13 Port Maintenance Control and Status Register (PMCSR)

SHAC I/O Address: 2000 425C$_{16}$

The port maintenance control and status register (PMCSR) is used for maintenance level control and status reporting. The CI port specification defines all but the two least significant bits. The format is shown in Figure 11–9 and the bit descriptions are listed in Table 11–6

The bits can be divided into the following categories:

- Status bits. These are set by the port to report various conditions. They are cleared by maintenance initialization or clearing the condition in another register. PMCSR does not include any status bits at this time.
- Function control bits are read/write by the port driver only. They are clear on a RESET.

  There are two classes of these bits:

  1. Init: This type of bit invokes a function (for example, initialization) by setting it. It always reads as zero, except while the function is active.
  2. Enable/disable: This type of bit causes an activity or state to exist while the bit is set. Clearing the bit stops the activity or changes the state. The bit always reads the most recently written value. The bit is never changed by the port.

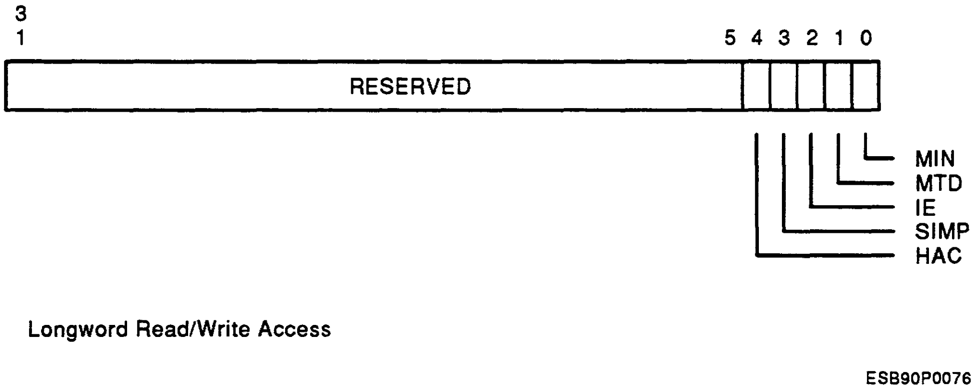


Figure 11–9 Port Maintenance Control and Status Register (PMCSR)

Table 11–6 Port Maintenance Control and Status Register (PMCSR) Bits

Data Bit	Name	Description
<31:5>	RESERVED	This bits are reserved. They should not be written; reads return UNPREDICTABLE results.
<4>	HAC	Host access feature. This bit must be zero, except for diagnostic purposes. This is an enable/disable class control bit.
<3>	SIMP	Simple SHAC Mode. *Must be* zero, except for diagnostic purposes. This is an enable/disable class control bit.

Table 11–6 (Cont.) Port Maintenance Control and Status Register (PMCSR) Bits

Data Bit	Name	Description
<2>	IE	Interrupt enable. When set, interrupts from the port to the SOC CPU are enabled. Power-up state is clear (interrupts disabled). This is an enable/disable class control bit.
<1>	MTD	Maintenance timer disable. Read/Write by SOC CPU. If set, the maintenance timer is turned off. Timer is set to the initial value and suspended. If clear, timer functions normally. Power-up state is clear (timer enabled). This is an enable/disable class control bit.
<0>	MIN	Maintenance init. Writing a one to this bit resets the port. Upon completion, the port is in the *uninitialized* state and MIN is clear. Writing a zero to this bit has no effect. It always reads as zero, except while the reset function is active. Although maintenance init resets the port, it is not equivalent to a write to the SHAC software chip reset register. In particular, the SHAC shared host memory address is not reset by maintenance init.

11.3.2 SHAC Specific Registers

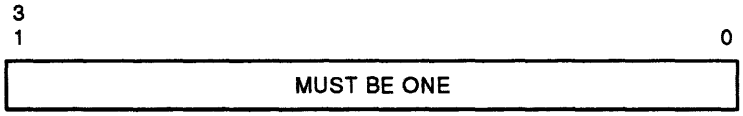
These registers, which are not defined in the CI port architecture, are used for additional maintenance level control.

11.3.2.1 SHAC Software Chip Reset Register (SSWCR)

SHAC I/O Address: 2000 4230_{16}

When the SOC CPU writes FFFF $FFFF_{16}$ to the SHAC software chip reset register (SSWCR), a chip reset is performed. The result is equivalent to that of the hardware chip reset that occurs following system power-up. On completion, all device registers are reset to their power-up state, and the port is in the *uninitialized* state. The format is shown in Figure 11–10.

SSWCR is write only by the SOC CPU and may be written to at any time. Its value when read is UNPREDICTABLE. The result if other than FFFF $FFFF_{16}$ is written to SSWCR is UNDEFINED.



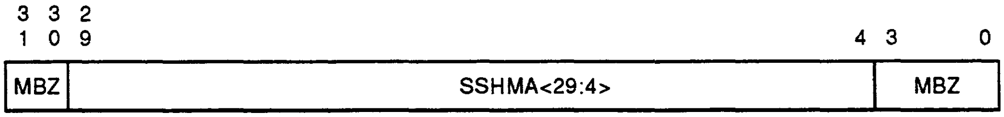
Longword Write Only Access

Figure 11–10 SHAC Software Chip Reset (SSWCR)

11.3.2.2 SHAC Shared Host Memory Address (SSHMA)

SHAC I/O Address: 2000 4244_{16}

The format for the SHAC shared host memory address is shown in Figure 11–11.



Longword Read/Write Access

Figure 11–11 SHAC Shared Host Memory Address (SSHMA)

Following chip reset, the SOC CPU writes into the SHAC shared memory address register (SSHMA) the physical address of the shared host memory header. The area must be octaword aligned and contiguous in physical memory.

SSHMA is read/write by the SOC CPU, but may be written only when the port is in the *uninitialized* state. Writing when the port is in any other state can produce unpredictable results.

12
KA660 Firmware

This chapter describes the KA660 functional firmware. The firmware is written in VAX–11 code which resides in the EPROM on the KA660 module. Typically KA660 firmware gains control whenever the onboard CPU halts, or more precisely, performs a processor restart operation. However, portions of the firmware can also be invoked by applications through a public subroutine linkage.

When the KA660 firmware is running, it provides services expected of a standard VAX console subsystem. In particular, the following services are available:

- Automatic restart or bootstrap of customer application images at power-up, on reset, or conditionally after processor halts
- Diagnostic tests executed both at power-up and by request, which verify the correct operation of the CPU and memory modules
- Operator interface providing complete examination or modification of the processor state

Conventions and Terminology

The following conventions and terminology are used in this chapter:

Convention	Meaning
<x:y>	Represents a bit field or position within a register or data structure. The bit field or position follows the structure name; the associated field name (if defined) typically follows the field definition and appears in parentheses. For example, PSL<20:16>(IPL) represents the five-bit field for the interrupt priority level (IPL) in the processor status longword (PSL).
x:y	Represents a range of integers from x through y.
20140030	Eight-digit numbers in this chapter are hexadecimal longwords, typically representing VAX-32 bit addresses or data.
456_{10},12_{16}	In sections where octal, decimal, and hexadecimal numbers may appear, the radix of a number is included to avoid confusion.

Terminology	Meaning
Firmware	Firmware is a generic term describing all program code located in the KA660 EPROM. It is sometimes referred to as the boot ROM, diagnostics ROM, or console ROM depending on the context.
Each major element of the firmware is referred to by other terms. For example:	
VMB, primary bootstrap	The boot program
Diagnostic, self-test	The ROM-based diagnostic program
Console, console program	The operator interface

12.1 General Description

The KA660 firmware provides the following services:

- Diagnostics that test all components on the board and verify that the module is working correctly
- Automatic/manual bootstrap of an operating system following processor halts
- Automatic/manual restart of an operating system following processor halts
- An interactive command language that allows the user to examine and alter the state of the processor
- Support of various terminals and devices as the system console
- Multilanguage support for displaying critical system messages and handling LK201 country-specific keyboards

The remainder of this section describes in detail the functions and external characteristics of the KA660 firmware.

The KA660 firmware is comprised of several major functional blocks of code. The halt **entry** code is invoked following system halts, resets, or severe errors. This code is responsible for saving the machine state and transferring control to the halt dispatcher code. The halt **dispatcher** code determines the nature of the halt, then transfers control to the appropriate subcode. The halt **exit** code is invoked whenever a transition is desired from a halted state to the running state. This code performs a restoration of the saved context prior to the transition. Figure 12–1 illustrates the KA660 firmware structural components and the functions are discussed in detail in Section 12.2.

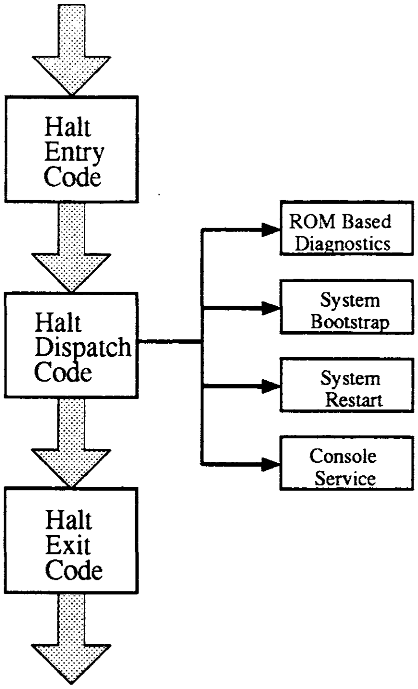


Figure 12–1 KA660 Firmware Structural Components

The *ROM-based diagnostics* consist of an initial power-up test and a series of functional component diagnostics invoked by a diagnostic executive. Section 12.3 describes the power-up and Section 12.8 describes the diagnostics.

Depending on the nature of the halt and the hardware context, the firmware attempts an *operating system restart* (Section 12.5), *a bootstrap operation* (Section 12.4), or transitions to *console I/O mode* (Section 12.6).

12.2 Halt Code

The main purpose of the halt code is to save the state of the machine on halt entry, invoke the dispatcher, and restore the state of the machine on exit to program I/O mode. Halt code is comprised of halt entry, halt dispatch, and halt exit codes.

12.2.1 Halt Entry - Saving Processor State

The entry code, residing at physical address 20040000, is executed whenever the KA660 halts. The value that the program counter contained when the processor was halted is saved in IPR 42 (PR$_SAVPC). On a power-up, the PR$_SAVPC register value is undefined.

The processor will halt for a variety of reasons. The reason for the halt is stored in PR$_SAVPSL<13:8>(RESTART_CODE), IPR 43. A complete list of the halt reasons and the associated console messages can be found in Table J–1 in Appendix J.

After a halt, the firmware first saves the current LED code then writes an "E" to the diagnostic LEDs. This action occurs within several instructions after the firmware has been invoked. The LED code is saved to let the user know that at least some instructions have been successfully executed.

The KA660 firmware unconditionally saves the contents of the following registers on any halt:

- R0 through R15, the general purpose registers
- PR$_SAVPSL, the saved PSL register
- PR$_SCBB, the system control block base register
- DLEDR, the diagnostic LED register

NOTE
The SSC programmable timer registers are not saved. In some cases, such as bootstrap, the timers are used by the firmware and previous "time" context is lost.

Several registers are unconditionally set to predetermined values by the firmware on any halt, processor init, or bootstrap. This action ensures that the firmware itself can run and it protects the board from physical damage.

The following is a list of such registers:

- The SSC configuration register (SSCCR)
- The SSC address match and mask registers (ADxMCH & ADxMSK)
- The CDAL bus timeout control register (CBTCR)
- The SSC timer interrupt vector registers (TIVRx)

Whenever the halt entry code is invoked, the firmware sets the console serial line baud rate based on the value read from the BDR and extends the halt protection from 8 Kbyte to 256 Kbyte to include all of the EPROM.

12.2.2 Halt Dispatch

The action taken by the firmware on a halt is depends primarily on the following information:

- The state of the BREAK enable switch, BDR<23>(HALT_ENABLE)
- The state of the console program mailbox, CPMBX<1:0>(HALT_ACTION)
- The user defined halt action (SET HALT)
- The halt code, PR$_SAVPSL<13:8>(RESTART_CODE)

In general, the BREAK enable switch governs whether or not a BREAK condition from the console serial line is recognized by the KA660. This switch also determines the default action taken on a power-up or other internal halt condition. By default, if BREAKs are enabled, the firmware invokes the console emulation code. If BREAKs are disabled, the firmware attempts a recovery operation.

However, the console program mailbox, CPMBX<1:0>(HALT_ACTION) (see Figure H–2) is used by operating systems to override the BREAK enable switch and instruct the firmware to invoke the console service, attempt to restart the operating system, or reboot the system following a halt, regardless of the setting of the BREAK enable switch.

The user-defined halt action invoked by using the SET HALT console command (see the description of the SET command in Section 12.7 is an alternative way to specify a default halt action. This feature allows users to specify auto-booting on power-ups, even when BREAKs are enabled.

For HALT instructions and error halt conditions, it is similar in function to the console program mailbox but has lower precedence and is only used when the console program mailbox is 0. This provides the user with a mechanism by which to specify what action should be taken, in the event that the operating system or user application does not set the console program mailbox.

The halt (or restart) code is automatically deposited in PR$_SAVPSL<13:8>(RESTART_CODE) on any halt condition. This field indicates the cause of the halt, and for the purpose of dispatching is divided into three categories:

02: External halts
03: Reset/power-up
xx: (All other values) HALT instruction and all error halts

Table 12–1 summarizes the action taken on all halt conditions, except external halts which are described in Section 12.2.2.1. The actual halt dispatch state machine is described in detail in Section I.1 of Appendix I.

Table 12–1 Halt Action Summary

Halt Code = 3	Break Enable Switch	User-Selected Halt Action (Number or Keyword)	Console Program Mailbox	Actions Performed by Firmware
T	1	0 (DEFAULT) 1 (RESTART) 3 (HALT)	x	• Run diagnostics • Invoke console code
T	1	2 (REBOOT) 4 (RESTART_ REBOOT)	x	• Run diagnostics • If successful, perform boot • If either fail, invoke console code
T	0	x	x	• Run diagnostics • If successful, perform boot • If either fail, invoke console code
F	1	0 (DEFAULT)	0	• Invoke console code
F	0	0 (DEFAULT)	0	• Perform restart • If restart fails, perform boot • If boot fails, invoke console code
F	x	1 (RESTART)	0	• Perform restart • If restart fails, invoke console code

Table 12–1 (Cont.) Halt Action Summary

Halt Code = 3	Break Enable Switch	User-Selected Halt Action (Number or Keyword)	Console Program Mailbox	Actions Performed by Firmware
F	x	2 (REBOOT)	0	• Perform boot • If boot fails, invoke console code
F	x	3 (HALT)	0	• Invoke console code
F	x	4 (RESTART_ REBOOT)	0	• Perform restart • If restart fails, perform boot • If boot fails, invoke console code
F	x	x	1	• Perform restart • If restart fails, invoke console code
F	x	x	2	• Perform boot • If boot fails, invoke console code
F	x	x	3	• Invoke console code

Halt Code = 3

T (true) indicates a reset or power-up condition.
F (false) indicates a HALT instruction or error halt condition.
x (don't care) indicates that the condition is "don't care".

Break Enable Switch

0 (off) indicates breaks are disabled.
1 (on) indicates breaks are enabled.
x (don't care) indicates that the condition is "don't care".

User Selected Halt Action

Numbers and keywords listed in this column can be used interchangeably.

Console Program Mailbox

This column reflects the value of the Console Program Mailbox. Refer to Section H.1.2 to interpret bit values.

Actions Performed by Firmware

These are the directions taken by the firmware after a halt.

Because the KA660 does not support battery backed up main memory, an operating system restart operation is not attempted on a power-up.

12.2.2.1 External Halts

Several conditions can trigger an external halt and different actions are taken depending on the condition.

An external halt can be caused by one of the following conditions:

1. A BREAK condition on the system console serial line, if the BREAK enable switch is set to "enabled". In this case BDR<23>(HALT_ENABLE) = 1 and the console code is invoked. **Ctrl/P may be established as the BREAK condition by using the SET CONTROLP ENABLE console command.**
2. The assertion of the BHALT line on the Q22-bus, causes an external halt if the SCR<14>(BHALT_ENABLE) bit in the CQBIC is set. As a result, the console code is invoked.
3. The negation of DCOK on the Q22-bus, if the SCR<7>(DCOK_ACTION) bit is set causes an external halt. (By default this bit is clear.) As a result, the console code is invoked.
4. Recognition of a valid MOP BOOT message by an appropriately initialized SGEC, if the REMOTE_BOOT_ENABLE jumper is in place (BDR<12>(REMOTE_BOOT_ENABLE) = 1). As a result, a bootstrap is attempted and if that fails, the console is entered.

   NOTE
   The firmware does not initialize the SGEC for this operation. The operating system must set up the SGEC to support this feature.

NOTE
The RESTART switch negates DCOK. The DCOK bit may also be negated by the DEQNA sanity timer, or any other Q22-bus module that chooses to implement the Q22-bus restart/reboot protocol. Because the SCR<7>(DCOK_ACTION) bit is cleared on power-up, the default consequence to deasserting DCOK is to generate a processor restart. Hence, pushing the RESTART button typically initiates a power-up sequence and destroys system state.

12.2.3 Halt Exit - Restoring Processor State

When the firmware exits, it uses the currently defined saved context. This context is initially determined by what was saved when the firmware code was invoked. However this context may be modified by console commands, or automatic operations such as an automatic bootstrap on power-up.

When restoring the context, the firmware will flush the CPU internal cache if enabled, and invalidate all translation buffer entries using the internal processor register PR$_TBIA, IPR 57.

In restoring the context, the console pushes the user's PSL and PC onto the user's interrupt stack, then executes a return from exception or interrupt instruction (REI) from that stack. This implies that the user's interrupt stack pointer (ISP) is valid before the firmware can exit. This is done automatically on a bootstrap. However, it is suggested that the stack pointer (SP) be set to a valid memory location before issuing the START or CONTINUE command. Furthermore, the user should validate the system control block base register (SCBB or PR$_SCBB) prior to executing a NEXT command, because the firmware uses the trace trap vector for this function. At power-up, the user ISP is set to 200 (hex) and the system control block base register is undefined.

12.3 Power-Up

At power-up, the KA660 firmware performs actions that are unique to this condition. Among these actions are the following:

- locating and identifying a console device
- language query
- diagnostic count-down

Certain actions depend on the state of the mode switch on the H3602-SA "test", "query", and "normal". This section describes the sequence of events which occur on power-up.

12.3.1 Identifying the Console Device

After power-up, the firmware attempts to determine what type of console device is present, so the device may be used to display further diagnostic progress. Normally, this is the device attached to the console serial line cable and the firmware sends the device attributes escape sequence (<ESC>[c) across the cable. This action determines the type of terminal attached and the functions it supports. Terminals that do not respond to the device attributes request correctly are assumed to be hard-copy devices.

Once a console device has been identified, the firmware displays the KA660 banner message (Figure 12–2) which contains the processor name, the version of the firmware, and the version of the VMB code.

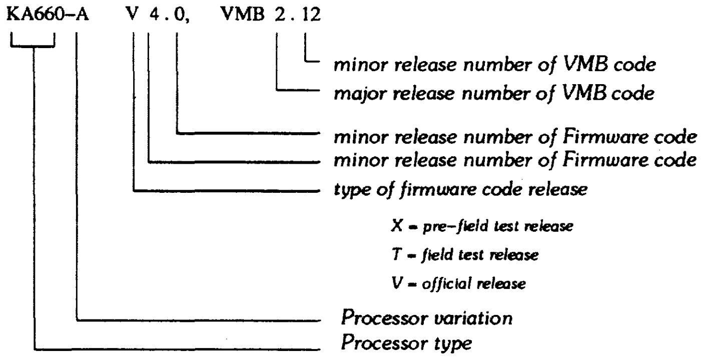

KA660–A V 4.0, VMB 2.12
minor release number of VMB code
major release number of VMB code
minor release number of Firmware code
minor release number of Firmware code
type of firmware code release
X - pre-field test release
T - field test release
V - official release
Processor variation
Processor type

Figure 12–2 Console Banner

If the designated console device supports DEC Multinational Character Set (MCS) **and** either the battery failed during power failure or the mode switch is set to query, the firmware prompts for the console language. The firmware first displays the language selection menu shown in Figure 12–3, Section 12.3.1.2.

After the language query, the firmware invokes the ROM-based diagnostics and eventually displays the console prompt.

12.3.1.1 Mode Switch Set to "Test"

If the mode switch is set to "test", the console serial line external loopback test is executed at the end of the IPT. **An external loopback connector should be inserted in the serial line connector on the H3602-SA panel prior to cycling power to invoke this test.** The purpose of this test is to verify that the console serial line connections from the KA660 through the H3602-SA panel are intact.

During this test the firmware toggles between two states, active and passive, each state is a few seconds long and each displays a different number on the LEDs.

During the active state (about 3 seconds long), the LEDs are set to "6". In this state the firmware reads the baud rate and mode switch, then transmits and receives a character sequence. If the mode switch has been moved from the test position, the firmware exits the test and continues as if on a normal power-up.

During the passive state (about 7 seconds long), the LEDs are set to "3".

If the firmware detects an error (parity, framing, overflow, or no characters), the firmware hangs with a "6" on the LEDs.

12.3.1.2 Mode Switch Set to "Query"

If the mode switch is set to "query" (or the firmware detects that the battery failed during a power loss), the firmware queries the user for a language which is used for displaying critical system messages.

Figure 12–3 shows the language selection menu.

```
 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
 (1..15):
```

Figure 12–3 Language Selection Menu

The user may select from one of the 15 supported languages. If no response is received within 30 seconds, the language defaults to English (United States/Canada). For those languages which do not have a unique keyboard, Figure 12–3 displays supported country specific keyboard variants in parentheses. **Language inquiry is performed only if the console device supports DEC MCS. Any console device that does not support DEC MCS, such as a VT100, defaults to English (United States/Canada).**

After completing language inquiry, the firmware proceeds as if the mode switch were set to "normal", as described in Section 12.3.1.3.

12.3.1.3 Mode Switch Set to "Normal"

If the mode selected is "normal", then the next step in the power-up sequence is to execute the bulk of ROM-based diagnostics. In addition to the message text, a countdown is displayed to indicated diagnostic test progress. Figure 12–4 shows a successful diagnostic countdown.

```
Performing normal system tests.
95..94..93..92..91..90..89..88..87..86..85..84..83..82..81..80..
79..78..77..76..75..74..73..72..71..70..69..68..67..66..65..64..
63..62..61..60..59..58..57..56..55..54..53..52..51..50..49..48..
47..46..45..44..43..42..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.
```

Figure 12–4 Normal Diagnostic Countdown

In the case of diagnostic failures, a diagnostic register dump is performed similar to the example shown in Figure 12–5. The remaining diagnostics execute and the countdown continues. For a detailed description of the register dump see Section 12.8.

```
95..94..93..92..91..90..89..88..87..86..85..84..83..82..81..80..
79..78..77..76..75..74..73..72..71..70..69..68..67..66..65..64..
63..62..61..60..59..58..57..56..55..54..53..52..51..50..49..48..
47..46..45..44..43..42..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..
?99 2 02 FF 0000 0000 00       ; SUBTEST_99_02, DE_Flush_Ena_Caches.IS
P1=00000003  P2=0000001F  P3=00000000  P4=00000000  P5=00000000
P6=00000000  P7=00000000  P8=00000000  P9=00000000 P10=00000000
r0=20063AC7  r1=0000002E  r2=00000099  r3=2014078C  r4=200629F5
r5=20062A26  r6=20063AD0  r7=00000000  r8=00000008 EPC=00000000
03..
Normal operation not possible.
```

Figure 12–5 Abnormal Diagnostic Countdown

If the diagnostics have successfully completed and halts are enabled, the firmware displays the console prompt (Figure 12–6) and enters console I/O mode.

```
>>>
```

```
    >>>
```

Figure 12–6 Console Prompt

If the diagnostics have successfully completed and halts are disabled, the firmware attempts to boot an operating system (Figure 12–7).

```
Loading system software.
No default boot device has been specified.
Devices:
-DIA0 (RF70)
-DIA1 (RF70)
-MUA0 (TK70)
-EZA0 (08-00-2B-03-82-78)
Device? [EZA0]:

(BOOT/R5:0 EZA0)


2..
-EZA0
```

Figure 12–7 Console Boot Display with No Default Boot Device

12.3.2 LED Codes

In addition to the console diagnostic countdown, a hexadecimal value is displayed on the diagnostic LEDs on the module and the H3602-SA panel. The purpose of the LED display is to improve fault isolation when there is no console terminal or when the hardware is incapable of communicating with the console terminal. Table 12–2 lists all LED codes and the associated actions which are performed at power-up. The LED code is changed before the corresponding test or action is performed.

Table 12–2 LED Codes

LED Value	Actions
F	Initial state on power-up, no code has executed
E	Entered ROM space, some instructions have executed
D	Waiting for power to stabilize (POK)
C	SSC RAM, SSC registers, and ROM checksum tests
B	Cache, interval timer, and virtual mode tests
A	FPA tests
9	Memory tests
8	CMCTL Memory, and I/O interaction tests
7	CQBIC (Q22-bus) tests
6	Console loopback tests
5	DSSI subsystem tests
4	Ethernet subsystem tests
3	Console I/O mode
2	Control passed to VMB
1	Control passed to secondary bootstrap
0	Program I/O mode, control passed to operating system

12.4 Operating System Bootstrap

Bootstrapping is the process by which an operating system loads and assumes control of the system. The KA660 supports bootstrap of the following operating systems: VMS, ULTRIX-32, and VAXELN. Additionally, the KA660 will boot MDM diagnostics and any user application image which conforms to the boot formats described in this section.

On the KA660 a bootstrap occurs whenever a BOOT command is issued at the console or whenever the processor halts and the conditions specified in the Table 12–1 for automatic bootstrap are satisfied.

12.4.1 Preparing for the Bootstrap

Prior to dispatching to the primary bootstrap (VMB), the firmware initializes the system to a known state. The following initialization sequence occurs:

1. Check the console program mailbox "bootstrap in progress" bit (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 EZA0.
4. Write a form of this BOOT request including the active boot flags and boot device on the console, for example "(BOOT/R5:0 DUA0)".
5. Set CPMBX<2>(BIP).
6. Initialize the Q22-bus Scatter/Gather map.
    a. Set IPCR<8>(AUX_HLT).
    b. Clear IPCR<5>(LMEAE).
    c. Perform an UNJAM.
    d. If an arbiter, map all vacant Q22-bus pages to the corresponding page in local memory and validate each entry if that page is "good".
    e. Perform an INIT.
    f. Set IPCR<5>(LMEAE).
7. Validate the PFN bit map. If invalid, rebuild it.
8. Search for a 128 Kbyte contiguous block of good memory as defined by the PFN bit map. If 128 Kbyte 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 bit map in bytes

   R3 = address of PFN bit map

   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 128 Kbyte good memory block + 512

   PC = base of 128 Kbyte 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 128 Kbyte 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. Local memory is partitioned as shown in Figure 12–8.

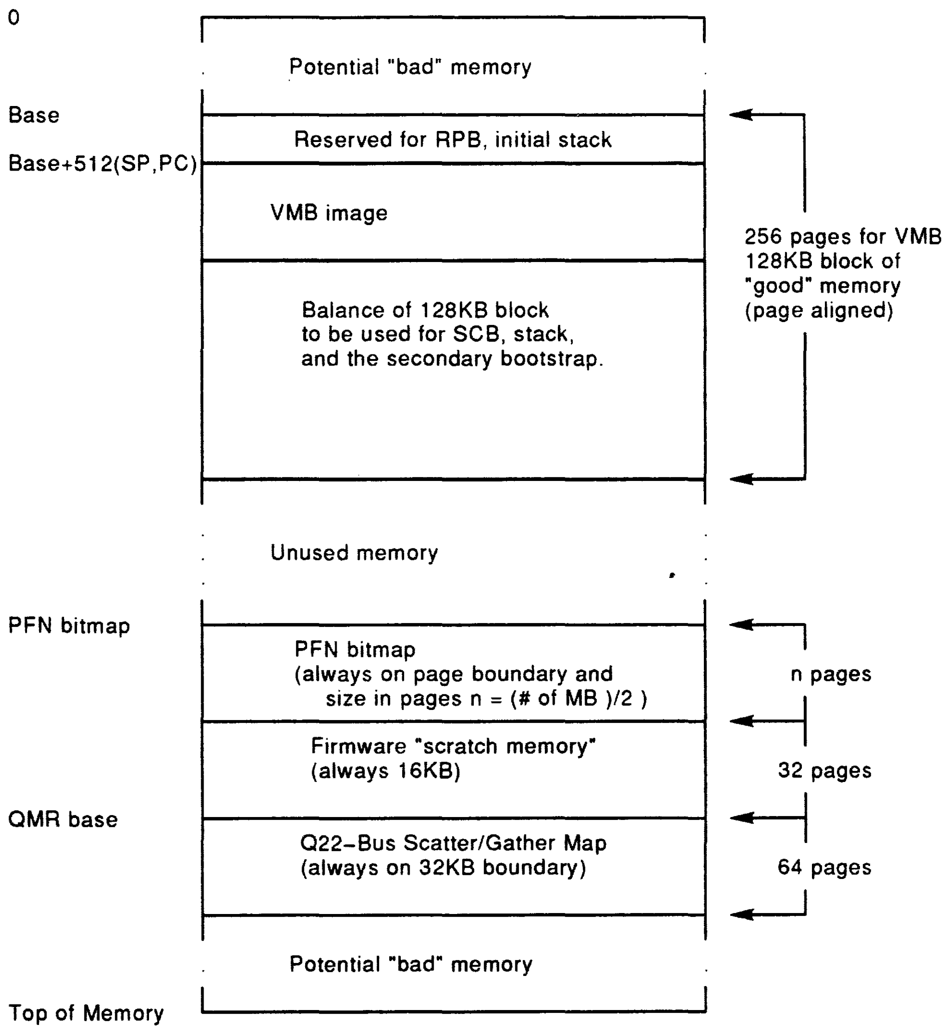

0
Potential "bad" memory
Base
Reserved for RPB, initial stack
Base+512(SP,PC)
VMB image
Balance of 128KB block to be used for SCB, stack, and the secondary bootstrap.
256 pages for VMB 128KB block of "good" memory (page aligned)
Unused memory
PFN bitmap
PFN bitmap (always on page boundary and size in pages n = (# of MB)/2)
n pages
Firmware "scratch memory" (always 16KB)
32 pages
QMR base
Q22-Bus Scatter/Gather Map (always on 32KB boundary)
64 pages
Potential "bad" memory
Top of Memory

Figure 12–8 Memory Layout Prior to VMB Entry

12.4.1.1 Boot Devices

The KA660 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 one of the following:

- the local Ethernet device, EZA0, if no default boot device has been specified
- the default boot device specified at initial power-up or via a SET BOOT command
- 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 an error message to be issued to the console. Otherwise the console makes no attempt to interpret or validate 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 12–3 correlates the boot device names expected in a BOOT command with the corresponding supported devices.

Table 12–3 KA660 Supported Boot Devices

	Boot Name[1]	Controller Type	Device Types
Disk:			
	[node$]DIAn	On-board DSSI	RF30, RF71
	DUcn	RQDX3 MSCP	RD52, RD53, RD54, RX33, RX50
		KDA50 MSCP	RA70, RA80, RA81, RA82, RA90
		KFQSA MSCP	RF30, RF71
		KLESI	RC25
	DLcn	RLV12	RL01, RL02
Tape:			
	[node$]MIAn	On-board DSSI	TF70
	MUcn	TQK50 MSCP	TK50
		TQK70 MSCP	TK70
		KLESI	TU81E
Network:			
	EZA0	On-board Ethernet	—
	XQcn	DEQNA	—
		DELQA	—
		DESQA	—
PROM:			
	PRA0	MRV11	—
	PRB0	On-board EPROM	—

[1] Boot device names consist of at least 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 dollar sign ($).

Table 12–3 presents a definitive list of boot devices which the KA660 supports. However, the KA660 could boot other devices which adhere to the MSCP standards.

12.4.1.2 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.

Figure 12–9 shows the location of the boot flags used by VMB in the boot flag longword and describes each flag's function.

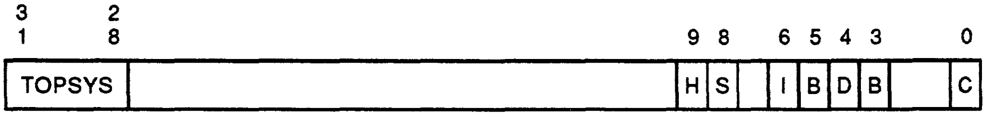


Figure 12–9 VMB Boot Flags (/R5:)

Table 12–4 VMB Boot Flags

Field	Name	Description
<31:28>	RPB$V_ TOPSYS	This field can be any value from 0 through F. This flag changes the top level directory name for the system disks with multiple operating systems. For example, if TOPSYS is 1, the top level directory name is [SYS1...]. **This does not apply to network bootstraps.**
<9>	RPB$V_ HALT	Halt before transfer. When this bit is set, VMB halts before transferring control to the application image.
<8>	RPB$V_ SOLICT	File name solicit. When this bit is set, VMB prompts the operator for the name of the application image file. A maximum of a 39-character file specification is permitted.
<6>	RPB$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.
<5>	RPB$V_ BOOBPT	Bootstrap breakpoint.
<4>	RPB$V_DIAG	Diagnostic bootstrap. When this bit is set, the load image requested over the network is [SYS0.SYSMAINT]DIAGBOOT.EXE.
<3>	RPB$V_ BBLOCK	Secondary bootstrap from bootblack. When this bit is set, VMB reads logical block number 0 of the boot device and tests it for conformance with the bootblack format. If it conforms, the block is executed to continue the bootstrap. No attempt to perform a Files–11 bootstrap is made.
<0>	RPB$V_ CONV	Conversational bootstrap.

12.4.2 Primary Bootstrap, VMB

Virtual Memory Boot (VMB) is the primary bootstrap for booting VAX processors. On the KA660, 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.

In certain cases, such as in 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 sequence 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). Refer to Table I–2.
4. Initialize the secondary bootstrap argument list. Refer to Table I–3 in Appendix E.
5. If not a PROM boot, locate a minimum of 3 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, "-DUA0".
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. **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 a minimum of 3 and a maximum of 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.**

Figure 12–10 shows a sample console display of a successful automatic bootstrap.

```
Loading system software.
(BOOT/R5:0 DUA0)

 2..
-DUA0
 1..0..
```

Figure 12–10 Successful Automatic Bootstrap

After a successful bootstrap operation, control is passed to the secondary bootstrap image with the memory layout as shown in Figure 12–11.

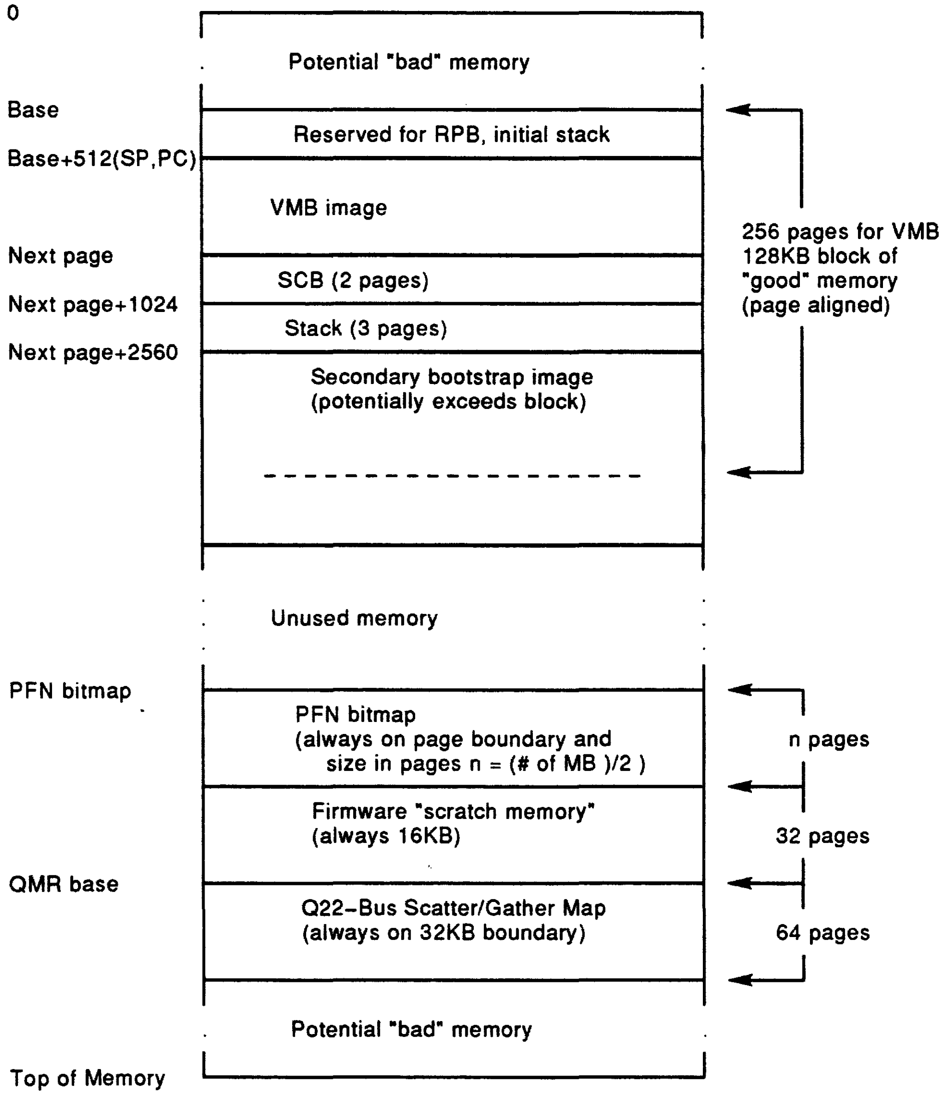


Figure 12–11 Memory Layout at VMB Exit

In the event that an operating system has an extraordinarily large secondary bootstrap which overflows the 128 Kbyte 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.

12.4.3 Device Dependent Bootstrap Procedures

As mentioned earlier, the KA660 supports bootstrapping from a variety of boot devices. The following sections describe the various device-dependent boot procedures.

12.4.3.1 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). VMS and ELN are the two standard operating systems that 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 12.4.1.2). If the boot image can't 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 12–12).
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 12–12. \

	31 24	23 16	15 0
BB+0:	1	n	any value
	low LBN		high LBN

(The next segment is also used as a PROM "signature block".)

	31 24	23 16	15 0
BB+(2*n)+0:	CHK	k	18(Hex)
	any value, most likely 0		
BB+(2*n)+8:	size in blocks of the image		
BB+(2*n)+12:	load offset		
BB+(2*n)+16:	offset into image to start		
BB+(2*n)+20:	sum of the previous three longwords		

Where:

1) the 18(hex) indicates this is a VAX instruction set
2) 18(hex) + "k" = the one's complement of "CHK"

Figure 12–12 Boot Block Format

12.4.3.2 PROM Bootstrap Procedure

The PROM bootstrap uses a variant of the boot block mechanism. VMB searches for a valid PROM signature block, the second segment of the boot block defined in Figure 12–12. If PRA0 is the selected "device", then VMB searches through Q22-bus memory on 16 Kbyte boundaries. If the selected "device" is PRB0, VMB checks the top 4096 byte block of the EPROM.

At each boundary, VMB:

1. Validates the readability of that Q22-bus memory page
2. 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 bootblack. If not, the next page will be tested.

Note that 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 16 Kbyte boundary is a candidate. Indeed, auxiliary bootstrap assumes that the image is in shared memory.

The PROM image is copied into main memory in 127 page "chunks" until the entire PROM is moved. All destination pages beyond the primary 128 Kbyte block are verified to make sure they are marked "good" in the PFN bit map. The PROM must be copied contiguously and if all required pages cannot fit into the memory immediately following the VMB image, the boot fails.

12.4.3.3 Network Bootstrap Procedure

Whenever a network bootstrap is selected on a KA660, the VMB code 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. (Refer to Appendix F for a complete description of supported MOP functions during bootstrap.) Once a network boot has been invoked, VMB turns on the designated network link and repeats load attempts, until either a successful boot occurs, a fatal controller error occurs, or VMB is halted from the operator console.

The KA660 uses network bootstraps to support the load of a standard operating system, a diagnostic image, or a user-designated program. The default image is the standard operating system; however, a user may select an alternate image by setting either the RPB$V_DIAG bit or the RPB$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. (See Figure 12–9 for a description of these bits.)

VMB accepts a maximum of 39 characters for a 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 extension .SYS. Therefore, the 39-character file specification need only consist of the filename if the default directory and extension attributes are used.

The KA660 VMB uses the MOP program load sequence for bootstrapping the module and the MOP "dump/load" protocol type for load related message exchanges. The types of MOP message used in the exchange are listed in Table F–1 and Table F–2 in Appendix F.

- VMB, the requester, starts by sending a REQ_PROGRAM message in the appropriate envelope (Table F–3 in Appendix F) to the MOP "dump/load" multicast address (Table F–4 in Appendix F). It then waits for a response in the form of a

VOLUNTEER message from another node on the network, the MOP load server. If a response is received, then the destination address is changed from the multicast address to the node address of the server. The same REQ_PROGRAM message is retransmitted to the server as an acknowledge which initiates the load.

- Next, VMB begins sending REQ_MEM_LOAD messages in response to any of the following:
  - a MEM_LOAD message, while there is still more to load
  - a MEM_LOAD_w_XFER, if it is the end of the image
  - a PARAM_LOAD_w_XFER, if it is the end of the image and operating system parameters are required
- The load number field in the load messages is used to synchronize the load sequence. At the beginning of the exchange, both the requester and server initialize the load number. The requester only increments the load number if a load packet has been successfully received and loaded. This forms the acknowledge to each exchange. The server will resend a packet with a specific load number, until it sees a request with the load number incremented. The final acknowledge is sent by the requester and has a load number equivalent to the load number of the appropriate LOAD_w_XFER message plus one.

- Because the request for load assistance is a MOP "must transact" operation, the network bootstrap continues indefinitely until a volunteer is found. The REQ_PROGRAM message is sent out in bursts of eight at four-second intervals; the first four in MOP Version 4 IEEE 802.3 format and the last four in MOP Version 3 Ethernet format. The backoff period between bursts doubles each cycle from an initial value of four seconds, to eight seconds, and so on, up to a maximum of five minutes. However, to reduce the likelihood of many nodes posting requests in lock-step, a random "jitter" is applied to the backoff period. The actual backoff time is computed as (.75+(.5*RND(x)))*BACKOFF, where 0≤x<1.

12.5 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 processor restart which results in firmware entry.

On the KA660 a restart occurs, if the conditions specified in Table 12–1 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. (Refer to Table I–2 in Appendix E for a detailed description of this data structure.) 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 CPMBX which it uses to avoid repeated attempts to restart a failing operating system. An additional restart in progress 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 "Restart failure." on the system console.

12.5.1 Locating the Restart Parameter Block (RPB)

The restart parameter block (RPB) is a page-aligned control block which can be identified by the first three longwords. Figure 12–13 shows the format of the RPB signature. (See Table I–2 in Appendix E for a complete description of the RPB.)

RPB: +00	physical address of the RPB
+04	physical address of the restart routine
+08	checksum of first 31 longwords of restart routine

Figure 12–13 Locating the Restart Parameter Block

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.

12.6 Console Service

The KA660 is by definition "halted", whenever the console program is running and the triple angle prompt (>>>) is displayed on the console terminal. When the processor is halted, the firmware provides most of the services of a standard VAX console through the device that is designated as the system console. The firmware also implements several commands which are not defined in the *VAX Architecture Reference Manual.* For a summary of the console commands see Table 12–11.

12.6.1 Console Control Characters

In console I/O mode, several characters have special meanings. Table 12–5 lists the characters and their meanings.

Table 12–5 Console Control Characters

Keyboard Key	Control Character	Meaning
Return	Return	This character ends a command line. No action is taken on a command until after it is terminated by a Return. A null line terminated by a Return is treated as a valid, null command. No action is taken, and the console reprompts for input. Return is echoed as return, line feed.
⊠	Delete	When the operator presses the delete key, the console deletes the character that the operator previously typed. What appears on the console terminal depends on whether the terminal is a video terminal or a hard copy terminal. For hard copy terminals, when the delete key is pressed, the console echoes with a backslash (\), followed by the character being deleted. If the operator deletes additional characters, the deleted characters are echoed. When the operator types a non-delete character, the console echoes another backslash, followed by the character typed. The result is to echo the characters deleted, surrounding them with backslashes. For video terminals, when the delete key is pressed, 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 the operator presses the delete key more times than there are characters on the line, the extra deletes are ignored. If the delete key is pressed on a blank line, it is ignored.
Ctrl/A	Ctrl/A or F14	Toggle insertion/overstrike mode for command line editing. By default, the console powers up to overstrike mode.
Ctrl/B	Ctrl/B or up arrow (or down arrow)	Recall previous command(s). Command recall is only operable if sufficient memory is available. This function may then be enabled and disabled using the SET RECALL command.
Ctrl/C	Ctrl/C	This character causes the console to echo ^C and to abort processing of a command. Ctrl/C has no effect as part of a binary load data stream. Ctrl/C clears Ctrl/S, and reenables output stopped by Ctrl/O.
Ctrl/D	Ctrl/D or left arrow	This character moves the cursor left one position.
Ctrl/E	Ctrl/E	Move the cursor to the end of the line.
Ctrl/F	Ctrl/F or right arrow	Moves the cursor one position to the right.
Ctrl/H	Ctrl/H, BACKSPACE or F12	Moves the cursor to the beginning of the line.

Table 12–5 (Cont.) Console Control Characters

Keyboard Key	Control Character	Meaning
Ctrl/O	Ctrl/O	This character 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 if it 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 Ctrl/Q.
Ctrl/Q	Ctrl/Q	This character causes the output to the console terminal to resume. Additional Ctrl/Qs are ignored. Ctrl/S and Ctrl/Q are not echoed.
Ctrl/S	Ctrl/S	Stops output to the console terminal until Ctrl/Q is typed. Ctrl/S and Ctrl/Q are not echoed.
Ctrl/U	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 the console prompts for another command.
Ctrl/R	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. When Ctrl/C is typed as part of a command line, the console deletes the line as it does with Ctrl/U.
Break	BREAK	If the console is in console I/O mode, BREAK is equivalent to Ctrl/C and is echoed as "^C".

NOTE

If the local console is in program I/O mode and halts are disabled, BREAK is ignored. If the console is in program I/O mode and halts are enabled, BREAK causes the processor to halt and enter console I/O mode.

Control characters are typed by pressing the character key while holding down the control key.

If an unrecognized control character (ASCII code less than 32 decimal or between 128 and 159 decimal) is typed it is echoed as an up arrow followed by the character with ASCII code 64 or greater. For example, BEL (ASCII code 7) is echoed as "^G", because capital G is ASCII code 7+64=71. When a control character is deleted using the delete key, it is echoed the same way. After echoing the control character, the console processes it like a normal character. Commands with control characters are invalid, unless they are part of a comment, and the console will respond with an error message.

12.6.2 Console Command Syntax

The console accepts commands of lengths up to 80 characters. It responds to longer commands with an error message. The count does not include deletes, deleted characters, or the terminating Return.

Commands may be abbreviated. Abbreviations are formed by dropping characters from the end of a keyword, as long as the resulting keyword is still unique. Most commands can be uniquely expressed with their first character.

Multiple adjacent spaces and tabs are treated as a single space by the console. Leading and trailing spaces and tabs are ignored. Tabs are echoed as spaces.

Command qualifiers can appear after the command keyword, or after any symbol or number in the command. A qualifier is any contiguous set of nonwhite-space characters that is started with a slash (ASCII code 47 decimal).

All numbers (addresses, data, counts) are in hexadecimal. Note, though, that symbolic register names number the registers in decimal. The console does not distinguish between upper and lower case either in numbers or in commands; both are accepted.

12.6.3 Console Command Keywords

The KA660 firmware implements a variant of the VAX SRM console command set. The only commands defined in the VAX SRM and not supported by the KA660 are MICROSTEP, LOAD, and @. The CONFIGURE, HELP, MOVE, SEARCH and SHOW command have been added to the command set to facilitate system debugging and access to system parameters. In general, however, the KA660 console is similar to other VAX consoles.

Table 12–6 lists command, parameter, and qualifier keywords.

Table 12–6 Command, Parameter, and Qualifier Keywords

	Command Keywords	
Processor Control	**Data Transfer**	**Console Control**
B*OOT	D*EPOSIT	CONF*IGURE
C*ONTINUE	E*XAMINE	F*IND
H*ALT	M*OVE	R*EPEAT
I*NITIALIZE	SEA*RCH	SET
N*EXT	X	SH*OW
S*TART		T*EST
U*NJAM		
	SET and SHOW Parameter Keywords	
BO*OT	BF*L(A)G	DE*VICE
DS*SI	ET*HERNET	HA*LT
H*OST	L*ANGUAGE	M*EMORY
Q*BUS	R*ECALL	RL*V12
U*QSSP	VERS*ION	T*RANSLATION

Table 12–6 (Cont.) Command, Parameter, and Qualifier Keywords

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
/ST*EP:	/U	/DU*P or /MA*INTENANCE
/WR*ONG		/DS*SI or /U*QSSP
		/DI*SK or /T*APE
		/SE*RVICE

The characters before the asterisk (*) indicate the minimal number of characters that are required to uniquely identify the keyword.

Table 12–11 provides a complete summary of the console commands, and Section 12.7 describes the console commands.

12.6.4 Console Command Qualifiers

All qualifiers in the console command syntax are global. That is, they may appear in any place 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.

Table 12–12 summarizes the qualifiers recognized by the KA660 console.

12.6.5 Console Numeric Expression Radix Specifiers

By default, the console treats any numeric expression, used as an address or a datum, as a hexadecimal integer. The user may override the default radix by using one of the specifiers listed in Table 12–7.

Table 12–7 Console Radix Specifiers

Form 1	Form 2	Radix
%b	^b	Binary
%o	^o	Octal
%d	^d	Decimal
%x	^x	Hexadecimal, default

For instance, the value 19 is by default hexadecimal, but it may also be represented as %b11001, %o31, %d25, and %x19 (or in the alternate form as ^b11001, ^o31, ^d25, and ^x19).

12.6.6 Command 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 6 address spaces: physical memory (/P qualifier), virtual memory (/V qualifier), general purpose registers (/G qualifier), internal processor registers (/I qualifier), protected memory (/U qualifier), and 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 reference is PHYSICAL.

The KA660 console supports symbolic references to addresses. A symbolic reference simultaneously defines the address space for a given symbol. Table 12–8 lists the symbolic addresses supported by the console. They are grouped according to address space.

Table 12–8 Console Symbolic Addresses

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
			/G - General Purpose Registers				
R0	00	R4	04	R8	08	R12 (AP)	0C
R1	01	R5	05	R9	09	R13 (FP)	0D
R2	02	R6	06	R10	0A	R14 (SP)	0E
R3	03	R7	07	R11	0B	R15 (PC)	0F
			/M - Processor Status Longword				
PSL	—						
			/I - Internal Processor Registers				
pr$_ksp	00	pr$_ pcbb	10	pr$_rxcs	20	—	30
pr$_esp	01	pr$_ scbb	11	pr$_ rxdb	21	—	31
pr$_ssp	02	pr$_ipl	12	pr$_txcs	22	—	32
pr$_usp	03	pr$_ astlv	13	pr$_ txdb	23	—	33
pr$_isp	04	pr$_sirr	14	—	24	—	34
—	05	pr$_sisr	15	pr$_ccr	25	—	35
—	06	—	16	—	26	—	36
—	07	—	17	pr$_ mser	27	pr$_ ioreset	37

Note: All symbolic values in this table are in hexadecimal.

Table 12–8 (Cont.) Console Symbolic Addresses

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
/I - Internal Processor Registers							
pr$_p0br	08	pr$_iccr	18	—	28	pr$_mapen	38
pr$_p0lr	09	—	19	—	29	pr$_tbia	39
pr$_p1br	0A	pr$_icr	1A	pr$_savpc	2A	pr$_tbis	3A
pr$_p1lr	0B	pr$_todr	1B	pr$_savpsl	2B	—	3B
pr$_sbr	0C	—	1C	—	2C	—	3C
pr$_slr	0D	—	1D	—	2D	—	3D
—	0E	—	1E	—	2E	pr$_sid	3E
—	0F	—	1F	—	2F	pr$_tbchk	3F
/P - Physical (VAX I/O Space)							
qbio	20000000	qbmem	30000000	qbmbr	20080010	—	—
rom	20040000	cacr	20084000	bdr	20084004	—	—
dscr	20080000	dser	20080004	dmear	20080008	dsear	2008000C
ipcr0	20001f40	ipcr1	20001f42	ipcr2	20001f44	ipcr3	20001f46
ssc_ram	20140400	ssc_cr	20140010	ssc_cdal	20140020	ssc_dledr	20140030
ssc_ad0mat	20140130	ssc_ad0msk	20140134	ssc_ad1mat	20140140	ssc_ad1msk	20140144
ssc_tcr0	20140100	ssc_tir0	20140104	ssc_tnir0	20140108	ssc_tivr0	2014010c
ssc_tcr1	20140110	ssc_tir1	20140114	ssc_tnir1	20140118	ssc_tivr1	2014011c
memcsr0	20080100	memcsr1	20080104	memcsr2	20080108	memcsr3	2008010c
memcsr4	20080110	memcsr5	20080114	memcsr6	20080118	memcsr7	2008011c
memcsr8	20080120	memcsr9	20080124	memcsr10	20080128	memcsr11	2008012c
memcsr12	20080130	memcsr13	20080134	memcsr14	20080138	memcsr15	2008013c
memcsr16	20080140	memcsr17	20080144	—	—	—	—
nicsr0	20008000	nicsr1	20008004	—	20008008	nicsr3	2000800C
nicsr4	20008010	nicsr5	20008014	nicsr6	20008018	nicsr7	2000801C
—	20008020	nicsr9	20008024	nicsr10	20008028	nicsr11	2000802C
nicsr12	20008030	nicsr13	20008034	nicsr14	20008038	nicsr15	2000803C

Table 12–8 (Cont.) Console Symbolic Addresses

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
			/P - Physical (VAX I/O Space)				
sgec_setup	20008000	sgec_poll	20008004	—	20008008	sgec_rba	2000800C
sgec_tba	20008010	sgec_status	20008014	sgec_mode	20008018	sgec_sbr	2000801C
—	20008020	sgec_wdt	20008024	sgec_mfc	20008028	sgec_verlo	2000802C
sgec_verhi	20008030	sgec_proc	20008034	sgec_bpt	20008038	sgec_cmd	2000803C
shac_sswcr	20004230	shac_sshma	20004244	shac_pqbbr	20004048	shac_psr	2000404c
shac_pesr	20004250	shac_pfar	20004254	shac_ppr	20004058	shac_pmcsr	2000405C
shac_pcq0cr	20004280	shac_pcq1cr	20004284	shac_pcq2cr	20004088	shac_pcq3cr	2000408C
shac_pdfqcr	20004290	shac_pmfqcr	20004294	shac_psrcr	20004098	shac_pecr	2000409C
shac_pdcr	200042A0	shac_picr	200042A4	shac_pmtcr	200040A8	shac_pmtecr	200040AC
			Any Address Space				
*	The last location successfully referenced in an EXAMINE or DEPOSIT command.						
+	The location immediately following the last location successfully 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, 8 for quadword). For other address spaces, the address is the last address referenced plus one.						
-	The location immediately preceding the last location successfully 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, 8 for quadword). For other address spaces, the address is the last addressed referenced minus one.						
@	The location addressed by the last location successfully referenced in an EXAMINE or DEPOSIT command.						

12.6.7 References to Processor Registers and Memory

The KA660 console is implemented in VAX MACRO code executing from EPROM. Actual processor registers cannot be modified by the console command interpreter. When the console is entered, the console saves the processor registers in console memory and all command references to them are directed to the corresponding saved values, not to the registers themselves.

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

The following registers are saved by the console, and any direct reference to these registers will be intercepted by the console and the access will be to the saved copies:

- R0 through R15 - the general purpose registers (GPRs)
- PR$_IPL - the interrupt priority level register (IPLR)
- PR$_SCBB - the system control block base register (SCBB)
- PR$_ISP - the interrupt stack pointer (ISP)
- PR$MAPEN - the memory management enable register (MAPEN)

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

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

12.7 Console Commands

The following section defines the conventions use to describe command syntax, and the commands accepted by the console when the KA660 is in console I/O mode.

Syntax Conventions

The following conventions are used to describe command syntax:

Table 12–9 Syntax Conventions

[ ]	Enclose an optional command elements.
{ }	Enclose a required command element.
...	Indicates a series of command elements.

The console allows you to override the default radix by using the following commands:

Table 12–10 Overriding the Default Radix

%d	Decimal (For example, %d1234)
%x	Hexadecimal (For example, %xFEEBFCEA)
%b	Binary (For example, %b1001)
%o	Octal (For example, %o1070)

The following example shows a console EXAMINE command that specifies a decimal value for the /N qualifier:

```
>>>EX/L/P/N:%d1023  0
```

BOOT

Format

BOOT *[qualifier] [boot_device[:]]*

Qualifiers

/R5:{boot_flags}
Boot flags is a 32-bit hex value, that is passed to VMB in R5. No interpretation of this value is performed by the console. Refer to Figure 12–9 for the bit assignments of R5. A default boot flags longword may be specified using the SET BFLAG command and displayed with the SHOW BFLAG command.

/{boot_flags}
Equivalent to the form above.

Arguments

[boot_device]
The boot 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 a string descriptor to this device name in R0. A default boot device may be specified using the SET BOOT command and displayed with the SHOW BOOT command. The factory default is the Ethernet device, EZA0.

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.

If a list of devices is specified, VMB attempts to boot from each device in turn and then transfers control to the first successfully booted image. In a list, network devices should always be placed last, because network bootstraps only terminate if a fatal hardware error occurs or an image is successfully loaded.

The console qualifies the bootstrap operation by passing a boot flags to VMB in R5. A more detailed description of the bootstrap process and how the default bootstrap device is determined is described in Section 12.4.

In the case where either the qualifiers or the device name is absent, then the corresponding default value is used. Explicitly stating the boot flags or the boot device overrides the current default value for the current boot request, but does not change the corresponding default value in BBURAM.

There are three mechanisms by which the default boot device and and boot flags may be set.

1. The operating system may write a default boot device and flags into the appropriate locations in BBURAM (refer to Appendix H).

2. The user may explicitly set the default boot device and boot flags with the console SET BOOT and SET BFLAG commands respectively.
3. The console will prompt the user for the default boot device, if any of the following conditions are met:
   - The power-up mode switch is set to query mode.
   - The console detects that the battery failed, and therefore the contents of BBURAM are no longer valid.
   - The console detects that the default boot device has not been explicitly set by the user. Either a previous device query timed out and defaulted to EZA0 or neither (1) nor (2) has been performed. Simply stated, the console will prompt the user on each and every power-up for a default boot device, until such a request has been satisfied.

On power-up if no default boot device is specified in BBURAM, the console issues a list of potential bootable devices and then queries the user for a device name. If no device name is entered within 30 seconds, EZA0 is used. However, EZA0 does not become the default boot device.

Examples

```
>>>show boot
DUA0
>>>show bflag
0
>>>b                    ! Boot using default boot flags and device.
(BOOT/R5:0 DUA0)

  2..
-DUA0

>>>bo EZA0              ! Boot using default boot flags and specified device.
(BOOT/R5:0 EZA0)

  2..
-EZA0

>>>boot/10              ! Boot using specified boot flags and default device.
(BOOT/R5:10 DUA0)

  2..
-DUA0

>>>boot /r5:220 EZA0    ! Boot using specified boot flags and device.
(BOOT/R5:220 EZA0)

  2..
-EZA0

>>>boot dia0,mua0,eza0
(BOOT/R5:0 DIA0,MUA0,EZA0)
  2..
```

CONFIGURE

Format

CONFIGURE

Qualifiers

None.

Arguments

None.

Description

CONFIGURE is similar to the VMS SYSGEN CONFIG utility. This feature simplifies system configuration by providing information that is typically available only with a running operating system.

The CONFIGURE command invokes an interactive mode that permits the user to enter Q22-bus device names, then generates a table of Q22-bus I/O page device CSR addresses and device vectors.

Examples

```
>>>configure
Enter device configuration, HELP, or EXIT
Device,Number? help
Devices:
 LPV11        KXJ11        DLV11J       DZQ11        DZV11        DFA01
 RLV12        TSV05        RXV21        DRV11W       DRV11B       DPV11
 DMV11        DELQA        DEQNA        DESQA        RQDX3        KDA50
 RRD50        RQC25        KFQSA-DISK   TQK50        TQK70        TU81E
 RV20         KFQSA-TAPE   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         ADQ32        DTC04
 DESNA        IGQ11        DIV32        KIV32        DTCN5        DTC05
 KWV32        KZQSA
Numbers:
 1 to 255, default is 1
Device,Number? kda50
Device,Number? kfqsa
Device is ambiguous
Device,Number? kfqsa-disk
Device,Number? kfqsa-tape
Device,Number? cxy08
Device,Number? cxa16
Device,Number? exit
```

```
Address/Vector Assignments
-772150/154 KDA50
-760334/300 KFQSA-DISK
-774500/260 KFQSA-TAPE
-760500/310 CXY08
-760520/320 CXA16
>>>
```

CONTINUE

Format

CONTINUE

Qualifiers

None.

Arguments

None.

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. Internally, the continue command pushes the user's PC and PSL onto the user's ISP, and then executes an REI instruction. This implies that the user's ISP is pointing to some valid memory.

Examples

```
>>>continue
>>>
```

DEPOSIT

Format

DEPOSIT *[qualifier_list] {address} {data} [data...]*

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 accessible only by the MTPR and MTPR instructions. The data size is always long.

/M
The address space is the Processor Status Longword (PSL).

/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 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 is overriden by the presence of this qualifier. This qualifier is not inherited.

/WRONG

The ECC bits for this data are forced to the value of 3 (ECC bits of 3 will always generate a double-bit error).

Arguments

{address}

A longword address that specifies the first location into which data is deposited. The address can be any legal address specifier as defined in Section 12.6.6 and Table 12–8.

{data}

The data to be deposited. If the specified data is larger than the deposit data size, the console ignores the command and issues an error response. If the specified data is smaller than the deposit data size, it is extended on the left with zeros.

[data]

Additional data to be deposited (up to a maximum of 6 values).

Description

Deposits the 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 long word and the default address is zero. If conflicting address space or data sizes are specified, the console ignores the command and issues an error response.

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.
    >>>
```

EXAMINE

Format

EXAMINE *[qualifier_list] [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 accessible only by the MTPR and MTPR instructions. The data size is always long.

/M
The address space is the Processor Status Longword (PSL).

/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 with 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 EXAMINE - <DATA>".

/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 this read access to main memory are ignored. Also, if specified, the ECC bits actually read are displayed in parenthesis following the datum. In the case of quadword and octaword data, the ECC bits shown apply to the most significant longword only.

/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 any legal address specifier as defined in Section 12.6.6 and Table 12–8. If no address is specified, "+" is assumed.

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 will cause the EXAMINE command to ignore ECC errors on reads from physical memory. Additionally, the EXAMINE command supports a /INSTRUCTION qualifier, which will disassemble the instructions at the current address.

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 of EPROM.
  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 MTPR    S^#2A,B^44(R1)
>>>e/ins                                            ! Look at next instruction.
  P 20040048   DB MTPR    S^#2B,B^48(R1)
>>>
```

FIND

Format

FIND *[qualifier-list]*

Qualifiers

/MEMORY

Search memory for a page-aligned block of good memory, 128 Kbyte in length. The search looks only at memory that is deemed usable by the bit map. This command leaves the contents of memory unchanged.

/RPB

Search all of physical memory for a restart parameter block. The search does not use the bit map to qualify which pages are looked at. The command leaves the contents of memory unchanged.

Arguments

None.

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.

Examples

```
>>>ex sp                                            ! Check the SP.
  G 0000000E 00000000
>>>find /mem                                        ! Look for a valid 128Kb.
>>>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.
>>>
```

HALT

Format

HALT

Qualifiers

None.

Arguments

None.

Description

This command has no effect and is included for compatibility with other consoles.

Examples

```
>>>halt                                            ! Pretend to halt.
>>>
```

HELP

Format

HELP

Qualifiers

None.

Arguments

None.

Description

This command has been included to help the console operator answer simple questions about command syntax and usage.

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 specifying a syntactically correct value
        ..         denotes one of an inclusive range of integers
        ...        denotes that the previous item may be repeated

Valid qualifiers:
    /B /W /L /Q /INSTRUCTION
    /G /I /V /P /M
    /STEP: /N: /NOT
    /WRONG /U
```

```
Valid commands:
    BOOT [/R5:<boot_flags> | /<boot_flags>] [<boot_device>[:]]
    CONFIGURE
    CONTINUE
    DEPOSIT [<qualifiers>] <address> [<datum> [<datum>]]
    EXAMINE [<qualifiers>] [<address>]
    FIND [/MEMORY | /RPB]
    HALT
    HELP
    INITIALIZE
    MOVE [<qualifiers>] <address> <address>
    NEXT [count]
    REPEAT <command>
    SEARCH [<qualifiers>] <address> <pattern> [<mask>]
    SET BFL(A)G <boot_flags>
    SET BOOT <boot_device>
    SET CONTROLP <0..1 | DISABLED | ENABLED>
    SET HALT <0..4 | DEFAULT | RESTART | REBOOT | HALT | RESTART_REBOOT>
    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>
    SET HOST/MAINTENANCE/UQSSP <physical_CSR_address>
    SET LANGUAGE <1..15>
    SET RECALL <0..1 | DISABLED | ENABLED>
    SHOW BFL(A)G
    SHOW BOOT
    SHOW DEVICE
    SHOW DSSI
    SHOW ETHERNET
    SHOW HALT
    SHOW LANGUAGE
    SHOW MEMORY [/FULL]
    SHOW RECALL
    SHOW RLV12
    SHOW QBUS
    SHOW UQSSP
    SHOW SCSI
    SHOW TRANSLATION <physical_address>
    SHOW VERSION
    START <address>
    TEST [<test_code> [<parameters>]]
    UNJAM
    X <address> <count>
>>>
```

INITIALIZE

Format

INITIALIZE

Qualifiers

None.

Arguments

None.

Description

A processor initialization is performed. The following registers are initialized, as specified in *The VAX Architecture Standard.*

PSL — 041F0000
IPL — 1F
ASTLVL — 4
SISR — 0
ICCS — bits <6> and <0> are clear, the rest are UNPREDICTABLE
RXCS — 0
TXCS — 80
MAPEN — 0
CPU cache — flushed and enabled
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 KA660 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.

Examples

```
>>>init
>>>
```

MOVE

Format

MOVE *[qualifier-list] {src_address} {dest_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.

/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 destination 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 is overriden by the presence of this qualifier. This qualifier is not inherited.

/WRONG
On reads, ECC errors on the access of data in main memory are ignored. On writes, the ECC bits for this data are forced to the value of 3 (ECC bits of 3 will always generate a double-bit error).

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 any legal address specifier as defined in Section 12.6.6 and Table 12–8. If no address is specified, "+" is assumed.

Description

The console copies the block of memory starting at the source address to a block beginning at the destination address. Typically, this command is used with the /N: qualifier to transfer large blocks of data. The destination will correctly reflect 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.

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
>>>move /n:4 200 0                          ! Move the data.
>>>ex /n:4 0                                ! Observe the destination.
  P 00000000 58DD0520
  P 00000004 585E04C1
  P 00000008 00FF8FBB
  P 0000000C 5208A8D0
  P 00000010 540CA8DE
>>>
```

NEXT

Format

NEXT *[count]*

Qualifiers

None.

Arguments

[count]
A value representing the number of macro instructions to execute.

Description

The NEXT command causes the processor to "step" the specified number of macro instructions. If no count is specified, "single-step" is assumed. The console then enters spacebar step mode as described in DEC STD 032. In this mode, subsequent spacebar strokes initiate single steps and pressing Return forces a return to the console prompt.

The console uses the trace and trace pending bits in the PSL, and the SCB trace pending vector to implement the NEXT function. This creates the following restrictions on the usage of the NEXT command:

- 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, due to the instructions executed in implementation, does not work where time critical code is being executed.
- 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 cannot be used in conjunction with other debuggers. This also implies that the user should validate PR$_SCCB before using the NEXT command.

Examples

```
>>>dep 1000 50D650D4                        ! Create a simple program.
>>>dep 1004 125005D1
>>>dep 1008 00FE11F9
>>>ex /instruction /n:5 1000                ! List it.
  P 00001000   D4 CLRL    R0
  P 00001002   D6 INCL    R0
  P 00001004   D1 CMPL    S^#05,R0
  P 00001007   12 BNEQ    00001002
  P 00001009   11 BRB     00001009
  P 0000100B   00 HALT
>>>dep pr$_scbb 200                         ! Set up a user SCBB...
>>>dep pc 1000                              ! ...and the PC.
>>>
>>>n                                        ! Single step...
  P 00001002   D6 INCL    R0                ! SPACEBAR
  P 00001004   D1 CMPL    S^#05,R0          ! SPACEBAR
  P 00001007   12 BNEQ    00001002          ! SPACEBAR
  P 00001002   D6 INCL    R0                ! CR
>>>n 5                                      ! ...or multiple step the program.
  P 00001004   D1 CMPL    S^#05,R0
  P 00001007   12 BNEQ    00001002
  P 00001002   D6 INCL    R0
  P 00001004   D1 CMPL    S^#05,R0
  P 00001007   12 BNEQ    00001002
>>>n 7
  P 00001002   D6 INCL    R0
  P 00001004   D1 CMPL    S^#05,R0
  P 00001007   12 BNEQ    00001002
  P 00001002   D6 INCL    R0
  P 00001004   D1 CMPL    S^#05,R0
  P 00001007   12 BNEQ    00001002
  P 00001009   11 BRB     00001009
>>>n
  P 00001009   11 BRB     00001009
>>>
```

REPEAT

Format

REPEAT *{command}*

Qualifiers

None.

Arguments

{command}

Description

The console repeatedly displays and executes the specified command. The repeating is stopped when the operator presses Ctrl/C. Any valid console command can be specified for the command with the exception of the REPEAT command.

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
>>>
```

SEARCH

Format

SEARCH *[qualifier_list] {address} {pattern} [mask]*

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 with 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 any legal address specifier as defined in Section 12.6.6 and Table 12–8. If no address is specified, "+" is assumed.

{pattern}

The target data.

[mask]

A longword containing the bits in the target which are to be "masked" out.

Description

The search command finds all occurrences 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 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.

Conceptually, a match condition occurs if the following condition is true:

```
 (pattern AND NOT mask)  EQUALS  (data AND NOT mask)

where:  pattern -- is the target data.
        mask    -- is the optional don't care bit mask (which defaults to 0).
        data    -- is the data (byte, word, long, quad) at the current address.
```

The command reports the address if the match condition is true, and there is no /NOT qualifier, or if the match condition is false and there is a /NOT qualifier. This is summarized in the following table:

/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, long or quad), unless overriden by the /STEP qualifier.

Examples

```
>>>dep /p/l/n:1000 0 0    ! Clear some memory.
>>>
>>>dep 300 12345678    ! Deposit some "search" data.
>>>dep 401 12345678
>>>dep 502 87654321
>>>
>>>search /n:1000 /st:1 0 12345678  ! Search for all occurrences...
  P 00000300 12345678    ! ...of 12345678 on any byte...
  P 00000401 12345678    ! ...boundary.
>>>search /n:1000 0 12345678   ! Then try on longword...
  P 00000300 12345678    ! ...boundaries.
>>>search /n:1000 /not 0 0   ! Search for all nonzero...
  P 00000300 12345678    ! ...longwords.
  P 00000400 34567800
  P 00000404 00000012
  P 00000500 43210000
  P 00000504 00008765
>>>search /n:1000 /st:1 0 1 FFFFFFFE  ! Search for "odd" longwords...
  P 00000502 87654321    ! ...on any boundary.
  P 00000503 00876543
  P 00000504 00008765
  P 00000505 00000087
>>>search /n:1000 /b 0 12   ! Search for all occurrences...
  P 00000303 12     ! ...of the byte 12.
  P 00000404 12
>>>search /n:1000 /st:1 /w 0 FE11  ! Search for all words which...
>>>                                             ! ...could be interpreted as...
>>>                                             ! ...a "spin" (10$: brb 10$).
>>>      ! Note, none found.
```

SET

Format

SET *{parameter} {value}*

Qualifiers

—

Depends on the parameters used.

Arguments

None.

Description

Sets the indicated console parameter to the indicated value. The following list describes the console parameters and their acceptable values:

Parameters

BFL(A)G

Set the default R5 boot flags. The argument to this parameter is {bitmap}. The value assigned to {bitmap} must be a hexadecimal number of up to 8 hex digits.

BOOT

Set the default boot device. The arguments to this qualifier are {device_name} and {device_list}. The value assigned to these arguments must be either a valid device name or device list as specified in Section 12.7, Console Commands on the BOOT command.

CONTROLP

Sets Ctrl/P as the console halt condition, instead of a BREAK. The arguments to this qualifier can be either a {value} or a {keyword}. The value assigned to this qualifier argument must be a 1 (or the keyword ENABLED) to set Ctrl/P recognition. The value must be a 0 (or the keyword DISABLE) to set BREAK recognition. In either case, the setting of the BREAK enable switch will determine whether or not a halt will occur.

HALT

Sets the user-defined halt action. The argument to this qualifier can be either a {value} or a {keyword}. Acceptable values are 0 through 4 or their corresponding keywords DEFAULT, RESTART, REBOOT, HALT, and RESTART_REBOOT. Refer to Table 12–1 for usage.

HOST

Invoke the DUP or MAINTENANCE driver on the selected node. Only SET HOST /DUP accepts a {value} parameter. Acceptable numbers that can be assigned to the {value} parameter are 0 (for bus 0) and 1 (for bus 1). The hierarchy of the SET HOST qualifiers listed next suggests the appropriate usage. Each qualifier supports only 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 /DUP is a task name for the selected DUP driver to execute.

> **NOTE**
> **The KA660 DUP driver supports only SEND DATA IMMEDIATE messages and those devices which support the SEND DATA messages.**

/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 KFQSA EEPROM configuration parameters. Note that SET HOST /MAINTENANCE does not accept a "task" value.

/UQSSP —

/SERVICE n — Specify the KFQSA controller number "n" of a KFQSA in service mode, where "n" is from 0 to 3. (The resulting fixed address of a KFQSA in service mode is 20001910+4*n.)
csr_address — Specify the Q22-bus I/O page CSR address for the KFQSA.

LANGUAGE

Set console language and keyboard type. The argument to this qualifier is {value}. 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 message mode. Acceptable values assigned to this qualifier argument are 1 through 15 and have the following meaning:

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

RECALL

This command allows you to enable or disable the recal function. The argument to this qualifier is {value}. Acceptable values to be assigned to this qualifier argument are 1 or the keyword ENABLE and 0 or the key word DISABLED. When recal is set, you can scroll through the recal buffer using the up arrow and down arrow keys.

Arguments

None.

Examples

```
   >>>
   >>>set bflag 220
   >>>
   >>>set boot EZA0
   >>>
   >>>set controlp disabled
   >>>
   >>>set halt reboot
   >>>
   >>>set host /dup /dssi 0
   Starting DUP server...

   DSSI Node 0 (SUSAN)
   Copyright © 1990  Digital Equipment Corporation
   DRVEXR V1.0  D  5-JUL-1990 15:33:06
   DRVTST V1.0  D  5-JUL-1990 15:33:06
   HISTRY V1.0  D  5-JUL-1990 15:33:06
   ERASE  V1.0  D  5-JUL-1990 15:33:06
   PARAMS V1.0  D  5-JUL-1990 15:33:06
   DIRECT V1.0  D  5-JUL-1990 15:33:06
   End of directory

   Task Name? params
   Copyright © 1990  Digital Equipment Corporation

   PARAMS> stat path

      ID  Path Block     Remote Node      DGS_S      DGS_R      MSGS_S     MSGS_R
      -- ------------  ---------------  ---------- ---------- ---------- ----
------
       0 PB  FF811ECC   Internal Path            0           0          0          0
       6 PB  FF811FD0  KFQSA  KFX V1.0           0           0          0          0
       1 PB  FF8120D4  KAREN  RFX V101           0           0          0          0
       4 PB  FF8121D8  WILMA  RFX V101           0           0          0          0
       5 PB  FF8122DC  BETTY  RFX V101           0           0          0          0
       2 PB  FF8123E0  DSSI1  VMS V5.0           0           0      14328      14328
       3 PB  FF8124E4  3      VMB BOOT           0           0         61         61

   PARAMS> exit
   Exiting...

   Task Name?

   Stopping DUP server...
   >>>
   >>>set host /dup/dssi 0 params
   Starting DUP server...

   DSSI Node 0 (SUSAN)
   Copyright © 1990  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...
>>>
>>>set host /maint/uqssp 20001468
UQSSP Controller (772150)

Enter SET, CLEAR, SHOW, HELP, EXIT, or QUIT
Node   CSR Address   Model
 0       772150       21
 1       760334       21
 4       760340       21
 5       760344       21
 7     ------ KFQSA ------
? help
Commands:
    SET <node> /KFQSA                   set KFQSA DSSI node number
    SET <node> <CSR_address> <model>    enable a DSSI device
    CLEAR <node>                        disable a DSSI device
    SHOW                                show current configuration
    HELP                                print this text
    EXIT                                program the KFQSA
    QUIT                                don't program the KFQSA
Parameters:
    <node>                              0 to 7
    <CSR_address>                       760010 to 777774
    <model>                             21 (disk) or 22 (tape)
? set 6 /kfqsa
? show
Node   CSR Address   Model
 0       772150       21
 1       760334       21
 4       760340       21
 5       760344       21
 6     ------ KFQSA ------
? exit
Programming the KFQSA...
>>>
>>>set language 5
>>>
>>>set recall 1
>>>
```

SHOW

Format

SHOW *{parameter}*

Qualifiers

—

Depends on the parameters used.

Parameters

BFL(A)G
Show the default R5 boot flags.

BOOT
Show the default boot device.

CONTROLP
Show the current state of Ctrl/P halt recognition, either ENABLED or DISABLED.

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 console 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 the hardware Ethernet address for all Ethernet adapters that can be found. Displays as blank, if no Ethernet adapter is present.

HALT
Show the user defined halt action. One of the following keywords are displayed: DEFAULT, RESTART, REBOOT, HALT, or RESTART_REBOOT. See Table 12–1 for usage.

LANGUAGE
Show the 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 bit map.

/FULL

Additionally show the normally inaccessible areas of memory, such as, the PFN bit map 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 Ctrl/C to terminate the command. The command disables the scatter/gather map for the duration of the command.

RECALL

Show the current state of command recall, either ENABLED or DISABLED.

RLV12

Show all RL01 and RL02 disks which appear on the Q22-bus.

SCSI

Show any SCSI devices in the system.

TRANSLATION

Show any virtual addresses which map to the specified physical address. The firmware uses the current values of page table base and length registers to perform its search; it is assumed that page tables have been properly built.

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 console 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 version of the firmware.

Qualifiers

—

Depends on the parameter used.

Arguments

None.

Description

Displays the console parameter indicated.

Examples

```
>>>
>>>show bflag
00000220
>>>
>>>show boot
 EZA0
>>>
>>>show device
DSSI Node 0 (SUSAN)
-DIA0 (RF30)

DSSI Node 1 (KAREN)
-DIA1 (RF30)

DSSI Node 3 (*)

DSSI Node 4 (WILMA)
-DIA4 (RF30)

DSSI Node 5 (BETTY)
-DIA5 (RF30)

DSSI Node 6 (KFQSA)

SCSI Adapter 0 (761300), SCSI ID 7
-DKA100 (DEC RZ31     (C) DEC)
-DKA300 (MAXTOR XT-8000S)

UQSSP Disk Controller 0 (772150)
-DUA0 (RF30)

UQSSP Disk Controller 1 (760334)
-DUB1 (RF30)

UQSSP Disk Controller 2 (760340)
-DUC3 (RF30)

UQSSP Disk Controller 3 (760344)
-DUD4 (RF30)

Ethernet Adapter
-EZA0 (08-00-2B-03-82-78)
>>>
>>>show dssi
DSSI Node 0 (SUSAN)
-DIA0 (RF30)

DSSI Node 1 (KAREN)
-DIA1 (RF30)

DSSI Node 3 (*)

DSSI Node 4 (WILMA)
-DIA4 (RF30)
```

```
DSSI Node 5 (BETTY)
-DIA5 (RF30)

DSSI Node 6 (KFQSA)
>>>
<>>>show ethernet
Ethernet Adapter
-EZA0 (08-00-2B-03-82-78)
>>>
>>>show halt
Reboot
>>>show language
English (United States/Canada)
>>>
>>>show memory
Memory 0: 00000000 to 003FFFFF, 4MB, 0 bad pages

Total of 4MB, 0 bad pages, 98 reserved pages
>>>
>>>show memory /full
Memory 0: 00000000 to 003FFFFF, 4MB, 0 bad pages

Total of 4MB, 0 bad pages, 98 reserved pages

Memory bit map
-003F3C00 to 003F3FFF, 2 pages

Console Scratch Area
-003F4000 to 003F7FFF, 32 pages

Qbus Map
-003F8000 to 003FFFFF, 64 pages

Scan of Bad Pages
>>>
>>>show qbus
Scan of Qbus I/O Space
-200000DC (760334) = 0000 (300) RQDX3/KDA50/RRD50/RQC25/KFQSA-DISK
-200000DE (760336) = 0AA0
-200000E0 (760340) = 0000 (304) RQDX3/KDA50/RRD50/RQC25/KFQSA-DISK
-200000E2 (760342) = 0AA0
-200000E4 (760344) = 0000 (310) RQDX3/KDA50/RRD50/RQC25/KFQSA-DISK
-200000E6 (760346) = 0AA0
-20001468 (772150) = 0000 (154) RQDX3/KDA50/RRD50/RQC25/KFQSA-DISK
-2000146A (772152) = 0AA0
-20001F40 (777500) = 0020 (004) IPCR

Scan of Qbus Memory Space
>>>
>>>show rlv12
>>>
>>>show scsi
SCSI Adapter 0 (761300), SCSI ID 7
-DKA100 (DEC RZ31     (C) DEC)
-DKA300 (MAXTOR XT-8000S)
>>>
>>>show translation 1000
 V 80001000
>>>
>>>show uqssp
UQSSP Disk Controller 0 (772150)
-DUA0 (RF30)

UQSSP Disk Controller 1 (760334)
-DUB1 (RF30)

UQSSP Disk Controller 2 (760340)
-DUC4 (RF30)
```

```
UQSSP Disk Controller 3 (760344)
-DUD5 (RF30)
>>>
>>>show version
KA660-A V4.0, VMB 2.12
>>>
```

START

Format

START *[address]*

Qualifiers

None.

Arguments

[address]
The address at which to begin execution. This is loaded in the user's PC.

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.

Examples

```
>>>start  1000
```

TEST

Format

TEST *[test_number [test_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 individual test.

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.

A more detailed explanation of the diagnostics may be found in Section 12.8.

Examples

```
>>>            ! Execute the power-up diagnostic script
>>>            ! Warning...this has the same affect as a power-up!
>>>
>>>test 0
95..94..93..92..91..90..89..88..87..86..85..84..83..82..81..80..
79..78..77..76..75..74..73..72..71..70..69..68..67..66..65..64..
63..62..61..60..59..58..57..56..55..54..53..52..51..50..49..48..
47..46..45..44..43..42..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
_____________________________________________________________________________
    20052400  SCB
    2005331C  De_executive
30  2005D474  MS650_Init_bit map  *** mark_Hard_SBEs ******
31  2005D234  MS650_Setup_CSRs    **********
32  2005CD7C  CMCTL regs          MEMCSR0_addr *********
33  2005CD38  CMCTL_powerup       *
34  20054A04  SSC_ROM             *
3E  2005B323  SOC_Cache_w_mem_errs sel_mask st_add LW0_pat LW1_pat test_add
****
3F  2005F298  MS650_FDM_Addr_shorts *** cont_on_err ******
40  2005FA80  MS650_count_pages   First_board Last_bd Soft_errs_allowed *******
41  20062BA0  Board_Reset         *
42  20054AC4  Chk_for_Interrupts  *****
43  2005AE00  SOC_DI_Cache_w_mem  cache_config start_add end_add add_incr ******
44  2005A600  SOC_D_Cache_w_Mem   cache_config start_add end_add add_incr ******
45  2005A1FC  SOC_Cache_mem_CQBIC cache_config start_add end_add add_incr
******
46  2005B670  SOC_Cache1_diag_mode cache_config addr_incr ********
47  2005F810  MS650_Refresh       start_a end incr cont_on_err time_seconds
*****
48  2005EED0  MS650_Addr_shorts   start_add end_add * cont_on_err pat2 pat3 ****
49  2005E928  MS650_FDM           *** cont_on_err ******
4A  2005E680  MS650_ECC_SBEs      start_add end_add add_incr cont_on_err ******
4B  2005E440  MS650_Byte_Errors   start_add end_add add_incr cont_on_err ******
4C  2005DEE4  MS650_ECC_Logic     start_add end_add add_incr cont_on_err ******
4D  2005DD54  MS650_Address       start_add end_add add_incr cont_on_err ******
4E  2005DB60  MS650_Byte          start_add end_add add_incr cont_on_err ******
4F  2005D8B4  MS650_Data          start_add end_add add_incr cont_on_err ******
51  20062C75  FPA                 *******
52  20055070  SSC_Prog_timers     which_timer wait_time_us ***
53  20055340  SSC_TOY_Clock       repeat_test_250ms_ea Tolerance ***
54  20054B99  Virtual_Mode        *********
55  200554F2  Interval_Timer      *
58  20061748  SHAC_RESET          dssi_bus port_number time_secs
59  200608A0  SGEC_LPBCK_ASSIST   time_secs **
5A  2005A100  SOC_CMCTL           dont_report_memory_bad repeat_count *
5C  20060E08  SHAC                shac_number *******
5F  2005FC54  SGEC                loopback_type no_ram_tests ******
60  20059D31  SSC_Console_SLU     start_BAUD end_BAUD ******
62  20055930  console_QDSS        mark_not_present selftest_r0 selftest_r1 *****
63  20055AAC  QDSS_any            input_csr selftest_r0 selftest_r1 ******
80  20059711  CQBIC_memory_LMGH   **********
81  20055594  Qbus_MSCP           IP_csr ******
82  20055759  Qbus_DELQA          device_num_addr ****
83  200569AA  QZA_LPBCK1          controller_number ********
84  20058050  QZA_LPBCK2          controller_number *********
85  20055C08  QZA_memory          incr test_pattern controller_number *******
86  200560C4  QZA_DMA             Controller_number main_mem_buf ********
87  20059290  QZA_EXTLPBCK        controller_number ****
90  20054FEE  CQBIC_registers     *
91  20054F84  CQBIC_powerup       **
99  20062E83  Flush_Ena_Caches    dis_flush_cache
9A  20061D88  INTERACTION         pass_count disable_device ****
9B  20062A60  Init_memory_4MB     *
9C  2005BF3F  List_CPU_registers  *
9D  2005CC1B  Utility             Expnd_err_msg get_mode init_LEDs clr_ps_cnt
9E  20055566  List_diagnostics    *
9F  20062F25  Create_A0_Script    **********
C1  200546B0  SSC_RAM_Data        *
C2  20054886  SSC_RAM_Data_Addr   *
C5  20059561  SSC_registers       *
C6  200545F4  SSC_powerup         *********
```

```
C7  2005965C  SSC_CBTCR_timeout  ***

Scripts
#   Description

A0  User defined scripts
A1  Powerup tests, Functional Verify, continue on error, numeric countdown
A3  Functional Verify, stop on error, test # announcements
A4  Loop on A3 Functional Verify
A5  Address shorts test, run fastest way possible
A6  Memory tests, mark only multiple bit errors
A7  Memory tests
A8  Memory acceptance tests, mark single and multi-bit errors, call A7
A9  Memory tests, stop on error
B5  SOC Cache debug script
>>>
>>>             ! Show the diagnostic state.
>>>
>>>t fe
 bit map=00FF3000, Length=00001000, Checksum=807F, Busmap=00FF8000
 Test_number=41, Subtest=00, Loop_Subtest=00, Error_type=00
 Error_vector=0000, Last_exception_PC=00000000, Severity=02
 Total_error_count=0000, Led_display=0C, Console_display=03, save_mchk_code=80
 parameter_1=00000000  2=00000000  3=00000000  4=00000000  5=00000000
 parameter_6=00000000  7=00000000  8=00000000  9=00000000 10=00000000
 previous_error=00000000, 00000000, 00000000, 00000000
 Flags=03FFFC10440E Set_mask=FF
 Return_stack=201406D4, Subtest_pc=20062BB8, Timeout=00030D40
>>>
>>>             ! Display the CPU registers.
>>>
>>>t 9c
  SBR=00FB8000     SLR=00002021  SAVPC=200449C9 SAVPSL=04190304     SCBB=20052400
 P0BR=80000000    P0LR=00100A80   P1BR=0A0A0A08   P1LR=000B0B0B      SID=14000006
 TODR=00AD7BC6    ICCS=00000000                  MAPEN=00000000    BDMTR=20084000
 TCR0=00000005    TIR0=109890DE  TNIR0=00000000  TIVR0=00000078    BDMKR=0000007C
 TCR1=00000001    TIR1=109DD93A  TNIR1=0000000F  TIVR1=0000007C
 RXCS=00000000    RXDB=0000000D   TXCS=00000000   TXDB=00000030
  SCR=0000D000    DSER=00000000  QBEAR=0000000F   DEAR=00000000    QBMBR=00FF8000
  BDR=08D0EFFF   DLEDR=0000000C  SSCCR=00D55537  CBTCR=00000004    IPCR0=0000

DSSI_0=00  (BUS_0)     PQBBR_0=03060022    PMCSR_0=00000000    SSHMA_0=0000CA20
     PSR_0=00000000     PESR_0=00000000     PFAR_0=00000000      PPR_0=00000000

NICSR0=1FFF0003   3=00004030   4=00004050   5=8039FF00   6=83E0F000  7=00000000
NICSR9=04E204E2  10=00040000  11=00000000  12=00000000  13=00000000 15=0000FFFF
NISA=08-00-2B-12-BC-
AC   RDES0=00441300   1=00000000   2=05EE0000  3=000046F0
                         TDES0=00008C80   1=07000000   2=00400000  3=000040FA

MEM_FRU 1      MCSR_0=80000017       1=80400017       2=80800017       3=80C00017
MEM_FRU 2      MCSR_4=00000000       5=00000000       6=00000000       7=00000000
MEM_FRU 3      MCSR_8=00000000       9=00000000      10=00000000      11=00000000
MEM_FRU 4      MCSR12=00000000      13=00000000      14=00000000      15=00000000
MEMCSR17=00000013  MEMCSR16=00000044  CSR16_page_address=00000000
MSER=00000000  CCR=00000010
>>>
```

UNJAM

Format

UNJAM

Qualifiers

None.

Arguments

None.

Description

An I/O bus reset is performed. This is implemented by writing 1 to IPR 55. Additionally, the SGEC and SHAC chips are explicitly software reset, because PR$_IORESET has no affect on them.

Examples

```
>>>unjam
>>>
```

X - Binary Load and Unload

Format

X *{address} {count} <CR> {line_checksum} {data} {data_checksum}*

Qualifiers

None.

Arguments

None.

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 console, and deposited into memory. If bit 31 of the count is set, data is to be read from memory and sent by the console. The remaining bits in the count are a positive number indicating the number of bytes to load or unload.

The console accepts the command upon receiving the carriage return. The next byte the console receives is the command checksum, which is not echoed. The command checksum is verified by adding all command characters, including the checksum and separating whitespace, (but not including the terminating carriage return or 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 console 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 contents of the register is nonzero, the data or checksum are in error, and the console responds with an error message.

If the command is a binary unload (bit 31 of the count is set), the console 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 2'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, and 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 when the Return key command is received.

It is possible to control the console serial line through the use of the control characters (Ctrl/C, Ctrl/S, Ctrl/O, and so on) during a binary unload. It is not possible during a binary load, as all received characters are valid binary data.

Data being loaded with a binary load command must be received by the console at a rate of at least one byte every 60 seconds. The command checksum that precedes the data must be received by the console within 60 seconds of the carriage return that terminates the command line. The data checksum must be received within 60 seconds of the last data byte. If any of these timing requirements are not met, the console aborts the transmission by issuing an error message and prompting for input.

The entire command, including the checksum, can be sent to the console as a single burst of characters at the console serial lines' specified character rate. The console is able to receive at least 4 Kbyte of data in a single X command.

! - Comment

Format

!

Qualifiers

None.

Arguments

None.

Description

The comment (!) command is used to include optional text which you can use to identify a command line or add descriptions. It can appear anywhere on the command line. All characters following the comment character are ignored.

Examples

```
>>>! The console ignores this line.
>>>
```

Table 12–11 provides a summary of the console commands.

Table 12–11 Console Command Summary

Command	Qualifier Elements	Argument Elements
BOOT	[/R5:{boot_flags} /{boot_flags}]	[boot_device]
CONFIGURE	—	—
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 BFL(A)G	—	{bit map}
SET BOOT	—	{device_name} or {device_list}
SET CONTROLP	—	{value} or {keyword}
SET HALT	—	{halt_action}
SET HOST	/DUP /DSSI /BUS:{value}	{node_number} [task]
SET HOST	/DUP /UQSSP {/DISK ! /TAPE } /DUP /UQSSP	{controller_number}{csr_address} [task][task]
SET HOST	/MAINTENANCE /UQSSP /SERVICE /MAINTENANCE /UQSSP	{controller_number}{csr_address}
SET LANGUAGE	—	{language_type}
SET RECALL	—	{value}
SHOW BFL(A)G	—	—
SHOW BOOT	—	—
SHOW CONTROLP	—	—
SHOW DEVICE	—	—
SHOW DSSI	—	—
SHOW ETHERNET	—	—
SHOW HALT	—	—

Table 12–11 (Cont.) Console Command Summary

Command	Qualifier Elements	Argument Elements
SHOW LANGUAGE	—	—
SHOW MEMORY	/FULL	—
SHOW QBUS	—	—
SHOW RECALL	—	—
SHOW RLV12	—	—
SHOW SCSI	—	—
SHOW TRANSLATION	—	{phys_address}
SHOW UQSSP	—	—
SHOW VERSION	—	—
START	—	[address]
TEST	—	{test_number} [parameters]
UNJAM	—	—
X	—	{address}{count}<CR>{line_checksum}{data} <{data_checksum}
!	—	—

Table 12–12 provides a summary of the console qualifiers.

Table 12–12 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	On writes, use the value of 3, which always generates double-bit errors. Ignore ECC errors on reads of main memory.

Table 12–12 (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 bit map, and so on)
/M	Machine state (PSL)
Command Specific	
/INSTRUCTION	EXAMINE command only. Disassemble the instruction at address specified.
/NOT	SEARCH command only. Invert the sense of the match.
/R5:{boot_flags}, /{boot_flags}	BOOT command only. Specify a function bit map to pass to VMB through R5. Refer to Figure 12–9 for a bit description of R5. Either form of the command is acceptable.
/RPB, /MEMORY	FIND command only. Search for valid RPB or good block of memory.
/DUP, /DSSI, /UQSSP, /DISK, /TAPE, /MAINTENANCE,	SET HOST command only. Refer to command description for usage.
/SERVICE	

Nomenclature for Table 12–11 and Table 12–12

UPPERCASE denotes the command or qualifier keyword.
{} denotes a mandatory item which must be syntactically correct.
[ ] denotes an optional item.
! denotes a logical OR condition.
boot_flags, count, size, address, and parameters denote hex longword values.
boot_device denotes a legal boot device name.
csr_address denotes a Q22-bus I/O page CSR address.
controller_number denotes a controller number from 0 to 255.
halt_action denotes the value of the user-defined halt action from 0 to 4.
language_type denotes the language value, from 1 to 15.
command denotes a console command other than REPEAT.
data, pattern, and mask denote hex values of the current size.
test_number denotes hex byte test number.

12.8 Diagnostics

The ROM-based diagnostics constitute the bulk of the firmware on the KA660. These diagnostics run automatically on power-up and can be executed interactively as a whole, or as individual tests using the TEST command (see Section 12.7, Console Commands). This section summarizes the operation of the ROM-based diagnostics.

The purpose of the ROM-based diagnostics is multifaceted:

1. During power-up, they determine if enough of the KA660 is working to allow the console to run.
2. During the manufacturing process, they verify that the board was correctly built.

2. During the manufacturing process, they verify that the board was correctly built.
3. In the field, they verify that the board is operational, and able to report all detected errors.
4. They allow sophisticated users and field service technicians to run individual diagnostics interactively, with the intent of isolating errors to the field replaceable unit (FRU).

To accomodate these requirements, the diagnostics are designed as a collection of individual parameterized tests. A data structure, called a *script*, and a program, called the *diagnostic executive*, orchestrate the running of these tests in the right order with the right parameters.

A script is a data structure that points to various tests. There are several scripts, one for the field, and several for manufacturing, depending on where on the manufacturing line the board is. Sophisticated users may also create their own scripts interactively. Additionally, the script contains other information:

- What parameters need to be passed to the test
- What is to be displayed, if anything, on the console
- What is to be displayed, if anything, on the LED
- What to do on errors (halt, loop, or continue)
- Where the tests may be run from

  For example, there are certain tests that can only be run from the EPROM. Other tests are PIC (Position Independent Code), and may be run from EPROM or main memory to save time in executing commands.

The diagnostic executive "interprets" scripts to determine what tests are to be run. There are several built-in scripts on the KA660 that are used for manufacturing, power-up, and field service personnel. The diagnostic executive automatically invokes the correct script based on the current environment of the KA660. Any script can be explicitly run with the TEST command from the console terminal.

The diagnostic executive is also responsible for controlling the tests so that when errors occur, they can be caught and reported to the user. The executive also ensures that when the tests are run, the machine is left in a consistent and well-defined state.

12.8.1 Error Reporting

Before a console is established, the only error reporting is through the KA660 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.

Figure 12–14 shows a typical error display.

```
?9A 2 02 FF 0000 0000 01    ; SUBTEST_9A_02, DE_INTERACTION.LIS      (1)

P1=00000002  P2=00000000  P3=00004000  P4=00008000  P5=0000C000      (2)
P6=00000000  P7=00000002  P8=00000002  P9=84004000 P10=00001FFF      (3)
r0=00000054  r1=00000040  r2=00000000  r3=0000C524  r4=00000014      (4)
r5=30002800  r6=0000C4E0  r7=20008000  r8=00004000 EPC=20057BBD      (5)

Normal operation not possible.
```

Figure 12–14 Diagnostic Register Dump

In Figure 12–14, the numbers in parentheses on the right side of the figure, refer to lines of the display and are not a part of the diagnostic dump. The information on these lines is summarized below.

1. Test summary containing six hexadecimal fields. ?9A
   This test identifies the diagnostic test.

   a. 2
      This is the severity level of a test failure, as dictated by the script. A severity level 2 error causes the display of this five-line error printout, and halts an autoboot to console I/O mode. A severity level 1 error displays the first line of the error printout, but does not interrupt an autoboot. Most tests have a severity level of 2.

   b. 02
      This is the subtestlog number. In conjunction with listing files, it isolates, to within a few instructions, where the diagnostic detected the error.

   c. FF
      This is the de_error 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. 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 0000

      This is the SCB vector (if non-zero) through which an unexpected exception or interrupt trapped, when the **de_error** field indicates an unexpected exception or interrupt (FE or EF).

   d. 0000
      This is the number of previous errors that have occurred.

   e. 01
      This loop_subtest is an additional subtestlog generated out of the context of the current test as specified by the current test number and subtestlog. Usually these logs occur in common subroutines called from a diagnostic test.

   f. SUBTEST_9A_02
      This subtest_symbol is a unique symbol which identifies the most recent subtestlog entry in the listing file.

g. DE_INTERACTION.LIS
   This listing_file is the name of the listing file which contains the failed diagnostic.

2. P1 through P5 are the first five parameters containing diagnostic state.
3. P6 through P10 are the last five parameters containing diagnostic state.
4. R0 through R4 are the first five GPRs at the moment the error was detected.
5. R5 through R8 are additional GPRs, and EPC is PC at the time of the error.

The use of parameters and registers varies with each test. The appropriate listing file should be consulted for interpretation of these parameters and registers in determining diagnostic state.

12.8.2 Diagnostic Interdependencies

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.

12.8.3 Areas Not Covered

The goal has been to achieve the highest possible coverage on the KA660 and the memory boards. However, the testing of the KA660 while running with memory management turned on is minimal. Also, due to the way the firmware is implemented (a polled environment running at IPL 31), the testing of interrupts is not extensive.

These diagnostics are not intended to be used as system-level tests. There are no tests that completely verify that access to the Q22-bus will work. Thus, a disk, a controller, the backplane, or portions of the CQBIC may be faulty, and the diagnostics may not detect the fault. Such a fault may later result as an inability to boot.

12.8.4 Diagnostic Scripts

Table 12–13 lists the firmware diagnostic scripts. Each entry in the following table corresponds to a test in a script which is executed. The tests can be invoked at the console prompt. Refer to the examples given for the TEST command in the command directory (Section 12.7). Refer to the table key for the meaning of each table entry.

Table Key

#..—This sequence number indicates the position of the test in the sequence of tests. The sequence number is displayed on the console terminal while a test is running and may or may not correspond to the test number. (Note: A dash (–) in this column indicates that no sequence number is displayed while the test is running.)
Test—This is the test number used to identify the specific test that is running.
LED—The Light Emitting Diode code is the HEX number code that is displayed on the console module while the test is running.
Name—This is the test name used to identify each test. The name is somewhat indicative of what function is being tested.
Conditions Set by the Firmware—Before a test is run, the firmware establishes a specific set of conditions under which the test will be run. The condition abbreviations are interpreted as follows:

NER-0 Error reporting ON
RPE-1 Error reporting OFF
CON-0 Continue on an error
STP-1 Stop on an error
SV1-1 Severity level 1
SV2-2 Severity level 2
VOF-0 Virtual mode ON
VON-1 Virtual mode OFF
RHP-0 Run halt protected
RHU-1 Run halt unprotected
ROM-0 Execute from ROM
RAM-2 Execute from RAM
FAST-3 Execute fast mode

Table 12–13 Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A1:				
95	9D	C	Utility	RPE - CON - SV2 - VOF - RHP - ROM
94	42	B	Check_for_intrs	RPE - CON - SV2 - VOF - RHP - ROM
93	33	8	CMCTL_chk_init	RPE - CON - SV2 - VOF - RHP - ROM
92	32	8	CMCTL_registers	RPE - CON - SV2 - VOF - RHP - ROM
91	31	8	CSR_setup	RPE - CON - SV2 - VOF - RHP - ROM
90	30	8	Map_setup	RPE - STP - SV2 - VOF - RHP - ROM
89	54	B	Virtual	RPE - CON - SV2 - VOF - RHP - ROM
88	49	8	Memory_test_fdm	RPE - CON - SV2 - VOF - RHP - ROM
87	60	6	Serial_line	RPE - CON - SV2 - VOF - RHP - ROM

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A1:				
85	90	7	Registers	RPE - CON - SV2 - VOF - RHP - ROM
84	C6	C	CSSC_chk_init	RPE - CON - SV2 - VOF - RHP - ROM
83	52	C	PROG_TIME	RPE - CON - SV2 - VOF - RHP - ROM
82	52	C	PROG_TIME	RPE - CON - SV2 - VOF - RHP - ROM
81	53	C	TOY	RPE - CON - SV2 - VOF - RHP - ROM
80	C1	C	SSC_RAM	RPE - CON - SV2 - VOF - RHP - ROM
79	34	C	ROM_logic	RPE - CON - SV2 - VOF - RHP - ROM
78	C5	C	SSC_registers	RPE - CON - SV2 - VOF - RHP - ROM
76	C7	C	CBTCR_timeout	RPE - CON - SV2 - VOF - RHP - ROM
75	46	B	SOC_cache_diag_mode	NER - CON - SV1 - VOF - RHU - ROM
74	46	B	SOC_cache_diag_mode	NER - CON - SV1 - VOF - RHU - ROM
73	46	B	SOC_cache_diag_mode	NER - CON - SV1 - VOF - RHU - ROM
72	46	B	SOC_cache_diag_mode	NER - CON - SV1 - VOF - RHU - ROM
71	46	B	SOC_cache_diag_mode	NER - CON - SV1 - VOF - RHU - ROM
70	46	B	SOC_cache_diag_mode	NER - CON - SV1 - VOF - RHU - ROM
69	46	B	SOC_cache_diag_mode	NER - CON - SV1 - VOF - RHU - ROM
68	46	B	SOC_cache_diag_mode	NER - CON - SV1 - VOF - RHU - ROM
67	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
66	4F	8	Memory_data	RPE - CON - SV2 - VOF - RHP - ROM
65	4E	8	Memory_byte	RPE - CON - SV2 - VOF - RHP - FAST
64	4D	8	Memory_addr	RPE - CON - SV2 - VOF - RHP - FAST
63	4C	8	Memory_ECC_error	RPE - CON - SV2 - VOF - RHP - FAST
62	4B	8	Mask_write_w_errs	RPE - CON - SV2 - VOF - RHP - FAST
61	4A	8	ECC_correction	RPE - CON - SV2 - VOF - RHP - FAST
60	3F	8	Mem_FDM_addr_shorts	RPE - CON - SV2 - VOF - RHP - FAST
59	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
58	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
57	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
56	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A1:				
55	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
54	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
53	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
52	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
51	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
50	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
47	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
46	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
45	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
44	48	8	Addr_shrts	RPE - CON - SV2 - VOF - RHP - FAST
43	47	8	Memory_refresh	RPE - CON - SV2 - VOF - RHP - FAST
42	40	8	Count_bad_pages	RPE - CON - SV1 - VOF - RHP - ROM
40	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
39	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
38	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
37	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
36	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
35	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
34	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
33	C2	C	SSC_RAM_addr_ shrts	RPE - CON - SV2 - VOF - RHP - ROM
32	80	7	CQBIC_memory	RPE - CON - SV2 - VOF - RHP - FAST
31	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
30	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
29	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
27	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
26	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
24	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
23	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A1:				
21	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
20	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
19	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
18	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
17	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
16	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
15	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
14	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
13	5A	8	SOC_CMCTL	RPE - CON - SV2 - VOF - RHP - RAM
12	51	A	FPA	RPE - CON - SV2 - VOF - RHP - FAST
11	5F	4	SGEC_func	RPE - CON - SV2 - VOF - RHP - ROM
10	5C	5	SHAC_func	RPE - CON - SV2 - VOF - RHP - ROM
09	9A	8	Interaction_func	RPE - CON - SV2 - VOF - RHP - FAST
08	83	7	Qza_lpbck1	RPE - CON - SV2 - VOF - RHP - ROM
07	84	7	Qza_lpbck2	RPE - CON - SV2 - VOF - RHP - ROM
06	85	7	Qza_memory	RPE - CON - SV2 - VOF - RHP - ROM
05	86	7	Qza_dma	RPE - CON - SV2 - VOF - RHP - ROM
04	99	B	Flush_ena_caches	RPE - CON - SV2 - VOF - RHP - ROM
03	41	C	Board_reset	RPE - CON - SV2 - VOF - RHP - ROM
script_A2:				
9D	9D	C	Utility	RPE - STP - SV2 - VOF - RHP - ROM
42	42	B	Check_for_intrs	RPE - STP - SV2 - VOF - RHP - ROM
C6	C6	C	CSSC_chk_init	RPE - STP - SV2 - VOF - RHP - ROM
60	60	6	Serial_line	RPE - STP - SV2 - VOF - RHP - ROM
52	52	C	PROG_TIME	RPE - STP - SV2 - VOF - RHP - ROM
52	52	C	PROG_TIME	RPE - STP - SV2 - VOF - RHP - ROM
53	53	C	TOY	RPE - STP - SV2 - VOF - RHP - ROM
C1	C1	C	SSC_RAM	RPE - STP - SV2 - VOF - RHP - ROM
34	34	C	ROM_logic	RPE - STP - SV2 - VOF - RHP - ROM

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A2:				
91	91	7	CQBIC_chk_init	RPE - STP - SV2 - VOF - RHP - ROM
C5	C5	C	SSC_registers	RPE - STP - SV2 - VOF - RHP - ROM
55	55	B	Interval_timer	RPE - STP - SV2 - VOF - RHP - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
99	99	B	Flush_ena_caches	RPE - STP - SV2 - VOF - RHP - ROM
90	90	7	Registers	RPE - STP - SV2 - VOF - RHP - ROM
32	32	8	CMCTL_registers	RPE - STP - SV2 - VOF - RHP - ROM
C7	C7	C	CBTCR_timeout	RPE - STP - SV2 - VOF - RHP - ROM
5C	5C	5	SHAC_func	RPE - STP - SV2 - VOF - RHP - ROM
script_A3:				
9D	9D	C	Utility	RPE - STP - SV2 - VOF - RHP - ROM
42	42	B	Check_for_intrs	RPE - STP - SV2 - VOF - RHP - ROM
33	33	8	CMCTL_chk_init	RPE - STP - SV2 - VOF - RHP - ROM
31	31	8	CSR_setup	RPE - STP - SV2 - VOF - RHP - ROM
30	30	8	Map_setup	RPE - STP - SV2 - VOF - RHP - ROM
54	54	B	Virtual	RPE - STP - SV2 - VOF - RHP - ROM
49	49	8	Memory_test_fdm	RPE - STP - SV2 - VOF - RHP - ROM
60	60	6	Serial_line	RPE - STP - SV2 - VOF - RHP - ROM
91	91	7	CQBIC_chk_init	RPE - STP - SV2 - VOF - RHP - ROM
90	90	7	Registers	RPE - STP - SV2 - VOF - RHP - ROM
C6	C6	C	CSSC_chk_init	RPE - STP - SV2 - VOF - RHP - ROM
52	52	C	PROG_TIME	RPE - STP - SV2 - VOF - RHP - ROM

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A3:				
52	52	C	PROG_TIME	RPE - STP - SV2 - VOF - RHP - ROM
53	53	C	TOY	RPE - STP - SV2 - VOF - RHP - ROM
C1	C1	C	SSC_RAM	RPE - STP - SV2 - VOF - RHP - ROM
C5	C5	C	SSC_registers	RPE - STP - SV2 - VOF - RHP - ROM
55	55	B	Interval_timer	RPE - STP - SV2 - VOF - RHP - ROM
C7	C7	C	CBTCR_timeout	RPE - STP - SV2 - VOF - RHP - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
46	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
4F	4F	8	Memory_data	RPE - STP - SV2 - VOF - RHP - ROM
4E	4E	8	Memory_byte	RPE - STP - SV2 - VOF - RHP - FAST
4D	4D	8	Memory_addr	RPE - STP - SV2 - VOF - RHP - FAST

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A3:				
4C	4C	8	Memory_ECC_error	RPE - STP - SV2 - VOF - RHP - FAST
4B	4B	8	Mask_write_w_errs	RPE - STP - SV2 - VOF - RHP - FAST
4A	4A	8	ECC_correction	RPE - STP - SV2 - VOF - RHP - FAST
3F	3F	8	Mem_FDM_addr_ shorts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
47	47	8	Memory_refresh	RPE - STP - SV2 - VOF - RHP - FAST
40	40	8	Count_bad_pages	RPE - STP - SV2 - VOF - RHP - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
44	44	B	SOC_D_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A3:				
80	80	7	CQBIC_memory	RPE - STP - SV2 - VOF - RHP - FAST
45	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
45	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
45	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
45	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
45	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
45	45	7	Ccache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
45	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
45	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
43	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
43	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
43	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
43	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
43	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
43	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
43	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
43	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
43	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
43	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
5A	5A	8	SOC_CMCTL	RPE - STP - SV2 - VOF - RHU - RAM
51	51	A	FPA	RPE - STP - SV2 - VOF - RHP - FAST
5F	5F	4	SGEC_func	RPE - STP - SV2 - VOF - RHP - ROM
5C	5C	5	SHAC_func	RPE - STP - SV2 - VOF - RHP - ROM
9A	9A	8	Interaction_func	RPE - STP - SV2 - VOF - RHP - FAST
99	99	B	Flush_ena_caches	RPE - STP - SV2 - VOF - RHP - ROM
41	41	C	Board_reset	RPE - STP - SV2 - VOF - RHP - ROM
9D	9D	C	Utility	RPE - STP - SV2 - VOF - RHP - ROM

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A5:				
3F	3F	8	Mem_FDM_addr_ shorts	RPE - CON - SV2 - VOF - RHU - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
script_A6:				
30	30	8	Map_setup	RPE - STP - SV2 - VOF - RHP - ROM
4F	4F	8	Memory_data	RPE - STP - SV2 - VOF - RHP - ROM
4D	4D	8	Memory_addr	RPE - STP - SV2 - VOF - RHP - FAST
4C	4C	8	Memory_ECC_error	RPE - STP - SV2 - VOF - RHP - FAST
4B	4B	8	Mask_write_w_errs	RPE - STP - SV2 - VOF - RHP - FAST
4A	4A	8	ECC_correction	RPE - STP - SV2 - VOF - RHP - FAST
3F	3F	8	Mem_FDM_addr_ shorts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
47	47	8	Memory_refresh	RPE - STP - SV2 - VOF - RHP - FAST
40	40	8	Count_bad_pages	RPE - STP - SV2 - VOF - RHP - ROM
80	80	7	CQBIC_memory	RPE - STP - SV2 - VOF - RHP - FAST
script_A7:				
4F	4F	8	Memory_data	RPE - STP - SV2 - VOF - RHP - ROM
4E	4E	8	Memory_byte	RPE - STP - SV2 - VOF - RHP - FAST
4D	4D	8	Memory_byte	RPE - STP - SV2 - VOF - RHP - FAST

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A7:				
4C	4C	8	Memory_ECC_error	RPE - STP - SV2 - VOF - RHP - FAST
4B	4B	8	Mask_write_w_errs	RPE - STP - SV2 - VOF - RHP - FAST
4A	4A	8	ECC_correction	RPE - STP - SV2 - VOF - RHP - FAST
3F	3F	8	Mem_FDM_addr_shorts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
48	48	8	Addr_shrts	RPE - STP - SV2 - VOF - RHP - FAST
47	47	8	Memory_refresh	RPE - STP - SV2 - VOF - RHP - FAST
40	40	8	Count_bad_pages	RPE - CON - SV2 - VOF - RHU - ROM
80	80	7	CQBIC_memory	RPE - CON - SV2 - VOF - RHU - FAST
41	41	C	Board_reset	RPE - STP - SV2 - VOF - RHP - ROM
script_A8:				
31	31	8	CSR_setup	RPE - STP - SV2 - VOF - RHP - ROM
30	30	8	Map_setup	RPE - STP - SV2 - VOF - RHP - ROM
49	49	8	Memory_test_fdm	RPE - STP - SV2 - VOF - RHP - ROM

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_A9:				
4F	4F	8	Memory_data	RPE - STP - SV2 - VOF - RHP - ROM
4E	4E	8	Memory_byte	RPE - STP - SV2 - VOF - RHP - FAST
4D	4D	8	Memory_addr	RPE - STP - SV2 - VOF - RHP - FAST
4C	4C	8	Memory_ecc_error	RPE - STP - SV2 - VOF - RHP - FAST
4B	4B	8	Mask_write_w_errs	RPE - STP - SV2 - VOF - RHP - FAST
47	47	8	Memory_refresh	RPE - STP - SV2 - VOF - RHP - FAST
40	40	8	Count_bad_pages	RPE - CON - SV2 - VOF - RHU - ROM
41	41	C	Board_reset	RPE - CON - SV2 - VOF - RHU - ROM
script_AD:				
—	41	C	Board_reset	NER - CON - SV2 - VOF - RHP - FAST
—	30	8	Map_setup	NER - CON - SV2 - VOF - RHP - FAST
—	4F	8	Memory_data	NER - CON - SV2 - VOF - RHP - FAST
—	4E	8	Memory_byte	NER - CON - SV2 - VOF - RHP - FAST
—	4D	8	Memory_addr	NER - CON - SV2 - VOF - RHP - FAST
—	4C	8	Memory_ECC_error	NER - CON - SV2 - VOF - RHP - FAST
—	4B	8	Memory_ECC_error	NER - CON - SV2 - VOF - RHP - FAST
—	4A	8	ECC_correction	NER - CON - SV2 - VOF - RHP - FAST
—	3F	8	Mem_FDM_addr_shorts	NER - CON - SV2 - VOF - RHP - FAST
—	48	8	Addr_shrts	NER - CON - SV2 - VOF - RHP - FAST
—	40	8	Count_bad_pages	NER - CON - SV1 - VOF - RHP - ROM
—	80	7	CQBIC_memory	NER - CON - SV2 - VOF - RHP - FAST
—	41	C	Board_reset	NER - CON - SV2 - VOF - RHP - FAST
script_AE:				
—	31	8	CSR_setup	NER - CON - SV2 - VOF - RHP - FAST
—	41	C	Board_reset	NER - CON - SV2 - VOF - RHP - FAST
script_AF:				
—	80	7	CQBIC_memory	NER - CON - SV2 - VOF - RHP - FAST
—	41	C	Board_reset	NER - CON - SV2 - VOF - RHP - FAST

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_B3:				
—	9D	C	Utility	NER - STP - SV2 - VOF - RHP - ROM
—	42	B	Check_for_intrs	NER - STP - SV2 - VOF - RHP - ROM
—	32	8	CMCTL_registers	NER - STP - SV2 - VOF - RHP - ROM
—	31	8	CSR_setup	NER - STP - SV2 - VOF - RHP - ROM
—	30	8	Map_setup	NER - STP - SV2 - VOF - RHP - ROM
—	54	B	Virtual	NER - STP - SV2 - VOF - RHP - ROM
—	49	8	Memory_test_fdm	NER - STP - SV2 - VOF - RHP - ROM
—	60	6	Serial_line	NER - STP - SV2 - VOF - RHP - ROM
—	90	7	Registers	NER - STP - SV2 - VOF - RHP - ROM
—	52	C	PROG_TIME	NER - STP - SV2 - VOF - RHP - ROM
—	52	C	PROG_TIME	NER - STP - SV2 - VOF - RHP - ROM
—	53	C	TOY	NER - STP - SV2 - VOF - RHP - ROM
—	C1	C	SSC_RAM	NER - STP - SV2 - VOF - RHP - ROM
—	34	C	ROM_logic	NER - STP - SV2 - VOF - RHP - ROM
—	C5	C	SSC_registers	NER - STP - SV2 - VOF - RHP - ROM
—	55	B	Interval_timer	NER - STP - SV2 - VOF - RHP - ROM
—	C7	C	CBTCR_timeout	NER - STP - SV2 - VOF - RHP - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	4F	8	Memory_data	NER - STP - SV2 - VOF - RHP - ROM
—	4E	8	Memory_byte	NER - STP - SV2 - VOF - RHP - FAST
—	4D	8	Memory_addr	NER - STP - SV2 - VOF - RHP - FAST
—	4C	8	Memory_ECC_error	NER - STP - SV2 - VOF - RHP - FAST
—	4B	8	Mask_write_w_errs	NER - STP - SV2 - VOF - RHP - FAST
—	4A	8	ECC_correction	NER - STP - SV2 - VOF - RHP - FAST
—	3F	8	Mem_FDM_addr_ shorts	NER - STP - SV2 - VOF - RHP - FAST

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_B3:				
—	48	8	Addr_shrts	NER - STP - SV2 - VOF - RHP - FAST
—	47	8	Memory_refresh	NER - STP - SV2 - VOF - RHP - FAST
—	40	8	Count_bad_pages	NER - STP - SV2 - VOF - RHP - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	C2	C	SSC_RAM_addr_shrts	NER - STP - SV2 - VOF - RHP - ROM
—	80	7	CQBIC_memory	NER - STP - SV2 - VOF - RHP - FAST
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
—	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
-	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
-	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST
-	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_B3:				
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST
—	5A	8	SOC_CMCTL	NER - STP - SV2 - VOF - RHP - FAST
—	51	A	FPA	NER - STP - SV2 - VOF - RHP - FAST
—	5F	4	SGEC_func	NER - STP - SV2 - VOF - RHP - ROM
—	5C	5	SHAC_func	NER - STP - SV2 - VOF - RHP - ROM
—	9A	8	Interaction_func	NER - STP - SV2 - VOF - RHP - FAST
—	99	B	Flush_ena_caches	NER - STP - SV2 - VOF - RHP - ROM
—	41	C	Board_reset	NER - STP - SV2 - VOF - RHP - ROM
—	9D	C	Utility	NER - STP - SV2 - VOF - RHP - ROM
script_B4:				
—	42	B	Check_for_intrs	NER - STP - SV2 - VOF - RHP - ROM
—	32	8	CMCTL_registers	NER - STP - SV2 - VOF - RHP - ROM
—	54	B	Virtual	NER - STP - SV2 - VOF - RHP - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	46	B	SOC_cache_diag_ mode	NER - CON - SV1 - VOF - RHU - ROM
—	49	8	Memory_test_fdm	NER - STP - SV2 - VOF - RHP - ROM
—	90	7	Registers	NER - STP - SV2 - VOF - RHP - ROM
—	52	C	PROG_TIME	NER - STP - SV2 - VOF - RHP - ROM
—	53	C	TOY	NER - STP - SV2 - VOF - RHP - ROM

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_B4:				
—	C1	C	SSC_RAM	NER - STP - SV2 - VOF - RHP - ROM
—	34	C	ROM_logic	NER - STP - SV2 - VOF - RHP - ROM
—	C5	C	SSC_registers	NER - STP - SV2 - VOF - RHP - ROM
—	55	B	Interval_timer	NER - STP - SV2 - VOF - RHP - ROM
—	C7	C	CBTCR_timeout	NER - STP - SV2 - VOF - RHP - ROM
—	4F	8	Memory_data	NER - STP - SV2 - VOF - RHP - ROM
—	4E	8	Memory_byte	NER - STP - SV2 - VOF - RHP - FAST
—	4D	8	Memory_addr	NER - STP - SV2 - VOF - RHP - FAST
—	4C	8	Memory_ECC_error	NER - STP - SV2 - VOF - RHP - FAST
—	4B	8	Mask_write_w_errs	NER - STP - SV2 - VOF - RHP - FAST
—	4A	8	ECC_correction	NER - STP - SV2 - VOF - RHP - FAST
—	48	8	Addr_shrts	NER - STP - SV2 - VOF - RHP - FAST
—	40	8	Count_bad_pages	NER - STP - SV2 - VOF - RHP - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	44	B	SOC_D_cache_w_memory	NER - CON - SV1 - VOF - RHU - ROM
—	C2	C	SSC_RAM_addr_shrts	NER - STP - SV2 - VOF - RHP - ROM
—	80	7	CQBIC_memory	NER - STP - SV2 - VOF - RHP - FAST
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	45	7	Cache_mem_cqbic	NER - CON - SV1 - VOF - RHU - ROM
—	43	B	SOC_DI_cache_w_memory	NER - CON - SV1 - VOF - RHU - FAST

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_B4:				
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - ROM
—	43	B	SOC_DI_cache_w_ memory	NER - CON - SV1 - VOF - RHU - FAST
—	5A	8	SOC_CMCTL	NER - STP - SV2 - VOF - RHP - FAST
—	5F	4	SGEC_func	NER - STP - SV2 - VOF - RHP - ROM
—	5C	5	SHAC_func	NER - STP - SV2 - VOF - RHP - ROM
—	9A	8	Interaction_func	NER - STP - SV2 - VOF - RHP - FAST
—	41	C	Board_reset	NER - STP - SV2 - VOF - RHP - ROM
—	9D	C	Utility	NER - STP - SV2 - VOF - RHP - ROM
script_B5:				
46	46	B	SOC_cache_diag_ mode	RPE - CON - SV2 - VOF - RHP - ROM
46	46	B	SOC_cache_diag_ mode	RPE - CON - SV2 - VOF - RHP - ROM
46	46	B	SOC_cache_diag_ mode	RPE - CON - SV2 - VOF - RHP - ROM
46	46	B	SOC_cache_diag_ mode	RPE - CON - SV2 - VOF - RHP - ROM
46	46	B	SOC_cache_diag_ mode	RPE - CON - SV2 - VOF - RHP - ROM
46	46	B	SOC_cache_diag_ mode	RPE - CON - SV2 - VOF - RHP - ROM
46	46	B	SOC_cache_diag_ mode	RPE - CON - SV2 - VOF - RHP - ROM
44	44	B	SOC_D_cache_w_ memory	RPE - CON - SV2 - VOF - RHP - ROM

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_B5:				
44	44	B	SOC_D_cache_w_memory	RPE - CON - SV2 - VOF - RHP - ROM
44	44	B	SOC_D_cache_w_memory	RPE - CON - SV2 - VOF - RHP - ROM
44	44	B	SOC_D_cache_w_memory	RPE - CON - SV2 - VOF - RHP - ROM
44	44	B	SOC_D_cache_w_memory	RPE - CON - SV2 - VOF - RHP - ROM
44	44	B	SOC_D_cache_w_memory	RPE - CON - SV2 - VOF - RHP - ROM
44	44	B	SOC_D_cache_w_memory	RPE - CON - SV2 - VOF - RHP - ROM
44	44	B	SOC_D_cache_w_memory	RPE - CON - SV2 - VOF - RHP - ROM
45	45	7	Cache_mem_cqbic	RPE - CON - SV2 - VOF - RHP - ROM
45	45	7	Cache_mem_cqbic	RPE - CON - SV2 - VOF - RHP - ROM
45	45	7	Cache_mem_cqbic	RPE - CON - SV2 - VOF - RHP - ROM
45	45	7	Cache_mem_cqbic	RPE - CON - SV2 - VOF - RHP - ROM
45	45	7	Cache_mem_cqbic	RPE - CON - SV2 - VOF - RHP - ROM
45	45	7	Cache_mem_cqbic	RPE - CON - SV2 - VOF - RHP - ROM
45	45	7	Cache_mem_cqbic	RPE - CON - SV2 - VOF - RHP - ROM
45	45	7	Cache_mem_cqbic	RPE - CON - SV2 - VOF - RHP - ROM
43	43	B	SOC_DI_cache_w_memory	RPE - CON - SV2 - VOF - RHP - FAST
43	43	B	SOC_DI_cache_w_memory	RPE - CON - SV2 - VOF - RHP - FAST
43	43	B	SOC_DI_cache_w_memory	RPE - CON - SV2 - VOF - RHP - FAST
43	43	B	SOC_DI_cache_w_memory	RPE - CON - SV2 - VOF - RHP - FAST
43	43	B	SOC_DI_cache_w_memory	RPE - CON - SV2 - VOF - RHP - FAST
43	43	B	SOC_DI_cache_w_memory	RPE - CON - SV2 - VOF - RHP - FAST
43	43	B	SOC_DI_cache_w_memory	RPE - CON - SV2 - VOF - RHP - FAST

Table 12–13 (Cont.) Diagnostic Scripts

#..	Test	LED	Name	Conditions
script_B5:				
43	43	B	SOC_DI_cache_w_memory	RPE - CON - SV2 - VOF - RHP - FAST
43	43	B	SOC_DI_cache_w_memory	RPE - CON - SV2 - VOF - RHP - ROM
43	43	B	SOC_DI_cache_w_memory	RPE - CON - SV2 - VOF - RHP - FAST
99	99	B	Flush_ena_caches	RPE - CON - SV2 - VOF - RHP - ROM

A
Q22-bus Specification

A.1 Introduction

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 A–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 KN220-AA 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 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 through 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.

A.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.

A.2 Q22-bus Signal Assignments

Table A–1 lists the data and address signal assignments. Table A–2 lists the control signal assignments. Table A–3 lists the power and ground signal assignments. Table A–4 lists the spare signal assignments.

Table A–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

Table A–1 (Cont.) Data and Address Signal Assignments

Data and Address Signal	Pin Assignment
BDAL13	BT2
BDAL14	BU2
BDAL15	BV2
BDAL16	AC1
BDAL17	AD1
BDAL18	BC1
BDAL19	BD1
BDAL20	BE1
BDAL21	BF1

Table A–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

Table A–2 (Cont.) Control Signal Assignments

Control Signal	Pin Assignment
BDMGI	AR2
System Control	
BHALT	AP1
BREF	AR1
BEVNT	BR1
BINIT	AT2
BDCOK	BA1
BPOK	BB1

Table A–3 Power and Ground Signal Assignments

Power and Ground	Pin Assignment
+5 B (battery) or	AS1
+12 B (battery)	
+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 A–4 Spare Signal Assignments

Spare	Pin Assignment
SSpare1	AE1
SSpare3	AH1
SSpare8	BH1
SSpare2	AF1
MSpareA	AK1
MSpareB	AL1
MSpareB	BK1
MSpareB	BL1
PSpare1	AU1
ASpare2	BU1

A.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 A–5.

Table A–5 Data Transfer Operations

Bus Cycle	Definition	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

The bus signals listed in Table A–6 are used in the data transfer operations described in Table A–5.

Table A–6 Bus Signals for Data Transfers

Signal	Definition	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.

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.

A.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.

A.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 4K 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 A–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.

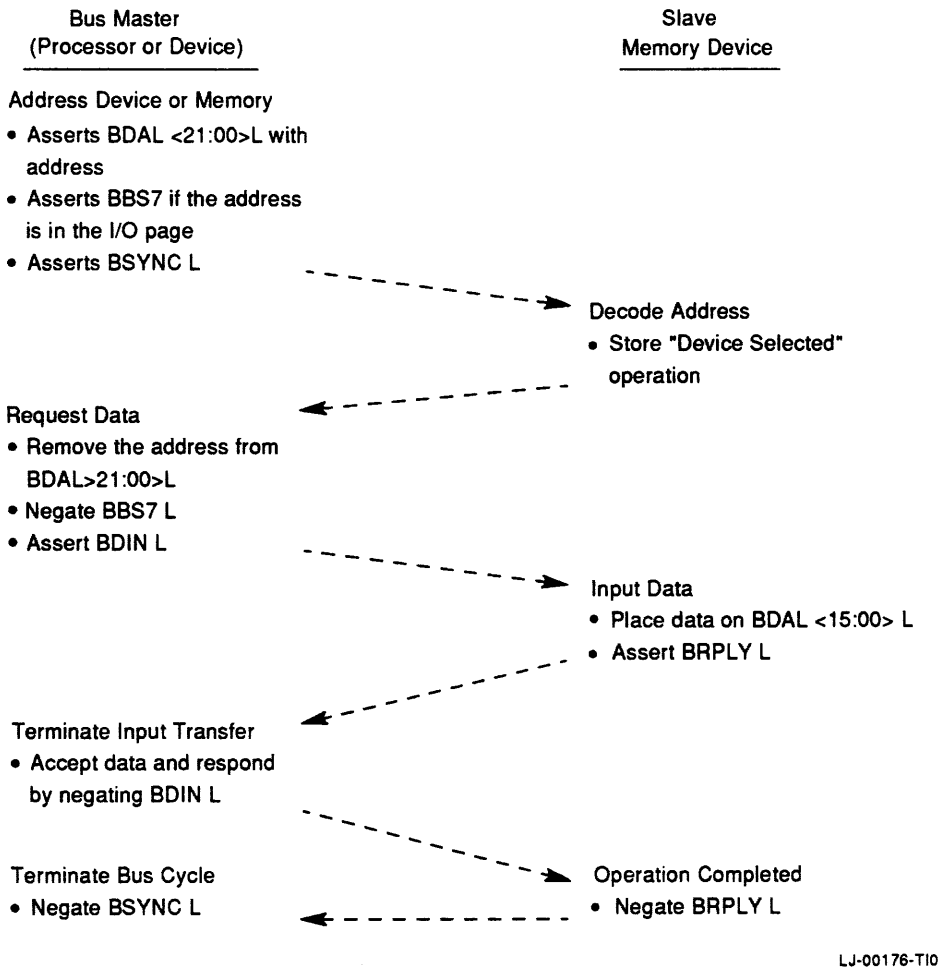


Figure A–1 DATI Bus Cycle

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:

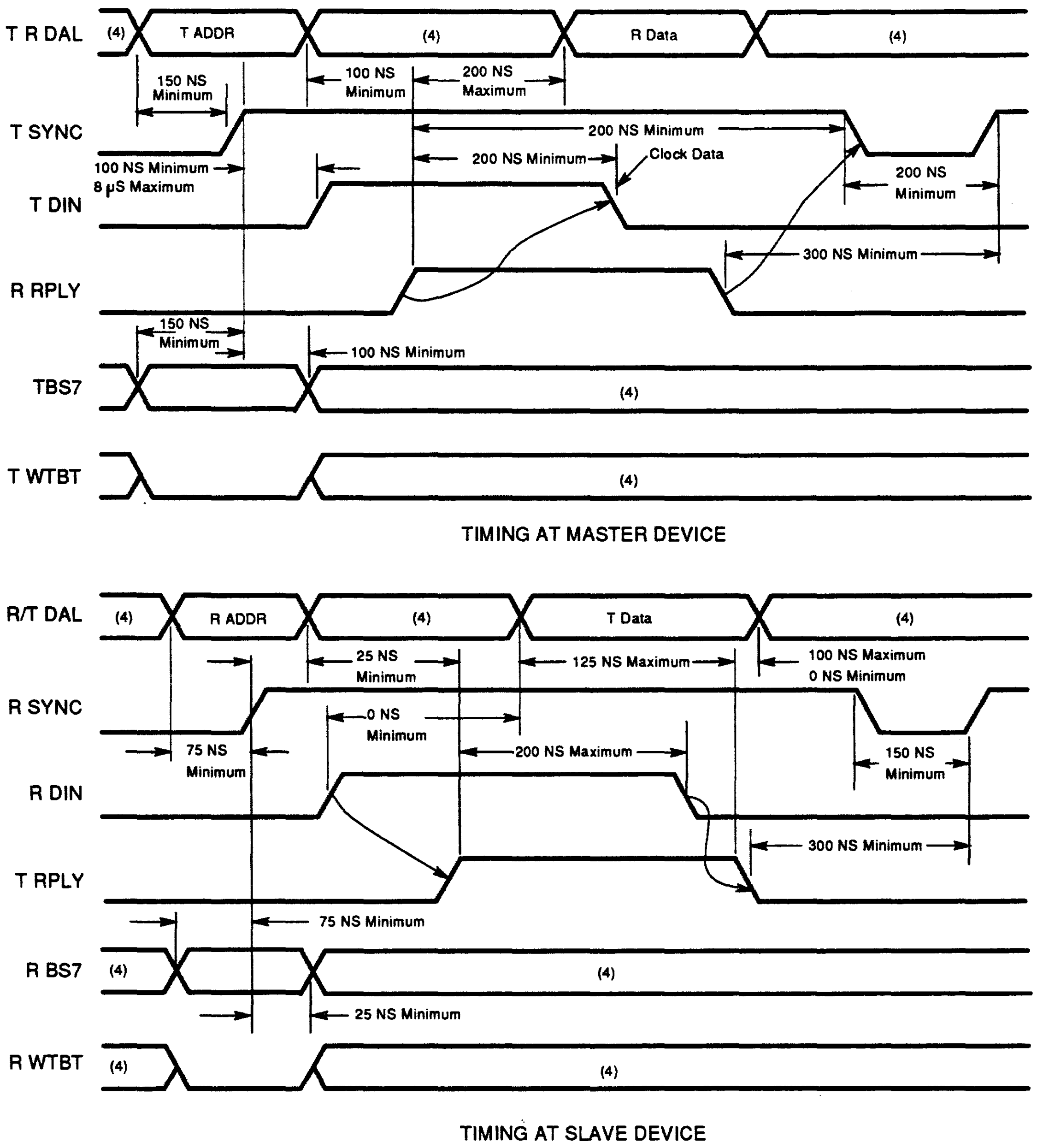
- BSYNC L must remain negated for 200 ns minimum.
- BSYNC L must not become asserted within 300 ns of previous BRPLY L negation.

Figure A–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

DATOB, shown in Figure A–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 part of a bus cycle when BWTBT L has been asserted by the bus master, or immediately following an input transfer part of a DATIOB 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.

LJ-00177-TI0

Figure A–2 DATI Bus Cycle Timing

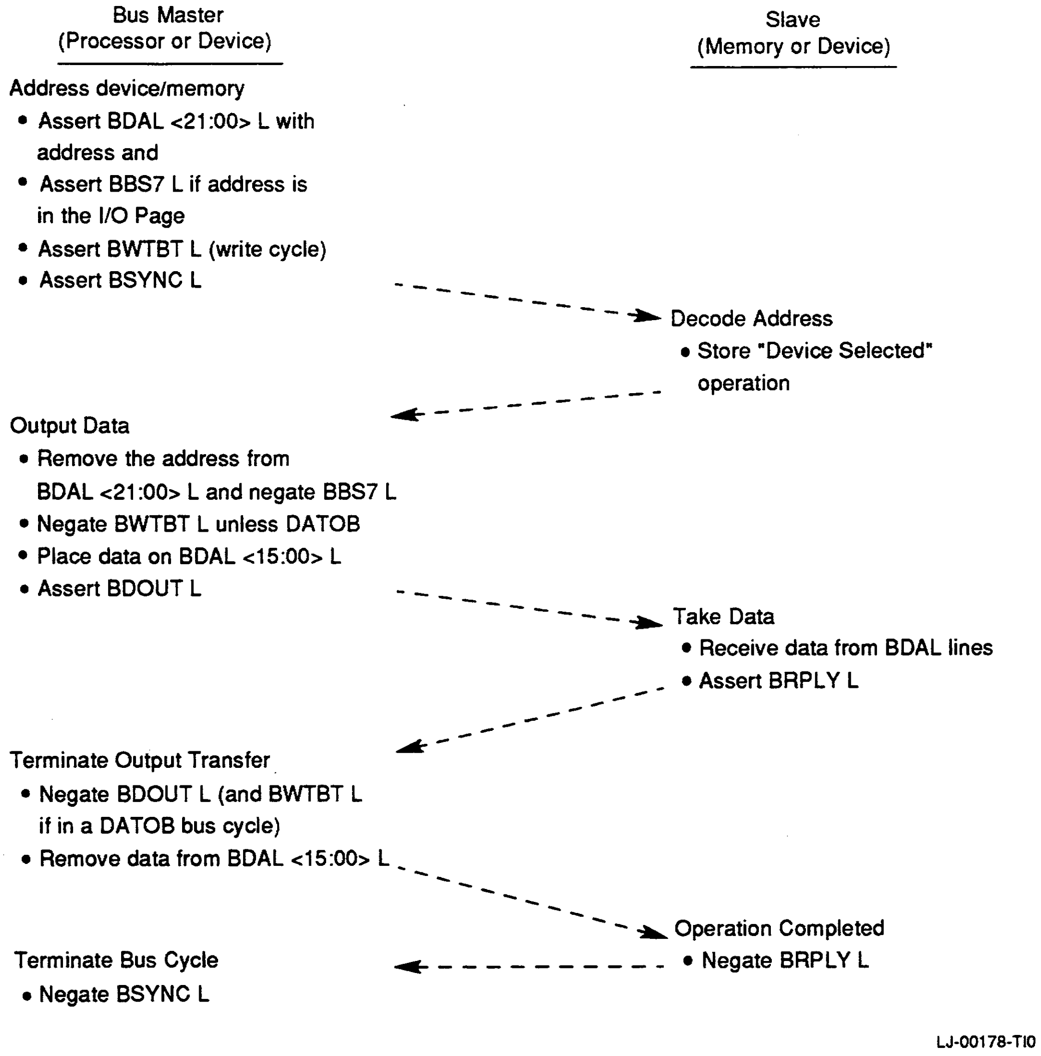

Bus Master
(Processor or Device)
Slave
(Memory or Device)
Address device/memory
• Assert BDAL <21:00> L with address and
• Assert BBS7 L if address is in the I/O Page
• Assert BWTBT L (write cycle)
• Assert BSYNC L
Decode Address
• Store "Device Selected" operation
Output Data
• Remove the address from BDAL <21:00> L and negate BBS7 L
• Negate BWTBT L unless DATOB
• Place data on BDAL <15:00> L
• Assert BDOUT L
Take Data
• Receive data from BDAL lines
• Assert BRPLY L
Terminate Output Transfer
• Negate BDOUT L (and BWTBT L if in a DATOB bus cycle)
• Remove data from BDAL <15:00> L
Operation Completed
• Negate BRPLY L
Terminate Bus Cycle
• Negate BSYNC L
LJ-00178-TI0

Figure A–3 DATO or DATOB Bus Cycle

The data transfer part of a DATOB bus cycle comprises a data setup and deskew time and a data hold and deskew time.

During the data setup and deskew time, thé 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.

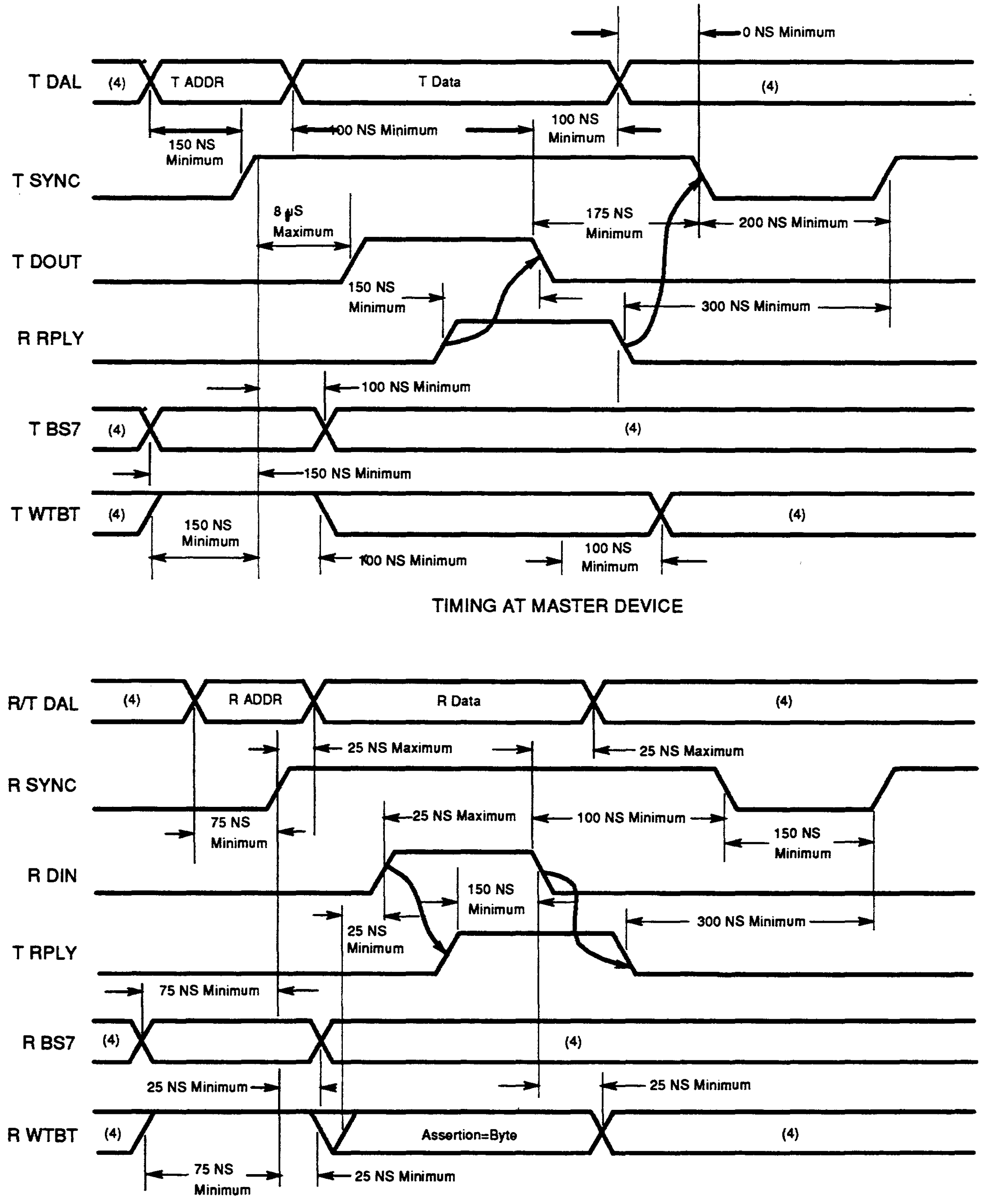
During a byte transfer, BDAL<00> L selects the high or low byte. This occurs in the addressing part of the cycle. If asserted, the high byte (BDAL<15:08> L) is selected; otherwise, the low byte (BDAL<07:00> L) is selected. An asserted BDAL 16 L at this time forces a parity error to be written into memory 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 DATOB bus cycle. Before the next cycle, BSYNC L must remain unasserted for at least 200 ns. Figure A–4 shows DATOB bus cycle timing.

DATIOB

The protocol for a DATIOB bus cycle is identical to the addressing and data transfer part of the DATI and DATOB bus cycles, and is shown in Figure A–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 (DATOB). 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 DATOB. Figure A–6 illustrates DATIOB bus cycle timing.



TIMING AT MASTER DEVICE

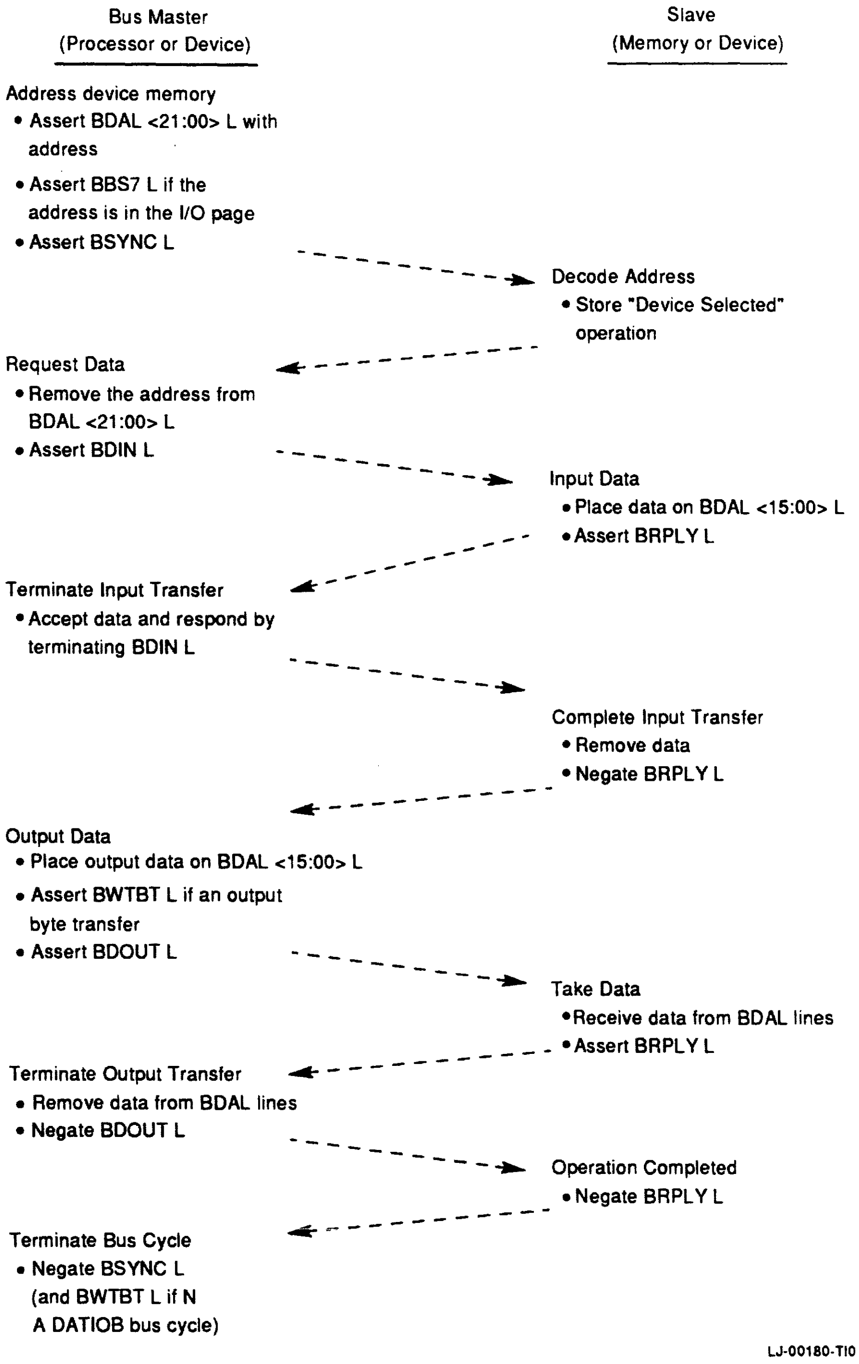
TIMING AT SLAVE DEVICE

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.

LJ-00179-TI0

Figure A–4 DATO or DATOB Bus Cycle Timing



Bus Master
(Processor or Device)

Slave
(Memory or Device)

Address device memory
- Assert BDAL <21:00> L with address
- Assert BBS7 L if the address is in the I/O page
- Assert BSYNC L

Decode Address
- Store "Device Selected" operation

Request Data
- Remove the address from BDAL <21:00> L
- Assert BDIN L

Input Data
- Place data on BDAL <15:00> L
- Assert BRPLY L

Terminate Input Transfer
- Accept data and respond by terminating BDIN L

Complete Input Transfer
- Remove data
- Negate BRPLY L

Output Data
- Place output data on BDAL <15:00> L
- Assert BWTBT L if an output byte transfer
- Assert BDOUT L

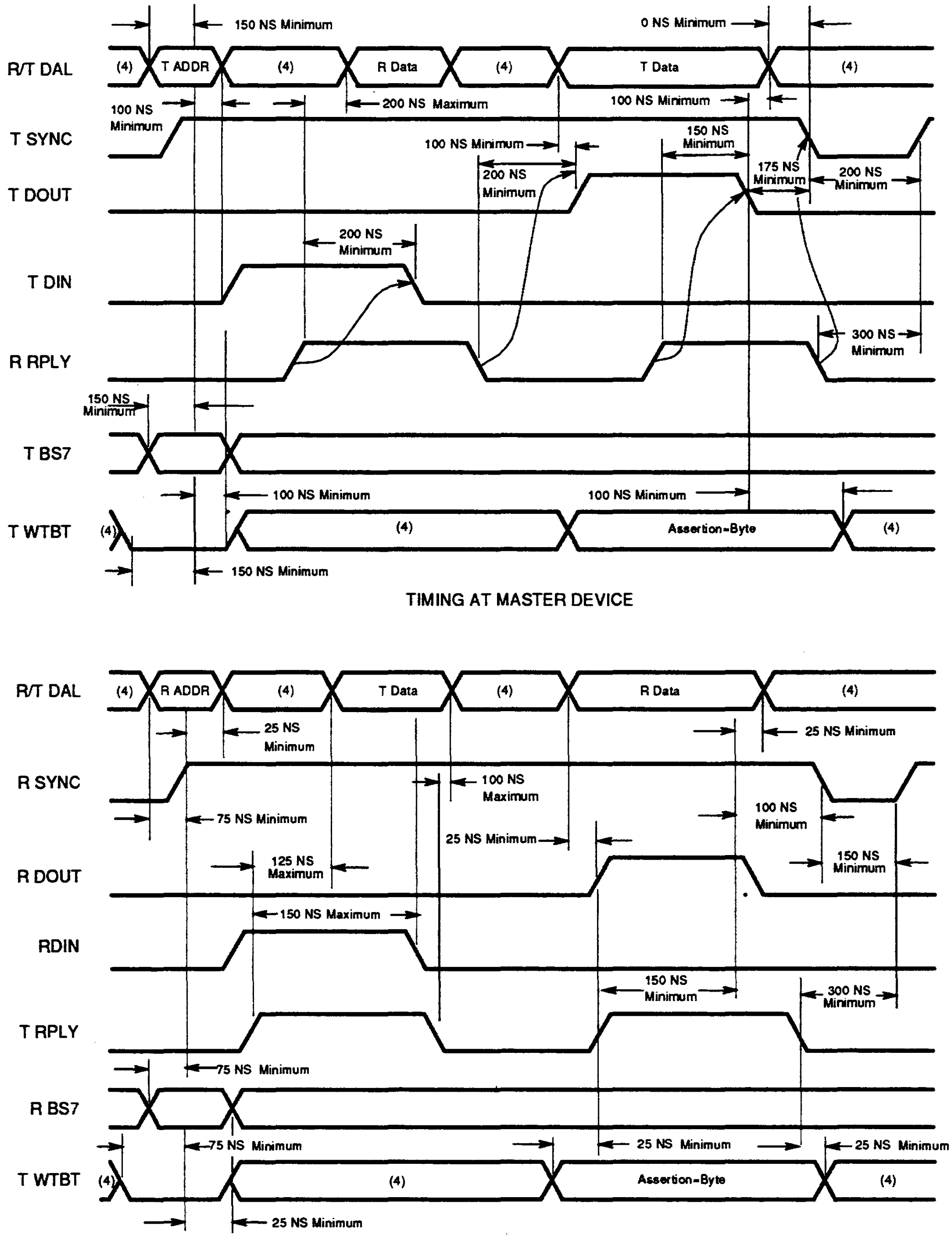
Take Data
- Receive data from BDAL lines
- Assert BRPLY L

Terminate Output Transfer
- Remove data from BDAL lines
- Negate BDOUT L

Operation Completed
- Negate BRPLY L

Terminate Bus Cycle
- Negate BSYNC L (and BWTBT L if N A DATIOB bus cycle)

Figure A–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.

LJ-00309-TI0

Figure A–6 DATIO or DATIOB Bus Cycle Timing

A.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 be supplied with 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

A.4.1 DMA Protocol

A DMA transaction can be divided into the following 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 is 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 A–7 shows the DMA protocol, and Figure A–8 shows DMA request/grant timing.

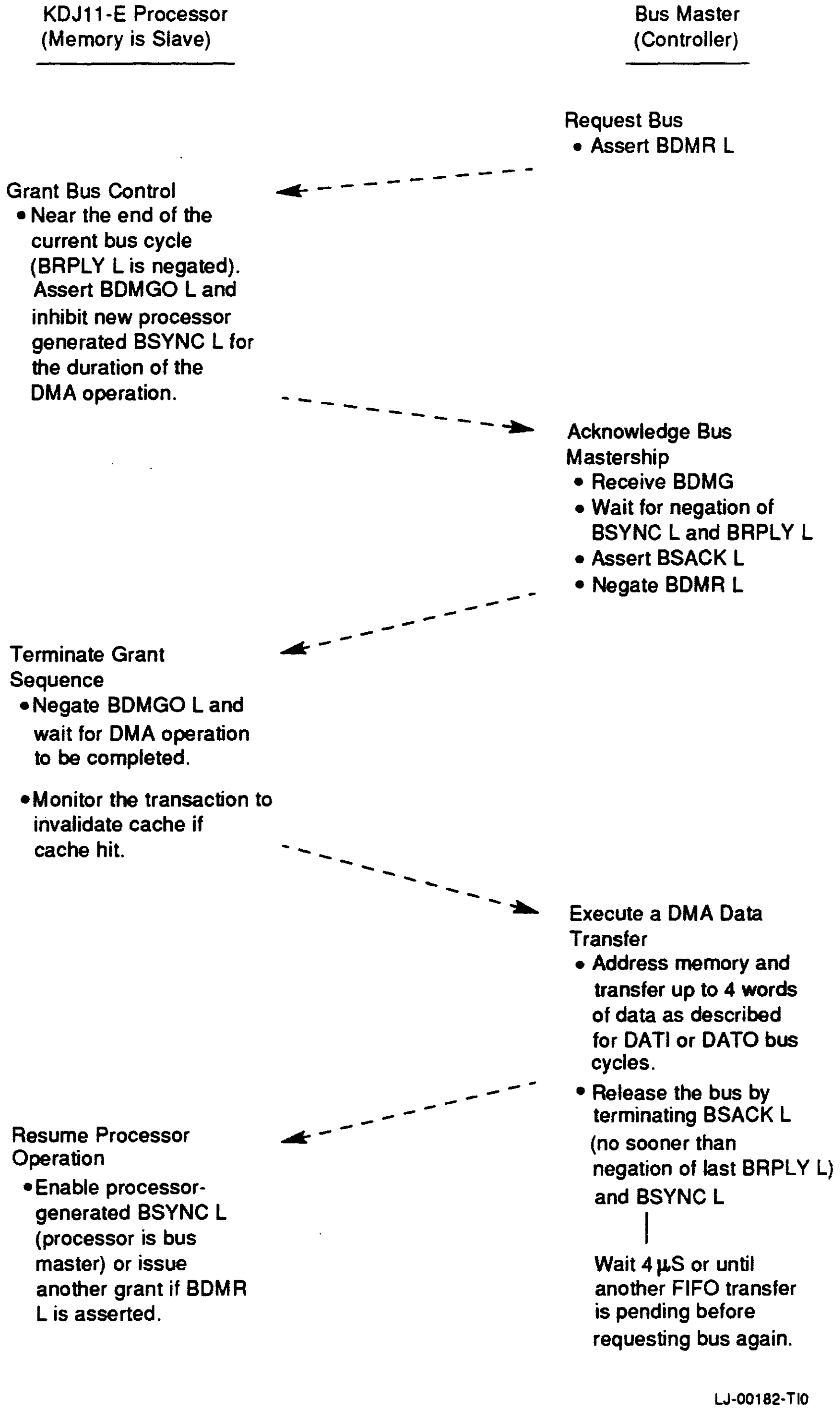
A.4.2 Block Mode DMA

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 A.4.2.1 and illustrated in Figure A–9.

The DATBO bus cycle is described in Section A.4.2.2 and illustrated in Figure A–10.



KDJ11-E Processor
(Memory is Slave)

Bus Master
(Controller)

Request Bus
- Assert BDMR L

Grant Bus Control
- Near the end of the current bus cycle (BRPLY L is negated). Assert BDMGO L and inhibit new processor generated BSYNC L for the duration of the DMA operation.

Acknowledge Bus Mastership
- Receive BDMG
- Wait for negation of BSYNC L and BRPLY L
- Assert BSACK L
- Negate BDMR L

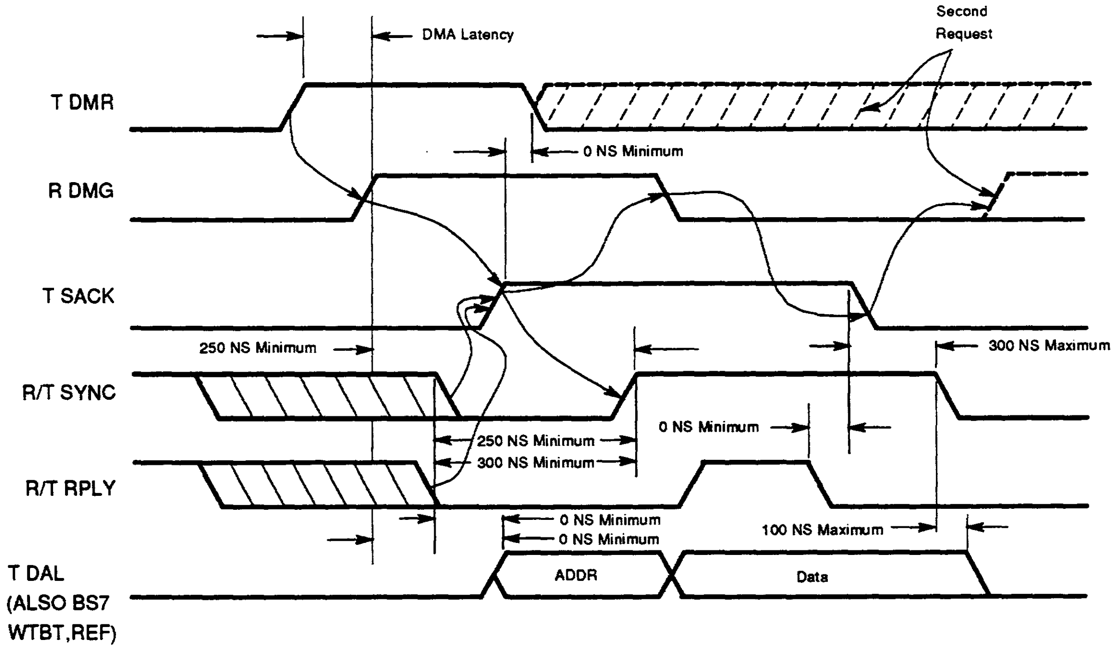
Terminate Grant Sequence
- Negate BDMGO L and wait for DMA operation to be completed.
- Monitor the transaction to invalidate cache if cache hit.

Execute a DMA Data Transfer
- Address memory and transfer up to 4 words of data as described for DATI or DATO bus cycles.
- Release the bus by terminating BSACK L (no sooner than negation of last BRPLY L) and BSYNC L

Wait 4 μS or until another FIFO transfer is pending before requesting bus again.

Resume Processor Operation
- Enable processor-generated BSYNC L (processor is bus master) or issue another grant if BDMR L is asserted.

Figure A–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.

LJ-00183-TI0

Figure A–8 DMA Request/Grant Timing

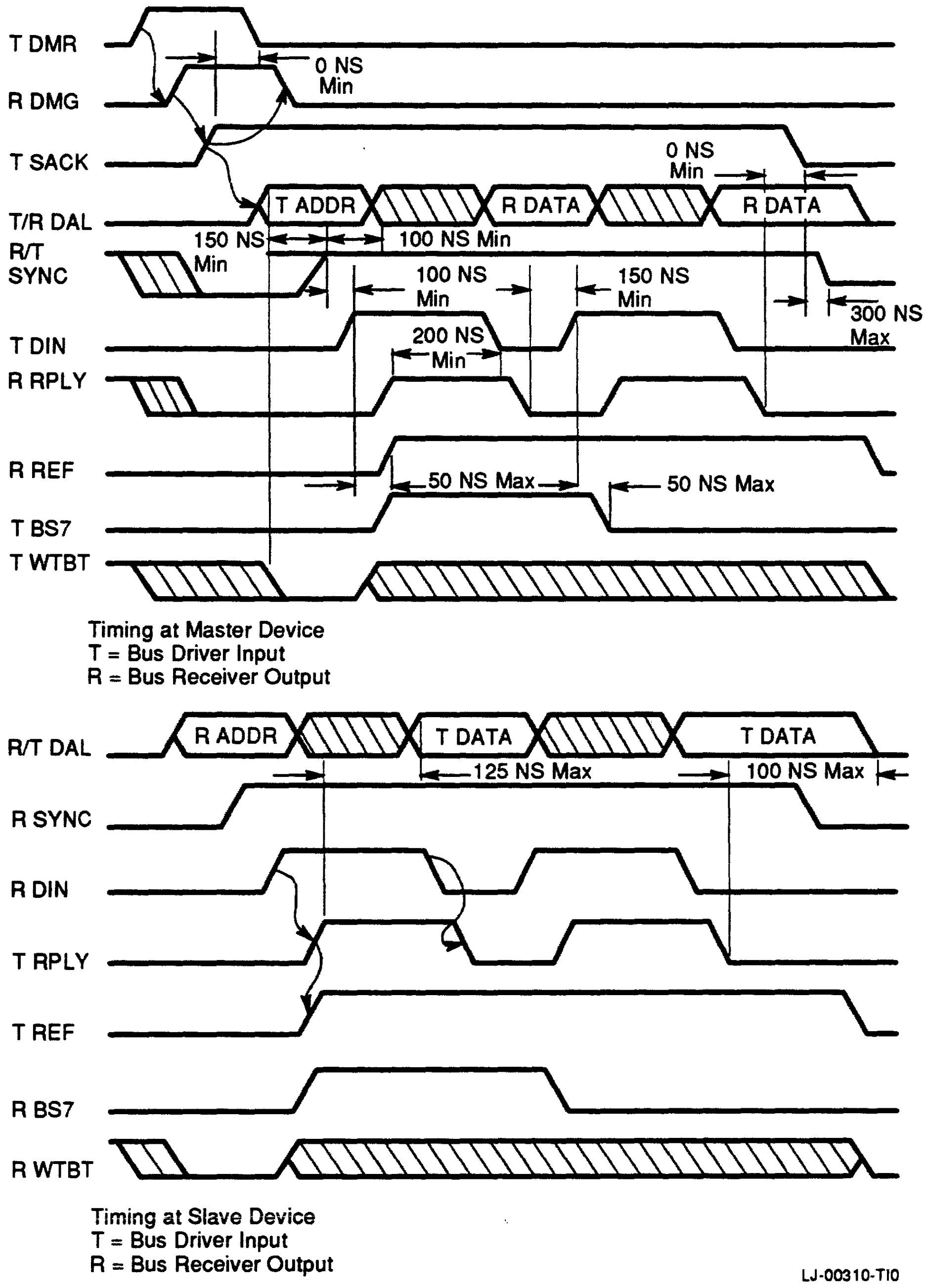


Figure A–9 DATBI Bus Cycle Timing

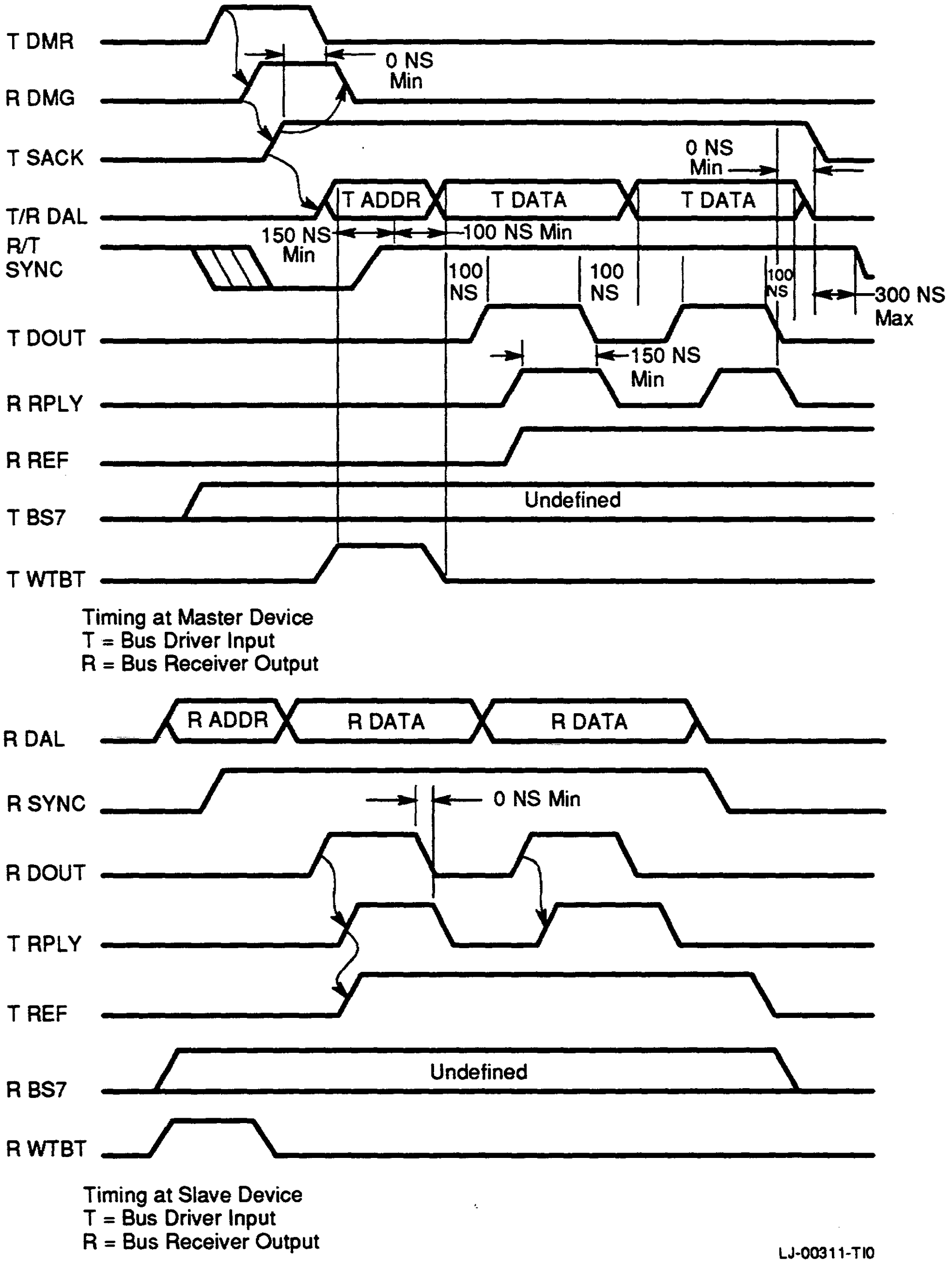


Figure A–10 DATBO Bus Cycle Timing

A.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 the timing relationship in 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.

A.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 8 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.

A.4.3 DMA Guidelines

The following is a list of 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.

A.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:

Signal	Definition
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

A.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.

A.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 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. The following list gives the interrupt levels and the corresponding Q22-bus interrupt request lines. For an explanation, refer to Section A.5.3.

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 A–11 shows the interrupt request/acknowledge sequence.

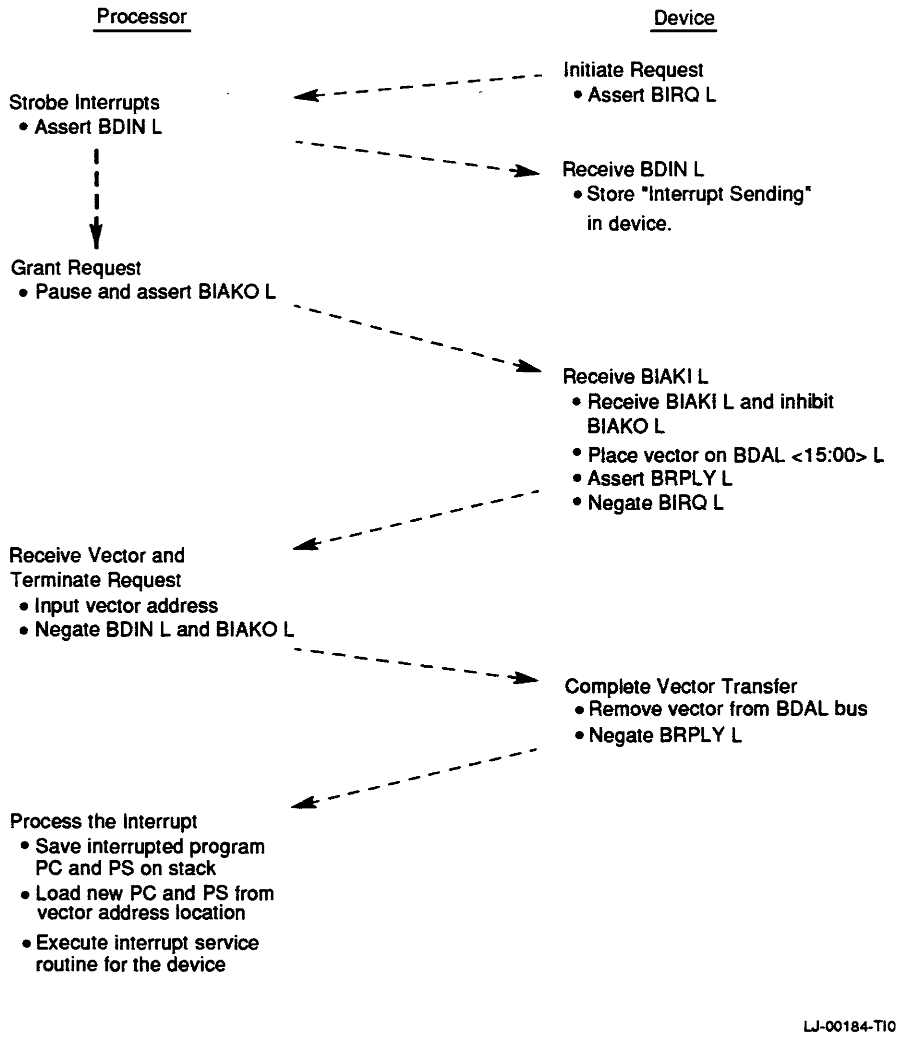

Processor
Device
Initiate Request
• Assert BIRQ L
Strobe Interrupts
• Assert BDIN L
Receive BDIN L
• Store "Interrupt Sending" in device.
Grant Request
• Pause and assert BIAKO L
Receive BIAKI L
• Receive BIAKI L and inhibit BIAKO L
• Place vector on BDAL <15:00> L
• Assert BRPLY L
• Negate BIRQ L
Receive Vector and Terminate Request
• Input vector address
• Negate BDIN L and BIAKO L
Complete Vector Transfer
• Remove vector from BDAL bus
• Negate BRPLY L
Process the Interrupt
• Save interrupted program PC and PS on stack
• Load new PC and PS from vector address location
• Execute interrupt service routine for the device
LJ-00184-TI0

Figure A–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 four-level interrupt scheme, it reacts as follows:

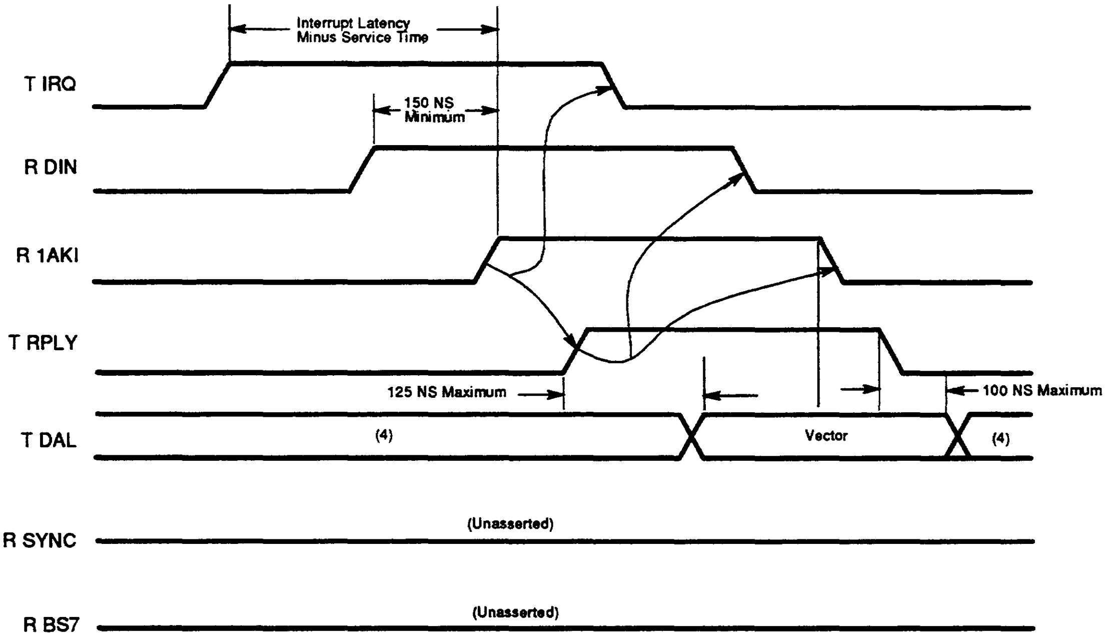
- If not requesting an interrupt, the device asserts BIAKO L and the acknowledge propagates to the next device on the bus.
- If the device is requesting an interrupt, it must check that no higher-level device is currently requesting an interrupt. This is done by monitoring higher-level request lines. The following table lists the lines that need to be monitored by devices at each priority level:

Device Priority Level	Line(s) Monitored
4	BIRQ5, BIRQ6
5	BIRQ6
6	BIRQ7
7	–

  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.

- 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 four-level interrupts (Figure A–12).



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.

LJ-00185-TI0

Figure A–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.

NOTE
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.

A.5.3 Q22-bus Four-Level Interrupt Configurations

If you have high-speed peripherals and desire better software performance, you can use the four-level interrupt scheme. Both position-independent and position-dependent configurations can be used with the four-level interrupt scheme.

Figure A–13 shows the position-independent configuration. This allows peripheral devices that use the four-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.

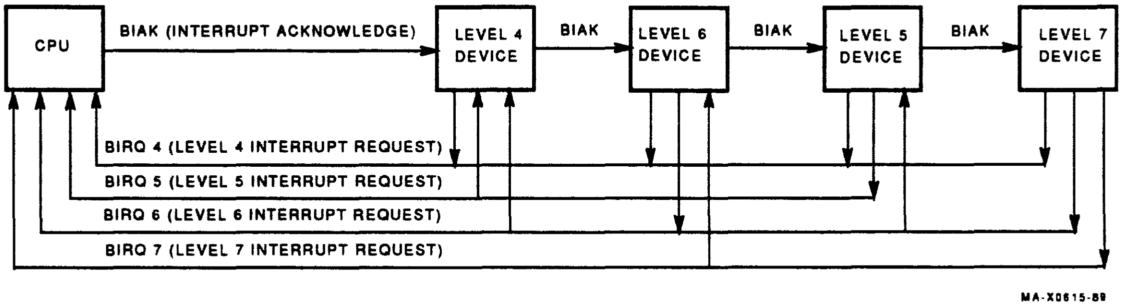

CPU
BIAK (INTERRUPT ACKNOWLEDGE)
LEVEL 4 DEVICE
BIAK
LEVEL 6 DEVICE
BIAK
LEVEL 5 DEVICE
BIAK
LEVEL 7 DEVICE
BIRQ 4 (LEVEL 4 INTERRUPT REQUEST)
BIRQ 5 (LEVEL 5 INTERRUPT REQUEST)
BIRQ 6 (LEVEL 6 INTERRUPT REQUEST)
BIRQ 7 (LEVEL 7 INTERRUPT REQUEST)
MA-X0615-89

Figure A–13 Position-Independent Configuration

Figure A–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.

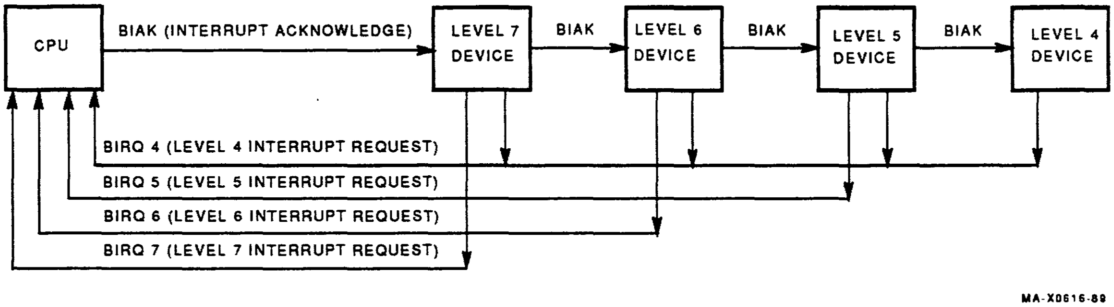

CPU
BIAK (INTERRUPT ACKNOWLEDGE)
LEVEL 7 DEVICE
BIAK
LEVEL 6 DEVICE
BIAK
LEVEL 5 DEVICE
BIAK
LEVEL 4 DEVICE
BIRQ 4 (LEVEL 4 INTERRUPT REQUEST)
BIRQ 5 (LEVEL 5 INTERRUPT REQUEST)
BIRQ 6 (LEVEL 6 INTERRUPT REQUEST)
BIRQ 7 (LEVEL 7 INTERRUPT REQUEST)
MA-X0616-89

Figure A–14 Position-Dependent Configuration

A.6 Control Functions

The following Q22-bus signals provide control functions:

Signal	Definition
BREF L	Memory refresh (also block mode DMA)
BHALT L	Processor halt
BINIT L	Initialize
BPOK H	Power OK
BDCOK H	DC power OK

A.6.1 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.

A.6.2 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.

A.6.3 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).

A.7 Q22-bus Electrical Characteristics

The input and output logic levels for Q22-bus signals are given in Section A.7.1.

A.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	
TTL logical high	0.4 Vdc maximum
	2.4 Vdc minimum

A.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.

A.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 ft).

A.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.

A.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.

A.7.6 Bus Termination

The 120-ohm Q22-bus must be terminated at each end by an appropriate terminator, as shown in Figure A–15. 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.

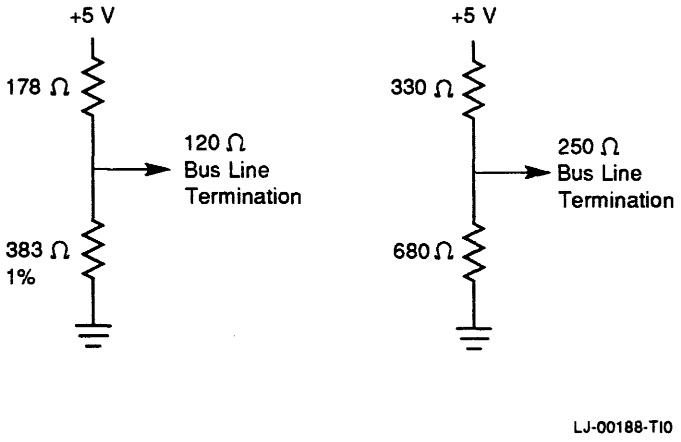


Figure A–15 Bus Line Terminations

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:

Bus Termination Characteristic	Value
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.

A.7.7 Bus Interconnecting Wiring

The following sections give specific information about bus interconnecting wiring.

A.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 and connector-module etch) 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.

A.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.

A.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 bused between backplanes on any interconnecting bus cables.

A.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 A–16). 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 in.) long.

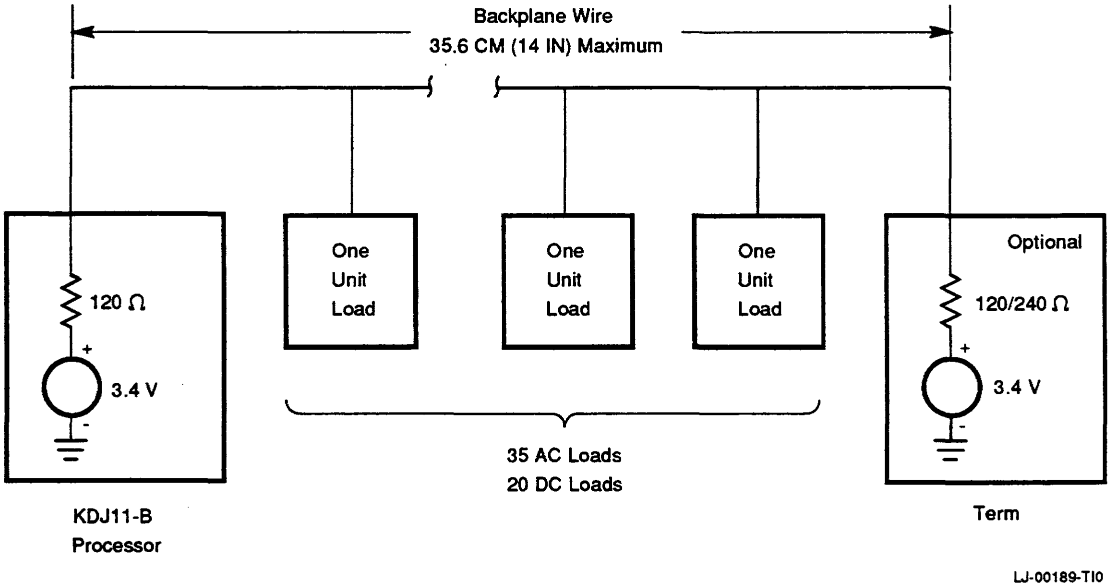
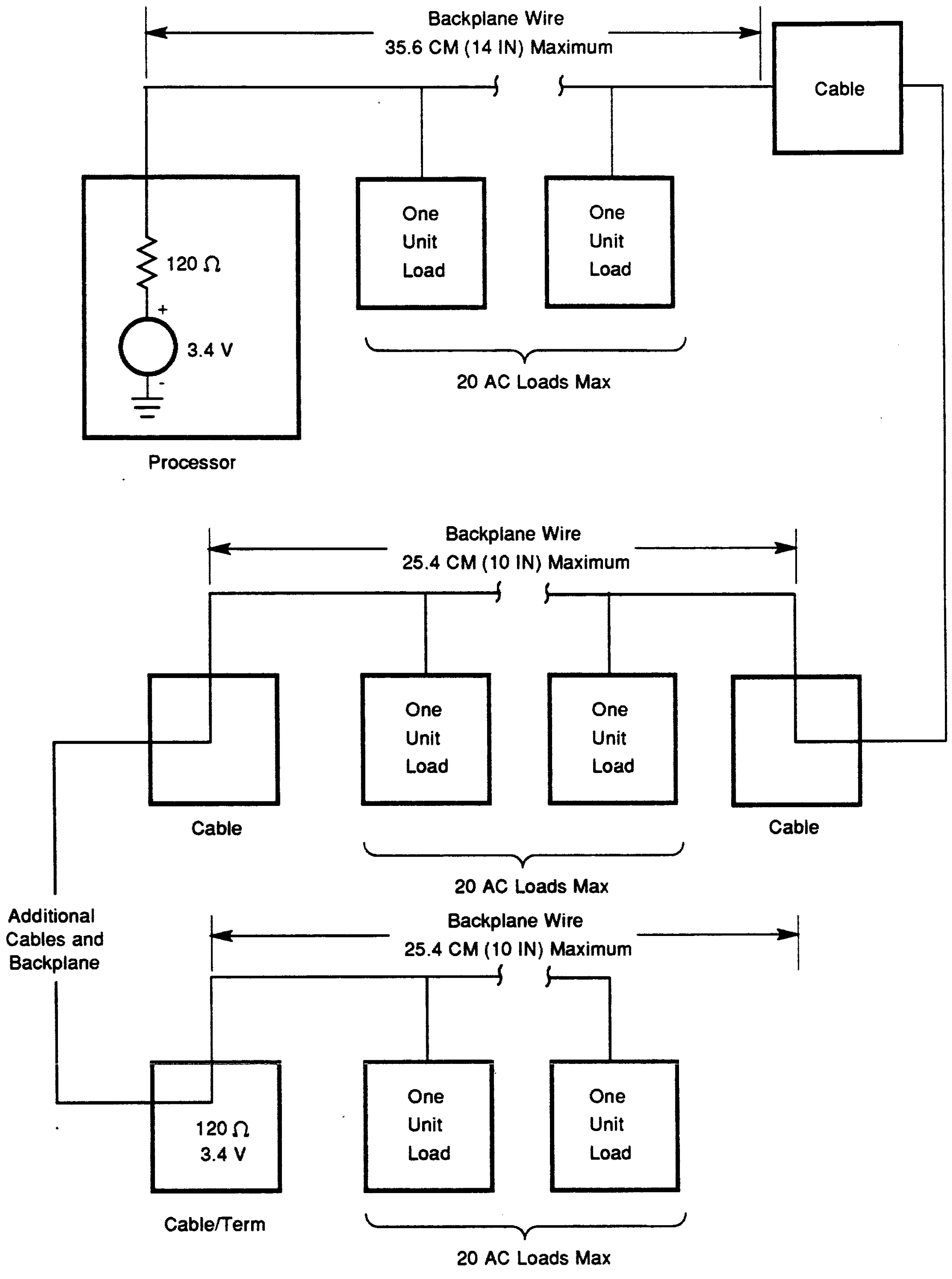


Figure A–16 Single-Backplane Configuration

Rules for configuring multiple backplane systems are as follows:

- Figure A–17 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 in.) 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 – 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 ft) or more in length.
- The cable(s) connecting the second backplane to the third backplane is 122 cm (4 ft) 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 ft).
- The cables used must have a characteristic impedance of 120 ohms.



Notes:

1. Two cables (max) 4.88 M (16 FT) (Max) total length.
2. 20 DC loads total (max).

Figure A–17 Multiple Backplane Configuration

A.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 through 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.

A.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 A–18.

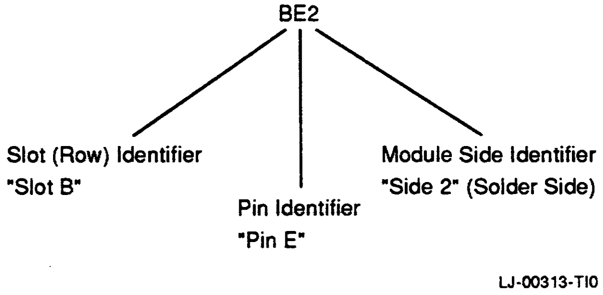


Figure A–18 Typical Pin Identification System

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 A–19.

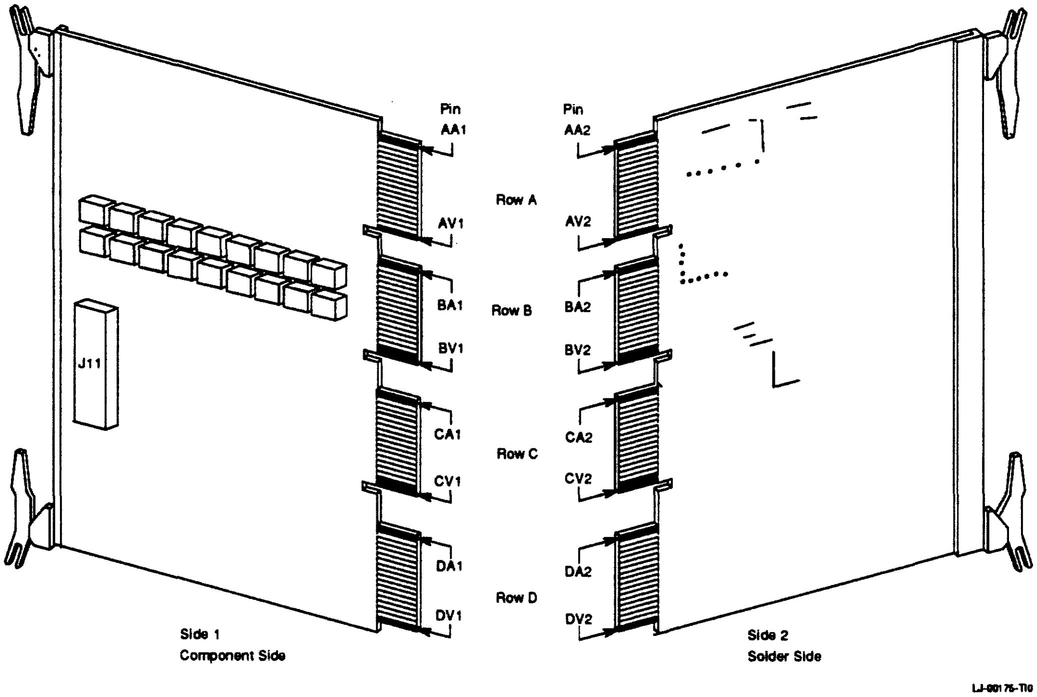


Figure A–19 Quad-Height Module Contact Finger Identification

Letters ranging from A through V (excluding G, I, O, and Q) identify a particular pin on a side of a slot. Table A–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 A–20.

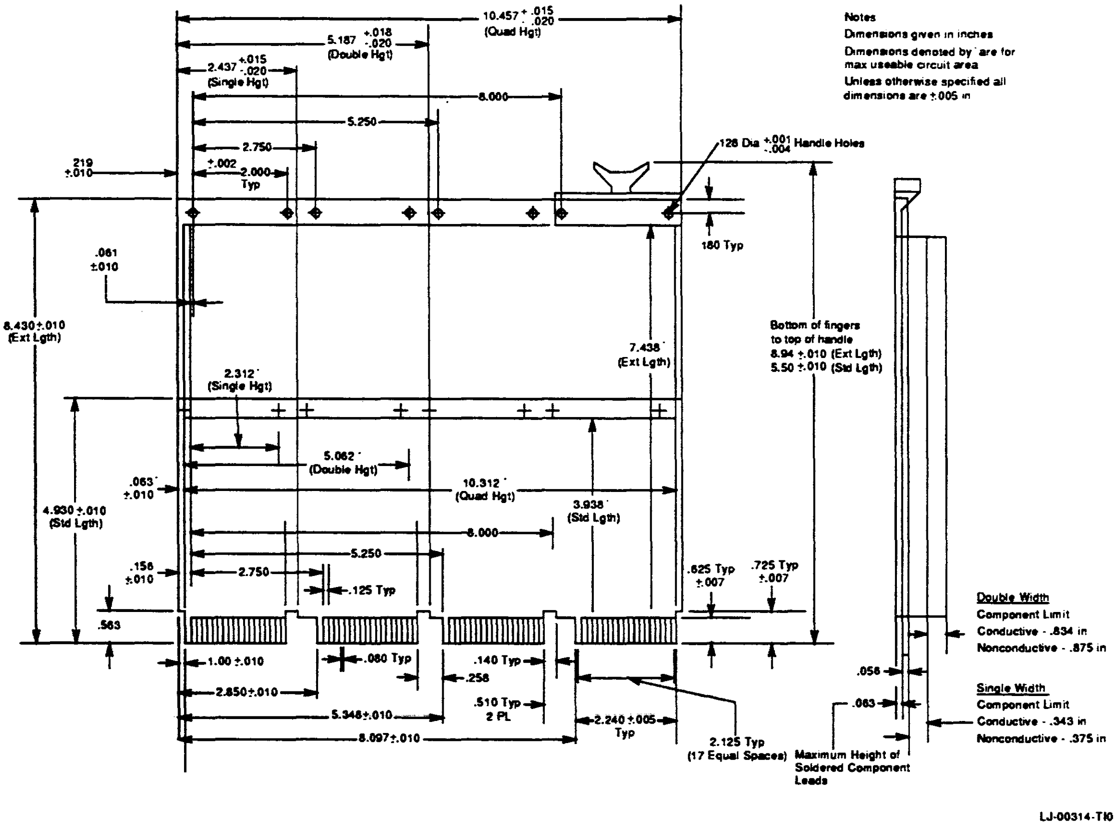


Figure A–20 Typical Q22-bus Module Dimensions

Table A–7 Bus Pin Identifiers

Bus Pin	Signal	Definition
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 bused 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.

Table A–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
AF1	SSPARE2	Special spare — not assigned or bused 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 bused 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 bused connection).
AL1	MSPAREB	Maintenance spare — normally connected together on the backplane at each option location (not a bused connection).
AM1	GND	Ground — system signal ground and dc return.
AN1	BDMR L	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 bused 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.
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.

Table A–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
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. Bused 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 bused address lines are address lines <21:18>. Currently not used during data time.
BF1	SSPARE7 BDAL21 L	In the Q22-bus, these bused address lines are address lines <21:18>. Currently not used during data time.
BH1	SSPARE8	Special spare — not assigned or bused in Digital's cable and backplane assemblies. Available for user interconnection.
BJ1	GND	Ground — system signal ground and dc return.
BK1 BL1	MSPAREB MSPAREB	Maintenance spare — normally connected together on the backplane at each option location (not a bused 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 through 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 bused 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 and 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.

Table A–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
AB2	-12	-12 V power — -12 Vdc power for (optional) devices requiring this voltage. 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.
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.
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.

Table A–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
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 BIRQ**n** 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.
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 A–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
BE2	BDAL2 L	Data/address lines — these 14 lines are part of the 16-line data/address bus.
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	

B
Specifications

B.1 Dimensions

The KA660-AA and MS650-BA are quad height modules with the following dimensions:

Module	Height (Inches)	Length (Inches)	Width (Inches) *
KA660	10.457 +0.015/ -0.020	8.430 +0.010/ -0.010	0.7 maximum
MS650	10.457 +0.015/ -0.020	8.430 +0.010/ -0.010	0.375 maximum

*Width, as defined for Digital Equipment modules, is the height of components above the surface of the module.

B.2 KA660-AA Connectors

The KA660-AA has five connector interfaces: two fingers that plug into rows A and B of the backplane (the A/B row fingers), two fingers that plug into rows C and D of the backplane (the C/D row fingers), a 50-pin connector that connects the KA660-AA and MS650-BA modules (the KA660-AA memory connector), a 50-pin connector that connects the KA660-AA to the DSSI bus, and a 40-pin connector that connects the KA640-AA with the H3602 CPU cover panel.

B.2.1 KA660-AA A/B Row Fingers

The KA660-AA A/B row fingers are compatible with the Q22-bus specification. The SRUN(L) signal appears on pin AF1.

B.2.2 KA660-AA C/D Row

The pinout of the KA660-AA C/D row fingers is included in the following section.

B.2.3 KA660-AA Configuration and Display Connector (J1)

The configuration and display connector is a 40-pin connector that connects the KA660 module to the H3602 console module. The connector features the following pinouts:

Table B–1 H3602 Connector Pinout

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	+12v	Fused +12 Vdc
18	GND	Ground
19	DTR H	Data Terminal Ready
20	GND	Ground
21	TXD L	Transmit Data
22	GND	Ground
23	GND	Ground
24	RXD L	Receive Data
25	RXD H	Receive Data
26	GND	Ground
27	+5v	Fused +5 Vdc
28 29 30	CONBITRATE2 L CONBITRATE1 L CONBITRATE0 L	Console bit rate <02:00>. These three bits determine the console baud rate. They are configured using the select switch on the inside of the H3602 console panel.

Table B–1 (Cont.) H3602 Connector Pinout

Pin	Mnemonic	Meaning
31	LED CODE0 L	LED code register bits <03:00>. When asserted, each of these four output signals turns on a corresponding LED on the module. LED CODE <03:00> are asserted (low) by power-up and by the negation of DCOK when the processor is halted. They are updated by the boot and diagnostic programs, through the boot and diagnostic register.
32	LED CODE1 L	
33	LED CODE2 L	
34	LED CODE3 L	
35	ENB HALT L	Halt Enable. This input signal controls the response to the halt conditions. If ENB HALT is asserted (low), then the KA660 halts and enters the console program if: 1. The program executes a Halt instruction in kernel mode 2. The console detects a break character. 3. The Q22-bus Halt line is asserted (but only if the KA660 is configured as an arbiter CPU). 4. The interprocessor communication register AUX HLT bit is set (but only if the KA660 is configured as an auxiliary CPU). If ENB HALT is negated, then the Halt line and break character are ignored and the ROM program responds to a halt instruction by restarting or rebooting the system. ENB HALT can be read by software, through the boot and diagnostic register. In the MicroVAX 3000 system, ENB HALT originates from a switch on the H3602 or cover panel.
36	BDCODE1 L	This 2-bit code can be read by software through the boot and diagnostic register. The KA660 ROM program may use boot and diagnostic code <01:00> to select various boot device or diagnostic test parameters at power-up and at system restart.
37	BDCODE0 L	
38	VBAT H	Battery backup voltage for the TOY clock.
39	GND	Ground
40	GND	Ground

NOTE
The KA660 module provides 10K pull-up resistors for the eight input signals.

B.2.4 KA660 DSSI and Private Memory Interconnect Connectors

The pinouts of the KA660 100-pin DSSI and PMI connector are as follows:

Table B–2 DSSI and PMI Connector Pinout

Pin Number	Signal Name
01	GND H

Table B–2 (Cont.) DSSI and PMI Connector Pinout

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

Table B–2 (Cont.) DSSI and PMI Connector Pinout

Pin Number	Signal Name
50	MD30 H
51	DSSIDATA0L
52	GND
53	DSSIDATA1L
54	GND
55	DSSIDATA2L
56	GND
57	DSSIDATA3L
58	GND
59	DSSIDATA4L
60	GND
61	DSSIDATA5L
62	GND
63	DSSIDATA6L
64	GND
65	DSSIDATA7L
66	GND
67	DSSIPARITYL
68	GND
69	DSSI19L
70	GND
71	DSSI21L
72	GND
73	VTERM
74	VTERM
75	VTERM
76	VTERM
77	VTERM
78	VTERM
79	GND
80	DSSI30L
81	GND
82	DSSI32L
83	GND
84	DSSI34L
85	GND
86	DSSIBSYL
87	GND
88	DSSIACKL
89	GND
90	DSSIRSTL
91	GND
92	DSSI42L
93	GND
94	DSSISELL
95	GND
96	DSSICDL
97	GND

Table B–2 (Cont.) DSSI and PMI Connector Pinout

Pin Number	Signal Name
98	DSSIREQL
99	GND
100	DSSIIOL

B.3 DC Power Consumption

The KA660 CPU module power requirements are as follows:

+5V±5%
+12V±5%

Typical currents are 10% less than the specified maximum.

B.4 Bus Loads

The KA660 CPU bus loads are as follows:

Module	DC	AC
KA660	1.4 loads	4.75 loads

B.5 Battery Backup Specifications

When DC power is supplied to the KA660 module, it charges the external batteries from +5 volts through a 240 ohm resistor.

When DC power is removed from the KA660 module, it drains the external batteries at a rate of 1.0 milliamps/hour. Because the batteries are rated at 180 mA/hr the battery backup lasts 180 hours.

NOTE
These batteries supply power to the KA660 time of year clock and SSC RAM only. There are no battery backup hooks for the memory system.

B.6 Operating Conditions

The KA660-AA module meets or exceeds Digital's requirements for operation in an add-on environment.

Temperature

Temperature should be +5 to +60° C (-40 to +140° F) with a change rate no greater than 20 +-2° C (36 +-4° F)/hour at sea level. The maximum temperature must be derated by 1.8° C/1000 meters (1 degree F/1000 feet) above sea level.

Humidity

Humidity should be 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

Altitude can be 2,400 meters (8,000 feet) with a change rate no greater than 300 meters /minute (1000 feet/minute).

Airflow

The airflow required to meet these specifications is 200 lfm.

B.7 Nonoperating Conditions (Less than 60 days)

Temperature

Tempature should be -40 to +66° C (-40 to +151° F) with a change rate no greater than 11 +-2° C (20 +-4° F)/ hour at sea level. The maximum temperature must be derated by 1.8° C/1000 meters (1 degree F/1000 feet) above sea level.

Humidity

Up to 95% noncondensing.

Altitude

Altitude can be as many as 4,900 meters (16,000 feet) with a change rate no greater than 600 meters/minute (2000 feet/minute).

B.8 Nonoperating Conditions (Greater than 60 days)

Temperature

Tempature should be +5 to +60° C (-40 to +140° F) with a change rate no greater than 20 +-2° C (36 +-4° F)/hour at sea level. The maximum temperature must be derated by 1.8° C/1000 meters (1 degree F/1000 feet) above sea level.

Humidity

Humidity can be 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

Altitude can be as many as 2,400 meters (8,000 feet) with a rate of change no greater than 300 meters/minute (1000 feet/minute).

Altitude can be 2,400 meters (8,000 feet) with a change rate no greater than 300 meters /minute (1000 feet/minute).

Airflow

The airflow required to meet these specifications is 200 lfm.

B.7 Nonoperating Conditions (Less than 60 days)

Temperature

Tempature should be -40 to +66° C (-40 to +151° F) with a change rate no greater than 11 +-2° C (20 +-4° F)/ hour at sea level. The maximum temperature must be derated by 1.8° C/1000 meters (1 degree F/1000 feet) above sea level.

Humidity

Up to 95% noncondensing.

Altitude

Altitude can be as many as 4,900 meters (16,000 feet) with a change rate no greater than 600 meters/minute (2000 feet/minute).

B.8 Nonoperating Conditions (Greater than 60 days)

Temperature

Tempature should be +5 to +60° C (-40 to +140° F) with a change rate no greater than 20 +-2° C (36 +-4° F)/hour at sea level. The maximum temperature must be derated by 1.8° C/1000 meters (1 degree F/1000 feet) above sea level.

Humidity

Humidity can be 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

Altitude can be as many as 2,400 meters (8,000 feet) with a rate of change no greater than 300 meters/minute (1000 feet/minute).

C Address Assignments

C.1 KA660 Physical Address Space

Table C–1 General Local Address Space Map

Address Range	Description
VAX Memory Space	
0000 0000—1FFF FFFF	Local Memory Space (512 Mbytes)
VAX I/O Space	
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 — 2007 FFFF	Local ROM Space
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)
2008 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)
3800 0000 — 3BFF FFFF	Reserved Local I/O Space (64 Mbytes)
3C00 0000 — 3FFF FFFF	Reserved Local I/O Space (64 Mbytes)

C.2 KA660 Detailed Physical Address Map

Table C–2 Detailed Local Address Space Map

Description	Address Range
VAX Memory Space	
Local Memory Space, 64 Mbytes (Q22-bus Map at top 32 Kbytes of Main Memory)	0000 0000—03FF FFFF
Reserved Memory Space (448 Mbytes)	0400 0000—1FFF FFFF
VAX I/O Space	
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 Comm Reg	2000 1F40
Reserved Q22-bus I/O Space	2000 1F44—2000 1FFF
Local Register I/O Space	**2000 2000—2003 FFFF**
Reserved Local Register I/O Space	2000 4202—2000 422F
SHAC SSWCR	2000 4230
Reserved Local Register I/O Space	2000 4234—2000 4043
SHAC SSHMA	2000 4244
SHAC PQBBR	2000 4248
SHAC PSR	2000 424C
SHAC PESR	2000 4250
SHAC PFAR	2000 4254
SHAC PPR	2000 4258
SHAC PMCSR	2000 425C
Reserved Local Register I/O Space	2000 4260—2000 427F
SHAC PCQ0CR	2000 4280
SHAC PCQ1CR	2000 4284
SHAC PCQ2CR	2000 4288
SHAC PCQ3CR	2000 428C
SHAC PDFQCR	2000 4290
SHAC PMFQCR	2000 4294
SHAC PSRCR	2000 4298

Table C–2 (Cont.) Detailed Local Address Space Map

Description	Address Range
Local Register I/O Space	**2000 2000—2003 FFFF**
SHAC PECR	2000 429C
SHAC PDCR	2000 42A0
SHAC PICR	2000 42A4
SHAC PMTCR	2000 42A8
SHAC PMTECR	2000 42AC
Reserved Local Register I/O Space	2000 42B0—2000 7FFF
NICSR0—Vector Add, IPL, Sync/Async	2000 8000
NICSR1—Polling Demand Register	2000 8004
NICSR2—Reserved	2000 8008
NICSR3—Receiver List Address	2000 800C
NICSR4—Transmitter List Address	2000 8010
NICSR5—Status Register	2000 8014
NICSR6—Command and Mode Register	2000 8018
NICSR7—System Base Address	2000 801C
NICSR8—Reserved	2000 8020*
NICSR9—Watchdog Timers	2000 8024*
NICSR10– Reserved	2000 8028*
NICSR11– Rev Num and Missed Frame Count	2000 802C*
NICSR12– Reserved	2000 8030*
NICSR13– Breakpoint Address	2000 8034*
NICSR14– Reserved	2000 8038*
NICSR15– Diagnostic Mode and Status	2000 803C
Reserved Local Register I/O Space	2000 8040—2003 FFFF
Local EPROM Space	**2004 0000—2007 FFFF**
µVAX System Type Register (In EPROM)	2004 0004
Local EPROM—(Halt Protected)	2004 0000—2007 FFFF
Local Register I/O Space	**2008 0000—201F FFFF**
Q22 System Configuration Register	2008 0000
Q22 System Error Register	2008 0004
Q22 Master Error Address Register	2008 0008
Q22 Slave Error Address Register	2008 000C
Q22-bus Map Base Register	2008 0010

*These are virtual registers. See Section 11.3.1.2 to access information.

Table C–2 (Cont.) Detailed Local Address Space Map

Description	Address Range
Local Register I/O Space	**2008 0000—201F FFFF**
Reserved Local Register I/O Space	2008 0014—2008 00FF
Main Memory Error Status Register	2008 0140
Main Memory Control/Diag Status Register	2008 0144
Reserved Local Register I/O Space	2008 0148—2008 3FFF
Boot and Diagnostic Register (32 Copies)	2008 4000—2008 407C
NI Station Address ROM	
DSSI ID Switch Settings	
Reserved Local Register I/O Space	2008 4080—2008 7FFF
Q22-bus Map Registers	2008 8000—2008 FFFF
Reserved Local Register I/O Space	2009 0000—2013 FFFF
SSC Base Address Register	2014 0000
SSC Configuration Register	2014 0010
CDAL Bus Timeout Control Register	2014 0020
Diagnostic LED Register	2014 0030
Reserved Local Register I/O Space	2014 0034—2014 006B
The following addresses allow those KA660 Internal Processor Registers that are implemented in the SSC chip (External, Internal Processor Registers) to be accessed via the local I/O page. These addresses are documented for diagnostic purposes only and should not be used by non-diagnostic programs.	
Time Of Year Register	2014 006C
Console Storage Receiver Status	2014 0070†
Console Storage Receiver Data	2014 0074†
Console Storage Transmitter Status	2014 0078†
Console Storage Transmitter Data	2014 007C†
Console Receiver Control/Status	2014 0080
Console Receiver Data Buffer	2014 0084
Console Transmitter Control/Status	2014 0088
Console Transmitter Data Buffer	2014 008C
Reserved Local Register I/O Space	2014 0090—2014 00DB
I/O Bus Reset Register	2014 00DC

†These registers are not fully implemented. Accesses yield unpredictable results.

Table C–2 (Cont.) Detailed Local Address Space Map

Description	Address Range
The following addresses allow those KA660 Internal Processor Registers that are implemented in the SSC chip (External, Internal Processor Registers) to be accessed via the local I/O page. These addresses are documented for diagnostic purposes only and should not be used by non-diagnostic programs.	
Reserved Local Register I/O Space	2014 00E0
Rom Data Register	2014 00F0‡
Bus Timeout Counter	2014 00F4‡
Interval Timer	2014 00F8‡
Reserved Local Register I/O Space	2014 00FC—2014 00FF
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 012F
BDR Address Decode Match Register	2014 0130
BDR Address Decode Mask Register	2014 0134
Reserved Local Register I/O Space	2014 0138—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

‡These registers are internal SSC registers used for SSC chip test purposes only. They should not be accessed by the CPU.

C.3 External, Internal Processor Registers

Several of the Internal Processor Registers (IPR's) on the KA660 are implemented in the SSC chip rather than the SOC CPU chip. These registers are referred to as External, Internal Processor Registers and are listed below.

Table C–3 External, Internal Processor Registers

IPR Number	Register Name	Mnemonic
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

C.4 Global Q22-bus Physical Address Space

Table C–4 Global Q22-bus Physical Address Map

Description	Address Range
Q22-bus Memory Space	
Q22-bus Memory Space (Octal)	0000 0000—1777 7777
Q22-bus I/O Space (BBS7 Asserted)	
Q22-bus I/O Space (Octal)	**1776 0000—1777 7777**
Reserved Q22-bus I/O Space	1776 0000—1776 0007
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 Comm Register	1777 7500
Reserved Q22-bus I/O Space	1777 7502—1777 7777

D

VAX Instruction Set

The information in this appendix is for reference only.

The standard notation for operand specifiers is:

name.access type data type

Name

A suggestive name for the operand in the context of the instruction. The name is the capitalized name of a register or block for implied operands.

Access type

A letter denoting the operand specifier access type.

a	Address operand
b	Branch displacement
m	Modified operand (both read and written)
r	Read-only operand
v	If not R*n*, same as a. Otherwise R[*n*+1]'R[*n*]
w	Write-only operand

Data type

A letter denoting the data type of the operand.

b	Byte
d	D_floating
f	F_floating
g	G_floating
l	Longword
q	Quadword
v	Field (used only in implied operands)
w	Word
*	Multiple longwords (used only in implied operands)

Implied operands

Locations that are accessed by the instruction, but not specified in an operand, are denoted by curly braces {}.

Abbreviations for Condition Codes

*	Conditionally set/cleared
-	Not affected
0	Cleared
1	Set

Abbreviations for Exceptions

rsv	Reserved operand fault
iov	Integer overflow trap
idvz	Integer divide by zero trap
fov	Floating overflow fault
fuv	Floating underflow fault
fdvz	Floating divide by zero fault
dov	Decimal overflow trap
ddvz	Decimal divide by zero trap
sub	Subscript range trap
prv	Privileged instruction fault

Integer Arithmetic And Logical Instructions

Opcode	Instruction	N Z V C	Exceptions
58	ADAWI add.rw, sum.mw	* * * *	iov
80	ADDB2 add.rb, sum.mb	* * * *	iov
C0	ADDL2 add.rl, sum.ml	* * * *	iov
A0	ADDW2 add.rw, sum.mw	* * * *	iov
81	ADDB3 add1.rb, add2.rb, sum.wb	* * * *	iov
C1	ADDL3 add1.rl, add2.rl, sum.wl	* * * *	iov
A1	ADDW3 add1.rw, add2.rw, sum.ww	* * * *	iov
D8	ADWC add.rl, sum.ml	* * * *	iov
78	ASHL cnt.rb, src.rl, dst.wl	* * * 0	iov
79	ASHQ cnt.rb, src.rq, dst.wq	* * * 0	iov
8A	BICB2 mask.rb, dst.mb	* * 0 -	
CA	BICL2 mask.rl, dst.ml	* * 0 -	
AA	BICW2 mask.rw, dst.mw	* * 0 -	
8B	BICB3 mask.rb, src.rb, dst.wb	* * 0 -	
CB	BICL3 mask.rl, src.rl, dst.wl	* * 0 -	
AB	BICW3 mask.rw, src.rw, dst.ww	* * 0 -	

Opcode	Instruction	N Z V C	Exceptions
88	BISB2 mask.rb, dst.mb	* * 0 -	
C8	BISL2 mask.rl, dst.ml	* * 0 -	
A8	BISW2 mask.rw, dst.mw	* * 0 -	
89	BISB3 mask.rb, src.rb, dst.wb	* * 0 -	
C9	BISL3 mask.rl, src.rl, dst.wl	* * 0 -	
A9	BISW3 mask.rw, src.rw, dst.ww	* * 0 -	
93	BITB mask.rb, src.rb	* * 0 -	
D3	BITL mask.rl, src.rl	* * 0 -	
B3	BITW mask.rw, src.rw	* * 0 -	
94	CLRB dst.wb	0 1 0 -	
D4	CLRL{=F} dst.wl	0 1 0 -	
7C	CLRQ{=D=G} dst.wq	0 1 0 -	
B4	CLRW dst.ww	0 1 0 -	
91	CMPB src1.rb, src2.rb	* * 0 *	
D1	CMPL src1.rl, src2.rl	* * 0 *	
B1	CMPW src1.rw, src2.rw	* * 0 *	
98	CVTBL src.rb, dst.wl	* * 0 0	
99	CVTBW src.rb, dst.wl	* * 0 0	
F6	CVTLB src.rl, dst.wb	* * * 0	iov
F7	CVTLW src.rl, dst.ww	* * * 0	iov
33	CVTWB src.rw, dst.wb	* * * 0	iov
32	CVTWL src.rw, dst.wl	* * 0 0	
97	DECB dif.mb	* * * *	iov
D7	DECL dif.ml	* * * *	iov
B7	DECW dif.mw	* * * *	iov
86	DIVB2 divr.rb, quo.mb	* * * 0	iov,idvz
C6	DIVL2 divr.rl, quo.ml	* * * 0	iov,idvz
A6	DIVW2 divr.rw, quo.mw	* * * 0	iov,idvz

Opcode	Instruction	N Z V C	Exceptions
87	DIVB3 divr.rb, divd.rb, quo.wb	* * * 0	iov,idvz
C7	DIVL3 divr.rl, divd.rl, quo.wl	* * * 0	iov,idvz
A7	DIVW3 divr.rw, divd.rw, quo.ww	* * * 0	iov,idvz
7B	EDIV divr.rl, divd.rq, quo.wl, rem.wl	* * * 0	iov,idvz
7A	EMUL mulr.rl, muld.rl, add.rl, prod.wq	* * 0 0	
96	INCB sum.mb	* * * *	iov
D6	INCL sum.ml	* * * *	iov
B6	INCW sum.mw	* * * *	iov
92	MCOMB src.rb, dst.wb	* * 0 -	
D2	MCOML src.rl, dst.wl	* * 0 -	
B2	MCOMW src.rw, dst.ww	* * 0 -	
8E	MNEGB src.rb, dst.wb	* * * *	iov
CE	MNEGL src.rl, dst.wl	* * * *	iov
AE	MNEGW src.rw, dst.ww	* * * *	iov
90	MOVB src.rb, dst.wb	* * 0 -	
D0	MOVL src.rl, dst.wl	* * 0 -	
7D	MOVQ src.rq, dst.wq	* * 0 -	
B0	MOVW src.rw, dst.ww	* * 0 -	
9A	MOVZBW src.rb, dst.wb	0 * 0 -	
9B	MOVZBL src.rb, dst.wl	0 * 0 -	
3C	MOVZWL src.rw, dst.ww	0 * 0 -	
84	MULB2 mulr.rb, prod.mb	* * * 0	iov
C4	MULL2 mulr.rl, prod.ml	* * * 0	iov
A4	MULW2 mulr.rw, prod.mw	* * * 0	iov
85	MULB3 mulr.rb, muld.rb, prod.wb	* * * 0	iov
C5	MULL3 mulr.rl, muld.rl, prod.wl	* * * 0	iov

Opcode	Instruction	N Z V C	Exceptions
A5	MULW3 mulr.rw, muld.rw, prod.ww	* * * 0	iov
DD	PUSHL src.rl, {-(SP).wl}	* * 0 -	
9C	ROTL cnt.rb, src.rl, dst.wl	* * 0 -	
D9	SBWC sub.rl, dif.ml	* * * *	iov
82	SUBB2 sub.rb, dif.mb	* * * *	iov
C2	SUBL2 sub.rl, dif.ml	* * * *	iov
A2	SUBW2 sub.rw, dif.mw	* * * *	iov
83	SUBB3 sub.rb, min.rb, dif.wb	* * * *	iov
C3	SUBL3 sub.rl, min.rl, dif.wl	* * * *	iov
A3	SUBW3 sub.rw, min.rw, dif.ww	* * * *	iov
95	TSTB src.rb	* * 0 0	
D5	TSTL src.rl	* * 0 0	
B5	TSTW src.rw	* * 0 0	
8C	XORB2 mask.rb, dst.mb	* * 0 -	
CC	XORL2 mask.rl, dst.ml	* * 0 -	
AC	XORW2 mask.rw, dst.mw	* * 0 -	
8D	XORB3 mask.rb, src.rb, dst.wb	* * 0 -	
CD	XORL3 mask.rl, src.rl, dst.wl	* * 0 -	
AD	XORW3 mask.rw, src.rw, dst.ww	* * 0 -	

Address Instructions

Opcode	Instruction	N Z V C	Exceptions
9E	MOVAB src.ab, dst.wl	* * 0 -	
DE	MOVAL{=F} src.al, dst.wl	* * 0 -	
7E	MOVAQ{=D=G} src.aq, dst.wl	* * 0 -	
3E	MOVAW src.aw, dst.wl	* * 0 -	

Opcode	Instruction	N Z V C	Exceptions
9F	PUSHAB src.ab, {-(SP).wl}	* * 0 -	
DF	PUSHAL{=F} src.al, {-(SP).wl}	* * 0 -	

7F	PUSHAQ{=D=G}
	src.aq, {-(SP).wl}
		* * 0 -	

3F	PUSHAW src.aw, {-(SP).wl}	* * 0 -	

Variable Length Bit Field Instructions

Opcode	Instruction	N Z V C	Exceptions
EC	CMPV pos.rl, size.rb, base.vb, {field.rv}, src.rl	* * 0 *	rsv
ED	CMPZV pos.rl, size.rb, base.vb, {field.rv}, src.rl	* * 0 *	rsv
EE	EXTV pos.rl, size.rb, base.vb, {field.rv}, dst.wl	* * 0 -	rsv
EF	EXTZV pos.rl, size.rb, base.vb, {field.rv}, dst.wl	* * 0 -	rsv
F0	INSV src.rl, pos.rl, size.rb, base.vb, {field.wv}	- - - -	rsv
EB	FFC startpos.rl, size.rb, base.vb, {field.rv}, findpos.wl	0 * 0 0	rsv
EA	FFS startpos.rl, size.rb, base.vb, {field.rv}, findpos.wl	0 * 0 0	rsv

Control Instructions

Opcode	Instruction	N Z V C	Exceptions
9D	ACBB limit.rb, add.rb, index.mb, displ.bw	* * * -	iov
F1	ACBL limit.rl, add.rl, index.ml, displ.bw	* * * -	iov
3D	ACBW limit.rw, add.rw, index.mw, displ.bw	* * * -	iov
F3	AOBLEQ limit.rl, index.ml, displ.bb	* * * -	iov
F2	AOBLSS limit.rl, index.ml, displ.bb	* * * -	iov
1E	BCC{=BGEQU} displ.bb		- - - -
1F	BCS{=BLSSU} displ.bb		- - - -
13	BEQL{=BEQLU} displ.bb		- - - -
18	BGEQ displ.bb		- - - -
14	BGTR displ.bb		- - - -
1A	BGTRU displ.bb		- - - -
15	BLEQ displ.bb		- - -

Opcode	Instruction	N Z V C	Exceptions
1B	BLEQU displ.bb		- - - -
19	BLSS displ.bb		- - - -
12	BNEQ{=BNEQU} displ.bb		- - - -
1C	BVC displ.bb		- - - -
1D	BVS displ.bb		- - - -
E1	BBC pos.rl, base.vb, displ.bb, {field.rv}	- - -	rsv
E0	BBS pos.rl, base.vb, displ.bb, {field.rv}	- - - - rsv	
E5	BBCC pos.rl, base.vb, displ.bb, {field.mv}	- - - -	rsv
E3	BBCS pos.rl, base.vb, displ.bb, {field.mv}	- - - -	rsv
E4	BBSC pos.rl, base.vb, displ.bb, {field.mv}	- - - -	rsv
E2	BBSS pos.rl, base.vb, displ.bb, {field.mv}	- - - -	rsv
E7	BBCCI pos.rl, base.vb, displ.bb, {field.mv}	- - - -	rsv
E6	BBSSI pos.rl, base.vb, displ.bb, {field.mv}	- - - -	rsv
E9	BLBC src.rl, displ.bb	- - - -	
E8	BLBS src.rl, displ.bb	- - - -	
11	BRB displ.bb	- - - -	
31	BRW displ.bw	- - - -	
10	BSBB displ.bb, {-(SP).wl}	- - - -	
30	BSBW displ.bw, {-(SP).wl}	- - - -	
8F	CASEB selector.rb, base.rb, limit.rb, displ.bw-list	* * 0 *	
CF	CASEL selector.rl, base.rl, limit.rl, displ.bw-list	* * 0 *	
AF	CASEW selector.rw, base.rw, limit.rw, displ.bw-list	* * 0 *	

Opcode	Instruction	N Z V C	Exceptions
17	JMP dst.ab	- - - -	
16	JSB dst.ab, {-(SP).wl}	- - - -	
05	RSB {(SP)+.rl}	- - - -	
F4	SOBGEQ index.ml, displ.bb	* * * -	iov
F5	SOBGTR index.ml, displ.bb	* * * -	iov

Procedure Call Instructions

Opcode	Instruction	N Z V C	Exceptions
FA	CALLG arglist.ab, dst.ab, {-(SP).w*}	0 0 0 0	rsv
FB	CALLS numarg.rl, dst.ab, {-(SP).w*}	0 0 0 0	rsv
04	RET {(SP)+.r*}	* * * *	rsv

Miscellaneous Instructions

Opcode	Instruction	N Z V C	Exceptions
B9	BICPSW mask.rw	* * * *	rsv
B8	BISPSW mask.rw	* * * *	rsv
03	BPT {-(KSP).w*}	0 0 0 0	
00	HALT {-(KSP).w*}	- - - -	prv
0A	INDEX subscript.rl, low.rl, high.rl, size.rl, indexin.rl,	* * 0 0	sub indexout.wl
DC	MOVPSL dst.wl	- - - -	
01	NOP	- - - -	
BA	POPR mask.rw, {(SP)+.r*}	- - - -	
BB	PUSHR mask.rw, {-(SP).w*}	- - - -	
FC	XFC {unspecified operands}	0 0 0 0	

Queue Instructions

Opcode	Instruction	N Z V C	Exceptions
5C	INSQHI entry.ab, header.aq	0 * 0 *	rsv
5D	INSQTI entry.ab, header.aq	0 * 0 *	rsv

Opcode	Instruction	N Z V C	Exceptions
0E	INSQUE entry.ab, pred.ab	* * 0 *	
5E	REMQHI header.aq, addr.wl	0 * * *	rsv
5F	REMQTI header.aq, addr.wl	0 * * *	rsv
0F	REMQUE entry.ab, addr.wl	* * * *	

Operating System Support Instructions

Opcode	Instruction	N Z V C	Exceptions
BD	CHME param.rw, {-(ySP).w*}	0 0 0 0	
BC	CHMK param.rw, {-(ySP).w*}	0 0 0 0	
BE	CHMS param.rw, {-(ySP).w*}	0 0 0 0	
BF	CHMU param.rw, {-(ySP).w*}	0 0 0 0	
	Where y=MINU(x), PSL<CURRENT_MODE>		
06	LDPCTX {PCB.r*, -(KSP).w*}	- - - -	rsv, prv
DB	MFPR procreg.rl, dst.wl	* * 0 -	rsv, prv
DA	MTPR src.rl, procreg.rl	* * 0 -	rsv, prv
0C	PROBER mode.rb, len.rw, base.ab	0 * 0 -	
0D	PROBEW mode.rb, len.rw, base.ab	0 * 0 -	
02	REI {(SP)+.r*}	* * * *	rsv
07	SVPCTX {(SP)+.r*, PCB.w*}	- - - -	prv

Floating Point Instructions

These instructions are implemented by the KA670 floating point accelerator.

Opcode	Instruction	N Z V C	Exceptions
60	ADDD2 add.rd, sum.md	* * 0 0	rsv,fov,fuv
40	ADDF2 add.rf, sum.mf	* * 0 0	rsv,fov,fuv
40FD	ADDG2 add.rg, sum.mg	* * 0 0	rsv,fov,fuv
61	ADDD3 add1.rd, add2.rd, sum.wd	* * 0 0	rsv,fov,fuv
41	ADDF3 add1.rf, add2.rf, sum.wf	* * 0 0	rsv,fov,fuv
41FD	ADDG3 add1.rg, add2.rg, sum.wg	* * 0 0	rsv,fov,fuv
71	CMPD src1.rd, src2.rd	* * 0 0	rsv
51	CMPF src1.rf, src2.rf	* * 0 0	rsv
51FD	CMPG src1.rg, src2.rg	* * 0 0	rsv

Opcode	Instruction	N Z V C	Exceptions
6C	CVTBD src.rb, dst.wd	* * 0 0	
4C	CVTBF src.rb, dst.wf	* * 0 0	
4CFD	CVTBG src.rb, dst.wg	* * 0 0	
68	CVTDB src.rd, dst.wb	* * * 0	rsv, iov
76	CVTDF src.rd, dst.wf	* * 0 0	rsv, fov
6A	CVTDL src.rd, dst.wl	* * * 0	rsv, iov
69	CVTDW src.rd, dst.ww	* * * 0	rsv, iov
48	CVTFB src.rf, dst.wb	* * * 0	rsv, iov
56	CVTFD src.rf, dst.wd	* * 0 0	rsv
99FD	CVTFG src.rf, dst.wg	* * 0 0	rsv
4A	CVTFL src.rf, dst.wl	* * * 0	rsv, iov
49	CVTFW src.rf, dst.ww	* * * 0	rsv, iov
48FD	CVTGB src.rg, dst.wb	* * * 0	rsv, iov
33FD	CVTGF src.rg, dst.wf	* * 0 0	rsv,fov,fuv
4AFD	CVTGL src.rg, dst.wl	* * * 0	rsv, iov
49FD	CVTGW src.rg, dst.ww	* * * 0	rsv, iov
6E	CVTLD src.rl, dst.wd	* * 0 0	
4E	CVTLF src.rl, dst.wf	* * 0 0	
4EFD	CVTLG src.rl, dst.wg	* * 0 0	
6D	CVTWD src.rw, dst.wd	* * 0 0	
4D	CVTWF src.rw, dst.wf	* * 0 0	
4DFD	CVTWG src.rw, dst.wg	* * 0 0	
6B	CVTRDL src.rd, dst.wl	* * * 0	rsv, iov
4B	CVTRFL src.rf, dst.wl	* * * 0	rsv, iov
4BFD	CVTRGL src.rg, dst.wl	* * * 0	rsv, iov
66	DIVD2 divr.rd, quo.md	* * 0 0	rsv,fov,fuv, fdvz
46	DIVF2 divr.rf, quo.mf * * 0 0	rsv,fov,fuv,fdvz	
46FD	DIVG2 divr.rg, quo.mg	* * 0 0	rsv,fov,fuv,fdvz
67	DIVD3 divr.rd, divd.rd, quo.wd	* * 0 0	rsv,fov,fuv,fdvz
47	DIVF3 divr.rf, divd.rf, quo.wf	* * 0 0	rsv,fov,fuv,fdvz

Opcode	Instruction	N Z V C	Exceptions
47FD	DIVG3 divr.rg, divd.rg, quo.wg	* * 0 0	rsv,fov,fuv,fdvz
72	MNEGD src.rd, dst.wd	* * 0 0	rsv
52	MNEGF src.rf, dst.wf	* * 0 0	rsv
52FD	MNEGG src.rg, dst.wg	* * 0 0	rsv
70	MOVD src.rd, dst.wd	* * 0 -	rsv
50	MOVF src.rf, dst.wf	* * 0 -	rsv
50FD	MOVG src.rg, dst.wg	* * 0 -	rsv
64	MULD2 mulr.rd, prod.md	* * 0 0	rsv,fov,fuv
44	MULF2 mulr.rf, prod.mf	* * 0 0	rsv,fov,fuv
44FD	MULG2 mulr.rg, prod.mg	* * 0 0	rsv,fov,fuv
65	MULD3 mulr.rd, muld.rd, prod.wd	* * 0 0	rsv,fov,fuv
45	MULF3 mulr.rf, muld.rf, prod.wf	* * 0 0	rsv,fov,fuv
45FD	MULG3 mulr.rg, muld.rg, prod.wg	* * 0 0	rsv,fov,fuv
62	SUBD2 sub.rd, dif.md	* * 0 0	rsv,fov,fuv
42	SUBF2 sub.rf, dif.mf	* * 0 0	rsv,fov,fuv
42FD	SUBG2 sub.rg, dif.mg	* * 0 0	rsv,fov,fuv
63	SUBD3 sub.rd, min.rd, dif.wd	* * 0 0	rsv,fov,fuv
43	SUBF3 sub.rf, min.rf, dif.wf	* * 0 0	rsv,fov,fuv
43FD	SUBG3 sub.rg, min.rg, dif.wg	* * 0 0	rsv,fov,fuv
73	TSTD src.rd	* * 0 0	rsv
53	TSTF src.rf	* * 0 0	rsv
53FD	TSTG src.rg	* * 0 0	rsv

Microcode-Assisted Emulated Instructions

The KA670 CPU provides microcode assistance for the macrocode emulation of these instructions. The CPU processes the operand specifiers, creates a standard argument list, and invokes an emulation routine to perform emulation.

Opcode	Instruction	N Z V C	Exceptions
20	ADDP4 addlen.rw, addaddr.ab, sumlen.rw, sumaddr.ab	* * * 0	rsv, dov

Opcode	Instruction	N Z V C	Exceptions
21	ADDP6 add1len.rw, add1addr.ab, add2len.rw, add2addr.ab, sumlen.rw, sumaddr.ab	* * * 0	rsv, dov
F8	ASHP cnt.rb, srclen.rw, srcaddr.ab, round.rb, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
35	CMPP3 len.rw, src1addr.ab, src2addr.ab	* * 0 0	
37	CMPP4 src1len.rw, src1addr.ab, src2len.rw, src2addr.ab	* * 0 0	
0B	CRC tbl.ab, inicrc.rl, strlen.rw, stream.ab	* * 0 0	
F9	CVTLP src.rl, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
36	CVTPL srclen.rw, srcaddr.ab, dst.wl	* * * 0	rsv, iov
08	CVTPS srclen.rw, srcaddr.ab, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
09	CVTSP srclen.rw, srcaddr.ab, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
24	CVTPT srclen.rw, srcaddr.ab, tbladdr.ab, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
26	CVTTP srclen.rw, srcaddr.ab, tbladdr.ab, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
27	DIVP divrlen.rw, divraddr.ab, divdlen.rw, divdaddr.ab, quolen.rw, quoaddr.ab	* * * 0	rsv,dov,ddvz
38	EDITPC srclen.rw, srcaddr.ab, pattern.ab, dstaddr.ab	* * * *	rsv, dov
39	MATCHC objlen.rw, objaddr.ab, srclen.rw, srcaddr.ab	0 * 0 0	
34	MOVP len.rw, srcaddr.ab, dstaddr.ab	* * 0 0	
2E	MOVTC srclen.rw, srcaddr.ab, fill.rb, tbladdr.ab, dstlen.rw, dstaddr.ab	* * 0 *	
2F	MOVTUC srclen.rw, srcaddr.ab, esc.rb, tbladdr.ab, dstlen.rw, dstaddr.ab	* * * *	
25	MULP mulrlen.rw, mulraddr.ab, muldlen.rw, muldaddr.ab, prodlen.rw, prodaddr.ab	* * * 0	rsv, dov
22	SUBP4 sublen.rw, subaddr.ab, diflen.rw, difaddr.ab	* * * 0	rsv, dov
23	SUBP6 sublen.rw, subaddr.ab, minlen.rw, minaddr.ab, diflen.rw, difaddr.ab	* * * 0	rsv, dov

E
Machine State on Power-Up

This appendix describes the state of the KA660 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<13:8>(RESTART_CODE) = 3).

The following sections describe data structures that are guaranteed to be constant over future versions of the KA660 firmware. Placement and/or existance of any other structure(s) is not implied.

E.1 Main Memory Layout and State

Main memory is tested and initialized by the firmware on power-up. Figure E–1 is a diagram of how main memory is partioned after diagnostics.

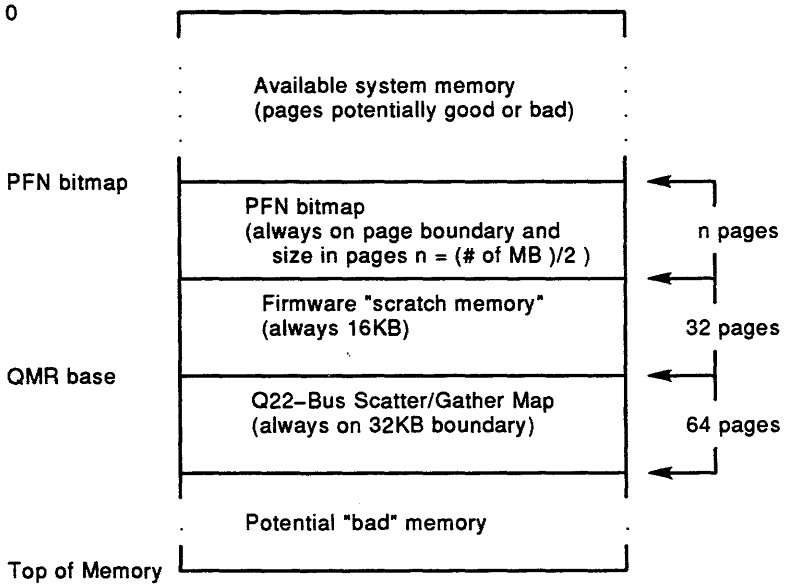

0
Available system memory
(pages potentially good or bad)
PFN bitmap
PFN bitmap
(always on page boundary and
size in pages n = (# of MB)/2)
n pages
Firmware "scratch memory"
(always 16KB)
32 pages
QMR base
Q22-Bus Scatter/Gather Map
(always on 32KB boundary)
64 pages
Potential "bad" memory
Top of Memory

Figure E–1 Memory Layout after Power-up Diagnostics

E.1.1 Reserved Main Memory

In order to build the scatter/gather map and the bitmap, the firmware attempts to find a physically contiguous page aligned 64 Kbyte block of memory at the highest possible address that has no multiple bit errors. Single bit errors are tolerated in this section.

Of the 64 Kbyte, the upper 32 Kbyte is dedicated to the Q22-bus scatter/gather map, as shown in Figure E–1. Of the lower portion, as many as 16 Kbyte at the bottom of the block is allocated to the Page Frame Number bitmap. The size of the PFN bitmap is dependent on the extent of physical memory, each bit in the bitmap maps one page (512 bytes) of memory. The remainder of the block between the bitmap and scater/gather map (minimumally 16 Kbytes) is allocated for the firmware.

E.1.1.1 PFN Bitmap

The PFN 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. The bitmap always starts on a page boundary. The bitmap requires 1 Kbyte for every 4 Mbytes of main memory, so an 8 Mbyte system requires 2 Kbytes, a 16 Mbyte system requires 4 Kbytes, a 32 Mbyte system requires 8 Kbytes, and a 64 Mbyte system requires 16 Kbytes. The bitmap does not map itself or anything above it. There may be memory above the bitmap which has both good and bad pages.

Each bit in the PFN 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 is good and can be used. A zero indicates that the page is bad and should not be used. By default, a page is flagged bad, if a multiple bit error occurs when referencing the page. Single bit errors, regardless of frequency, will not cause a page to be flagged bad.

The PFN bitmap is protected by a checksum stored in the BBURAM. The checksum is a simple byte wide, two's complement checksum. The sum of all bytes in the bitmap and the bitmap checksum should result in zero. Operating systems that modify the bitmap are encouraged to update this checksum to faciliate diagnosis by service personnel.

E.1.1.2 Scatter/Gather Map

On power-up, the scatter/gather map is initialized by the firmware to map to the first 4 Mbyte of 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 marked bad by the PFN bitmap.

On a processor halt other than power-up, the contents of the scatter/gather map is undefined and is dependent on operating system usage.

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 20088000 to 2008FFFC, 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.

E.1.1.3 Firmware Scratch Memory

This section of memory is reserved for the firmware. However, it is only used after successful execution of the memory diagnostics and initialization of the PFN bitmap and scatter/gather map. This memory is primarily used for diagnostic purposes.

E.1.2 Contents of Main Memory

The contents of main memory are undefined after the diagnostics have run. Typically, non-zero test patterns are left in memory.

The diagnostics scrub all of main memory so that no power-up induced errors remain in the memory system. On the KA660 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) all power-up induced errors.

E.2 Memory Controller Registers

The KA660 firmware assigns bank numbers to the MEMCSRs in ascending order without attempting to disable physical banks that contain errors. High order unused banks are set to zero. Error loggers should capture the following bits from each MEMCSR register:

MEMCSR<31> (bank enable bit). As the firmware always assigns banks in ascending order, knowing which banks are enabled is sufficient information to derive the bank numbers. MEMCSR<1:0> (bank usage). This field determines the size of the banks on the particular memory board.

Additional information should be captured from the MEMCSR32, MEMCSR33, MEMCSR34, MEMCSR35, and MEMCSR36 as needed.

E.2.1 On-Chip Cache

The CPU on-chip cache is tested during the power-up diagnostics, flushed and then turned off. The cache is again turned on by the BOOT and the INIT command. Otherwise, the state of the on-chip cache is disabled.

E.2.2 Translation Buffer

The CPU translation buffer is tested by diagnostics on power-up, but not used by the firmware because it runs in physical mode. The translation buffer can be invalidated by using PR$_TBIA, IPR 57.

E.2.3 Halt Protected Space

On the KA660 halt protected space spans the 256KB EPROM from 20040000 to 2007FFFF.

The firmware always runs in halt protected 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 then halts before booting.

F
Maintenance Operations Protocol (MOP) Support

F.1 Network Listening

While the KA660 is waiting for a load volunteer during bootstrap, 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 the following:

- A remote console server that generates COUNTERS messages in response to REQ_ COUNTERS messages, unsolicited SYSTEM_ID messages every 8 to 12 minutes, and solicited SYSTEM_ID messages in response to REQUEST_ID messages, and recognizes BOOT messages.
- A loopback server that responds to Ethernet LOOPBACK messages by echoing the message to the requester.
- An IEEE 802.2 responder which replies to both XID and TEST messages.

During network bootstrap operation, the KA660 complies with the requirements for a primitive node. The firmware listens only to MOP Load/Dump, MOP Remote Console, Ethernet Loopback Assistance, and IEEE 802.3 XID/TEST messages (listed in Table F–4) directed to the Ethernet physical address of the node. All other Ethernet protocols are filtered by the network device driver.

The MOP functions and message types which are supported by the KA660 are summarized in the following tables:

Table F–1 KA660 Network Maintenance Operations Summary

Function	Role	Transmit	Receive
		MOP Ethernet and IEEE 802.3 Messages [1]	
Dump	Requester	—	—
	Server	—	—

[1] All unsolicited messages are sent in Ethernet (MOP V3) and IEEE 802.2 (MOP V4), until the MOP version of the server is known. All solicited messages are sent in the format used for the request.

Table F–1 (Cont.) KA660 Network Maintenance Operations Summary

Function	Role	Transmit		Receive
		MOP Ethernet and IEEE 802.3 Messages [1]		
Load	Requester	REQ_ PROGRAM[2]	to solicit	VOLUNTEER
		REQ_MEM_ LOAD	to solicit and ACK	MEM_LOAD
			or	MEM_LOAD_w_XFER
			or	PARAM_LOAD_w_XFER
	Server	—		—
Console	Requester	—		—
	Server	COUNTERS	in response to	REQ_COUNTERS
		SYSTEM_ID[3]	in response to	REQUEST_ID BOOT
Loopback	Requester	—		—
	Server	LOOPED_ DATA[4]	in response to	LOOP_DATA
		IEEE 802.2 Messages[5]		
Exchange ID	Requester	—		—
	Server	XID_RSP	in response to	XID_CMD
Test	Requester	—		—
	Server	TEST_RSP	in response to	TEST_CMD

[1] All unsolicited messages are sent in Ethernet (MOP V3) and IEEE 802.2 (MOP V4), until the MOP version of the server is known. All solicited messages are sent in the format used for the request.

[2] 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.

[3] 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.

[4] 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 when sent in Ethernet frames omit the additional length field (padding is disabled).

[5] IEEE 802.2 support of XID and TEST is limited to Class 1 operations.

Table F–2 Supported MOP Messages

Message Type	Message Fields						
DUMP/LOAD							
MEM_LOAD_w_	Code	Load #	Load addr		Image data		Xfer addr
XFER	00	nn	aa-aa-aa-aa		None		aa-aa-aa-aa
MEM_LOAD	Code	Load #	Load addr		Image data		
	02	nn	aa-aa-aa-aa		dd-...		
REQ_	Code	Device	Format	Program	SW ID	Procesr	Info
PROGRAM	08	25			[3]		(see SYSTEM_ID)
		LQA	01	02	C-17 [1]	00	
		??	V3	Sys	C-128	Sys	
		KA660	04 V4		[2]		
					If C[1]		
					>00		
					Len		
					00 No		
					ID		
					FF OS		
					FE		
					Maint		
REQ_MEM_	Code	Load #	Error				
LOAD	0A	nn	ee				
PARM_LOAD_	Code	Load #	Prm	Prm	Prm val		Xfer addr
w_XFER	14	nn	typ	len	Target name [1]		aa-aa-aa-aa
			01	I-16	Target addr [1]		
			02	I-06	Host name [1]		
			03	I-16	Host addr [1]		
			04	I-06	Host time [1]		
			05	0A	Host time [2]		
			06	08			
			00 End				
VOLUNTEER	Code						
	03						
REMOTE CONSOLE							
REQUEST_ID	Code	Rsrvd	Recpt #				
	05	xx	nn-nn				

[1] MOP V3.0 only.

[2] MOP x4.0 only.

[3] Software ID field is load from the string stored in the 40 byte field, RPB$T_FILE, of the RPB on a solicited boot.

Table F–2 (Cont.) Supported MOP Messages

Message Type	Message Fields							
REMOTE CONSOLE								
SYSTEM_ID	Code	Rsrvd	Recpt	Info type		Info	Info value	
	07	xx	#	01-00 Version		len	04-00-00	
			nn-nn	02-00 Functions		03	00-59	
			or	07-00 HW addr		02	ee-ee-ee-ee-ee-ee	
			00-00	64-00 Device		06	25 or ??	
				90-01 Datalink		01	01	
				91-01 Bufr size		01	06-04	
						02		
REQ_ COUNTERS	Code	Recpt						
	09	#						
		nn-nn						
COUNTERS	Code	Recpt	Counter block					
	0B	#						
		nn-nn						
BOOT [4]	Code	Verification		Procesr	Control	Dev ID	SW ID	Script
	06	vv-vv-vv-vv-vv-vv-				C-17	[3]	ID [2]
		vv-vv		00	xx		(see	C-128
				Sys			REQ_	
							PROGRAM)	
LOOPBACK								
LOOP_DATA	Skpcnt	Skipped bytes		Function		Forward addr		Data
		bb-...		00-02 Forward		ee-ee-ee-ee-ee-ee		dd-...
	nn-			data				
	nn							
LOOPED_DATA	Skpcnt	Skipped bytes		Function		Recpt #		Data
		bb-...		00-01 Reply		nn-nn		dd-...
	nn-							
	nn							
IEEE 802.2								
XID_CMD/RSP	Form	Class	Rx window size (K)					
	81	01	00					
TEST_CMD/RSP	Optional data.							

[2] MOP x4.0 only.

[3] Software ID field is load from the string stored in the 40 byte field, RPB$T_FILE, of the RPB on a solicited boot.

[4] A BOOT message is not verified, because in this context, a boot is already in progress. However, a received BOOT message will cause the boot backoff timer to be reset to its minimum value.

Table F–3 Ethernet and IEEE 802.3 Packet Headers

Ethernet MOP Message Format (MOP V3)

Dest_address	Src_address	Prot	Len	MOP msg	Pad	CRC
dd-dd-dd-dd-dd-dd	ss-ss-ss-ss-ss-ss	60-01	nn-nn	dd-...	xx-...	cc-cc
		60-02	nn-nn	dd-...		
		90-00	dd-...			

IEEE 802.3 SNAP SAP MOP Message Format (MOP V4)

Dest_address	Src_address	Len	DSAP	SSAP	Ctl	P_ID	MOP_msg	CRC
dd-dd-dd-dd-dd-dd	ss-ss-ss-ss-ss-ss	nn-nn	AA	AA	03	08-00-2B-60-01 08-00-2B-60-02 08-00-2B-90-00	dd-...	cc-cc

IEEE 802.3 XID/TEST Message Format (MOP V4)

Dest_address	Src_address	Len	DSAP	SSAP	Ctl [1]	Data	CRC
dd-dd-dd-dd-dd-dd	ss-ss-ss-ss-ss-ss	nn-nn	aa	bb	cc	ff-tt-ss (XID) Optional data (TEST)	cc-cc

[1] XID and TEST messages are identified in the IEEE 802.2 control field with binary 101x1111 and 111x0011, respectively. "x" denotes the Poll/Final bit which gets echoed in the response.

Table F–4 MOP Multicast Addresses and Protocol Specifiers

Function	Address	IEEE Prefix [1]	Protocol	Owner
Dump/Load	AB-00-00-01-00-00	08-00-2B	60-01	Digital
Remote Console	AB-00-00-02-00-00	08-00-2B	60-02	Digital
Loopback Assistance	CF-00-00-00-00-00[2]	08-00-2B	90-00	Digital

[1] MOP 4.0 only.

[2] Not used.

F.2 MOP Counters

The counters listed in Table F–5 are kept for the Ethernet boot channel. All counters are unsigned integers. V4 counters roll over on overflow. All V3 counters "latch" at their maximum value to indicate overflow. Unless otherwise stated, all counters include both normal and multicast traffic. Furthermore, they include information for all protocol types. Frames received and bytes received counters do not include frames received with errors. Table F–5 displays the byte lengths and ordering of all the counters in both MOP Version 3.0 and 4.0.

Table F–5 MOP Counter Block

	V3		V4		
Name	**Off**	**Len**	**Off**	**Len**	**Description**
TIME_SINCE_CREATION	00	2	00	16	**Time since last zeroed** This is the time which has elapsed, since the counters were last zeroed. It provides a frame of reference for the other counters by indicating the amount of time they cover. For MOP V3, this time is the number of seconds. MOP V4 uses the UTC Binary Relative Time format.
Rx_BYTES	02	4	10	8	**Bytes received** This is the total number of user data bytes successfully received. This does not include Ethernet data link headers. This number is the number of bytes in the Ethernet data field, which includes any padding or length fields when they are enabled. These are bytes from frames that passed hardware filtering. When the number of frames received is used to calculate protocol overhead, the overhead plus bytes received provides a measurement of the amount of Ethernet bandwidth (over time) consumed by frames addressed to the local system.
Tx_BYTES	06	4	18	8	**Bytes sent** This is the total number of user data bytes successfully transmitted. This does not include Ethernet data link headers or data link generated retransmissions. This number is the number of bytes in the Ethernet data field, which includes any padding or length fields when they are enabled. When the number of frames sent is used to calculate protocol overhead, the overhead plus bytes sent provides a measurement of the amount of Ethernet bandwidth (over time) consumed by frames sent by the local system.
Rx_FRAMES	0A	4	20	8	**Frames received** This is the total number of frames successfully received. These are frames that passed hardware filtering. It provides a gross measurement of incoming Ethernet usage by the local system. It provides information used to determine the ratio of the error counters to successful transmits.

Table F–5 (Cont.) MOP Counter Block

	V3		V4		
Name	**Off**	**Len**	**Off**	**Len**	**Description**
Tx_FRAMES	0E	4	28	8	**Frames sent** This is the total number of frames successfully transmitted. This does not include data link generated retransmissions. It provides a gross measurement of outgoing Ethernet usage by the local system. It provides information used to determine the ratio of the error counters to successful transmits.
Rx_MCAST_BYTES	12	4	30	8	**Multicast bytes received** This is the total number of multicast data bytes successfully received. This does not include Ethernet data link headers. This number is the number of bytes in the Ethernet data field. In conjunction with total bytes received, it provides a measurement of the percentage of this system's receive bandwidth (over time) that was consumed by multicast frames addressed to the local system.
Rx_MCAST_FRAMES	16	4	38	8	**Multicast frames received** This is the total number of multicast frames successfully received. In conjunction with the total frames received, it provides a gross percentage of the Ethernet usage for multicast frames addressed to this system.
Tx_INIT_DEFFERED	1A	4	40	8	**Frames sent[1], initially deferred** This is the total number of times that a frame transmission was deferred on its first transmission attempt. In conjunction with the total frames sent, it measures Ethernet contention with no collisions.
Tx_ONE_COLLISION	1E	4	48	8	**Frames sent[1], single collision** This is the total number of times that a frame was successfully transmitted on the second attempt, after a normal collision on the first attempt. In conjunction with the total frames sent, it measures Ethernet contention at a level where there are collisions but the backoff algorithm still operates efficiently.

[1] Only one of these three counters will be incremented for a given frame.

Table F–5 (Cont.) MOP Counter Block

	V3		V4		
Name	**Off**	**Len**	**Off**	**Len**	**Description**
Tx_MULTI_COLLISION	22	4	50	8	**Frames sent[1], multiple collisions** This is the total number of times that a frame was successfully transmitted on the third or later attempt, after normal collisions on previous attempts. In conjunction with the total frames sent, it measures Ethernet contention at a level where there are collisions and the backoff algorithm no longer operates efficiently. NO SINGLE FRAME IS COUNTED IN MORE THAN ONE OF THE ABOVE THREE COUNTERS.
TxFAIL_COUNT	26	2	—	—	**Send failure count[2]** This is the total number of times a transmit attempt failed. Each time the counter is incremented, a type of failure is recorded. When the read-counter function reads the counter, the list of failures is also read. When the counter is set to zero, the list of failures is cleared. In conjunction with the total frames sent, it provides a measure of significant transmit problems. TxFAIL_BITMAP contains the possible reasons.
TxFAIL_BITMAP	2C	2	—	—	**Send failure reason bit map[2]** This bit map lists the types of transmit failures that occurred as summarized here: 0 - Excessive collisions 1 - Carrier detect failed 2 - Short circuit 3 - Open circuit 4 - Frame too long 5 - Remote failure to defer
TxFAIL_EXCESS_COLLS	—	—	58	8	**Send failure - excessive collisions** The maximum number of retransmissions due to collisions was exceeded. This indicates an overload condition on the Ethernet.

[1] Only one of these three counters will be incremented for a given frame.

[2] V3 send/receive failures are collapsed into one counter with bit map indicating which failures occurred.

Table F–5 (Cont.) MOP Counter Block

	V3		V4		
Name	**Off**	**Len**	**Off**	**Len**	**Description**
TxFAIL_CARIER_CHECK	—	—	60	8	**Send failure - carrier check failed** The data link did not sense the receive signal that is required to accompany the transmission of a frame. This indicates a failure in either the transmitting or receiving hardware. It could be caused by either the transceiver, the transceiver cable, or a babbling controller that has been cut off.
TxFAIL_SHRT_CIRCUIT	—	—	68	8	**Send failure - short circuit**[3] There is a short somewhere in the local area network coaxial cable or the transceiver or controller/transceiver cable has failed. This indicates a problem either in local hardware or global network. The two can be distinguished by checking to see if other systems are reporting the same problem.
TxFAIL_OPEN_CIRCUIT	—	—	70	8	**Send failure - open circuit**[3] There is a break somewhere in the local area network coaxial cable. This indicates a problem either in local hardware or global network. The two can be distinguished by checking to see if other systems are reporting the same problem.
TxFAIL_LONG_FRAME	—	—	78	8	**Send failure - frame too long**[3] The controller or transceiver cut off transmission at the maximum size. This indicates a problem with the local system. Either it tried to send a frame that was too long or the hardware cut off transmission too soon.
TxFAIL_REMOTE_DEFER	—	—	80	8	**Send failure - remote failure to defer**[3] A remote system began transmitting after the allowed window for collisions. This indicates either a problem with some other system's carrier sense or a weak transmitter.

[3] Always zero.

Table F–5 (Cont.) MOP Counter Block

	V3		V4		
Name	**Off**	**Len**	**Off**	**Len**	**Description**
RxFAIL_COUNT	2A	2	—	—	**Receive failure count**[2] The total number of frames received with some data error. Includes only data frames that passed either physical or multicast address comparison. This counter includes failure reasons in the same way as the send failure counter. In conjunction with the total frames received, it provides a measure of data-related receive problems. RxFAIL_BITMAP contains the possible reasons.
RxFAIL_BITMAP	2C	2	—	—	**Receive failure reason bit map**[2] This bit map lists the types of receive failures that occurred as summarized here: 0 - Block check failure 1 - Framing error 2 - Frame too long
RxFAIL_BLOCK_CHECK	—	—	88	8	**Receive failure - block check error** A frame failed the CRC check. This indicates several possible failures, such as, EMI, late collisions, or improperly set hardware parameters.
RxFAIL_FRAMING_ERR	—	—	90	8	**Receive failure - framing error** The frame did not contain an integral number of 8-bit bytes. This indicates several possible failures, such as, EMI, late collisions, or improperly set hardware parameters.
RxFAIL_LONG_FRAME	—	—	98	8	**Receive failure - frame too long**[3] The frame was discarded because it was outside the Ethernet maximum length and could not be received. This indicates that a remote system is sending invalid length frames.
UNKNOWN_ DESTINATION	2E	2	A0	8	**Unrecognized frame destination** This is the number of times a frame was discarded because there was no portal with the protocol type or multicast address enabled. This includes frames received for the physical address, the broadcast address, or a multicast address.

[2] V3 send/receive failures are collapsed into one counter with bit map indicating which failures occurred.

[3] Always zero.

Table F–5 (Cont.) MOP Counter Block

	V3		V4		
Name	**Off**	**Len**	**Off**	**Len**	**Description**
DATA_OVERRUN	30	2	A8	8	**Data overrun** This is the total number of times the hardware lost an incoming frame because it was unable to keep up with the data rate. In conjunction with the total frames received, it provides a measure of hardware resource failures. The problem reflected in this counter is also captured as an event.
NO_SYSTEM_BUFFER	32	2	B0	8	**System buffer unavailable**[3] This is the total number of times no system buffer was available for an incoming frame. In conjunction with the total frames received, it provides a measure of system buffer-related receive problems. The problem reflected in this counter is also captured as an event. This can be any buffer between the hardware and the user buffers (those supplied on receive requests). Further information as to potential different buffer pools is implementation specific.
NO_USER_BUFFER	34	2	B8	8	**User buffer unavailable**[3] This is the total number of times no user buffer was available for an incoming frame that passed all filtering. These are the buffers supplied by users on receive requests. In conjunction with the total frames received, it provides a measure of user buffer-related receive problems. The problem reflected in this counter is also captured as an event.
FAIL_COLLIS_DETECT	—	—	C0	8	**Collision detect check failure** This is the approximate number of times that collision detect was not sensed after a transmission. If this counter contains a number roughly equal to the number of frames sent, either the collision detect circuitry is not working correctly or the test signal is not implemented.

[3] Always zero.

G
ROM Partitioning

This appendix describes ROM partitioning and subroutine entry points that are public and are guaranteed to be compatible over future versions of the KA660 firmware. An entry point is the address at which any subroutine or subprogram will start execution.

G.1 Firmware EPROM Layout

The KA660 has 256 Kbytes of EPROM. Unlike previous Q22-bus based MicroVAX processors, there is no duplicate decoding of the EPROM into halt-protected and halt-unprotected spaces. The entire EPROM is halt-protected.

20040000	Branch instruction
20040006	System Id Extension
20040008	CP$GETCHAR_R4
2004000C	CP$MSG_OUT_NOLF_R4
20040010	CP$READ_WTH_PRMPT_R4
20040014	Rsvd Mfg L200 Testing
20040018	Def Boot Dev Dscr Ptr
2004001c	Def Boot Flags Ptr
	Console, diagnostic and boot code
	EPROM checksum
	Reserved for Digital
2005F800	4 pages reserved for customer use
2005FFFC	

Figure G–1 KA660 EPROM Layout

The first instruction executed on halts is a branch around the System Id Extention (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. This is documented further in the next section.

The fixed area checksum is the sum of longwords from 20040000 to the checksum inclusive. This checksum is distinct from the checksum that the rest of the console uses in that the fixed checksum is calculated on a stationary series of longwords, whereas the other console checksum is calculated on a set of longwords that my vary in size according to the size of the firmware image.

The console, diagnostic, and boot code constitute the bulk of the KA660 firmware. This code is field upgradable. The console checksum is from 20044000 to the checksum inclusive.

The memory between the console checksum and the user area at the end of the EPROM is reserved for DIGITAL for future expansion of the KA660 firmware. The contents of this area is set to FF.

The last 4096 bytes of EPROM is reserved for customer use and is not included in the console checksum. During a PROM bootstrap with PRB0 as the selected boot device, this block is the tested for a PROM "signature block". Refer to Section 12.4.3.2 and Figure 12–12 for a description of the boot block mechanism.

G.1.1 Call-Back Entry Points

The KA660 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.

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 in Table G–1 and are described in the following sections.

Table G–1 Call-Back Entry Points

CP$GET_CHAR_R4	20040008
CP$MSG_OUT_NOLF_R4	2004000C
CP$READ_WTH_PRMPT_R4	20040010

G.1.1.1 CP$GET_CHAR_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.

;--------------------------------------------------------------------
```

G.1.1.2 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$:     .ascii  /This is a message/     ; Message.
10$:                                    ;

;--------------------------------------------------------------------
```

G.1.1.3 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, Ctrl/U, and Ctrl/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; therefore, the caller does not need to 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$:     .ascii  /Prompt> /              ; Prompt string.
10$:

;----------------------------------------------------------------------
```

G.1.2 Boot Information Pointers

Two longwords located in EPROM are used as pointers to the default boot device descriptor and the default boot flags, because the actual location of this data may change in successive versions of the firmware. Any software that uses these pointers should refer to them at the addresses in halt-protected space.

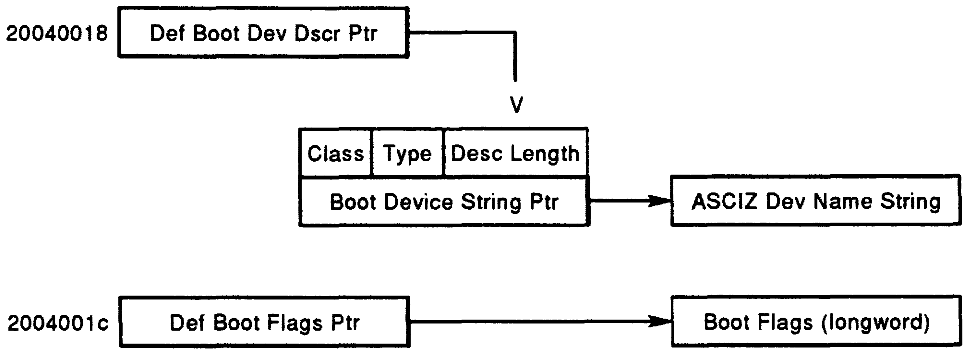


Figure G–2 Boot Information Pointers

The following macro defines the boot device descriptor format:

```
; Default Boot Device Descriptor
;
boot_device_descriptor::
        base = .
        . = base + dsc$w_length
        .word   nvr$s_boot_device

        . = base + dsc$b_dtype
        .byte   dsc$k_dtype_z

        . = base + dsc$b_class
        .byte   dsc$k_class_z

        . = base + dsc$a_pointer
        .long   nvr_base + nvr$b_boot_device

        . = base + dsc$s_dscdef1
```

H

Battery Backed-up RAM Partitioning

This appendix describes the KA660 firmware partitions of the SSC 1-Kbyte battery backed-up (BBU) RAM.

H.1 SSC RAM Layout

The KA660 firmware uses the 1 Kbyte of BBU RAM on the SSC for storage of firmware specific data structures and other information that must be preserved across power cycles. This BBU RAM resides in the SSC chip starting at address 20140400. The BBU RAM should not be used by the operating systems except as documented in the following sections. This BBU RAM is not reflected in the bit map built by the firmware.

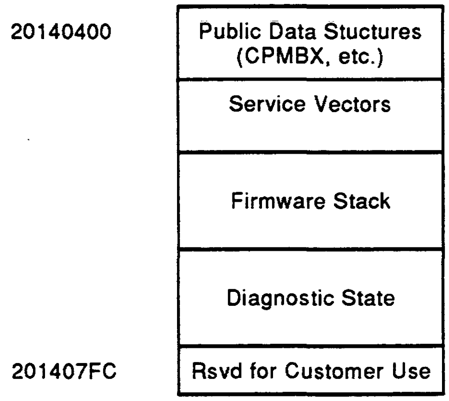


Figure H–1 KA660 SSC BBU RAM Layout

H.1.1 Public Data Structures Area

The public data structures area contains the console program mailbox Section H.1.2.

Fields that are designated as reserved or internal use should not be written, because there is no protection against such corruption.

H.1.2 Console Program Mailbox (CPMBX)

The console program mailbox (CPMBX) is a software data structure located at the beginning of BBU RAM (20140400). The CPMBX is used to pass information between the KA660 firmware and diagnostics, VMB, or an operating system. It consists of three bytes referred to here as NVR0, NVR1, and NVR2.

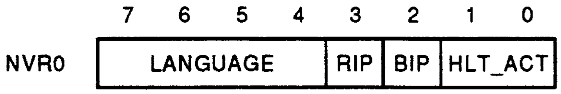


Figure H–2 NVR0 (20140400) : Console Program Mailbox (CPMBX)

Field	Name	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 12–1 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.

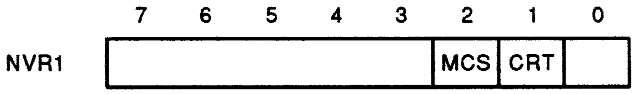


Figure H–3 NVR1 (20140401)

Field	Name	Description
2	MCS	If set, this field indicates that the attached terminal supports Multinational Character Set (MCS). If clear, MCS is not supported.
1	CRT	If set, this field indicates that the attached terminal is a CRT. If clear, it indicates that the terminal is hard copy.

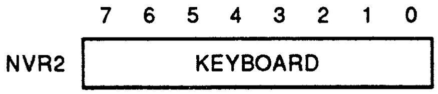


Figure H–4 NVR2 (20140402)

Field	Name	Description
7:0	KEYBOARD	This field indicates the national keyboard variant in use.

H.1.3 Firmware Stack Area

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.

H.1.4 Diagnostic State Area

This area is used by the firmware resident diagnostics. This section is not documented here.

H.1.5 USER Area

The KA660 console reserves the last longword (address 201407FC) of the BBU RAM for customer use. This location is not tested by the console firmware. Its value is undefined.

I
Data Structures

This appendix contains definitions of key global data structures which are used by the KA660 firmware.

I.1 Halt Dispatch State Machine

The KA660 halt dispatcher determines what actions the firmware will take on halt entry based on the machine state. The dispatcher is implemented as a state machine, which uses a single bit map control word and the transition Table I–1 to process all halts. The transition table is sequentially searched for matches with the current state and control word. If there is a match, a transition occurs to the next state.

The control word is comprised of the following information:

- **Halt type** is used for resolving external halts. It is valid only if the halt code is 00.

  000 : power-up state
  001 : halt in progress
  010 : negation of Q22-bus DCOK
  011 : console BREAK condition detected
  100 : Q22-bus BHALT
  101 : SGEC BOOT_L asserted (trigger boot)

- **Halt code** is a compressed form of SAVPSL<13:8>(RESTART_CODE).

  00 : RESTART_CODE = 2, external halt
  01 : RESTART_CODE = 3, power-up/reset
  10 : RESTART_CODE = 6, halt instruction
  11 : RESTART_CODE = any other, error halts

- **Mailbox action** is passed by an operating system in CPMBX<1:0>(HALT_ACTION).

  00 : restart, boot, halt
  01 : restart, halt
  10 : boot, halt
  11 : halt

- **User action** is specified with the SET HALT console command.

  000 : default
  001 : restart, halt
  010 : boot, halt
  011 : halt
  100 : restart, boot, halt

- **HEN** is the BREAK (halt) enable switch, BDR<23>.
- **ERR** is the error status.

- **TIP** is the trace in progress.
- **DIP** is the diagnostics in progress.
- **BIP** is the bootstrap in progress CPMBX<2>.
- **RIP** is the restart in progress CPMBX<3>.

A transition to a next state occurs if a match is found between the control word and a current state entry in the table. The firmware does a linear search through the table for a match. Therefore, the order of the entries in the transition table is important. The control longword is reassembled before each transition from the current machine state. The state machine transitions are shown in Table I–1.

Table I–1 Firmware State Transition Table

Current State	Next State	Halt Type	Halt Code	Mailbox Action	User Action	HEN-ERR-TIP-DIP-BIP-RIP
		Perform conditional initialization.[1]				
ENTRY	->RESET INIT	xxx	01	xx	xxx	x - x - x - x - x - x
ENTRY	->BREAK INIT	011	00	xx	xxx	x - x - x - x - x - x
ENTRY	->TRACE INIT	xxx	10	xx	xxx	x - 0 - 1 - x - x - x
ENTRY	->OTHER INIT	xxx	xx	xx	xxx	x - x - x - x - x - x
		Perform common initialization. [2]				
RESET INIT	->INIT	xxx	xx	xx	xxx	x - x - x - x - x - x
BREAK INIT	->INIT	xxx	xx	xx	xxx	x - x - x - x - x - x
TRACE INIT	->INIT	xxx	xx	xx	xxx	x - x - x - x - x - x
OTHER INIT	->INIT	xxx	xx	xx	xxx	x - x - x - x - x - x
		Check for external halts. [3]				
INIT	->BOOTSTRAP	010	00	xx	xxx	0 - x - x - x - x - x
INIT	->BOOTSTRAP	101	00	xx	xxx	x - x - x - x - x - x
INIT	->HALT	xxx	00	xx	xxx	x - x - x - x - x - x
		Check for pending (NEXT) trace.[4]				

[1] Perform a unique initialization routine on entry. In particular, power-ups, BREAKs, and TRACEs require special initialization. Any other halt entry performs a default initialization.

[2] After performing conditional initialization, complete common initialization.

[3] Halt on all external halts, except:

if DCOK (unlikely) and halts are disabled, perform a bootstrap
if SGEC remote trigger, bootstrap

[4] Unconditionally enter the TRACE state, if the TIP flag is set and the halt was due to a HALT instruction. From the TRACE state the firmware exits, if TIP is set and ERR is clear, otherwise it halts.

Table I–1 (Cont.) Firmware State Transition Table

Current State	Next State	Halt Type	Halt Code	Mailbox Action	User Action	HEN-ERR-TIP-DIP-BIP-RIP
INIT	–>TRACE	xxx	10	xx	xxx	x - x - 1 - x - x - x
TRACE	–>EXIT	xxx	10	xx	xxx	x - 0 - 1 - x - x - x
TRACE	–>HALT	xxx	xx	xx	xxx	x - x - x - x - x - x
		Check for bootstrap conditions.[5]				
INIT	–>BOOTSTRAP	xxx	01	xx	xxx	0 - 0 - 0 - 0 - 0 - 0
INIT	–>BOOTSTRAP	xxx	01	xx	010	1 - 0 - 0 - 0 - 0 - 0
INIT	–>BOOTSTRAP	xxx	01	xx	100	1 - 0 - 0 - 0 - 0 - 0
INIT	–>BOOTSTRAP	xxx	1x	10	xxx	x - 0 - 0 - 0 - 0 - 0
INIT	–>BOOTSTRAP	xxx	1x	00	010	x - 0 - 0 - 0 - 0 - 0
INIT	–>BOOTSTRAP	xxx	1x	00	100	x - 0 - 0 - 0 - 0 - 1
INIT	–>BOOTSTRAP	xxx	1x	00	100	x - 1 - 0 - 0 - 0 - x
INIT	–>BOOTSTRAP	xxx	1x	00	000	0 - 0 - 0 - 0 - 0 - 1
RESTART	–>BOOTSTRAP	xxx	1x	00	000	0 - 1 - 0 - 0 - 0 - x
		Check for restart conditions. [6]				
INIT	–>RESTART	xxx	1x	01	xxx	x - 0 - 0 - 0 - 0 - 0
INIT	–>RESTART	xxx	1x	00	001	x - 0 - 0 - 0 - 0 - 0
INIT	–>RESTART	xxx	1x	00	100	x - 0 - 0 - 0 - 0 - 0
INIT	–>RESTART	xxx	1x	00	000	0 - 0 - 0 - 0 - 0 - 0
		Perform common exit processing, if no errors. [7]				
BOOTSTRAP	–>EXIT	xxx	xx	xx	xxx	x - 0 - x - x - x - x
RESTART	–>EXIT	xxx	xx	xx	xxx	x - 0 - x - x - x - x
HALT	–>EXIT	xxx	xx	xx	xxx	x - 0 - x - x - x - x
		Exception transitions, just halt. [8]				
INIT	–>HALT	xxx	xx	xx	xxx	x - x - x - x - x - x
BOOT	–>HALT	xxx	xx	xx	xxx	x - x - x - x - x - x

[5] Perform a bootstrap if:

power-up and halts are disabled
power-up and halts are enabled and user action is 2 or 4
not power-up and mailbox is 2
not power-up, mailbox is 0, and user action is 2
not power-up, restart failed, mailbox is 0, and user action is 0 or 4

[6] Restart the operating system, if not power-up, and if:

mailbox is 1
mailbox is 0 and user action is 1 or 4
mailbox is 0, user action is 0, and halts are disabled

[7] Exit after halts, bootstrap or restart. The exit state transitions to program I/O mode.

[8] Guard block that catches all exception conditions. In all cases, just halt.

Table I–1 (Cont.) Firmware State Transition Table

Current State	Next State	Halt Type	Halt Code	Mailbox Action	User Action	HEN-ERR-TIP-DIP-BIP-RIP
RESTART	->HALT	xxx	xx	xx	xxx	x - x - x - x - x - x
HALT	->HALT	xxx	xx	xx	xxx	x - x - x - x - x - x
TRACE	->HALT	xxx	xx	xx	xxx	x - x - x - x - x - x
EXIT	->HALT	xxx	xx	xx	xxx	x - x - x - x - x - x

x = don't care

I.2 Restart Parameter Block

VMB typically utilizes the low portion of memory unless there are bad pages in the first 128 Kbytes. The first page in its block is used for the restart parameter block (RPB), through which it communicates to the operating system. Usually, this is page 0.

VMB will initialize the RPB as follows:

Table I–2 Restart Parameter Block fields

(R11)+	Field Name	Description
00:	RPB$L_BASE	Physical address of base of RPB
04:	RPB$L_RESTART	Cleared
08:	RPB$L_CHKSUM	-1
0C:	RPB$L_RSTRTFLG	Cleared
10:	RPB$L_HALTPC	R10 on entry to VMB (HALT PC)
10:	RPB$L_HALTPSL	PR$_SAVPSL on entry to VMB (HALT PSL)
18:	RPB$L_HALTCODE	AP on entry to VMB (HALT CODE)
1C:	RPB$L_BOOTR0	R0 on entry to VMB **NOTE** **The field RPB$W_R0UBVEC, which overlaps the high order word of RPB$L_BOOTR0, is set by the boot device drivers to the SCB offset (in the second page of the SCB) of the interrupt vector for the boot device.**
20:	RPB$L_BOOTR1	VMB version number. The high-order word of the version is the major ID and the low-order word is the minor ID.
24:	RPB$L_BOOTR2	R2 on entry to VMB
28:	RPB$L_BOOTR3	R3 on entry to VMB

Table I–2 (Cont.) Restart Parameter Block fields

(R11)+	Field Name	Description
2C:	RPB$L_ BOOTR4	R4 on entry to VMB **NOTE** **The 48-bit booting node address is stored in RPB$L_BOOTR3 and RPB$L_BOOTR4 for compatibility with VAXELN V1.1. (This field is only initialized this way when performing a network boot.)**
30:	RPB$L_ BOOTR5	R5 on entry to VMB
34:	RPB$L_IOVEC	Physical address of boot driver's I/O vector of transfer addresses
38:	RPB$L_ IOVECSZ	Size of BOOT QIO routine
3C:	RPB$L_ FILLBN	LBN of secondary bootstrap image
40:	RPB$L_FILSIZ	Size of secondary bootstrap image in blocks
44:	RPB$Q_ PFNMAP	The PFN bit map is an array of bits, where each bit has the value 1, if the corresponding page of memory is valid, or the value 0, if the corresponding page of memory contains a memory error. Through use of the PFNMAP, the operating system can avoid memory errors by avoiding known bad pages altogether. The memory bit map is always page-aligned, and describes all the pages of memory from physical page #0 to the high end of memory, but excluding the PFN bit map itself and the Q-bus map registers. If the high byte of the bit map spans some pages available to the operating system and some pages of the PFN bit map itself, the pages corresponding to the bit map itself will be marked as bad pages. The first longword of the PFNMAP descriptor contains the number of bytes in the PFNMAP; the second longword contains the physical address of the bit map.
4C:	RPB$L_ PFNCNT	Count of good pages of physical memory, but not including the pages allocated to the Q22-bus scatter/gather map, the console scratch area, and the PFN bit map at the top of memory.
50:	RPB$L_ SVASPT	0
54:	RPB$L_ CSRPHY	Physical address of CSR for boot device
58:	RPB$L_ CSRVIR	0
5C:	RPB$L_ ADPPHY	Physical address of ADP really the address of QMRs - ^x800 to look like a UBA adapter
60:	RPB$L_ ADPVIR	0
64:	RPB$W_UNIT	Unit number of boot device
66:	RPB$B_ DEVTYP	Device type code of boot device

Table I–2 (Cont.) Restart Parameter Block fields

(R11)+	Field Name	Description
67:	RPB$B_SLAVE	Slave number of boot device
68:	RPB$T_FILE	Name of secondary bootstrap image (defaults to [SYS0.SYSEXE]SYSBOOT.EXE). This field (up to 40 bytes) is overwritten with the input string on a solicit boot. **NOTE** **1 : For VMS, the RPB$T_FILE must contain the root directory string SYS*n*. on a non-network bootstrap. This string is parsed by SYSBOOT (that is, SYSBOOT does not use the high nibble of BOOTR5).** **2 : The RPB$T_FILE is overwritten to contain the boot node name for compatibility with VAXELN V1.1. (This field is only initialized this way when performing a network boot.)**
90:	RPB$B_CONFREG	Array (16 bytes) of adapter types (NDT$_UB0 - UNIBUS)
A0:	RPB$B_HDRPGCNT	Count of header pages
A1:	RPB$W_BOOTNDT	Boot adapter nexus device type. Used by SYSBOOT and INIADP (OF SYSLOA) to configure the adapter of the boot device (changed from a byte to a word field in Version 12 of VMB).
B0:	RPB$L_SCBB	Physical address of SCB
BC:	RPB$L_MEMDSC	Count of pages in physical memory including both good and bad pages. The high 8 bits of this longword contain the TR #, which is always zero for KA660.
C0:	RPB$L_MEMDSC+4	PFN of the first page of memory. This field is always zero for KA660, even if page #0 is a bad page. **NOTE** **No other memory descriptors are used.**
104:	RPB$L_BADPGS	Count of bad pages of physical memory.
108:	RPB$B_CTRLLTR	Boot device controller number biased by 1. In VMS, this field is used by INIT (in SYS) to construct the boot device's controller letter. A zero implies this field has not been initialized; if it is initialized, then A=1, B=2, and so on. (This field was added in Version 13 of VMB.)
nn:		The rest of the RPB is zeroed.

I.3 VMB Argument List

The VMB code will also initialize an argument list as follows (the address of the argument list is passed in the AP):

Table I–3 VMB Argument List

(AP)+	Field Name	Description
04:	VMB$L_ FILECACHE	Quadword filename
0C:	VMB$L_LO_ PFN	PFN of first page of physical memory (always zero, regardless of where 128 Kbytes of good memory starts).
10:	VMB$L_HI_ PFN	PFN of last page of physical memory
14:	VMB$Q_ PFNMAP	Descriptor of PFN bit map. First longword contains count of bytes in bit map. Second longword contains physical address of bit map. (Same rules as for RPB$Q_PFNMAP listed previously.)
1C:	VMB$Q_ UCODE	Quadword
24:	VMB$B_ SYSTEMID	48-bit (actually a quadword is allocated) booting node address which is initialized when performing a network boot. This field is copied from the target system address parameter of the parameters message. (The DECnet HIORD value is added if the field was 2 bytes.)
2C:	VMB$L_ FLAGS	Set as needed
30:	VMB$L_CI_ HIPFN	Cluster interface high PFN
34:	VMB$Q_ NODENAME	Boot node name which is initialized when performing a network boot. This field is copied from the target system name parameter of the parameters message.
3C:	VMB$Q_ HOSTADDR	Host node address (this value is only initialized when booting over the network). This field is copied from the host system address parameter of the parameters message.
44:	VMB$Q_ HOSTNAME	Host node name (this value is only initialized when performing a network boot). This field is copied from the host system name parameter of the parameters message.
4C:	VMB$Q_TOD	Time of day (this value is only initialized when performing a network boot). The time of day is copied from the first 8 bytes of the host system time parameter of the parameters message. (The time differential values are NOT copied.)
54:	VMB$L_ XPARAM	Pointer to data retrieved from request of the parameter file
58:		The rest of the argument list is zeroed.

J
Error Messages

The error messages issued by the KA660 firmware fall into three catagories: halt code messages, VMB error messages, and console messages.

J.1 Machine Check Register Dump

Some error conditions, such as machine check, generate an error summary register dump preceeding the error message. For example, examining a nonexistent memory location results in the following display:

```
>>>ex 1ffffff

 PCSTS=0000088A  PCERR=02000003  BCSTS=01E00011  BCERR=02000000
 BCCTL=0000000E  RMESR=80441044  RMEAR=02000000 RIOEAR=00080188
  CEAR=00000000  MCDSR=3E391700  CBTCR=00004000   DSER=00000080
 QBEAR=0000000A   DEAR=00000000  IPCR0=0000
?7D MACHINE CHECK  80000011 20140544 20043870 00000000 0300D00E 0000001D

 PCSTS=0000080A  PCERR=00009A90  BCSTS=01800000  BCERR=20044BB0
 BCCTL=0000000E  RMESR=00440044  RMEAR=02000000 RIOEAR=00080188
  CEAR=00000000  MCDSR=3E391700  CBTCR=00004000   DSER=00000000
 QBEAR=0000000A   DEAR=00000000  IPCR0=0000
?7B SOFT ERROR
```

J.2 Halt Code Messages

Except on power-up, which is not treated as an error condition, the following halt messages are issued by the firmware whenever the processor halts.

For example, if the processor encounters a HALT instruction while in kernel mode, the processor halts and the firmware displays the following message before entering console I/O mode.

```
?06 HLT INST
PC = 800050D3
```

The number preceding the halt message is the halt code. This number is obtained from SAVPSL<13:8>(RESTART_CODE), IPR 43, which is saved on any processor restart operation.

Table J–1 HALT Messages

Code	Message	Description
?02	EXT HLT	External halt, caused by a console BREAK condition, a Q22-bus BHALT_L, or a DBR<AUX_HLT> bit, was set while enabled.
_03	–	Power-up, no halt message is displayed. However, the presence of the firmware banner and diagnostic countdown indicates this halt reason.
?04	ISP ERR	In attempting to push state onto the interrupt stack during an interrupt or exception, the processor discovered that the interrupt stack was mapped NO ACCESS or NOT VALID.
?05	DBL ERR	The processor attempted to report a machine check to the operating system, and a second machine check occurred.
?06	HLT INST	The processor executed a HALT instruction in kernel mode.
?07	SCB ERR3	The SCB vector had bits <1:0> equal to 3.
?08	SCB ERR2	The SCB vector had bits <1:0> equal to 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 change mode had bit <0> set.
?0C	SCB RD ERR	A hard memory error occurred while the processor was trying to read an exception or interrupt vector.
?10	MCHK AV	An access violation or an invalid translation occurred during machine check exception processing.
?11	KSP AV	An access violation or translation not valid occurred during processing of a kernel stack not valid exception.
?12	DBL ERR2	Double machine check error. A machine check occured while trying to service a machine check.
?13	DBL ERR3	Double machine check error. A machine check occured while trying to service a kernel stack not valid exception.
?19	PSL EXC5[1]	PSL<26:24> = 5 on interrupt or exception.
?1A	PSL EXC6[1]	PSL<26:24> = 6 on interrupt of exception.
?1B	PSL EXC7[1]	PSL<26:24> = 7 on interrupt or exception.
?1D	PSL REI5[1]	PSL<26:24> = 5 on an REI instruction
?1E	PSL REI6[1]	PSL<26:24> = 6 on an REI instruction.
?1F	PSL REI7[1]	PSL<26:24> = 7 on an REI instruction.

[1] For the last six cases, the VAX architecture does not allow execution on the interrupt stack while in a mode other than kernel. In the first three cases, an interrupt is attempting to run on the interrupt stack while not in kernel mode. In the last three cases, an REI instruction is attempting to return to a mode other than kernel and still run on the interrupt stack.

J.3 VMB Error Messages

The error messages listed in Table J–2 are issued by VMB.

Table J–2 VMB Error Messages

Code	Message	Description
?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	BADIRECTORY	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
?55	IVDEVNAM	Invalid device name
?56	DRVERR	Drive error

J.4 Console Error Messages

The error messages listed in Table J–3 are issued in response to a console command that has errors.

Table J–3 Console Error Messages

Code	Message	Description
?61	CORRUPTION	The console program database has been corrupted.
?62	ILLEGAL REFERENCE	Illegal reference. The requested reference would violate virtual memory protection, the address is not mapped, the reference is invalid in the specified address space, or the value is invalid in the specified destination.

Table J–3 (Cont.) Console Error Messages

Code	Message	Description
?63	ILLEGAL COMMAND	The command string cannot be parsed.
?64	INVALID DIGIT	A number has an invalid digit.
?65	LINE TOO LONG	The command was too large for the console to buffer. The message is issued only after receipt of the terminating carriage return.
?66	ILLEGAL ADRRESS	The address specified falls outside the limits of the address space.
?67	VALUE TOO LARGE	The value specified does not fit in the destination.
?68	QUALIFIER CONFLICT	Two different data sizes are specified for an EXAMINE command.
?69	UNKNOWN QUALIFIER	The switch is unrecognized.
?6A	UNKNOWN SYMBOL	The symbolic address in an EXAMINE or DEPOSIT command is unrecognized.
?6B	CHECKSUM	The command or data checksum of an X command is incorrect. If the data checksum is incorrect, this message is issued, and is not abbreviated to "Illegal command".
?6C	HALTED	The operator entered a HALT command.
?6D	FIND ERROR	A FIND command failed either to find the RPB or 128 Kbytes of good memory.
?6E	TIME OUT	During an X command, data failed to arrive in the time expected (60 seconds).
?6F	MEMORY ERROR	A machine check occurred with a code indicating a read or write memory error.
?70	UNIMPLEMENTED	Unimplemented function.
?71	NO VALUE QUALIFIER	The qualifier does not take a value.
?72	AMBIGUOUS QUALIFIER	There were not enough unique characters to determine the qualifier.
?73	VALUE QUALIFIER	The qualifier requires a value.
?74	TOO MANY QUALIFIERS	Too many qualifiers supplied for this command.
?75	TOO MANY ARGUMENTS	Too many arguments supplied for this command.
?76	AMBIGUOUS COMMAND	There were not enough unique characters to determine the command.
?77	TOO FEW ARGUMENTS	Insufficient arguments were supplied for this command.
?78	TYPEAHEAD OVERFLOW	The typeahead buffer overflowed.
?79	FRAMING ERROR	A framing error was detected on the console serial line.
?7A	OVERRUN ERROR	An overrun error was detected on the console serial line.
?7B	SOFT ERROR	A soft error occurred.

Table J–3 (Cont.) Console Error Messages

Code	Message	Description
?7C	HARD ERROR	A hard error occurred.
?7D	MACHINE CHECK	A machine check occurred.

Glossary

BFLAG

Boot flags is the longword supplied in the SET BFLAG and BOOT /R5: commands which qualify the bootstrap operation. SHOW BFLAG displays the current value.

BHALT

Q22-bus HALT signal is usually tied to the front panel halt switch.

BIP

Boot in progress flag in CPMBX<2>

CPMBX

Console program mailbox is used to pass information between operating systems and the firmware.

CQBIC

CVAX to Q22-bus interface chip

DCOK

Q22-bus signal indicating DC power is stable. This signal is usually tied to the front panel restart switch.

DNA

Digital Network Architecture

EPROM

Erasable programmable read-only memory is used on some products to store firmware. Commonly used synonyms are PROM or ROM. Erasable by using ultraviolet light.

DE

Diagnostic executive is a component of the ROM-based diagnostics responsible for set-up, execution, and clean-up of component diagnostic tests.

Firmware

Firmware in this document refers to the VAX instruction code residing in EPROM at physical address 20040000 on the KA660. The firmware consists of diagnostic, bootstrap, console, and halt entry and exit code.

GPR

General purpose registers on KA660 are the sixteen standard VAX longword registers R0 through R15. The last four registers, R12 through R15, are also known by their unique mnemonics: AP (argument pointer), FP (frame pointer), SP (stack pointer), and PC (program counter), respectively.

IPL
Interrupt priority level ranges from 0 to 31 (0 to 1F hex).

IPR
Internal processor registers on KA660 are those implemented by the CVAX chip set. These longword registers are only accessible with the instructions MTPR (Move To Processor Register) and MFPR (Move From Processor Register) and require kernel mode privileges. This document uses the prefix "PR$_" when referencing these registers.

LED
Light Emitting Diode

MSCP
Mass storage control protocol is used in Digital disks and tapes.

MOP
Maintenance operations protocol specifies message protocol for network loopback assistance, network bootstrap, and remote console functions.

Microsecond
One-millionth of a second (10e-6 seconds).

Millisecond
One-thousandth of a second (10e-3 seconds).

BBURAM
Nonvolatile RAM; on the KA660 this is 1 Kbyte of battery backed-up RAM on the SSC.

PC
Program counter or R15

PCB
Process control block is a data structure pointed to by the PR$_PCBB register and contains the current process' hardware context.

PFN
Page frame number is an index of a page (512 bytes) of local memory. A PFN is derived from the bit field <23:09> of a physical address.

PR$_ICCS
Interval clock control and status, IPR 24

PR$_IPL
Interrupt priority level, IPR 18

PR$_MAPEN
Memory management mapping enable, IPR 56

PR$_PCBB
Process control block base register, IPR 16

PR$_RXCS
Receive console status, IPR 32

PR$_RXDB
Receive data buffer, IPR 33

PR$_SAVISP
Saved interrupt stack pointer, IPR 41

PR$_SAVPC
Saved program counter, IPR 42

PR$_SAVPSL
Saved program status longword, IPR 43

PR$_SCBB
System control block base register, IPR 17

PR$_SISR
Software interrupt summary register, IPR 21

PR$_TODR
Time of day register, IPR 27; is commonly referred to as the time of year register or TOY clock.

PR$_TXCS
Transmit console status, IPR 34

PR$_TXDB
Transmit data buffer, IPR 35

PSL
Processor status longword is the VAX extension of the processor status word (PSW).

PSW
The processor status word (lower word) contains instruction condition codes and is accessible by nonprivileged users; however, the upper word contains system status information and is accessible by privileged users.

QBMBR
Q22-bus map base register found in the CQBIC determines the base address in local memory for the scatter/gather registers.

QDSS
Q22-bus video controller for workstations

QNA
Q22-bus Ethernet controller module

QMR
Q22-bus map register

RAM
Random access memory

RIP
Restart in progress flag in CPMBX<3>

RPB
Restart parameter block is a software data structure used as a communication mechanism between firmware and the operating system. Information in this block is used by the firmware to attempt an operating system (warm) restart.

SCB
System control block is a data structure pointed to by PR$_SCBB. It contains a list of longword exception and interrupt vectors.

SGEC
Second generation Ethernet chip

SHAC
Single host adapter chip

SOC
System on a chip

SP
Stack pointer or R14

SRM
Standard reference manual as in VAX SRM

SSC
System support chip

VMB
Virtual memory boot is the portion of the firmware dedicated to booting the operating system.

Index

digital