KA670 CPU Module Technical Manual

Order Number EK-KA670-TM-001

digital equipment corporation
maynard, massachusetts

First Edition, April 1990

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

Contents

Architecture

3 Central Processor and Floating Point Unit

4 Cache and Main Memory

5 The Console Line, TOY Clock, and Bus System

8 KA670 Error Handling

Firmware

9 Firmware

F Maintenance Operation Protocol (MOP) Support

G ROM Partitioning

H RAM Partitioning

I Data Structures

J Error Messages

Glossary

Index

Examples

Figures

Tables

About This Manual

The *KA670 CPU Module Technical Manual* documents the functional, physical, and environmental characteristics of the KA670 CPU module. The manual also includes information on the MS670 memory expansion modules.

There are two versions of the KA670 CPU module, KA670-AA and KA670-BA. This manual covers both versions. The KA670-BA CPU module is designed for use with workstations and servers. The KA670-BA is functionally equivalent to the KA670-AA, except that it does not support multiuser VMS and ULTRIX operating system licenses.

Audience

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.

Organization

The manual is divided into three parts.

Overview and Installation

- Chapter 1, "Overview," introduces the KA670 CPU module, the MS670 memory module, and the H3604 console module, including module features and specifications.
- Chapter 2, "Installation and Configuration," describes the procedures for installing and configuring the CPU, memory, and console modules in the Q22-bus backplanes and system enclosures.

Architecture

- Chapter 3, "Central Processor and Floating Point Unit," describes the functions of the central processing unit (P-chip) and the floating point unit (F-chip).
- Chapter 4, "Cache and Main Memory," describes the operation of the KA670 CPU module's cache memory as well as the feature of main memory.
- Chapter 5, "The Console Line, TOY Clock, and Bus System," describes the console serial line and the time-of-year clock. The chapter also provides an overview of the KA670 bus system.
- Chapter 6, "KA670 Boot and Diagnostic Facility," describes the boot and diagnostic registers, EPROM memory, battery backed-up RAM and hardware initialization.
- Chapter 7, "Interface Subsystems," describes the interfaces the KA670 CPU module uses for the Q22-bus, Ethernet, and mass storage bus.

- Chapter 8, “KA670 Error Handling,” describes unexpected KA670 system error exceptions and interrupts, from the macrocoder’s point of view.

Firmware

- Chapter 9, “Firmware,” describes the entry dispatch code, boot diagnostics, device booting sequence, console program, and console commands.

Appendices

- Appendix A, “Q22-bus Specification,” describes the low-end member of Digital’s bus family. All of Digital’s microcomputers, such as the MicroVAX 3500, MicroVAX 3600, and MicroPDP-11, use the Q22-bus.
- Appendix B, “Specifications,” describes the physical, electrical, and environmental characteristics of the KA670 CPU module.
- Appendix C, “Address Assignments,” provides a map of VAX memory space.
- Appendix D, “VAX Instruction Set,” is a list of the VAX instructions, provided for reference only.
- Appendix E, “Machine State on Power-Up,” describes the state of the KA670 after a power-up halt.
- Appendix F, “Maintenance Operation Protocol (MOP) Support,” describes the maintenance operation protocol (MOP) support features in the KA670 firmware.
- Appendix G, “ROM Partitioning,” describes the public ROM partitioning and subroutine entry points that are guaranteed to be compatible over future versions of the KA670 firmware.
- Appendix H, “RAM Partitioning,” describes how the KA670 firmware partitions the 1 kilobyte of battery backed-up RAM.
- Appendix I, “Data Structures,” describes the global data structures used by the KA670 firmware.
- Appendix J, “ Error Messages ,” provides a list of the expected responses to error conditions that may be encountered during various transactions on the KA670 module.
- The glossary defines many of the acronyms and new terms used in this manual.

Conventions

The following conventions are used in this manual:

Table 1 Conventions

Convention	Meaning
<x:y>	Represents a bit field, a set of lines, or a set of 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.
2014 0030	Eight-digit numbers in this document 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.
Return	Keys or switches that are labeled on the equipment appear in a box.
Ctrl C	For key sequences that begin with the Ctrl key, hold down Ctrl and press the second key.
Caution	Contains information to prevent damage to equipment.
Note	Contains general information.
variable	The names of variable command parameters and options appear in italics.
{}	Encloses a required part of a console command.
[ ]	Encloses an option to a console command.
...	Represents a list command elements.

Related Documents

The following documents are related to the KA670 CPU:

KA670 CPU System Maintenance Manual	EK–347AA–MG
MicroVAX Maintenance Kit	QZ–K19AA–GZ130
VAX Architecture Handbook	EB–26115–46
VAX Architecture Reference Manual	EY–3459E–DB

You can order these documents by phone or mail.

Continental USA and Puerto Rico

Call 800–258–1710 or mail to:

Digital Equipment Corporation
P.O. Box CS2008
Nashua, NH 03061

New Hampshire, Alaska, and Hawaii

Call 1-603-884-6660.

Outside the USA and Puerto Rico Mail to:

Digital Equipment Corporation
Attn: Accessories and Supplies Business Manager
c/o Local Subsidiary or Digital-Approved Distributor

Overview and Installation

- Chapter 1, Overview
- Chapter 2, Installation and Configuration

1
Overview

This chapter describes the KA670 CPU module, MS670 memory module, and H3604 console module.

1.1 KA670 CPU Module

The KA670 (Figure 1–1) is a quad-height VAX processor module for the Q22-bus. The KA670 is designed for use in high-speed, real-time applications and in multiuser, multitasking environments. The KA670 uses a cache memory to maximize performance.

Photograph of KA670 CPU module.

Figure 1–1 KA670 CPU Module

The KA670 is used in the MicroVAX 4000-300 system, which is housed in the BA440 enclosure. There are no jumpers or switches to configure. Fuses are located on the H3604 console module.

The KA670 can be configured only as an arbiter CPU on the Q22-bus, where it arbitrates bus mastership and fields bus interrupt requests and any on-board interrupt requests.

The KA670 uses a 100-pin ribbon cable to communicate with the H3604 CPU console module. The module contains configuration switches, Ethernet and DSSI connectors, and a LED display. Section 1.7 describes the H3604 module.

A single KA670 CPU module can support up to four MS670 memory modules. The KA670 and MS670 modules mount in dedicated backplane slots in the BA440 enclosure. The KA670 CPU module communicates with the MS670 memory modules across a memory interconnect located on a 270-pin backplane connector. The backplane connector also connects the subsystem with the Q22-bus and one DSSI bus. Together, the CPU and memory modules form a VAX subsystem that uses the DSSI bus to communicate with mass storage devices and the Q22-bus to communicate with I/O devices. Figure 1–2 is a block diagram of the subsystem major functions.

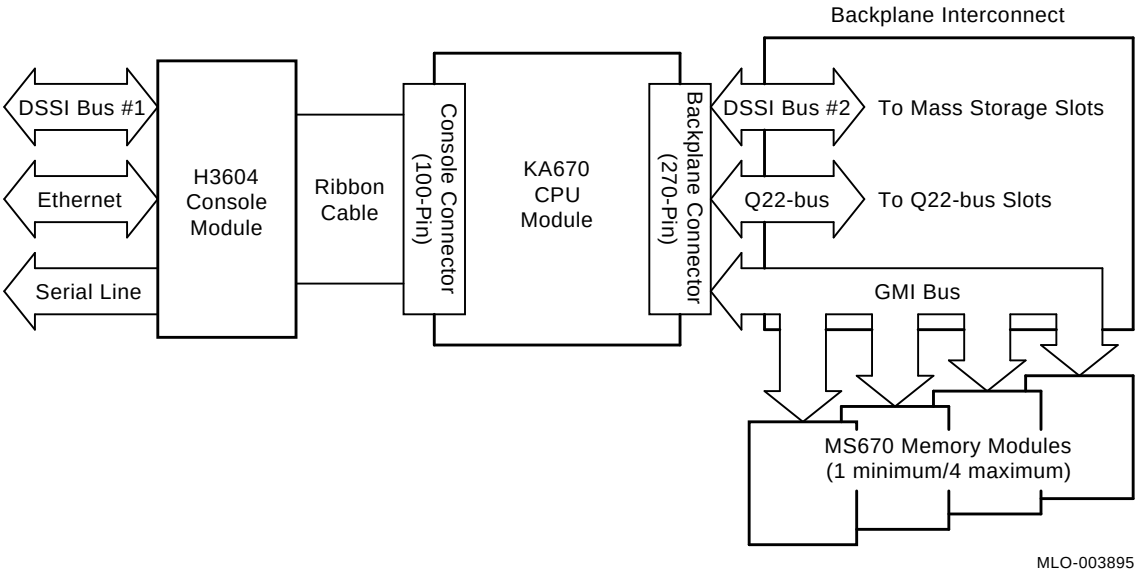


Figure 1–2 KA670 CPU Module Block Diagram

1.1.1 Module Components

The KA670 CPU is a quad-height module that mounts in a dedicated CPU backplane slot. The MS670 memory modules mount in four dedicated memory backplane slots. The CPU module is fingerless and uses a 270-pin high-density, right-angle connector to connect to the backplane.

KA670 CPU module includes the following major hardware components. Figure 1–3 shows chip locations, using the chip identification numbers.

- DC520 (P-chip): VAX central processor with a 143 MHz clock
- DC523 (F-chip): Floating point accelerator
- DC592 (C-chip): Two-level cache and its bank of associated RAM chips

- DC561 (G-chip): Main memory controller
- DC521: Clock
- DC527 (CQBIC): Q22-bus interface
- DC541 (SGEC): Ethernet interface
- DC542 (SHAC): DSSI interface chips (2)
- DC511 (SSC): System support chip
- DC509: Clock
- Two firmware ROMs: 256 kilobytes (Each is 128 kilobytes by 8.)
- 100-pin connector to the H3604 console module
- 270-pin connector to the backplane carrying signals for the Q22-bus, the DSSI bus, and the memory interconnect

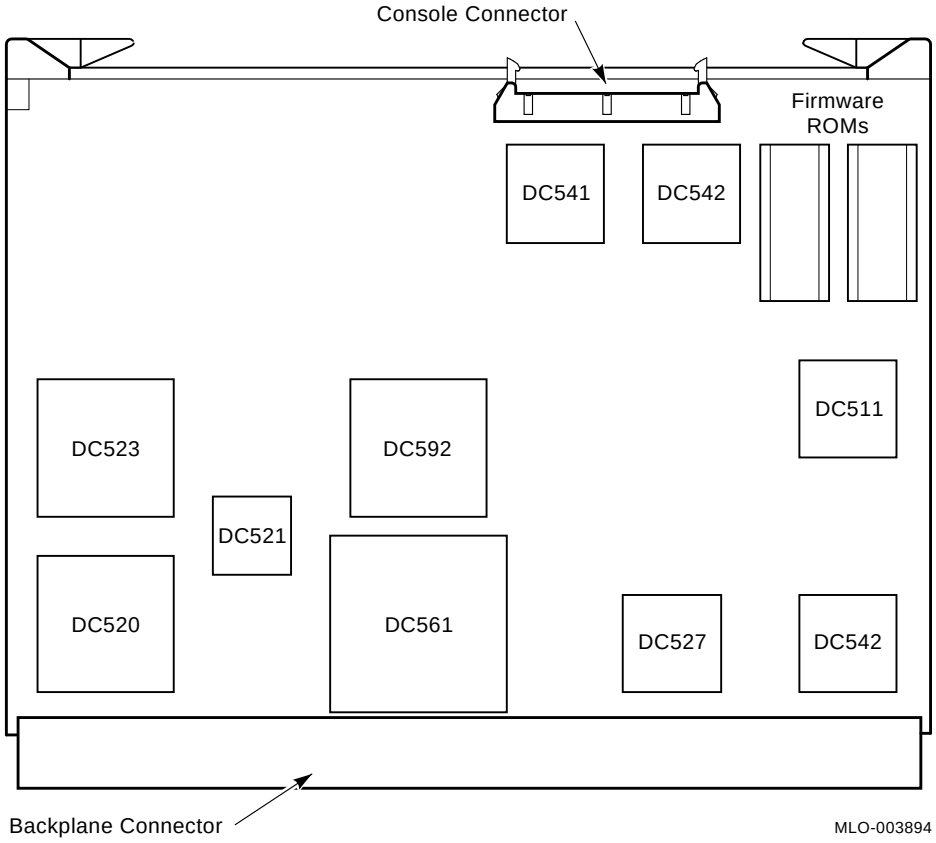


Figure 1–3 KA670 CPU Module Component Side

The KA670 CPU is designed for use in high-speed, real-time applications and in multiuser, multitasking environments. The KA670 CPU incorporates a two-level cache to maximize system performance. Estimated compute performance for the KA670-AA CPU is 8.0 times that of a VAX 11/780 system.

Functionally, the KA670-AA CPU module is divided into four major areas:

- Central processing subsystem

- System support subsystem
- I/O subsystem
- Main memory controller

1.2 Central Processing Subsystem

The central processing subsystem contains a CPU chip, a floating point accelerator (FPA) chip, the cache RAMs, and a cache controller chip.

1.2.1 Central Processing Unit (P-Chip (DC520))

The CPU chip is the heart of the KA670 module. The CPU executes the 181 instructions in the MicroVAX chip subset of the VAX instruction set. It is implemented by the CPU chip (REX520, DC520), which is in a 224-pin surface-mount package. The CPU chip achieves a 28 ns microcyle at an operating frequency of 143 Mhz. The processor also supports full VAX memory management with demand paging and a 4 gigabyte virtual address space.

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

- CMPC3
- CMPC5
- LOCC
- SCANC
- SKPC
- SPANC

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

- 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.

1.2.2 Floating Point Accelerator (F-Chip (DC523))

The floating point accelerator is implemented by the F-chip, which executes the VAX f_, d_, and g_ floating point instructions. The F-chip receives opcode information from the P-chip, and receives operands directly from memory or the P-chip. The result of the floating point is always returned to the P-chip.

The floating point accelerator executes 61 floating point instructions and 2 longword-length integer multiply instructions in the VAX base instruction group. The F-chip is in a 224-pin surface mount package.

1.2.3 The Cache

The KA670 processor module uses a two-level cache to maximize CPU performance. The first level is the *primary* cache, consisting of 2 kilobytes on the central processing chip (P-chip). The second level is the *backup* cache, consisting of 24 16K-by-4 static RAMs and a cache controller chip.

The cache controller chip is implemented with the backup cache chip, (C-chip, DC592), which is in a 224-pin surface mount package. The C-chip contains the tag store and the control logic for the backup cache RAMs, as well as a copy of the primary cache tag store to guarantee primary cache coherence between memory and processor. The chip also provides an additional bus interface for invalidate filtering, to improve performance.

1.3 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.

1.3.1 System Support Chip (SSC (DC511))

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

- Word-wide ROM unpacking
- 1 kilobyte of battery backed-up RAM
- Halt-arbitration logic
- Console serial line
- Interval timer with 10 ms interrupts
- VAX standard time-of-year clock with battery backup
- IORESET register
- Programmable CDAL bus timeout (CPU data/address lines)
- Two programmable timers
- A register to control the diagnostic LEDs

* A ceramic-body device with leads on four sides.

1.3.2 Firmware ROMs

Resident firmware ROM is on two 128 Kbyte by 8 EPROM chips. The firmware gains control when the CPU halts. The firmware contains programs that provide the following services:

- Board initialization
- Power-up self-testing of the KA670 and MS670 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

See Chapter 9 for details on KA670 firmware.

1.3.3 Boot and Diagnostic Register

The boot and diagnostic register (BDR) allows the firmware and the operating system to read KA670 configuration bits.

1.3.4 Station Address ROM

The station address ROM contains the network address of the system. This is implemented in a 32-byte by 8-bit ROM (6331).

1.4 I/O Subsystem

The I/O subsystem contains the following:

- 2 DSSI mass storage interfaces
- Ethernet interface
- Q22-bus interface

1.4.1 DSSI Mass Storage Interface (SHAC (DC542))

The two single-host adapter chips (SHAC) implement the DSSI bus interfaces. One SHAC interfaces to the KA670 system console module, while the other SHAC interfaces to the KA670 backplane. The DSSI interface allows each DSSI bus on the KA670 to transmit packets of data to, and receive packets from, up to seven other DSSI devices. These devices include the RF-series integrated storage elements (ISEs), a KFQSA module, a second KA670 module, or a KA640 module.

Each SHAC is in a 164-pin CERQUAD package. The SHAC facilitates scatter and gather mapping along with internal FIFO buffering.

The DSSI bus improves system performance, because it has a higher transfer rate than the Q22-bus and it relieves the Q22-bus of disk traffic. The DSSI bus has eight data lines, one parity line, and eight control lines. The ISEs have built-in controllers, so many functions can be handled without host or adapter intervention.

1.4.2 Ethernet Interface (SGEC (DC541))

The Ethernet interface handles communications between the CPU module and other nodes on the Ethernet. The interface is implemented with the second generation Ethernet controller chip (SGEC, DC541) on-board network interface. Used in connection with the H3604 console module, the SGEC allows the KA670 to connect to either a ThinWire or standard Ethernet. The SGEC supports the Ethernet data link layer and the CP bus parity protection. The SGEC chip is in a 84 pin package. The chip facilitates scatter and gather mapping along with dual internal FIFO buffering.

1.4.3 Q22-bus Interface (CQBIC (DC527))

The KA670 includes a Q22-bus interface that allows communication between the KA670 and other devices on the bus. It is implemented with the CP bus to Q22-bus asynchronous adapter chip (CQBIC, DC527). The CQBIC is in a 132-pin CERQUAD surface mount package. The KA670 does not provide Q22-bus termination. The backplane provides the termination resistors. The Q22-bus interface supports the following functions:

- Programmable and direct mapping functions
- Masked and unmasked longword reads and writes from CPU to the Q22-bus memory and I/O space and to the interface registers
- Up to 16-word, block mode writes from Q22-bus to main memory
- Up to 2-word, block mode transfers between the CPU and Q22-bus devices
- Transfers from CPU to local Q22-bus memory space

1.5 Memory Support Subsystem

This subsystem provides support for the KA670 memory subsystem. The memory support subsystem contains a memory controller, a bus adapter, and a G-chip interface.

1.5.1 Memory Controller/Bus Adapter (G-Chip (DC561))

The memory controller and bus adapter are implemented by the memory controller chip (G-chip, DC561). The G-chip is a dual-ported ECC memory controller and a bus adapter. As a memory controller, the G-chip controls transactions between the GMI, RDAL bus, and the CP bus. In addition, the G-chip is responsible for assisting with maintaining primary and backup cache coherency with the memory system.

The G-chip controls communication among the P-chip, the CQBIC, and the SGEC and SHAC chips. The G-chip controls and passes data to or from one, two, three, or four buffered memory modules.

As a bus adapter, the G-chip controls transactions between the higher performance RDAL bus and the lower performance CP bus. The CP bus port to the G-chip provides a peripheral bus for direct memory access (DMA) by peripheral functions. The CP bus is a peripheral bus on the KA670 and does not support the P-chip on this system.

The G-chip is in a 332-pin, high-performance tape package (HPTP). The tape package is a surface mountable chip carrier with 12.5 mil lead spacing.

1.6 MS670 Memory Module

The MS670–BA is a 32 Mbyte, double-sided board, with an access time of 100 ns in a 39-bit-wide array (32 bits of data and 7 bits of error correction code) implemented with a 1 Mbyte dynamic RAM in SOJ surface mount packages.

The module mounts in a dedicated memory backplane slot. The module is fingerless and uses a 150-pin, high-density, right-angle connector to connect to the backplane. Figure 1–4 is a photograph of the MS670 memory module.

Figure 1–4 MS670 Memory Module

1.7 H3604 Console Module

The H3604 console module (Figure 1–5) allows the KA670 CPU module to interface to a serial line console device, a DSSI bus, and the Ethernet. The H3604 is wide enough to cover the five slots dedicated to the KA670 and its four MS670 modules. Five adhesive tags are included for the user to name the modules in the respective slots.

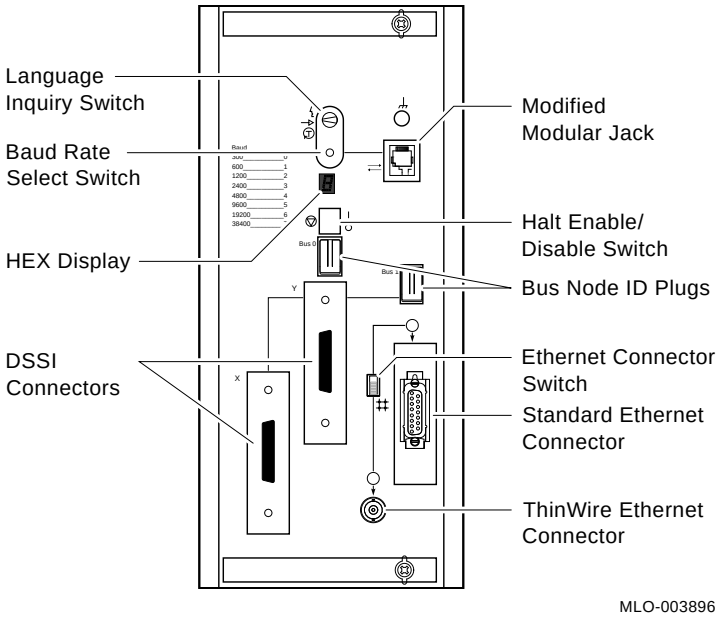


Figure 1–5 H3604 Console Module (Front View)

The H3604 module contains the following connectors to allow CPU communication:

- A console serial line (with baud rate switch)
- Two Ethernet connectors (with switch)
- Two 50-ping DSSI connectors that allow daisy-chaining of one DSSI bus, terminators for both DSSI connectors, and two bus node ID plugs

The H3604 module also has four feature selection switches:

- Baud rate select switch for the serial console line
- Power-up mode switch
- Break enable/disable switch from the console keyboard Break key (default) or Ctrl P, depending on the state of SSCCR <15>. If used, Ctrl P must be reset after each halt action. If this switch is set to the enable position (1), the system does not autoboot on power-up. Instead, the system enters console I/O mode and displays the >>> prompt.
- Ethernet connector switch to selects the following:
  - A 15-conductor connector for a standard Ethernet cable
  - A male BNC connector for a ThinWire Ethernet coaxial cable

  LEDs indicate the selected connector and valid +12 Vdc for that connector.

In addition, the H3604 module contains the following features:

- Console serial line drivers and receivers
- Hexidecimal display
- Battery charger and low voltage detection
- 25.6 kHz TOY clock oscillator
- -9 V dc/dc converter
- Ethernet serial transceiver chip (SIA)
- Fused current surge protection

Inside the door of the H3604 module are a DSSI circuit fuse and two jumpers. The fuse prevents shorts from the accidental grounding of the DSSI cable power pin. The jumpers must be in place to give the bus node number 7 to both of the SHAC DSSI bus controllers on the CPU board. (The two DSSI buses are separate.)

There are two connectors from the H3604 module to the internal BA440. One is a 4-pin power connection to a small printed circuit card that inserts next to the KA670 CPU in the backplane. The other is the 100-pin connector to the KA670 CPU module.

2
Installation and Configuration

This chapter describes how to install the KA670 in a system. The chapter discusses the following topics:

- Installing the KA670 and MS670 modules
- Configuring the KA670
- KA670 connectors

2.1 Installing the KA670 and MS670 Memory Modules

NOTE
You can use the KA670 and MS670 modules only in BA440 system enclosures that use high-density backplane connector slots.

The KA670 CPU module and the MS670 memory modules must be installed in the five rightmost backplane slots. Note that the KA670 module installs in backplane slot J5, and the memory modules install in slots J4 through J1.

To install the KA670 and MS670 modules:

1. Install the KA670 CPU in slot J5 of the Q22-bus/CD backplane.
2. Install MS670 memory modules in slots J4 through J1 next to to the KA670 CPU.
   - If you only use one memory module , you can install it in any of the slots J4 through J1.
   - If you use more than one memory module, you must install the first memory module in J4, the second in J3, and so on. Do not leave a gap between memory modules.
3. Install a 100-pin ribbon cable between the KA670 CPU and the console module.

Figure 2–1 shows the positions of the KA670 CPU and the memory modules in the backplane.

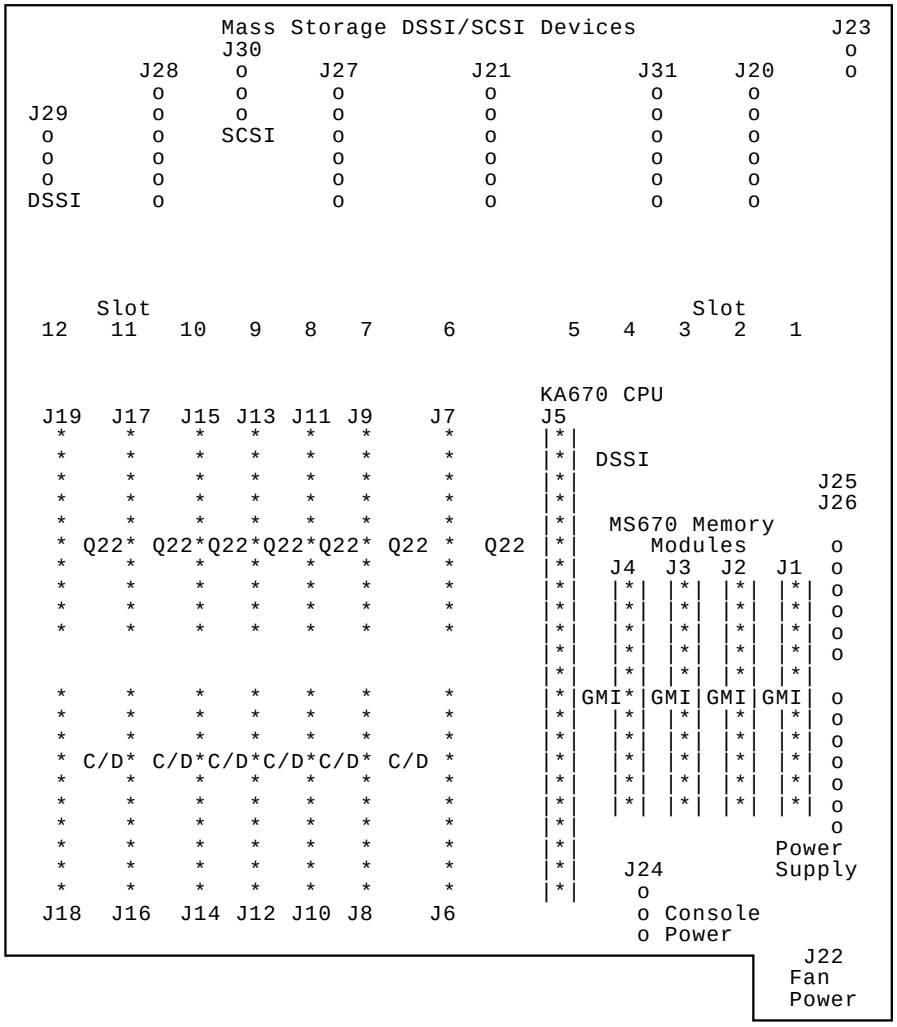


Figure 2–1 Backplane

2.2 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 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 *KA670 CPU System Maintenance Manual* for switch settings and CSR and interrupt vector jumper settings for supported options.

2.3 Mass Storage Configuration

There is space for four mass storage devices—either three integrated storage elements (ISEs) and one TK70 tape drive, 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 they 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 2.3.3 describes the procedure for accessing DUP through the VMS operating system.

2.3.1 Changing the Node Name

Each ISE has a node name that is maintained in EPROM onboard 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 EEPROM.
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

2.3.2 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 EEPROM.

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

2.3.3 Accessing RF-series Firmware in VMS, Through DUP

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

Use the VMS operating system to access the ISE frmware if you want to look up or view parameter settings, but not change them. To change ISE parameter settings, enter the ISE frmware 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 by using the following VMS command:

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

2.3.3.1 Allocation Class

When a KA670 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 nonzero 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..
>>>
```

2.4 DSSI Cabling, Device Identity, and Bus Termination

The ISEs in one particular BA440 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.

There are two DSSI ports in the KA670 system. One DSSI port is routed along the backplane and exits the enclosure at the left edge, from a connector near the ISE slots. The other DSSI port is configured by means of the DSSI connector on the H3604 panel. If unused, DSSI connectors must be terminated.

There is no terminator on the KA670. The near-end termination is contained on the backplane for the internal DSSI bus, and provided by the pluggable connectors for the external bus.

All DSSI devices on the same bus must have unique identifiers. On the face of the H3604 console module, you can see the two DSSI bus ID plugs (Figure 1–5). These ID plugs provide an identity for each DSSI bus. Because the DSSI buses are separate, the two ID plugs may be identical.

2.5 KA670 Connectors

The KA670 CPU module uses two connectors, J1 and J2. J1 is a 270-pin connector that mates with the backplane. J2 is the connector for the 100-pin ribbon cable that goes to the console module. Users configure the KA670 through the H3604 console module. Figure 1–3 shows the location of the connectors on the KA670 module.

Architecture

- Chapter 3, Central Processor and Floating Point Unit
- Chapter 4, Cache and Main Memory
- Chapter 5, The Console Line, TOY Clock, and Bus System
- Chapter 6, KA670 Boot and Diagnostic Facility
- Chapter 7, Interface Subsystems
- Chapter 8, KA670 Error Handling

3
Central Processor and Floating Point Unit

This chapter describes the functions of the central processing unit (P-chip) and the floating point unit (F-chip).

3.1 Central Processor

The central processor of the KA670 supports the MicroVAX chip subset (plus six additional string instructions) of the VAX instruction set and data types, as well as full VAX memory management. The central processor is implemented with a single VLSI chip called the P-chip (REX520).

3.1.1 Processor State

The *processor state* is that portion of the state of a process which is stored in processor registers rather than in memory. The processor state is composed of 16 general-purpose registers (GPRs), the processor status longword (PSL), and the internal processor registers (IPRs).

Nonprivileged software can access the GPRs and the processor status word (bits <15:00> of the PSL). Only privileged software can access the IPRs and bits <31:16> of the PSL. 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.

3.1.1.1 General-Purpose Registers

The KA670 implements 16 general-purpose registers as specified in the *VAX Architecture Reference Manual*. These registers are used for temporary storage, accumulators, and as base and index registers for addressing. The general-purpose registers are R0 to R15. The bits of a register are numbered from the right, <0> to <31>. Figure 3–1 shows the format of a general-purpose register. Table 3–1 describes the registers.

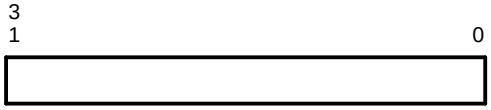


Figure 3–1 General-Purpose Register

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

Table 3–1 General-Purpose Register Descriptions

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*. The AP contains the address of the base of this data structure.

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

3.1.1.2 Processor Status Longword

The KA670 processor status longword (PSL) is implemented as specified in the *VAX Architecture Reference Manual* . See that manual for a detailed description of this register's operation.

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. Nonprivileged software can access bits <15:00>, but only privileged software can access bits <31:16>. Processor initialization sets the PSL to 041F 0000_{16}. Figure 3–2 shows the format of the processor status longword. Table 3–2 lists the bits and definitions.

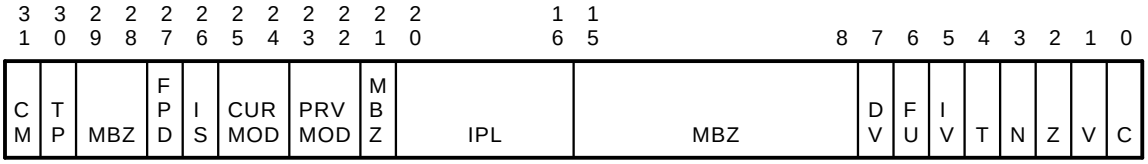


Figure 3–2 Processor Status Longword

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.

Table 3–2 explains the properties of each internal process register:

Table 3–2 Internal Process Register Descriptions

PSL Data Bit	Name	Definition
<31>	CM	Compatibility mode. This bit always reads as zero, loading a one into this bit is a NOP.
<30>	TP	Trace pending
<29:28>	MBZ	Must be written as zero
<27>	FPD	First part done
<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 KA670 hardware; the bit can be used by macrocode that 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

3.1.1.3 Internal Processor Registers

The KA670 internal processor registers (IPRs) can be accessed by using the MFPR and MTPR privileged instructions. Each IPR falls into one of the following five categories:

1. Implemented by KA670 as specified in the *VAX Architecture Reference Manual* .
2. P-chip implementation that is unique or different from that specified in the *VAX Architecture Reference Manual*.
3. Not implemented by KA670. Read as zero, NOP on writes.
4. Not implemented by KA670. Access causes a reserved operand fault.
5. Not fully implemented by KA670. Access causes unpredictable results.

Table 3–3 provides information on each IPR.

There are different categories of IPRs. Section 3.1.1.3.1 lists category 1 IPRs and the section where they are described. Section 3.1.1.3.2 lists category 2 IPRs and the section where they are described.

Table 3–3 KA670 Internal Processor Registers

Decimal	Hex.	Register Name	Mnemonic	Type	Scope	Impl.	Init?	Category
0	0	Kernel stack pointer	KSP	RW	PROC	REX520		1
1	1	Executive stack pointer	ESP	RW	PROC	REX520		1
2	2	Supervisor stack pointer	SSP	RW	PROC	REX520		1
3	3	User stack pointer	USP	RW	PROC	REX520		1
4	4	Interrupt stack pointer	ISP	RW	CPU	REX520		1
5–7	5–7	Reserved						3
8	8	P0 base register	P0BR	RW	PROC	REX520		1
9	9	P0 length register	P0LR	RW	PROC	REX520		1
10	A	P1 base register	P1BR	RW	PROC	REX520		1
11	B	1 length register	P1LR	RW	PROC	REX520		1
12	C	System base register	SBR	RW	CPU	REX520		1
13	D	System length register	SLR	RW	CPU	REX520		1
14–15	E–F	Reserved						
16	10	Process control block base	PCBB	RW	PROC	REX520		1
17	11	System control block base	SCBB	RW	CPU	REX520		1
18	12	Interrupt priority level	IPL	RW	CPU	REX520	Yes	1
19	13	AST level	ASTLVL	RW	PROC	REX520	Yes	1
20	14	Software interrupt request register	SIRR	W	CPU	REX520		1

Type

R –Read-only register
W –Write-only register
RW–Read/write register

Scope –Processor register's scope

CPU –CPU-wide register
PROC–Per-process register

Impl. –Chip in which the processor register is implemented.

REX520 –REX520 chip (P-chip)
SSC –System support chip
C-chip –C-chip

Init? –Initialized on module RESET (power-up, or negation of DCOK)

Category–Processor register category

Table 3–3 (Cont.) KA670 Internal Processor Registers

Decimal	Hex.	Register Name	Mnemonic	Type	Scope	Impl.	Init?	Category
21	15	Software interrupt summary register	SISR	RW	CPU	REX520	Yes	1
22–23	16–17	Reserved						3
24	18	Interval counter control status	ICCS	RW	CPU	REX520		2
25–26	19–1A	Reserved						3
27	1B	Time-of-year register	TODR	RW	CPU	SSC		1
28	1C	Console storage receiver status	CSRS	RW	CPU	SSC	Yes	5
29	1D	Console storage receiver data	CSRD	R	CPU	SSC	Yes	5
30	1E	Console storage transmitter status	CSTS	RW	CPU	SSC	Yes	5
31	1F	Console storage transmitter data	CSTD	W	CPU	SSC	Yes	5
32	20	Console receiver control/status	RXCS	RW	CPU	SSC	Yes	2
33	21	Console receiver data buffer	RXDB	R	CPU	SSC	Yes	2
34	22	Console transfer control/status	TXCS	RW	CPU	SSC	Yes	2
35	23	Console transfer data buffer	TXDB	W	CPU	SSC	Yes	2
36–37	24–25	Reserved						3
38	26	Machine check error register	MCESR	W	CPU	REX520		2
39	27	Reserved						3
40	28	Accelerator control and status register	ACCS	RW	CPU	REX520	Yes	2

Type

R –Read-only register
W –Write-only register
RW–Read/write register

Scope –Processor register's scope

CPU –CPU-wide register
PROC–Per-process register

Impl. –Chip in which the processor register is implemented.

REX520 –REX520 chip (P-chip)
SSC –System support chip
C-chip –C-chip

Init? –Initialized on module RESET (power-up, or negation of DCOK)

Category–Processor register category

Table 3–3 (Cont.) KA670 Internal Processor Registers

Decimal	Hex.	Register Name	Mnemonic	Type	Scope	Impl.	Init?	Category
41	29	Reserved						3
42	2A	Console saved PC	SAVPC	R	CPU	REX520		2
43	2B	Console saved PSL	SAVPSL	R	CPU	REX520		2
44–46	2C–2E	Reserved						3
47	2F	Translation buffer tag	TBTAG	W	CPU	REX520		2
48–54	30–36	Reserved						3
55	37	I/O system reset register	IORESET	W	CPU	SSC		2
56	38	Memory management enable	MAPEN	RW	CPU	REX520	Yes	1
57	39	Translation buffer invalidate all	TBIA	W	CPU	REX520		1
58	3A	Translation buffer invalidate single	TBIS	W	CPU	REX520		1
59	3B	Translation buffer data	TBDATA	W	CPU	REX520		2
60–61	3C–3D	Reserved						3
62	3E	System identification	SID	R	CPU	REX520		1
63	3F	Translation buffer check	TBCHK	W	CPU	REX520		1
64–111	40–6F	Reserved						3
112	70	Backup cache reserved register	BC112	RW	CPU	C-chip		5
113	71	Backup cache tag store	BCBTS	RW	CPU	C-Chip		2
114	72	Backup cache P1 tag store	BCP1TS	RW	CPU	C-Chip		2

Type

R –Read-only register
W –Write-only register
RW–Read/write register

Scope –Processor register's scope

CPU –CPU-wide register
PROC–Per-process register

Impl. –Chip in which the processor register is implemented.

REX520 –REX520 chip (P-chip)
SSC –System support chip
C-chip –C-chip

Init? –Initialized on module RESET (power-up, or negation of DCOK)

Category–Processor register category

Table 3–3 (Cont.) KA670 Internal Processor Registers

Decimal	Hex.	Register Name	Mnemonic	Type	Scope	Impl.	Init?	Category
115	73	Backup cache P2 tag store	BCP2TS	RW	CPU	C-Chip		2
116	74	Backup cache refresh register	BCRFR	RW	CPU	C-Chip		2
117	75	Backup cache index register	BCIDX	RW	CPU	C-Chip		2
118	76	Backup cache status register	BCSTS	RW	CPU	C-Chip	Yes	2
119	77	Backup cache control register	BCCTL	RW	CPU	C-Chip	Yes	2
120	78	Backup cache error register	BCERR	R	CPU	C-Chip		2
121	79	Backup cache flush backup tag store	BCFBTS	W	CPU	C-Chip		2
122	7A	Backup cache flush primary tag store	BCFPTS	W	CPU	C-Chip		2
123	7B	Vector interface error status register	VINTSR	RW	CPU	C-Chip		2
124	7C	Primary cache tag store	PCTAG	RW	CPU	REX520		2
125	7D	Primary cache index register	PCIDX	RW	CPU	REX520		2
126	7E	Primary cache error address register	PCERR	RW	CPU	REX520		2
127	7F	Primary cache status register	PCSTS	RW	CPU	REX520	Yes	2
128–255	80–FF	Reserved						3
>255	>FF	Reserved						4

Type

R –Read-only register
W –Write-only register
RW–Read/write register

Scope –Processor register's scope

CPU –CPU-wide register
PROC–Per-process register

Impl. –Chip in which the processor register is implemented.

REX520 –REX520 chip (P-chip)
SSC –System support chip
C-chip –C-chip

Init? –Initialized on module RESET (power-up, or negation of DCOK)

Category–Processor register category

ACCESS TO CATEGORY 3 REGISTERS

Category 3 processor registers in the previous table are passed to the RDAL by the P-chip. Since these registers are not implemented by the KA670 module, the SSC terminates the EPR read or write transaction after the period specified by the SSC bus timeout control register.

During this time, the CPU does not execute any other instructions, and no other DAL transactions are possible. Therefore, category 3 processor registers should not be referenced during normal system operation, as this may cause device or CPU timeouts to occur.

3.1.1.3.1 KA670 VAX Standard Internal Processor Registers

Internal Processor Registers (IPRs) that are implemented as specified in the *VAX Architecture Reference Manual* are classified as category 1 IPRs. See the *VAX Architecture Reference Manual* for details on the operation and use of these registers.

The category 1 registers listed in Table 3–4 are also referenced in other sections of this manual:

Table 3–4 Category 1 Internal Processor Registers

Number				
Decimal	**Hex**	**Register Name**	**Mnemonic**	**Section**
18	12	Interrupt priority level	IPL	3.1.6.1
20	14	Software interrupt request	SIRR	3.1.6.1
21	15	Software interrupt summary	SISR	3.1.6.1
27	1B	Time-of-year clock	TODR	Section 5.2
56	38	Memory management enable	MAPEN	3.1.5.2
57	39	Translation buffer invalidate all	TBIA	3.1.5.2
58	3A	Translation buffer invalidate single	TBIS	3.1.5.2
62	3E	System identification	SID	Section 3.1.7
63	3F	Translation buffer check	TBCHK	3.1.5.2

3.1.1.3.2 KA670 Unique Internal Processor Registers

Internal processor registers (IPRs) that are implemented uniquely on the KA670 are classified as category two IPRs. For example, category 2 IPRs are not contained in, or do not fully conform to, the *VAX Architecture Reference Manual*. Category 2 IPRs are described in detail in this manual. See the sections listed in Table 3–5 for a description of these registers:

Table 3–5 Category 2 Internal Processor Registers

Number				
Decimal	**Hex**	**Register Name**	**Mnemonic**	**Section**
24	18	Interval clock control/status	ICCS	5.2.2
32	20	Console receiver control/Status	RXCS	5.1.1.1
33	21	Console receiver data buffer	RXDB	5.1.1.2
34	22	Console transmit control/status	TXCS	5.1.1.3
35	23	Console transmit data buffer	TXDB	5.1.1.4
38	26	Machine check error register	MCESR	3.1.6.4
40	28	Accelerator control and status	ACCS	3.1.8
42	2A	Console saved PC	SAVPC	3.1.6.6
43	2B	Console saved PSL	SAVPSL	3.1.6.6
47	2F	Translation buffer tag	TBTAG	3.1.5.2
55	37	I/O system reset register	IORESET	6.5.3.1
59	3B	Translation buffer data	TBDATA	3.1.5.2
113	71	Backup cache tag store	BCBTS	4.1.3.5.1
114	72	Backup cache P1 tag store	BCP1TS	4.1.3.5.2
115	73	Backup cache P2 tag store	BCP2TS	4.1.3.5.2
116	74	Backup cache refresh register	BCRFR	4.1.3.5.3
117	75	Backup cache index register	BCIDX	4.1.3.5.4
118	76	Backup cache status register	BCSTS	4.1.3.5.5
119	77	Backup cache control register	BCCTL	4.1.3.5.6
120	78	Backup cache error register	BCERR	4.1.3.6.1
121	79	Backup cache flush backup tag store	BCFBTS	4.1.3.6.2
122	7A	Backup cache flush primary tag store	BCFPTS	4.1.3.6.3
123	7B	Vector interface error status register	VINTSR	-
124	7C	Primary cache tag store	PCTAG	4.1.2.5.4
125	7D	Primary cache index register	PCIDX	4.1.2.5.3
126	7E	Primary cache error address register	PCERR	4.1.2.5.2
127	7F	Primary cache status register	PCSTS	4.1.2.5.1

3.1.2 Process Structure

A *process* is a single thread of execution. The context of the current process is contained in the process control block (PCB), which is pointed to by the process control block base register (PCBB). The KA670 implements these structures as defined in the *VAX Architecture Reference Manual*. See that manual for a description of the PCB and the PCBB.

3.1.3 Data Types

The KA670 CPU supports the following subset of the VAX data types:

- 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 by macrocode emulation.

3.1.4 Instruction Set

The KA670 CPU implements the following subset of the VAX instruction set types in microcode:

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

The P-chip (REX520) 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

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

- CRC
- EDITPC

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

- Octaword
- Compatibility mode instructions

Appendix D lists the entire KA670 instruction set. The appendix indicating which instructions are implemented in the floating point accelerator (FPA) and which instructions have microcode assists to speed up macrocode emulation.

3.1.5 Memory Management

The KA670 implements VAX Memory Management in full, as defined in the *VAX Architecture Reference Manual*. System space addresses are virtually mapped through single-level page tables, and process space addresses are virtually mapped through two-level page tables. See the *VAX Architecture Reference Manual* for descriptions of the virtual-to-physical address translation process, and the format for VAX page table entries (PTEs).

3.1.5.1 Translation Buffer

To reduce the overhead associated with translating virtual addresses to physical addresses, the P-chip employs a 64-entry, fully associative, translation buffer for caching VAX PTEs. Each entry can store a 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 to 8).
- IPR 57 (TBIA) or IPR 58 (TBIS) is written to.

Each entry is divided into two parts—a 24-bit tag register and a 27-bit PTE register. The tag register stores the virtual page number (VPN) of the virtual page that the corresponding PTE Register maps, and a valid bit (TB.V) that indicates the tag contains a valid VPN. The PTE register stores the 21-bit page frame number (PFN) field, the PTE.V bit, the PTE.M bit, and the 4-bit PROT field from the corresponding VAX PTE.

During virtual-to-physical address translation, the contents of the 64 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 and the TB.V bit indicates the entry is valid, then a translation buffer “hit” has occurred. 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 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.

3.1.5.2 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 or disabled through IPR 56 (MAPEN). Writing a 0 to this register with a MTPR instruction disables memory management. Writing a 1 to this register with a MTPR instruction enables memory management. Writes to this register flush the translation buffer. To determine whether or not memory management is enabled, IPR 56 is read using the MFPR instruction.

Translation buffer entries that map a particular virtual address can be invalidated by writing the virtual address to IPR 58 (TBIS), using the MTPR instruction. *Whenever software changes (1) a valid page table entry for the system or current process region, or (2) 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 using the MTPR instruction to write a virtual address within that page to IPR 63 (TBCHK) . If the translation buffer contains a valid translation for the page, the condition code V bit (bit<1> of the PSL) is set. The TBIS, TBIA, and TBCHK IPRs are write only. The operation of an MFPR instruction from any of these registers is undefined.

There are three pairs of base and length registers that specify the base and length of the P0, P1, and S0 spaces:

- IPR 8 (P0BR) and IPR 9 (P0LR)
- IPR 10 (P1BR) and IPR 11 (P1LR)
- IPR 12 (SBR) and IPR 13 (SLR)

The base and length of the P0, P1, and S0 page tables may be changed by writing the appropriate address or length to any of the following registers:

IPR 8 (P0BR)
IPR 9 (P0LR)
IPR 10 (P1BR)
IPR 11 (P1LR)
IPR 12 (SBR)
IPR 13 (SLR)

Whenever the location or size of the system map is changed by changing the SBR (IPR 12) or SLR (IPR 13), the entire translation buffer must be cleared. The P-chip accomplishes this by flushing the TB on any change to SBR and SLR, or to P0BR, P1BR, P0LR, and P1LR.

When a process context is loaded with the LDPCTX instruction, all TB entries that map process-space pages are automatically cleared. System-space mappings are preserved.

Two IPRs are used by diagnostic software to test the translation buffer:

IPR 47 (TBTAG)(Format shown in Figure 3–3.)
IPR 59 (TBDATA)(Format shown in Figure 3–4.)

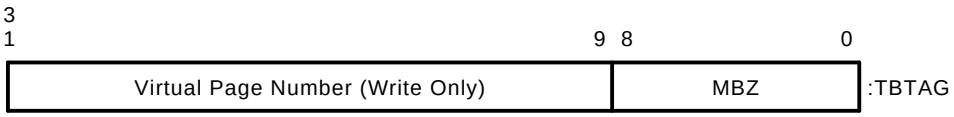


Figure 3–3 Translation Buffer Tag (TBTAG)—(IPR 47_{10} $2F_{16}$)

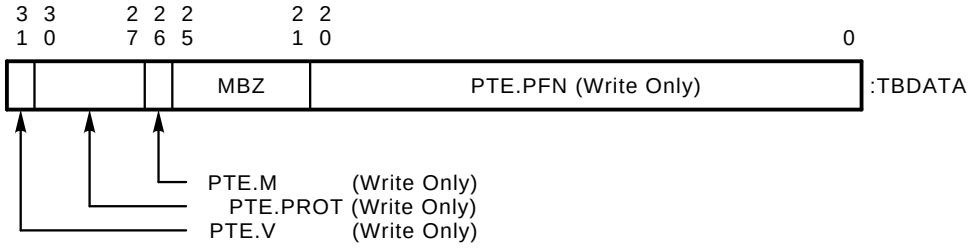


Figure 3–4 Translation Buffer Data (TBDATA)—(IPR 59_{10} $3B_{16}$)

Diagnostic software may use IPR 47 (TBTAG) and IPR 59 (TBDATA) to test the operation of the translation buffer. A write to TBTAG writes bits <31:9> of the source data into the VPN field of the current tag location and clears the TB.V bit. A subsequent write to TBDATA interprets the source data as a PTE; writes PTE.V, PTE.M, PTE.PROT, and PTE.PFN into the current PTE location; sets the TB.V bit; and increments the NLU pointer.

These registers are provided for diagnostic purposes only and should not be written during normal operation. Writes to these registers must be done under very controlled conditions to achieve the desired results. Specifically, the following restrictions apply:

- The NLU pointer must be in a known state. A TBIA will initialize the NLU pointer to the first location in the array.
- Memory management must be enabled during the use of TBTAG and TBDATA, because writing to MAPEN implicitly does a TBIA and resets the NLU pointer.
- Data- and instruction-stream references during the use of TBTAG and TBDATA must not be allowed to change the NLU pointer.

NOTE
The TBIS, TBIA, TBCHK, TBTAG, and TBDATA IPRs are write only. An MFPR instruction used to read any of these registers will cause the P-chip (REX520) to initiate a reserved operand fault.

3.1.6 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).

3.1.6.1 Interrupts

Interrupts can be divided into two classes: nonmaskable and maskable. For more information on error recovery and analysis, see Chapter 8.

Nonmaskable interrupts cause a halt through 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. Nonmaskable interrupts cannot be blocked by raising the processor IPL, but can be blocked by running out of the halt protected address space. The exception is nonmaskable interrupts that generate a halt code of 3. Nonmaskable 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 system control block (SCB).

Table 3–6 lists the various interrupt conditions for the KA670, along with their associated priority levels and SCB offsets.

Table 3–6 Interrupt Priority Levels

Priority Level	Interrupt Condition	SCB Offset
Nonmaskable	BDCOK and BPOK negated, then asserted on Q22-bus (power up)	*
	BDCOK negated, then asserted while BPOK asserted on Q22-bus (power up)	†
	BHALT asserted on Q22-bus	†
	BREAK generated by the console device	†
1F	Unused	
1E	BPOK negated on Q22-bus	0C
1D	Uncorrectable main memory errors (MASKED writes only)	60
	Main memory NXM errors on writes	60
	RDAL data parity errors on writes	60
	CP bus NXM/TIMEOUT on a write	60
	Q22-bus NXM/NOSACK on a write	60

*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 3–6 (Cont.) Interrupt Priority Levels

Priority Level	Interrupt Condition	SCB Offset
	Q22-bus NOGRANT on a write	60
1C:1B	Unused	
1A	Correctable main memory errors	54
	Uncorrectable main memory errors (I-Stream)	54
	RDAL data parity errors on I-stream or nonrequest D-stream	54
	Primary cache tag parity errors (writes or I-Stream)	54
	Primary cache data parity errors (I-Stream)	54
	CP Bus NXM/TIMEOUTS errors (I-Stream)	54
19:18	Unused	
17	BR7 L asserted	Q-bus vector plus 200_{16}
16	Interval timer interrupt	C0
	BR6 L asserted	Q-bus vector plus 200_{16}
15	BR5 L asserted	Q-bus vector plus 200_{16}
14	Console terminal	F8,FC
	Programmable timers	78,7C
	Mass storage interface 1 (DSSI port 1) (external)	108
	Mass storage interface 2 (DSSI port 2) (internal)	104
	Network interface	10C
	Interprocessor doorbell	204
	BR4 L asserted	Q-bus vector plus 200_{16}
13:10	Unused	
0F:01	Software interrupt requests	84-BC

NOTE

Because the Q22-bus does not allow differentiation between the four bus grant levels (for example, a level 7 device could respond to a level 4 bus grant), the KA670 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 KA670 maintains the IPL at the priority of the interrupt for all other interrupts.

The interrupt system is controlled by three IPRs:

- IPR 18, the interrupt priority level register (IPLR) (Figure 3–5)
  Used for loading the processor priority field in the PSL (bits<20:16>).
- IPR 20, the software interrupt request register (SIRR) (Figure 3–6)
  Used for creating software interrupt requests.
- IPR 21, the software interrupt summary register (SISR) (Figure 3–7)
  Records pending software interrupt requests at levels 1 to 15.

See the *VAX Architecture Reference Manual* for more information on these registers.

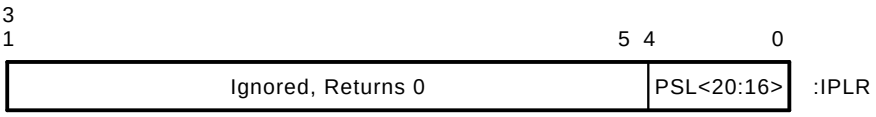


Figure 3–5 Interrupt Priority Level Register (IPLR)— (IPR 18_{10} 12_{16})

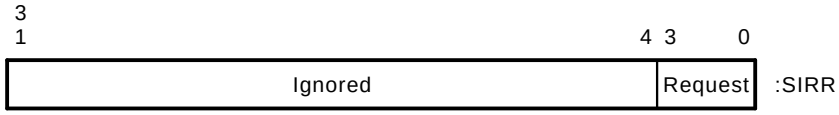


Figure 3–6 Software Interrupt Request Register (SISR)— (IPL 20_{10} 14_{16})

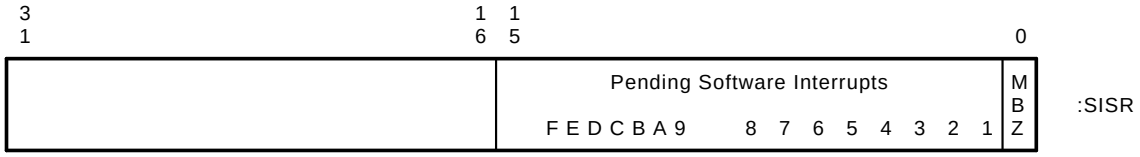


Figure 3–7 Software Interrupt Summary Register (SISR)— (IPL 21_{10} 15_{16})

3.1.6.2 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 normally would have been executed, and the instruction can be restarted.

A *fault* is an exception that occurs during an instruction. A fault leaves the registers and memory in a consistent state, so eliminating the fault condition and restarting the instruction gives 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**. That is, 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. The aborted instruction may or may not be the instruction that caused the abort. 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. Table 3–7 lists exceptions by class. All the exceptions listed (except machine check) are described in greater detail in the *VAX Architecture Reference Manual*.

Table 3–7 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 fault or abort		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
Serious System Failure Exceptions		
Console error halt	Abort	*
Interrupt stack not valid	Abort	*
Kernel stack not valid	Abort	08
Machine checks consisting of the following:	Abort	04
P-cache tag and data parity errors (D-stream reads) B-cache data parity errors (D-stream reads) RDAL data parity errors (nonrequested bytes only) Main memory uncorrectable errors (D-stream) Main memory read NXM errors CP bus read parity errors Q22-bus NXM/NOSACK errors (D-stream reads) Q22-bus read device parity errors Q22-bus read NO GRANT errors CP bus timeout/NXM read errors		

*Dispatched by resident firmware rather than through the SCB.

Exceptions save the PC and PSL. In some cases, exceptions also save one or more parameters on the stack. Most exceptions do not change the IPL of the processor or cause the exception to be sent to the appropriate service routine through the SCB.

However, exceptions in the *Serious System Failure* class set the processor IPL to 1F. The *interrupt stack not valid* exception and exceptions that occur while an interrupt or another exception is being serviced are sent to the appropriate service routine by the resident firmware.

3.1.6.3 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 3–8:

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

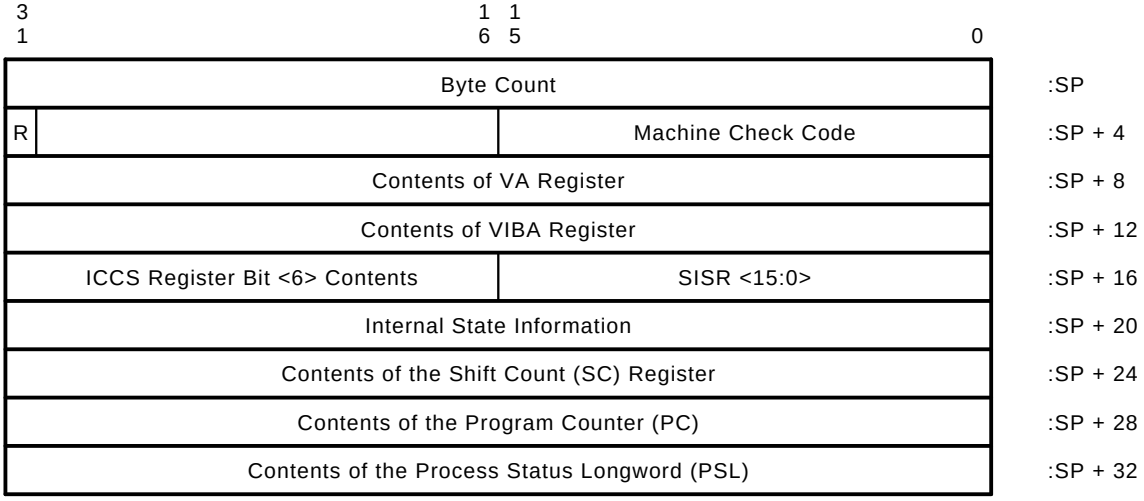


Figure 3–8 Information Saved on a Machine Check Exception

The following paragraphs explain the diagram of the stack pointer.

Byte Count

The byte count is <31:0> (00000018_{16}, 24_{10}) The byte count value indicates the number of bytes of information that follow on the stack, excluding the PC and PSL.

VAX Restart Bit (R)

Bit <31> of the longword at (SP)+4 after a machine check is the VAX restart bit (R). If R is 1, no state was by the instruction executing when the error was detected. If R is 0, the state was changed by the instruction.

Machine Check Code Parameter

Bits <15:0> of the longword at (SP)+4 after a machine check contain the machine check parameter code. This code value indicates the type of machine check that occurred. A list of the possible machine check codes (in hexadecimal) and their associated causes follows:

- **Floating Point Errors** (Table 3–8)

  These codes indicate that the FPA or CPU chips detected an error during the execution of a floating point instruction.

There are two most likely cause of these types of machine checks:

A problem internal to the P-chip or FPA chips
A problem with the interconnect between the two chips

Machine checks due to floating point errors may be retried, depending on the state of the VAX restart bit of the longword at (SP)+4, and the FIRST PART DONE flag (captured in PSL <27>). The error may be retried only if the VAX restart bit is set and the FIRST PART DONE flag is cleared. Otherwise, the error is unrecoverable; depending on the current mode, the current process or the operating system should be terminated, or the FPA should be disabled. The information pushed on the stack by this type of machine check is from the instruction that caused the machine check.

Table 3–8 Floating Point Errors

Hex Code	Error Description
01	A protocol error was detected by the FPA chip during an F-chip operand/result transfer.
02	An illegal opcode was detected by the FPA chip.
03	The FPA chip detected an operand parity error.
04	Unknown status was returned by the FPA chip.
05	The returned FPA chip result had a parity error.

- **Memory Management Errors** (Table 3–9)

  These codes indicate that the microcode in the P-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 P-chip. Machine checks due to memory management errors may be retried. 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 registers should be logged.

Table 3–9 Memory Management Errors

Hex Code	Error Description
08	A memory management error occurred while the P-chip was handling an access control violation/translation not valid fault. The READ/WRITE that caused the error missed the translation buffer. This error may be retried if the VAX restart bit or the FIRST PART DONE flag is set.
09	A memory management error occurred while the P-chip was handling an access control violation/translation not valid fault. The READ/WRITE that caused the error hit the translation buffer. This error may be retried if the VAX restart bit or the FIRST PART DONE flag is set. The fact that the errant reference hit the translation buffer means that the P-chip is the most likely cause of the error.

- **Interrupt Error** (Table 3–10)

  This code indicates that the interrupt controller in the P-chip 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 P-chip. Machine checks due to unused IPL errors may be retried. A nonvectored interrupt generated by a serious error condition

(memory error, power failure or processor halt) has probably been lost. The operating system should be terminated.

Table 3–10 Interrupt Errors

Hex Code	Error Description
0A	A hardware interrupt was requested at an unused interrupt priority level (IPL). This error may be retried if the VAX restart bit or the FIRST PART DONE flag is set.

- **Microcode Errors** (Table 3–11)

  These codes indicate that the microcode detected an impossible situation while the instruction was executing. Note that most erroneous branches in the P-chip microcode will cause random microinstructions to be executed. The most likely cause of this type of machine check is a problem internal to the P-chip. Machine checks due to microcode errors may be retried. Depending on the current mode, either the current process or the operating system should be terminated.

Table 3–11 Microcode Errors

Hex Code	Error Description
0B	An impossible state (for example, an undefined state bit combination in the microsequencer) was detected during a MOVC3 or MOVC5 instruction (not move forward, move backward, or fill). This error may be retried if the FIRST PART DONE flag is set.
0C	An undefined trap code was produced by the P-chip. This error may be retried if the VAX restart bit is set and the FIRST PART DONE flag is cleared.
0D	An undefined control store address was reached by the microsequencer. This error may be retried if either the VAX restart bit or the FIRST PART DONE flag is set.

- **Read Errors** (Table 3–12)

  These codes indicate that an error was detected while the P-chip was trying to read from either the primary cache, backup cache, main memory, or Q22-bus. The most likely cause of this type of machine check is determined from the state of the PCSTS, PCERR, BCSTS, BCERR DSER, MEMCSR32, MEMCSR33, and MEMCSR34.

  Machine checks due to read errors may be retried depending on the state of the VAX restart flag, the FIRST PART DONE flag, and the PCSTS<trap2> double-error bit. If either the FIRST PART DONE flag or VAX RESTART flag is set, and PCSTS<trap2> are cleared, then the error may be retried. Otherwise, the error is unrecoverable; 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 3–12 Read Errors

Hex Code	Error Description
10	A primary cache tag or data parity error occurred during a read.
11	An RDAL bus (error terminated cycle) or data parity error occurred during a read.

- **Write Error** (Table 3–13)

  This code indicates that an error was detected while the P-chip was trying to write to either the primary cache, backup cache, or main memory. This is an unexpected MCHK abort response in the KA670, because ERR L should never be accepted on a write cycle.

Table 3–13 Write Errors

Hex Code	Error Description
12	An RDAL bus error (for example, ERR L terminated cycle) occurred on a write or clear write buffer.

- **RDAL Bus Errors** (Table 3–14)

  This code indicates that the P-chip detected that the RDAL bus was in an undefined state. This machine check is not recoverable.

Table 3–14 RDAL Bus Errors

Hex Code	Error Description
13	An undefined RDAL bus state was detected by the P-chip.

Contents of the P-Chip's Internal Virtual Address (VA) Register

After a machine check, the location at (SP)+8 captures the contents of the P-chip's VA register at the time of the machine check. After a machine check of 10 or 11, the (SP)+8 location represents the virtual address of the memory location that was being read when the error occurred.

After a machine check of 12 (an RDAL bus error write or clear), the (SP)+8 location represents the virtual address of a location that was being referenced either during or after the error. Therefore, the contents of this field cannot be used for error recovery if the machine check occurred on a write operation.

Contents of the P-Chip's Internal VIBA Register

After a machine check, the location at (SP)+12 captures the contents of the P-chip's VIBA register at the time of the machine check. After a machine check, this field represents the virtual address of the last I-stream fetch plus four.

ICCS Register Bit<6> Contents

After a machine check, the location at (SP)+16 bit<22> captures the contents of the P-chip's interval clock control and status (ICCS) register's bit<6> the interrupt enable (IE) at the time of the machine check.

SISR Register Bits<15:0> Contents

After a machine check, the location at (SP)+16 bits<15:0> captures the contents of the P-chip's software interrupt summary register's (SISR) bits<15:0> at the time of the machine check.

Internal State Information

The internal state information field is divided into five subfields (Table 3–15).

Table 3–15 Internal State Information Field

Bits	Description
<31:24>	Delta PC (PC –backup PC)
<20:18>	The access Type (AT) at machine check time. The 3-bit code is interpreted as follows:
	<000> —Read access
	<001> —Write access
	<010> —Modify access
	<101> —Address access
	<110> —Variable bit access
	<111> —Branch access
<17:16>	The data length (DL) at machine check time. The 2-bit code is interpreted as follows:
	<00> —Byte long
	<01> —Word long
	<10> —Longword long
	<11> —Quadword long
<15:8>	Opcode—This field captures the opcode of the instruction being read or executed at the time of the machine check.
<3:0>	Register Number (RN) —This field captures the number of the register that was the destination of the instruction being executed at the time of the machine check.

Contents of the Shift Count (SC) Register

After a machine check, the location at (SP)+24 captures the contents of the P-chip's shift count (SC) register at the time of the machine check. The P-chip uses this register in different ways, depending on the instruction being executed.

Contents of the Program Counter (PC)

PC<31:0>—After a machine check, the location at (SP)+ 28 captures the virtual address of the start of the instruction being executed at the time of the machine check.

Contents of the Process Status Longword (PSL)

After a machine check, the location at (SP)+ 32 captures the contents of the PSL at the time of the machine check.

NOTE
The software must acknowledge machine checks by writing a 0 to the MCESR (IPR 38).

3.1.6.4 Machine Check Error Register (MCESR) IPR 38

The machine check error register (IPR 38, MCESR) provides the mechanism by which software acknowledges receipt of a machine check. MCESR is a write-only register and has the format shown in Figure 3–9:

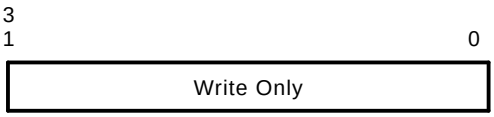


Figure 3–9 Machine Check Error Register (MCESR)— (IPR 38_{10} 26_{16})

When the P-chip microcode invokes the software machine check handler, it sets a MACHINE CHECK IN PROGRESS flag. If a machine check or memory management exception occurs while this flag is set, the microcode initiates a console double-error halt.

Software should clear the MACHINE CHECK IN PROGRESS flag in the machine check handler as soon as possible , by writing a 0 to IPR MCESR. Doing so re-enables normal machine check and memory management exception reporting.

3.1.6.5 System Control Block (SCB)

The system control block (SCB) consists of two pages in main memory that contain the vectors used to send interrupts and exceptions to the appropriate service routines. The SCB is pointed to by IPR 17, the system control block base register (SCBB). Figure 3–10 shows the format of the system control block format, and Table 3–16 describes the format.

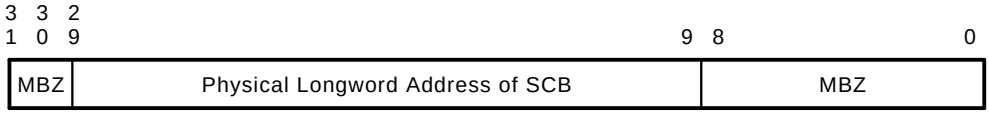


Figure 3–10 System Control Block Base Register (SCBB)— (IPL 17_{10} 11_{16})

Table 3–16 The System Control Block Format

SCB Offset	Interrupt/Exception Name	Type	Number of Params	Notes
00	Passive release	Interrupt	0	IPL is raised to request IPL.
04	Machine check	Abort	6	Parameters reflect machine state.

Table 3–16 (Cont.) The System Control Block Format

SCB Offset	Interrupt/Exception Name	Type	Number of Params	Notes
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/vector alignment fault	Fault	2	Parameters are virtual address, status code.
24	Translation not valid	Fault		2 Parameters are virtual address, status code.
28	Trace pending (TP)	Fault	0	
2C	Breakpoint instruction	Fault	0	
30	Unused	-	-	Compatibility mode in other VAX machines.
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	Memory soft error notification (corrected read error)	Interrupt	0	IPL is 1A.
58-5C	Unused	-	-	
60	Memory hard error notification	Interrupt	0	IPL is 1D.
64	Unused	-	-	
68	Vector unit disabled	Fault	0	Vector instructions.
6C-74	Unused	-	-	
78	Programmable timer 0	Interrupt	0	IPL is 14.
7C	Programmable timer 1	Interrupt	0	IPL is 14.

Table 3–16 (Cont.) The System Control Block Format

SCB Offset	Interrupt/Exception Name	Type	Number of Params	Notes
80	Unused	-	-	
84	Software level 1	Interrupt	0	
88	Software level 2	Interrupt	0	Ordinarily used for AST delivery.
8C	Software level 3	Interrupt	0	Ordinarily used for process scheduling.
90-BC	Software levels 4–15	Interrupt	0	
C0	Interval timer	Interrupt	0	IPL is 16.
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.
108	Mass storage interface 1 (DSSI PORT 1)	Interrupt	0	IPL is 14.
104	Mass storage interface 2 (DSSI PORT 2)	Interrupt	0	IPL is 14.
D8-DC	Unused	-	-	
F0	Network interface	Interrupt	0	IPL is 14.
F4	Unused	-	-	
F8	Console receiver	Interrupt	0	IPL is 15.
FC	Console transmitter	Interrupt	0	IPL is 15.
204	Interprocessor doorbell	Interrupt	0	IPL is 14.

Vectors in the range of 100 to FFFC are used to directly vector interrupts from the external bus. The SCBB vector index is determined from bits <15:2> of the value supplied by external hardware.

The new PSL priority level is determined by either the external interrupt request level that caused the interrupt or by bit <0> of the value supplied by external hardware.

If bit<0> is 0, the new IPL level is determined by the interrupt request level being serviced. IRQ<3> sets the IPL to 17_{16}; IRQ<2>, 16_{16}; IRQ<1>, 15_{16} ; and IRQ<0>, 14_{16}. If bit<0> of the value supplied by external hardware is 1, then the new IPL is forced to 17_{16}.

The ability to force the IPL to 17_{16} supports an external bus, such as the Q22-bus, that cannot guarantee that the device generating the SCBB vector index is the device that originally requested the interrupt.

For example, the Q22-bus has four separate interrupt request signals that correspond to IRQ<3:0>, but only one signal to daisy chain the interrupt grant. Furthermore, devices on the Q22-bus are ordered so that higher priority devices are electrically closer to the

bus master. If an IRQ<1> is being serviced, there is no guarantee that a higher priority device will not intercept the grant.

Software must determine the level of the device that was serviced and set the IPL to the correct value. Only device vectors in the range of 100 to $FFFC_{16}$ should be used, except by devices emulating console storage and terminal hardware.

3.1.6.6 The Hardware Halt Procedure

The hardware halt procedure is the method used by the hardware to assist the firmware in emulating a processor halt. The hardware halt procedure saves the following from IPR 42 (SAVPC) and IPR 43 (SAVPSL):

IPR 42 (SAVPC)	Current value of the PC
IPR 43 (SAVPSL)	Current value of the PSL, MAPEN<0>, a halt code, and the valid bit

Figure 3–11 and Figure 3–12 show the formats for (SAVPC) and (SAVPSL), respectively.

SAVPSL<14> (valid bit) is set to 0 if the PSL is valid, and set to 1 if the PSL is invalid. The valid bit is undefined after a halt caused by a system reset.

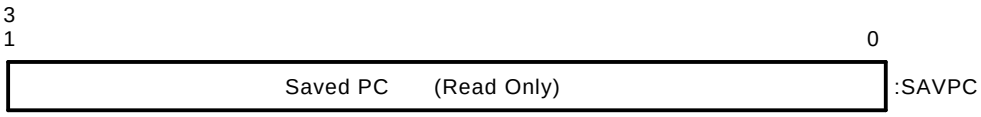


Figure 3–11 Console Saved PC (SAVPC)— (IPR 42_{10} $2A_{16}$)

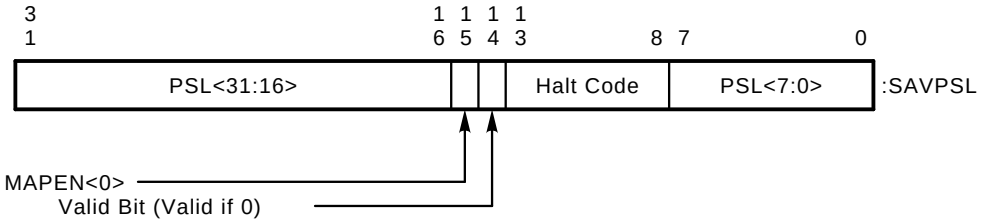


Figure 3–12 Console Saved PSL (SAVPSL)— (IPR 43_{10} $2B_{16}$)

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 3–17 shows the state of the CPU after a halt.

Table 3–17 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)

Table 3–17 (Cont.) CPU State After a Halt

Register	New Contents
SAVPSL<13:8>	Saved halt code
SP	Current interrupt stack (IPR 4)
PSL	041F 0000 $_{16}$
PC	2004 0000 $_{16}$
MAPEN	0
ICCS	0 (for a halt code of 3)
SISR	0 (for a halt code of 3)
ASTLVL	0 (for a halt code of 3) (asynchronous system traps level register)
All else	Undefined

The firmware uses the halt code in combination with hardware event indicators to send the interrupt or exception responsible for the halt to the appropriate firmware routine (either console emulation, power-up, reboot, or restart). Table 3–18 lists the interrupts and exceptions that can cause a halt, along with their corresponding halt codes and event indicators.

Table 3–18 HALT Codes

Halt Code	Interrupt Condition	Event Indicator
Halt Codes for Unmaskable Interrupts		
2	**External Halt (P-chip HALT_L pin asserted).**	
	BHALT asserted on the Q22-bus.	DSER<15>
	BDCOK negated and asserted on the Q22-bus while BPOK stays asserted (Q22-bus reboot/restart).	DSER<14>
	BREAK generated by the console.	RXDB<11>
3	**Hardware Reset (P-chip RESET pin negated)**	
	BDCOK and BPOK negated then asserted on the Q22-bus (power-up).	-
	BDCOK negated and asserted on the Q22-bus while BPOK stays asserted (Q22-bus reboot/restart).	-
Halt Codes for Exceptions		
6	Halt instruction executed in Kernel Mode.	
Halt Codes for Exceptions that Occur 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.	

Table 3–18 (Cont.) HALT Codes

Halt Code	Interrupt Condition	Event Indicator
10	Access 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.	
3F	Power-up self-test failed in the P-chip (microcoded).	

3.1.7 System Identification

The KA670 firmware and operating system software references two registers to determine the processor on which they are running. The first register is the system identification (SID) register , an internal processor register. The second register is the system identification extension (SIE) register, a firmware register in the KA670 EPROM.

3.1.7.1 System Identification Register

The system identification (SID) register (IPR 62) is a 32-bit, read-only register implemented in the CPU chip. The SID 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. Figure 3–13 shows the format of the SID register. Table 3–19 lists the bit definitions.

Figure 3–13 System Identification Register (SID)— (IPR 62_{10} $3E_{16}$)

Table 3–19 System Identification Register (SID)

Field	Name	RW	Description
<31:24>	CPU type	RO	The CPU type is the processor-specific identification code.
<23:8>	Reserved	RO	Reserved for future use.
<7:0>	Version	RO	Version of the microcode.

3.1.7.2 System Identification Extension Register (20040004)

The system identification extension (SIE) register is an extension of the SID register, 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 depend on the CPU type in SID<31:24>.

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 KA670 ROM. Figure 3–14 shows the format of the SIE register. Table 3–20 lists the definitions of the register bits.

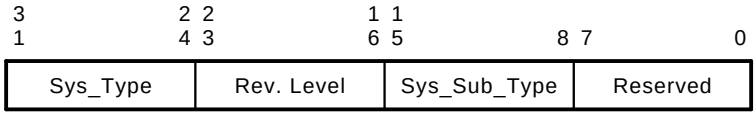


Figure 3–14 System Type Register (SYS_TYPE)

Table 3–20 System Type Register (SYS_TYPE)

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 field indentifies the resident version of the firmware EPROM, encoded as two hexadecimal digits. For example, if the banner displays V5.0, then this field is 50_{16}.
15:8	Sys_sub_ type	RO	This field indentifies the particular system subtype. 01 : KA650 02 : KA640 03 : KA655 04 : KA670
7:0	Reserved		This field is reserved.

3.1.8 Accelerator Control and Status Register (ACCS) IPR 40

The accelerator control and status register (IPR 40, ACCS) provides the FPU with the ability to generate bad data parity on write operations. Figure 3–15 shows the format of the ACCS. Table 3–21 lists the register bit definitions.

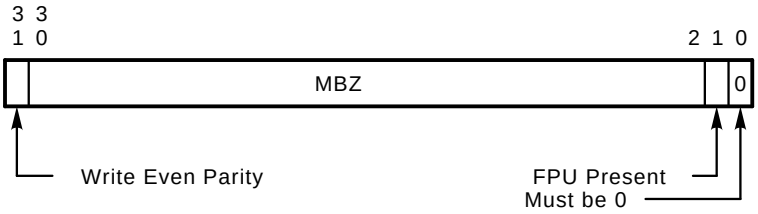


Figure 3–15 Accelerator Control and Status Register (ACCS)—(IPR 40_{10} 28_{16})

NOTE
The M bit should be set in any PTE that maps pages to be written while write even parity is enabled. Failure to do so may result in a PTE being written with bad parity during an M bit update.

Table 3–21 Accelerator Control and Status Register Bit Definitions

Data Bit	Name	Type	Definition
<31>	Write even parity	Write only	This bit enables the generation of bad data parity for write operations. If the bit is set to 1, all subsequent cache or memory writes and F-chip operand transfers are done with bad data parity on all bits. This bit is used for diagnostics only, and should never be set during normal operations. The write even parity bit is automatically cleared if ACCS is read with an MFPR instruction. Also, the write-even-parity state is cleared at the start of any interrupt, exception, or console halt, to prevent an exception stack frame from being written with bad data parity. The write even parity bit is cleared during a reset.
<30:2>	Reserved		Must be read as 0.
<1>	FPU present	Read/write	This bit enables use of the F-chip. If the FPU present bit is set to 1, floating point and longword-length integer multiply instructions are passed to the F-chip for execution. If the FPU present is set to 0, the execution of a floating point instruction results in a reserved instruction fault. Since an F-chip is included on every KA670 module, the FPU present bit should be set during normal operation. If an F-chip error is detected, the F-chip may be disabled if the operating system emulation software is loaded. This bit is cleared during a reset.
<0>	Must be zero	–	Reserved. Must be read as 0.

3.1.9 CPU References

CPU references are divided into three groups:

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

3.1.9.1 Instruction-Stream Read References

The CPU has an instruction prefetcher for prefetching program instructions from either cache or main memory. The prefetcher uses a 16-byte (4-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 generates an aligned quadword, request instruction-stream (I-stream) read reference.

3.1.9.2 Data-Stream Read References

Whenever the CPU needs data immediately to continue processing, a demand data-stream (D-stream) read reference is generated. 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.

All data read references are translated into an appropriate combination of masked and unmasked, aligned quadword read references. The reasons for the translation are that (1) the CPU does not impose any restrictions on data alignment other than the aligned operands of the add aligned word interlocked (ADAWI) and interlocked queue instructions, and (2) memory can only be accessed one aligned quadword at a time.

If the required data is . . .	Then the following is generated . . .
A byte, a word within a quadword, or an aligned quadword	A single, aligned quadword, demand D-stream read reference
A word that crosses a quadword boundary or an unaligned quadword	Two successive, aligned quadword, demand D-stream read references
Larger than a quadword	The data is divided into a number of successive, aligned quadword, demand D-stream reads, with no optimization.

3.1.9.3 Write References

Whenever data is stored or moved, a write reference is generated. All data write references are translated into an appropriate combination of masked and unmasked, aligned quadword write references. The reason for the translation is that (1) the CPU does not impose any restrictions on data alignment (other than the aligned operands of the ADAWI and interlocked queue instructions), and (2) memory can only be accessed one aligned quadword at a time.

If the required data is . . .	Then the following is generated . . .
A byte, a word within a quadword, or an aligned quadword	A single, aligned quadword, write reference
A word that crosses a quadword boundary or an unaligned quadword	Two successive, aligned quadword, write references

If the required data is . . .	Then the following is generated . . .
Larger than a quadword	The data is divided into a number of successive, aligned quadword writes

3.2 KA670 Floating Point Accelerator

The KA670 module includes a floating point accelerator (FPA) chip to enhance the performance of floating point and certain integer calculations. These functions are implemented by the F-chip.

3.2.1 Floating Point Accelerator Data Types

The KA670 floating point accelerator supports the following data types:

- F_floating
- D_floating
- G_floating
- Byte (conversion to and from floating formats)
- Word (conversion to and from floating formats)
- Longword (conversion to and from floating formats and multiply)

3.2.2 Floating Point Accelerator Instructions

The KA670 FPU chip processes the following VAX instructions:

- F_floating add, subtract, multiply, divide, convert, move, compare, negate, and test instructions. ACBF, EMODF, and POLYF are emulated, not processed by the FPU chip.
- D_floating add, subtract, multiply, divide, convert, move, compare, negate, and test instructions. ACBD, EMODD, and POLYD are emulated, not processed by the FPU chip.
- G_floating add, subtract, multiply, divide, convert, move, compare, negate, and test instructions. ACBG, EMODG, and POLYG are emulated, not processed by the FPU chip.
- Longword-length integer multiply instructions.

If the FPU chip is absent or disabled, the execution of a floating point instruction results in a reserved instruction exception. The execution of a longword-length integer multiply instruction is done by the FPU chip microcode.

3.2.3 Operand and Result Transfer

The CPU and FPU chips work together to execute instructions accelerated by the FPU chip. The CPU parses the opcode and instruction specifiers, then sends opcode and operands to the FPU.

Operands from the GPRs, the instruction stream, and the primary cache are explicitly transferred from the CPU to the FPU. Floating point short literals are transferred in unexpanded form; it is the FPU's responsibility to expand them to the correct format.

Operands from the backup cache or from memory are returned to both the CPU and the FPU simultaneously—they are not received by the CPU and rebroadcast to the FPU.

When the FPU receives the last operand for an instruction, the FPU begins to compute the result. In parallel, the CPU completes any instruction setup (for example, parsing a destination specifier). The CPU then requests the result from the FPU and stalls until the result is returned. Finally, the CPU stores the result in GPR or memory and sets the PSL condition codes.

The FPU tests for exception conditions and reports them to the CPU, in response to the request for the result. Detected exceptions include reserved operands, floating divide by zero, floating overflow, floating underflow, and data parity errors.

3.2.4 Power-Up State

At power-up, the CPU microcode disables the FPU as part of the chip initialization process. Until the FPU is enabled, the execution of any floating point instruction results in a reserved instruction exception. The console should enable the FPU by setting bit <1> of the accelerator control and status (ACCS) processor register, then test the operation of the FPU. If the FPU fails these tests, the console should clear ACCS<1> again.

Console Programmer's Note

The FPU does not accept memory operands in I/O space. Because the FPU executes longword-length integer multiply instructions as well as floating point instructions, it may not produce correct results or report operand parity errors if it is enabled during the execution of the console code from the boot ROM.

Therefore, the FPU should be disabled on any console entry, by writing a 0 to bit 1 of the ACCS processor register. This action causes the CPU to execute the integer instructions in microcode and invoke a reserved instruction fault for the floating point instructions.

The FPU is normally tested during the power-up self-test. In this case, it is the responsibility of the console programmer to understand the restrictions involved and perform the test in a controlled manner.

4
Cache and Main Memory

This chapter describes the operation and features of the KA670's cache memory and main memory controller.

4.1 KA670 Cache Memory

To maximize CPU performance, the KA670 incorporates a two-level cache hierarchy. The primary cache consists of 2 kilobytes of memory contained entirely in the CPU chip. The backup cache consists of the C-chip and twenty-four 16K × 4 static RAMs. The C-chip contains the tag store and the control logic for the backup cache RAMs. The backup cache is a 128-kilobyte cache used in combination with the CPU to provide a performance boost for the system.

The C-chip also serves to filter invalidates that may come from a memory controller, so not all invalidates have to be broadcast on the data and address lines. To filter invalidates, the C-chip maintains a copy of the primary cache tag store and uses an invalidate bus (I-bus). The I-bus can be used by DMA devices to determine if a memory location is cached in either cache. Using the I-bus eliminates the need to run an invalidate cycle of the RDAL for every DMA. Therefore, only those DMAs that hit in either cache cause an invalidate cycle saving RDAL bandwidth.

4.1.1 Cacheable References

Any reference stored by the primary or backup cache is called a *cacheable reference*. The primary and backup caches store CPU read references to the VAX memory space (bit <29> of the physical address equals 0) only. They do not store references to the VAX I/O space or DMA references by the Q22-bus interface. Two types of CPU references are stored—request instruction-stream read references and demand data-stream read references other than read-lock references.

If the CPU generates . . .	Then . . .
A noncacheable reference or a cacheable reference not stored in the primary cache	A single quadword reference of the same type is generated on the RDAL bus.
A cacheable reference stored in the primary cache	No reference is generated on the RDAL bus.

4.1.2 Primary Cache Overview

The primary cache is a 2-kilobyte cache, directly mapped, with a quadword fill and allocate (block) size. The cache is read-allocate, no-write-allocate, and write-through. The primary cache tag store contains one tag and one valid bit for each primary cache block. There are 256 tags mapping 256 quadword data blocks. Each tag entry includes an 18-bit tag, 1 valid bit, and 1 parity bit. Each data block contains 8 data bytes and 8 parity bits, one for each data byte.

4.1.2.1 Primary Cache Organization

The primary cache is arranged in 64 rows, with 4 quadwords and 4 tag entries per row. Figure 4–1 shows the format.

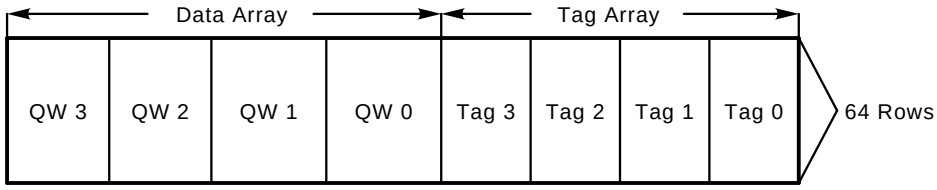


Figure 4–1 Primary Cache Data and Tag Layout

Each tag entry of the memory is organized as shown in Figure 4–2.

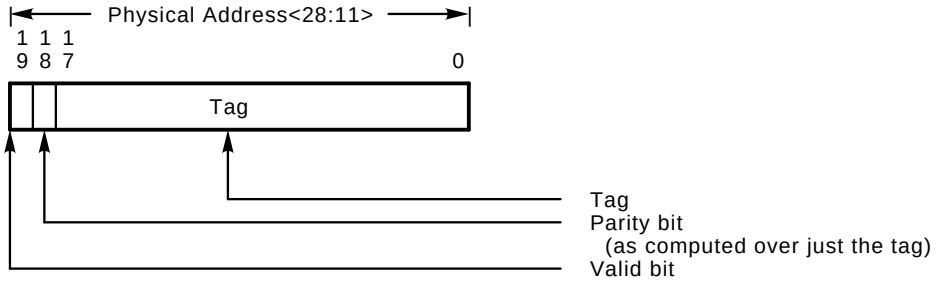


Figure 4–2 Primary Cache Tag Entry

The tag consists of bits <28:11> of the physical address (PA). The tag parity bit is the odd parity computed over 18 address bits, PA<28:11>. It is computed by the primary cache. The valid bit is used to indicate whether or not the corresponding entry in the primary cache is valid. The valid bit is not included in the tag parity calculation.

The data array has each quadword logically arranged as shown in Figure 4–3.

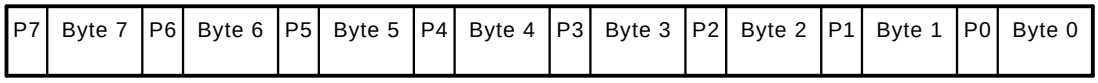


Figure 4–3 Primary Cache Data Entry

Each primary cache entry consists of one quadword. Odd parity information is maintained separately for each byte. The primary cache neither generates nor checks parity on data; the primary cache only stores parity information. The CPU (P-chip) bus interface unit (BIU) takes the responsibility of checking parity for the data. If a parity error is detected on the data coming from or written into the primary cache, the primary cache may be flushed or switched off by the resulting microtrap routine.

4.1.2.2 Primary Cache Address Translation

The physical addresses supplied to the primary cache consist of 28 bits (address<29:2>). Bit <2> of the physical address selects a longword out of the quadwords of the primary cache. Bits <8:3> select one of the rows of the primary cache memory. Because there are four tag entries in each row, two bits of the address (bits <10:9>) are used to select one of the four columns. Bits <28:11> are stored as tags in the primary cache. Bit <29> of the address specifies I/O space. I/O space addresses are not cached.

Whenever the CPU requires an instruction or data, the contents of the primary cache are checked to determine if the referenced location is stored there. The cache contents are checked by translating the physical address as shown in Figure 4–4.

On noncacheable references, the reference is never stored in the cache. So a primary cache miss occurs, and a single quadword reference is generated on the RDAL bus.

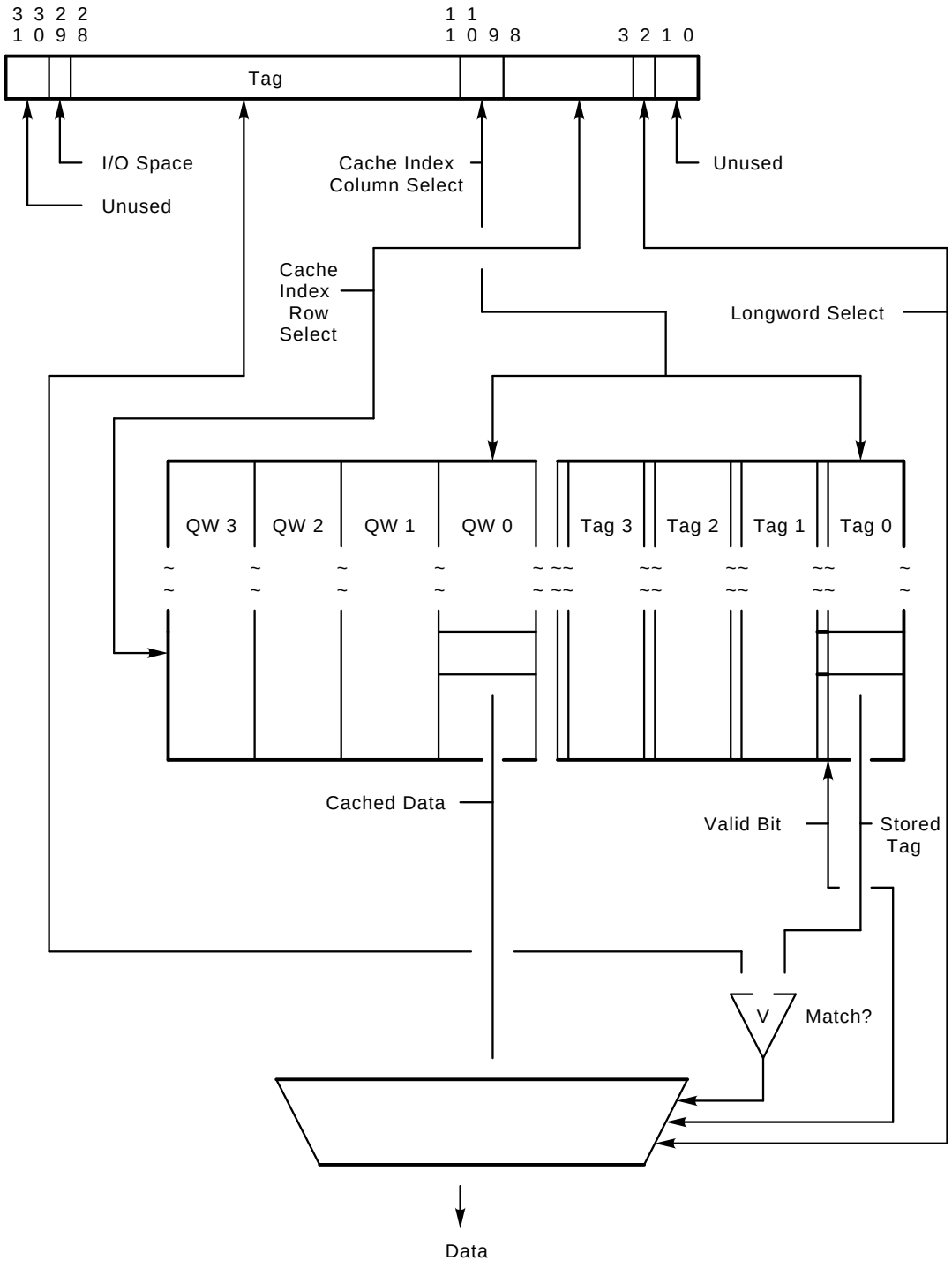


Figure 4–4 Primary Cache Physical Address Translation

4.1.2.3 Primary Cache Data Block Allocation

Cacheable references that miss the primary cache initiate a quadword read to on the RDAL bus. When the requested quadword is supplied by the backup cache or the main memory controller, the requested quadword is passed on to the CPU; a data block is allocated in the cache to store the quadword.

Since the KA670 supports 512 megabytes (64 mega-quadwords) of physical memory, up to 1 mega-quadwords share each row (four data blocks) of the cache. Contiguous programs larger than 2 kilobytes and noncontiguous programs separated by 2 kilobytes will overwrite themselves when cache data blocks are allocated.

4.1.2.4 Primary Cache Behavior on Writes

On CPU-generated write references, the primary cache is write-through. For all CPU write references that hit the primary cache, the contents of the referenced location in main memory is updated as well as the copy in the cache.

On DMA write references that hit the primary cache, the cache entry containing the copy of the referenced location is invalidated.

4.1.2.5 Primary Cache Internal Processor Registers

The primary cache includes four registers that may be accessed using IPR reads (move from processor register, MFPR) and writes (move to processor register, MTPR). These four registers are used for controlling the primary cache operation, storing status, and diagnostics and error recovery. Table 4–1 lists the four registers.

Table 4–1 Primary Cache Internal Processor Registers

		IPR Number		
Register Name	**Mnemonic**	**Hex**	**Decimal**	**Type**
Primary cache tag array	PCTAG	7C	124	Read/write
Primary cache index register	PCIDX	7d	125	Read/write
Primary cache error address register	PCERR	7E	126	Read/write
Primary cache status register	PCSTS	7F	127	Read/write

4.1.2.5.1 Primary Cache Status Register (PCSTS)—IPR 127

The primary cache status register (PCSTS) is used to control the primary cache's mode of operation, flush the cache, and maintain information about all errors detected by the CPU (not only primary cache errors). The PCSTS register is considered locked to errors that result in an interrupt, if the interrupt bit (PCSTS<5>) or trap1 bit (PCSTS<7>) is set. For errors that result in a trap, this register is considered locked only if trap1 is set.

Figure 4–5 shows the format of the status register. Table 4–2 lists bit definitions.

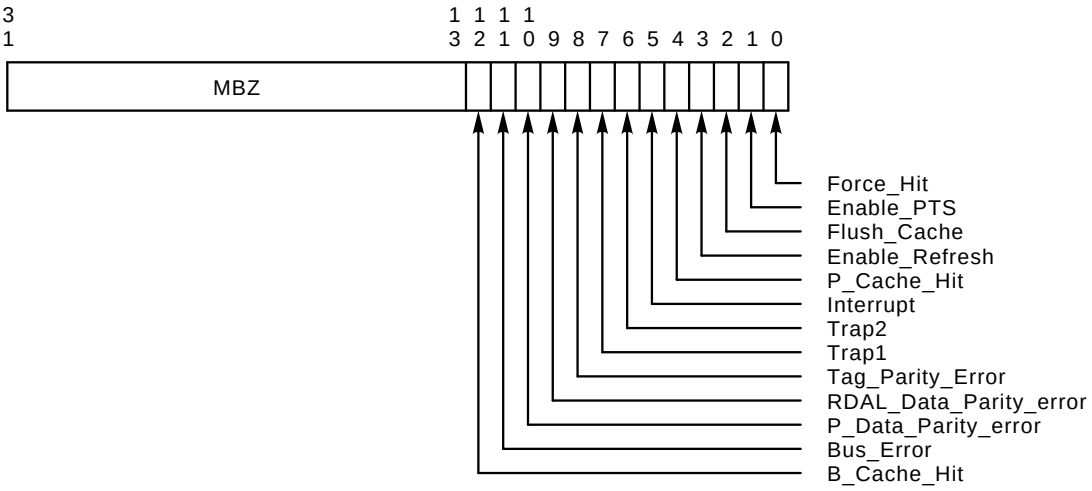


Figure 4–5 Primary Cache Status Register (PCSTS)— (IPR 127_{10} $7F_{16}$)

Table 4–2 Primary Cache Status Register

Data Bit	Name	Definition
<31:13>	MBZ	Must be zero. Always read as 0s. Writes have no effect.
<12>	B_cache_hit	Backup cache hit (read only). This bit indicates that the error condition causing the primary cache status register to lock was a reference that hit in the backup cache. For RDAL parity errors, this bit can be used to determine who was driving the the RDAL bus. If B_cache_hit is set, the source was the backup cache. If B_cache_hit is clear, the memory subsystem was the source. This bit is updated for any reference that has an associated error, if the primary cache status register is not already locked. B_cache_hit is cleared on reset.
<11>	Bus_error	Bus error (read only). This bit is set when an RDAL read, write, or clear-write-buffer command results in an error. If the RDAL command was an I-stream read, the interrupt bit (PCSTS<5>) is also set. The error is reported as an IPL $1A_{16}$ soft error interrupt. If the RDAL command was a D-stream read, write or clear-write-buffer, trap1 (PCSTS<7>) or trap2 (PCSTS<6>) is also set. The error results in a machine check. This bit is updated for any reference that has an associated error, if the primary cache status register is not already locked. Bus_error is cleared on reset.

Table 4–2 (Cont.) Primary Cache Status Register

Data Bit	Name	Definition
<10>	P_data_parity	Primary cache data error (read only). This bit is set when a read hits in the primary cache and the requested data has a parity error. If the RDAL command was an I-stream read, the interrupt bit (PCSTS<5>) is also set. The error is reported as an IPL $1A_{16}$ soft error interrupt. If the RDAL command was a D-stream read, write or clear-write-buffer, trap1 (PCSTS<7>) or trap2 (PCSTS<6>) is also set. The error results in a machine check. This bit is updated for any reference that has an associated error, if the primary cache status register is not already locked. P_data_error is cleared on reset.
<9>	RDAL_data_parity	RDAL data parity error (read only). This bit is set when the data returned in response to a non-I/O space RDAL read has a parity error. If the error is detected on the nonrequested longword of a D-stream read, or on either longword of an I-stream read, the interrupt bit (PCSTS<5>) is also set. The error is reported as an IPL $1A_{16}$ soft error interrupt. If the RDAL data parity error is detected on the requested longword of a D-stream read, trap1 (PCSTS<7>) or trap2 (PCSTS<6>) is also set. The error results in a machine check. This bit is updated for any reference that has an associated error, if the primary cache status register is not already locked. RDAL_data_parity is cleared on reset.
<8>	Tag_parity_error	Primary cache tag parity error (read only). This bit is set if a primary cache tag parity error is detected during a read, write, or invalidate reference, providing the PCSTS register has not been not locked by a previous error. If tag_parity_error is set, the interrupt bit (PCSTS<5>) is also set. The error is reported as an IPL $1A_{16}$ soft error interrupt. If the reference was a D-stream read that hit, trap1 (PCSTS<7>) or trap2 (PCSTS<6>) is also set. The error results in a machine check. This bit is updated for any reference that has an associated error, if the primary cache status register is not already locked. Tag_parity_error is cleared on reset.

Table 4–2 (Cont.) Primary Cache Status Register

Data Bit	Name	Definition
<7>	Trap1	Write one to clear. This bit is set when an error detected by the CPU results in a machine check. PCSTS<12:8> and the primary cache error address register (PCERR IPR 126), are latched until trap1 is cleared. If this bit is set, the primary cache is *not* automatically flushed. However, it is automatically disabled (although enable_PTS PCSTS<1> is not changed). Trap1 is cleared on reset and by writing 1 to it with an MTPR instruction.
<6>	Trap2	Write one to clear. This bit is set when an error detected by the CPU results in a machine check and trap1 (PCSTS<7>) is already set. When trap2 is set, it indicates that a nested error occurred and that PCSTS<12:8> and the primary cache error address register (PCERR IPR 126) contain information about the first error that set trap1. This should be considered a fatal error condition. If trap2 is set, the primary cache is *not* automatically flushed. However, it is automatically disabled (although enable_PTS (PCSTS<1>) is not changed). Trap2 is cleared on reset and by writing 1 to it with an MTPR instruction.
<5>	Interrupt	Write one to clear. This bit is set when an error detected by the CPU results in an interrupt at IPL $1A_{16}$. PCSTS<12:8> are latched unless the interrupt bit or trap1 (PCSTS<7>) was previously set; they remain latched until the interrupt bit is cleared or another error sets trap1. If the interrupt bit is set, the primary cache is automatically disabled (although enable_PTS (PCSTS<1>) is not changed). The interrupt bit is cleared on reset and by writing 1 to it with an MTPR instruction.
<4>	P_cache_hit	Primary cache hit (read only). This bit is the latched output value of the tag comparator. This bit is updated for all D-stream reads, writes, or invalidate cycles. It may be used to test the primary cache hit logic. P_cache_hit is cleared on reset and should be used for diagnostic purposes only.
<3>	Enable_refresh	Enable refresh (read/write). When this bit is set, the automatic refresh of the primary cache take place and the refresh counter increments. When this bit is a cleared, refresh is disabled, the refresh counter does not increment, and the refresh timer logic is disabled. This bit should be set during normal primary cache operations. Enable_refresh is cleared on reset.

Table 4–2 (Cont.) Primary Cache Status Register

Data Bit	Name	Definition
<2>	Flush_cache	Flush primary cache (write only). This bit is used to clear all valid bits in the primary cache tag array. If this bit is written with a 1, the primary cache is flushed. The hardware then clears this bit in the next cycle, so that it is always read as a 0. **NOTE** **The state of the primary cache is unpredictable if enable_PTS is 0 (PCSTS<1>). Therefore, the primary cache should be flushed before it is enabled. This may be done as a separate IPR write of flush_cache before enable_PTS is set in the primary cache status register. It may also be done by setting flush_cache and enable_PTS in the same IPR write to the primary cache status register.**
<1>	Enable_PTS	Enable primary cache (read/write). This bit enables or disables normal operation of the primary cache. If the bit is set, both I-stream and D-stream references are cached, and primary cache tag and data parity errors are reported. I/O references are never cached. If the bit is cleared, all references (read, write, and invalidate) result in a miss. enable_PTS is cleared on reset.
<0>	Force_hit	Force a primary cache hit (read/write) When this bit is set, the primary cache forces a hit for all memory references. Memory write requests still go to the external memory. I/O references are not affected (they are not cached). When this bit is set, the following are disabled: primary cache tag parity error reporting associated with D-stream reads, writes, and invalidates, and primary cache data parity errors associated with D-stream reads. RDAL errors (parity errors associated with the data present on the RDAL or bus errors) are not affected by this bit. Force_hit should not be used to satisfy I-stream reads, since primary cache tag or data parity errors detected during I-stream reads may cause a loop. Force_hit may be used to initialize the primary cache data array. Force_hit is cleared on a reset. This bit is for diagnostics only and should be cleared during normal operation. **NOTE** **When the primary cache is off (enable_PTS = 0) and force_hit is set, the operation of the primary cache is unpredictable.**

4.1.2.5.2 Primary Cache Error Address Register (PCERR)—IPR 126

For read commands, the primary cache error address register (PCERR) latches and holds the physical address of an error that causes trap1 (PCSTS<7>) to set. Since write errors are asynchronous to the instruction pipeline, the address latched for write commands is not the address of the error. For write commands, no address is available. The PCERR register remains locked until trap1 is cleared in the primary cache status register (PCSTS).

The PCERR register also provides visibility into the refresh counter and refresh timer. An IPR write (MTPR) to the PCERR register updates the refresh counter and timer. An IPR write to the PCERR register loads the refresh counter with bits <8:3> of the data, and the refresh timer with bits <15:9> of the data.

An IPR read (MFPR) of the primary cache error address register (PCERR) reads the error address out if trap1 (PCSTS<7>) is set. If trap1 is not set, an IPR read is used to read the refresh timer in bits <15:9> and the value of the refresh counter in bits <8:3>.

Access to the refresh counter and refresh timer is provided for diagnostics only.

Figure 4–6 shows the format for the primary cache error address register.

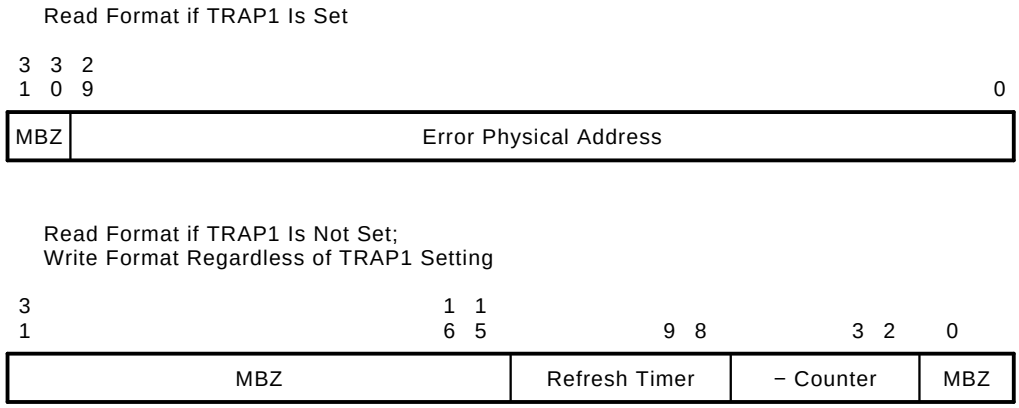


Figure 4–6 Primary Cache Error Address Register (PCERR) (IPR 126 $_{10}$ 7E$_{16}$)

If enable_refresh (PCSTS<3>) is set, a read of the refresh timer and counter (through the PCERR register) following an IPR write to the timer and counter will result in a different value than the value written. The reason is as follows.

When the enable_refresh (PCSTS<3>) bit is set, the refresh counter is incremented for every NOP operation. The refresh timer is incremented for every cycle during which the operation is not a NOP or an IPR write to the PCERR register. Due to the internal latency involved in the execution of MxPRs, the count values of the refresh counter and timer may change.

To keep the count values of the refresh counter and timer unchanged, the enable_refresh (PCSTS<3>) bit should be cleared.

4.1.2.5.3 Primary Cache Index Register (PCIDX)–IPR 125

The primary cache index register (PCIDX) provides the mechanism for reading and writing the tag array of the primary cache. During IPR (MTPR) writes to the primary cache tag array register (PCTAG) (Section 4.1.2.6), the contents of the PCIDX register are used to index the desired tag entry in the array. Therefore, the PCIDX register must be written with the desired index before performing an IPR write (MTPR) to the PCTAG register.

Figure 4–7 shows the format of this register.

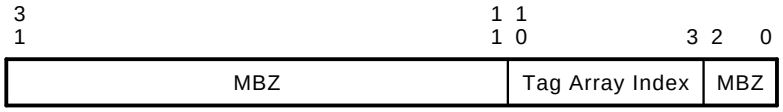


Figure 4–7 Primary Cache Index Register (PCIDX)—(IPR 125 $_{10}$ $7D_{16}$)

4.1.2.5.4 Primary Cache Tag Array Register (PCTAG)—IPR 124

The primary cache tag array register (PCTAG) is a 32-bit logical register that provides the mechanism for reading and writing the tag array of the primary cache.

Figure 4–8 shows the format for this register.

Figure 4–8 Primary Cache Tag Array Register (PCTAG)—(IPR 124 $_{10}$ $7C_{16}$)

4.1.2.6 Writing and Reading the Primary Cache Tag Array

During an IPR read/write of the primary cache tag array register (PCTAG), the primary cache index register (PCIDX) supplies the index for the tag entry to be accessed. To write a tag entry in the primary cache, first the index of the tag entry is written in the PCIDX register by issuing an MTPR. Then an MTPR is issued for the primary cache tag array register (PCTAG), with the desired value of the valid bit, parity bit, and tag address<28:11> in data bits<31:30> and <28:11>.

In order to read a tag entry in the primary cache, the index of the tag entry is written in the PCIDX register by issuing an MTPR instruction. Then an MFPR instrcution is issued for the primary cache tag array register (PCTAG).

4.1.2.7 Primary Cache Error Recovery

When an error is detected in the primary cache, the primary cache latches error information in the PCSTS and PCERR registers, then becomes disabled. The exact type of error can be determined from the information in the PCSTS register and the way the error was reported.

If the error was a tag parity error, the entire tag store must be written with “invalid tag with good parity.” The PCERR register contains the address of the tag in error only if the tag parity error is reported as a machine check.

For all other errors, the primary cache should simply be flushed by writing 1 to the flush_cache bit in the primary cache status register (PCSTS<2>).

To complete error recovery, the primary tag store in the C-chip should be flushed and the error bits should be cleared in the PCSTS. The primary cache should be enabled if the error rate is such that the primary cache would remain disabled. The primary tag store in the C-chip must also be disabled.

The following is the recommended sequence for bringing the primary cache back to normal operation:

1. Save the primary cache status register.
2. Save the primary cache error address register.
3. If the error was a tag parity error, write all tags in the primary cache, as follows:
    1. Write the primary cache index register with the next index.
    2. Write the primary cache tag array register with tag = 0 (arbitrarily chosen), parity = 0 (odd parity for chosen tag value), and valid = 0.
4. Flush the primary tag store in the C-chip by writing a 0 to the backup cache flush primary tag store (BCFPTS) register (IPR 122).
5. Logically OR the flush_cache bit (<2>) into the saved value of the primary cache status register, then write the resulting value back into the primary cache status register. This step clears any error bits that were set, flushes the cache, and enable its if it was enabled before.

4.1.2.8 Primary Cache Initialization

At power-up, the primary cache must be initialized. The console firmware should load the primary cache status register (PCSTS) with the desired values for the force_hit, enable_PTS, and enable_refresh bits. The firmware should clear the interrupt, trap1, and trap2 bits in the PCSTS register. The firmware should also invalidate the entire primary cache by issuing an IPR write (MTPR) to PCSTS, writing a 1 in bit <2> (flush_cache). Then each tag store entry should be loaded with an invalid tag with good parity. Each entry may be written with a write to PCIDX, followed by a write to PCTAG.

4.1.2.9 Primary Cache Diagnostics

The primary cache may be tested by reading and writing tags with PCIDX and PCTAG. Error detection may be tested by constructing an error and then reading the state from PCSTS and PCERR. The primary cache refresh counter and timer may be tested by reading and writing the primary cache error register (PCERR).

4.1.2.10 Error Handling by the Primary Cache

The primary cache is responsible for latching any error signals that occur for the following:

- Primary cache tag parity error
- Primary cache data parity error
- RDAL data parity error
- RDAL bus error
- F-chip result parity error

The latter four errors are detected by the CPU (P-chip). Errors are reported in one of two ways: as a soft error interrupt at IPL $1A_{16}$, or as a machine check.

When an error is detected, the primary cache sets trap1, trap2, or the interrupt bit in the primary cache status register (PCSTS) and conditionally latches other bits to indicate the type of error. When trap1, trap2, or the interrupt biut are set in the PCSTS register, the primary cache is automatically disabled.

Primary cache tag parity errors are reported if a tag parity error is detected during a read, write, or invalidate reference; and if the primary cache status register has not been already locked by a previous error (for example, enable_PTS = 1, trap1 = 0, trap2 = 0, interrupt = 0, and force_hit = 0). Tag parity errors are always reported as an interrupt. If the reference was a D-stream read that hit, the error is also reported as a machine check.

Primary cache data parity errors are reported if a data parity error is detected during a read reference that hit in the primary cache (unless force_hit (PCSTS<0>=1). Primary cache data parity errors are reported as a machine check if the reference was a D-stream read. If the reference was an I-stream read, primary cache data parity errors are reported as an interrupt.

RDAL data parity errors are reported if a data parity error is detected during a non-I/O space read reference that missed in the primary cache. RDAL data parity errors detected on the requested longword of a D-stream read are reported as a machine check. RDAL parity errors detected on the nonrequested longword of a D-stream read, or on either longword of an I-stream read, are reported as an interrupt.

RDAL bus errors are reported if a read, write, or clear write buffer command is terminated with an error (RDAL bus signal ERR_L asserted). Bus errors detected during D-stream read, write, or clear write buffer commands are reported as a machine check. Bus errors detected during an I-stream read are reported as an interrupt.

NOTE

RDAL bus errors may also be reported for an EPR read or read interrupt vector command that is terminated with the RDAL signal ERR_L. In those cases, however, the PCSTS register does not lock and the CPU processes the error entirely by microcode, with no error reported to the software.

F-chip result parity errors are reported if a data parity error is detected during a result transfer from the F-chip. Result parity errors are always reported as a machine check.

Errors reported as interrupts do the following:

- Set the interrupt bit (PCSTS<5>).
- Update bits PCSTS<12:8> with information describing the error.

However, if either interrupt or trap1 is already set when the error is detected, bits PCSTS<12:8> are not updated. Bits PCSTS<12:8> reflect the first error detected.

Errors reported as a machine check do three things:

- Set trap1 in the primary cache status register (PCSTS).
- Update bits PCSTS<12:8>.
- Load the primary cache error address register (PCERR).

However, if trap1 is already set when the error is detected, trap2 is set and neither bits PCSTS<12:8> nor the PCERR register are updated. This causes bits PCSTS<12:8> and the PCERR register to reflect the first error detected. Note that the state corresponding to the machine check overwrites any information latched due to a previous interrupt. It is assumed that errors reported as a machine check are more important than those reported as an interrupt.

In the CPU, primary cache data parity errors are reported only if the read reference hits in the cache. However, primary cache tag parity errors are reported whenever the error is detected. The following are two reasons for this inconsistency:

1. Primary cache tag entries can be directly written without any side effects, using MTPR macro instructions. There is no direct and easy way of writing the primary cache data array.
2. If primary cache tag parity errors are reported only under hits, there is a possibility that a stuck-at fault in the tag array might not get detected for a long time. Meanwhile, the system will run at degraded performance. This is undesirable.

Figure 4–9 shows the resulting status register values for each error type.

Error Conditions				Resulting PCSTS Register Values												
Error	Command	LW	PC hit?	<12>5 BCH	<11> BER	<10> PDP	<9> DDP	<8> PTP	<7> TR1	<6> TR2	<5> INT	<4> HIT	<3>1 RFR	<2> FLS	<1>7 PON	<0> FHT
PRIMARY CACHE TAG PARITY ERROR	D-READ	X	YES	0	0	0	0	1	1	0	1[3]	X	1	0	1	0
PRIMARY CACHE TAG PARITY ERROR	D-READ	X	NO	0	0	0	0	1	0	0	1	X	1	0	1	0
PRIMARY CACHE TAG PARITY ERROR	I-READ	X	YES	0	0	0	0	1	0	0	1	X	1	0	1	0
PRIMARY CACHE TAG PARITY ERROR	I-READ	X	NO	0	0	0	0	1	0	0	1	X	1	0	1	0
PRIMARY CACHE TAG PARITY ERROR	WRITE	X	X	0	0	0	0	1	0	0	1	X	1	0	1	0
PRIMARY CACHE TAG PARITY ERROR	INVAL	X	X	0	0	0	0	1	0	0	1	X	1	0	1	0
PRIMARY CACHE TAG PARITY ERROR	OTHER[6]	X	X	0	0	0	0	0	0	0	0	X	1	0	X	0
PRIMARY CACHE DATA PARITY ERROR	D-READ	X	YES	0	0	1	0	0	1	0	0	X	1	0	1	0
PRIMARY CACHE DATA PARITY ERROR	I-READ	X	YES	0	0	1	0	0	0	0	1	X	1	0	1	0
PRIMARY CACHE DATA PARITY ERROR	OTHER[6]	X	X	0	0	0	0	0	0	0	0	X	1	0	X	0
RDAL DATA PARITY ERROR	D-READ	1	NO	BCH	0	0	1	0	1	0	0	X	1	0	X	0
RDAL DATA PARITY ERROR	D-READ	2	NO	BCH	0	0	1	0	0	0	1	X	1	0	X	0
RDAL DATA PARITY ERROR	I-READ	X	NO	BCH	0	0	1	0	0	0	1	X	1	0	X	0
RDAL DATA PARITY ERROR	OTHER[6]	X	X	0	0	0	0	0	0	0	0	X	1	0	X	0
RDAL BUS ERROR	D-READ	X	NO	BCH	1	0	0[4]	0	1	0	0	X	1	0	X	0
RDAL BUS ERROR	I-READ	X	NO	BCH	1	0	0[4]	0	0	0	1	X	1	0	X	0
RDAL BUS ERROR	WRITE	X	X	BCH	1	0	0	0	1	0	0	X	1	0	X	0
RDAL BUS ERROR	CL WR BUF	X	X	BCH	1	0	0	0	1	0	0	X	1	0	X	0
RDAL BUS ERROR	OTHER[6]	X	X	0	0	0	0	0	0	0	0	X	1	0	X	0
F-CHIP RSLT PARITY ERROR	RD RSLT	X	X	0	0	0	0	0	1	0	0	X	1	0	X	0

Figure 4–9 Primary Cache Detectable Single Errors

Notes:

1. In all of these cases, it is assumed that enable_refresh (PCSTS<3>) is set to 1 and force_hit (PCSTS<0>) is set 0. This is the normal state of the cache, and other states may change the way errors are reported.
2. The primary cache must be enabled to get a primary cache tag or data parity error. The primary cache may or may not be enabled when a DAL data parity error, RDAL bus error, or an F-chip result parity error is detected.

3. Primary cache tag parity errors always cause an interrupt request. If the error was the result of a D-stream read that hit, a microtrap is also started.
4. If a read transaction is terminated by ERR_L, data parity is ignored. Therefore, the RDAL data parity error bit in the status register is never set for a read terminated in ERR_L.
5. B_cache_hit is always loaded when an error is detected. However, primary cache tag and data parity errors are detected as part of a primary cache reference, so the B_cache_hit is always be a 0 for those errors.
6. Commands that have error detection inhibited do not set trap1, trap2, and the interrupt bit in the primary cache status register. For example, tag parity errors are inhibited for commands that do not access the tag store. Similarly, RDAL bus errors are not reported for EPR read, EPR write, or read interrupt vector transactions terminated by ERR_L; those commands are handled specially by the microcode.

The resulting values shown in Figure 4–9 assume that trap1, trap2, and the interrupt bit are all 0 when the error is detected. In that case, bits PCSTS<12:8> are updated as shown. If trap1, trap2, or the interrupt bit are 1 when the error is detected, bits are updated as shown in Figure 4–10.

STATE BEFORE ERROR				STATE AFTER ERROR						
7 TR1	6 TR2	5 INT	TYPE	7 TR1	6 TR2	5 INT	LOAD <12:8>	LOCK ADDR	ERROR REG	DISABLE PRIMARY CACHE?	NOTES:
0	0	0	INT	0	0	1	YES	NO		YES	SINGLE INTERRUPT
0	0	1	INT	0	0	1	NO	NO		YES	MULTIPLE INTERRUPT
0	1	0	INT	0	1	1	YES	NO		YES	POSSIBLE BUT NOT LIKELY
0	1	1	INT	0	1	1	NO	NO		YES	POSSIBLEBUT NOT LIKELY
1	0	0	INT	1	0	1	NO	NO		YES	INTERRRUPT AFTER TRAP
1	0	1	INT	1	0	1	NO	NO		YES	MULTIPLE INTERRUPT AFTER TRAP
1	1	0	INT	1	1	1	NO	NO		YES	INTERRUPT AFTER MULTIPLE TRAP
1	1	1	INT	1	1	1	NO	NO		YES	MULTIPLE INTERRUPT AFTER MULTIPLE TRAP
0	0	0	TRAP	1	0	0	YES	YES		YES	SINGLE TRAP
0	0	1	TRAP	1	0	1	YES	YES		YES	TRAP AFTER INTERRUPT
0	1	0	TRAP	1	1	0	YES	YES		YES	POSSIBLE BUT NOT LIKELY
0	1	1	TRAP	1	1	1	YES	YES		YES	POSSIBLE BUT NOT LIKELY
1	0	0	TRAP	1	1	0	NO	NO		YES	DOUBLE TRAP
1	0	1	TRAP	1	1	1	NO	NO		YES	DOUBLE TRAP AFTER INTERRUPT
1	1	0	TRAP	1	1	0	NO	NO		YES	MULTIPLE TRAP
1	1	1	TRAP	1	1	1	NO	NO		YES	MULTIPLE TRAP AFTER INTERRUPT

Figure 4–10 Primary Cache Detectable Double Errors

4.1.3 Backup Cache Overview

The backup cache is 128 kilobytes, direct-mapped, quadword access size, with an octaword fill (subblock) size, and a 4-octaword allocate (block) size. The cache is read-allocate, no-write-allocate, and write-through.

Because the data bus (D_BUS) is 8 bytes wide, the cache RAMs are organized as 8 bytes wide by 16K locations deep. There are also 8 bits of parity (1 bit corresponding to each data byte). Fourteen bits of address are needed to access the cache. Bits <16:3> of the VAX physical address are used as the cache index.

When returning data to the primary cache, the backup cache returns a quadword. This is the **fill** size of the primary cache.

4.1.3.1 Backup Cache Organization

The backup cache tag store is organized such that **one tag** and **four valid bits** correspond to each **four-octaword block** of the cache. Each valid bit corresponds to one octaword subblock. When a cache tag miss occurs on a read, a block is allocated, a subblock is filled, and the corresponding valid bit is set. When a cache tag compare is successful but the valid bit is not set, a subblock is filled from memory, and the corresponding valid bit is set. Figure 4–11 shows the backup cache organization.

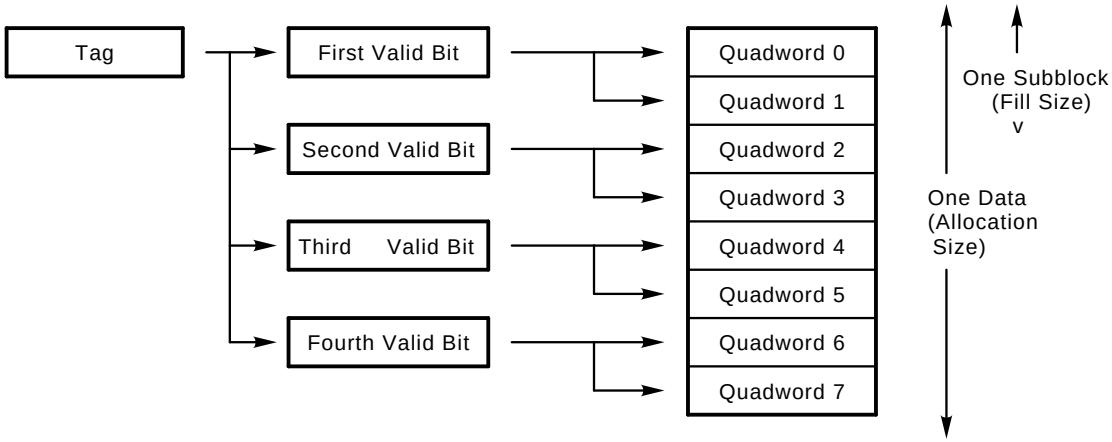


Figure 4–11 Tag and Valid Bits as They Correspond to Backup Cache Data

4.1.3.2 Backup Cache Address Translation

The physical addresses supplied to the backup cache consist of 28 bits (address<29:2>). Bits <5:4> of the physical address select one of the four valid bits that cover two quadwords in a backup cache row, as shown in Figure 4–11.

There are 16K quadwords of data. Fourteen bits of the address (<16:3>) select one quadword. Because there are 2K tag entries (one tag covers 8 quadword data entries), 11 bits of the address (<16:6>) are used to select one tag. Bits <28:17> are stored as tags in the backup cache. Bit <29> of the address specifies I/O space; this bit is not used, because I/O space addresses are not cached.

On noncacheable references, the reference is never stored in the cache. Therefore, a backup cache miss occurs and an octaword reference is generated on the RDAL bus.

Whenever the CPU requires an instruction or data not found in the primary cache, the contents of the backup cache is checked to determine if the referenced location is stored there. The cache contents are checked by translating the physical address as shown in Figure 4–12.

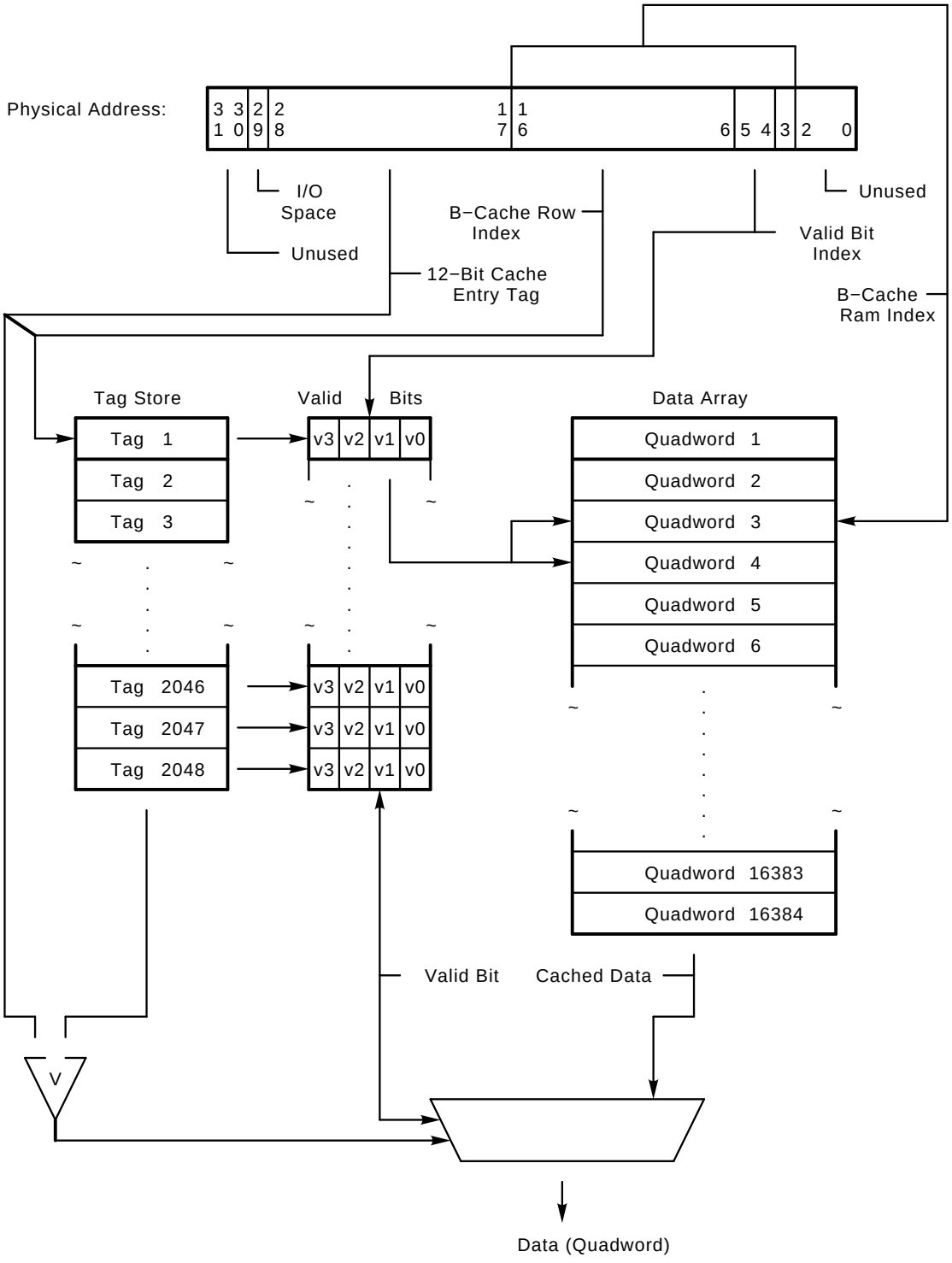


Figure 4–12 Backup Cache Physical Address Translation

4.1.3.3 Backup Cache Data Block Allocation

On cacheable references that miss the primary cache, a quadword read is initiated on the RDAL bus. If the requested quadword cannot be found in the backup cache:

- An octaword is provided by the main memory controller.
- Both caches allocate a data block for storing the data. (The primary cache allocates and fills a quadword; the backup cache allocates 4 octawords but only fills 1 octaword.
- The requested quadword is passed on to the CPU.

Since the KA670 supports 512 megabytes (32 mega-octawords) of physical memory, up to 4K octawords *share* each data block (8 quadwords) of the cache. Contiguous programs larger than 128 kilobytes, or noncontiguous programs separated by 128 kilobytes, will overwrite themselves in the backup cache when cache data blocks are allocated.

4.1.3.4 Backup Cache Behavior on Writes

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

On DMA write references that hit the cache, the cache entry containing the copy of the referenced location is invalidated.

4.1.3.5 Backup Cache External Processor Registers

Several C-chip registers may be accessed using EPR reads (MFPRs) and EPR writes (MTPRs). The following sections detail the structure of the registers and how the access of the registers is accomplished. During the EPR access, RDAL address bits <10:3> tell which EPR is being accessed.

The C-chip contains some vector registers that are not used on the KA670 module, since it does not have a vector processor. This manual discusses only one of these registers briefly, the vector interface error status register (VINTSR). Table 4–3 lists the C-chip EPR registers and their numbers.

Table 4–3 Backup Cache External/Internal Processor Registers

Register Name	Mnemonic	EPR Number		Type
		Hex	Decimal	
C-Chip Nonvector Registers				
Backup cache tag store	BCBTS	71	113	Read/write
Primary tag store, first half	BCP1TS	72	114	Read/write
Primary tag store, second half	BCP2TS	73	115	Read/write
Refresh register	BCRFR	74	116	Read/write
Index register	BCIDX	75	117	Read/write
Status register	BCSTS	76	118	Read/write
Control register	BCCTL	77	119	Read/write
Error address register	BCERR	78	120	Read only
Flush backup tag store	BCFBTS	79	121	Write only
Flush primary tag store	BCFPTS	7A	122	Write only

The following sections show the contents of each register.

4.1.3.5.1 Backup Cache Backup Tag Store (BCBTS)—EPR 113

The backup cache backup tag store (BCBTS) register is used to access the backup cache tag store, valid bits, and parity bits. The tag store tag, valid bits, and parity may be written explicitly using an EPR write (MTPR) of the BCBTS register; they may be read using an EPR read (MFPR) of the BCBTS register.

Figure 4–13 shows the format for the register. Table 4–4 lists bit descriptions.

On an EPR read of the BCBTS register, the C-chip responds according to the format in the figure. The backup tag store row and column index fields in the backup cache index (BCIDX) register bits <16:6> are used as the index to the tag array. So the BCIDX register must have been previously written using an EPR write (MTPR), to ensure predictable results from the EPR read (MFPR) of the BCBTS register.

On an EPR write of the BCBTS register, the C-chip writes the data into the tag store according to the format shown in the next figure. The backup tag store row and column index fields of the BCIDX register are used as the index to the tag array. So the BCIDX register must have been previously written using an EPR write (MTPR), to ensure predictable results from the EPR write (MFPR) of the BCBTS register.

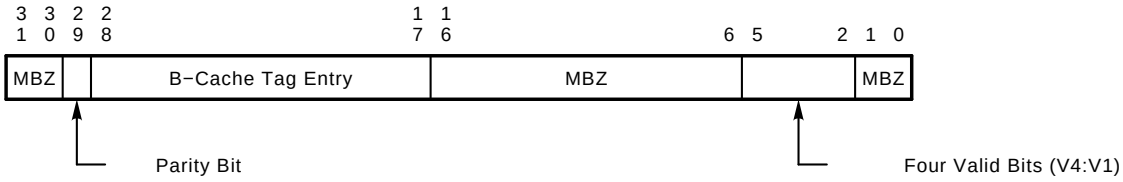


Figure 4–13 Backup Cache Backup Tag Store Register (BCBTS)—(EPR 113 $_{10}$71$_{16}$)

Table 4–4 Backup Cache Backup Tag Store Register Bits

Data Bit	Name	Description
<31:30>	MBZ	Read as 0. Writes ignored.
<29>	Parity bit	Read/write. The parity bit corresponding to the odd parity, as calculated on the tag.
<28:17>	B-cache tag	Read/write. Backup cache entry tag. The tag portion of the tag store entry.
<16:6>	MBZ	Read as 0. Writes ignored.
<5:2>	Four valid bits	Read/write. The four valid bits of the tag store entry.
<0:1>	MBZ	Read as 0. Writes ignored.

Table 4–5 shows the correspondence between bits BCBTS<5:2>, the valid bit selected by physical address bits <5:4>, and the subblock number in the tag store.

Table 4–5 Tag Store Subblock Numbers

BCBTS Bit Set	Address <5:4>	Subblock Number
2	00	1
3	01	2
4	10	3
5	11	4

4.1.3.5.2 C-Chip's Primary Cache Tag Store Access, Using BCP1TS and BCP2TS, EPR 114 and 115

Figure 4–14 defines the format of the C-chip's copy of the primary cache tag store.

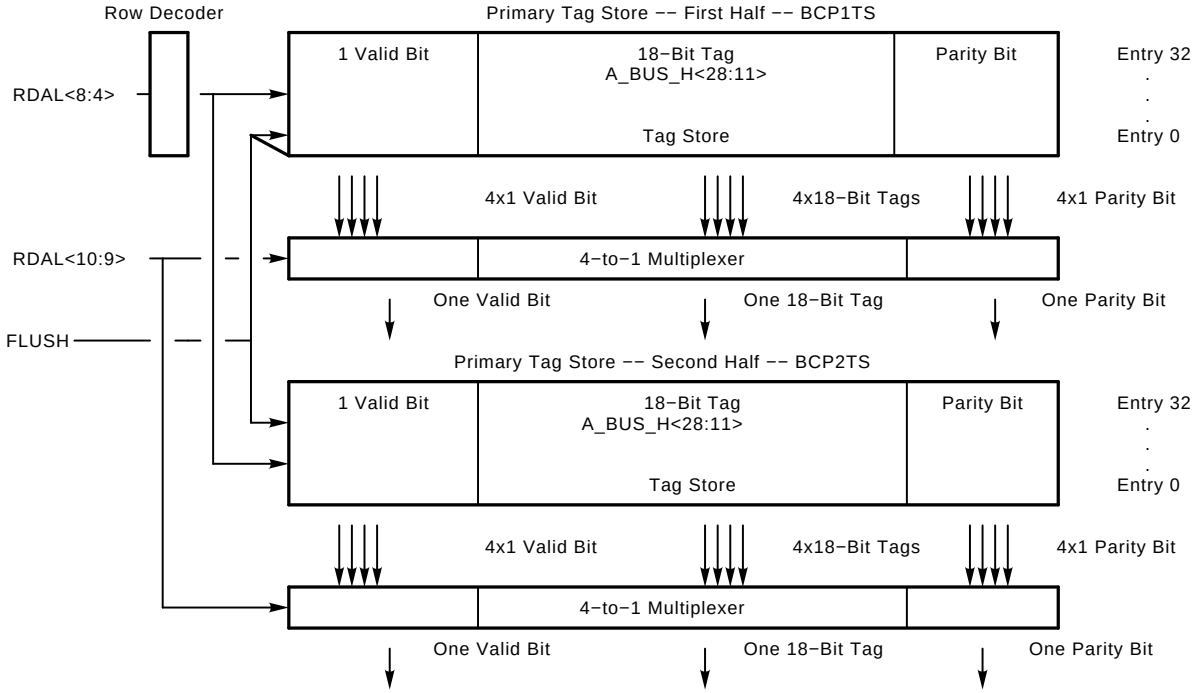


Figure 4–14 The Primary Cache Tag Store ~~C~~-Chip Copy

The backup cache primary tag store contains one tag and one valid bit for each quadword block in the primary cache. There are 256 quadword blocks in the primary cache, so the primary tag store contains 256 entries of 20 bits each. Each entry consists of an 18-bit tag (bits <28:11> of the physical address), one valid bit, and one parity bit.

Figure 4–15 defines the format of the VAX physical address as used in the C-chip's primary tag store addressing during external processor operations. The C-chip copy of the primary tag store is organized in two banks, with 32 rows and 4 columns each. The tag store row is indexed using bits <8:4> of the address. The tag store column is indexed using bits <10:9> of the address. On a primary tag store access, both halves of the tag store are accessed and a hit is calculated separately in each half.

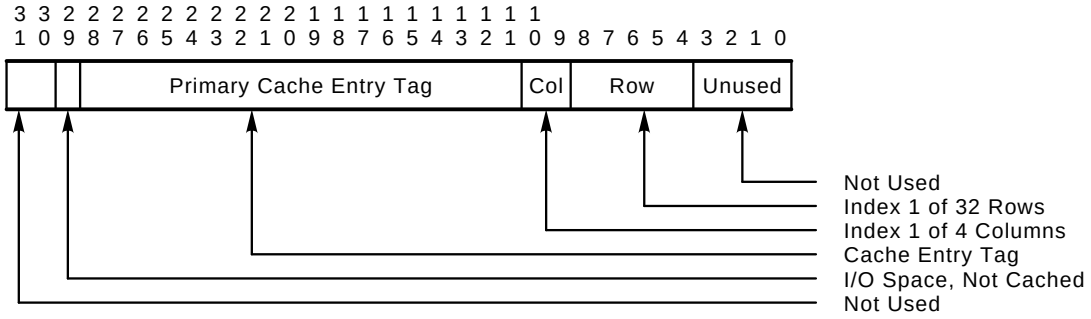


Figure 4–15 VAX Physical Address in C-Chip's Primary Tag Store Addressing (EPR Operations)

The tag store tag, valid bit, and parity may be written explicitly using an EPR write (MTPR) of the tag store; they may be read using an EPR read (MFPR) of the tag store.

EPR 114 (BCP1TS) is the access to the first bank of the C-chip's copy of the primary tag store. EPR 115 (BC2TS) is the access to the second bank.

On an EPR read (MFPR) of the C-chip's copy of the primary tag store, the C-chip responds by driving the data bus according to the format shown in Figure 4–16. The primary cache tag store row and column index fields of the backup cache index (BCIDX) register, bits <10:4>, are used as the index to the tag array. So the BCIDX register must have been previously written using an EPR write (MTPR), to ensure predictable results from the EPR read (MFPR) of the tag store.

On an EPR write (MTPR) of the C-chip's copy of the primary tag store, the C-chip writes the contents of the data bus into the tag store according to the format in the next figure. Again, the primary cache tag store row and column index fields of the BCIDX register are used as the index to the tag array. So the BCIDX register must have been previously written using an EPR write (MTPR), to ensure predictable results from the EPR write (MTPR) of the tag store. Table 4–6 lists the bit descriptions of the primary tag store register.

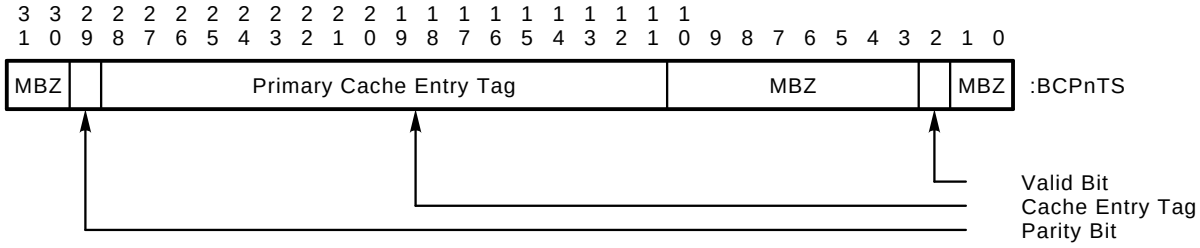


Figure 4–16 Data Bus Format to Access the Primary Tag Store (C-Chip Copy)

Table 4–6 Primary Tag Store Register Bits

Data bit	Name	Description
<31:30>	MBZ	Read as 0. Write as 0.
<29>	Parity bit	The parity bit corresponding to the odd parity as calculated on the tag.
<28:11>	Cache entry tag	The tag portion of the tag store entry.
<10:3>	MBZ	Read as 0. Write as 0.
<2>	Valid bit	The valid bit of the tag store entry.
<1:0>	MBZ	Read as 0. Write as 0.

4.1.3.5.3 Backup Cache Refresh Register (BCRFR)—EPR 116

The backup cache refresh pointer register (BCRFR) contains separate addresses to refresh the backup tag store and the primary tag store. Bits BCRFR<16:9> contain the backup tag store refresh address, which corresponds to the backup tag store row index. Bits BCRFR<8:4> contain the primary tag store refresh address, which corresponds to the primary tag store row index. Both tag stores are refreshed at the same time, when the contents of the register is written with an MTPR. Figure 4–17 shows the format for the BCRFR register. Table 4–7 lists bit descriptions.

When the enable_refresh status bit (BCCTL<3>) is set and a refresh is done, each refresh address field is incremented separately. In this manner, the C-chip's primary tag store is completely refreshed after 32 refresh microcycles, and the backup tag store is completely refreshed after 256 refresh microcycles.

When the enable_refresh bit (BCCTL<3>) is not set, the refresh addresses are only changed explicitly through an EPR write (MTPR). The tag store rows are only refreshed when they are accessed explicitly through reads, writes, EPR reads, or EPR writes. In addition, the BCRFR register is used instead of the backup cache index (BCIDX) register to access the backup tag store and the C-chip's primary tag store during EPR operations on those registers.

The BCRFR register may be written using an EPR write (MTPR) or read using EPR read (MFPR). If enable_refresh (BCCTL<3>) is set when the EPR operation is done, the result of the operation is unpredictable.

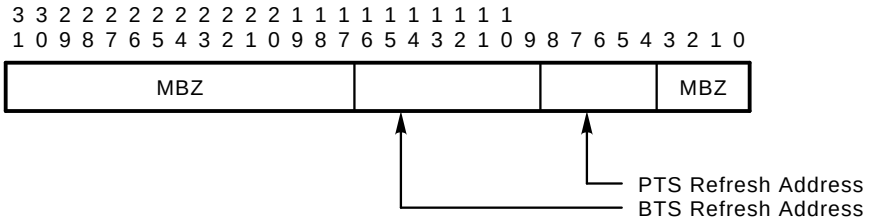


Figure 4–17 C-Chip Refresh Register (BCRFR)—(EPR 116 $_{10}$ 74_{16})

Table 4–7 C-Chip Refresh Register Bits

Data Bit	Name	Description
<31:17>	MBZ	Read as 0. Write as 0.
<16:9>	Backup cache tag store refresh address	This field contains the row address of the backup tag store. The field is incremented each time a refresh is done, if enable_refresh (BCCTL<3>) is set.
<8:4>	Primary cache tag store refresh address	This field contains the row address of the primary cache tag store. The field is incremented each time a refresh is done, if enable_refresh (BCCTL<3>) is set. Note: both halves of the C-chips' primary cache tag store are refreshed.
<3:0>	MBZ	Read as 0. Write as 0.

4.1.3.5.4 Backup Cache Index Register (BCIDX)—EPR 117

The backup cache index register (BCIDX) is used to access the backup tag store and the C-chip's copy of the primary tag store through EPR reads (MFPR) and writes (MTPR). When the backup tag store is accessed, the bits that correspond to the backup tag store index are used. When the primary tag store is accessed, the bits that correspond to the primary tag store index are used. Figures 4–18 and 4–19 show the formats for the backup and primary tag store registers, respectively. Tables 4–8 and 4–9 list the bit descriptions.

The entire BCIDX register may be read using an MFPR, while writes (MTPR) to BCIDX only modify bits <16:4>.

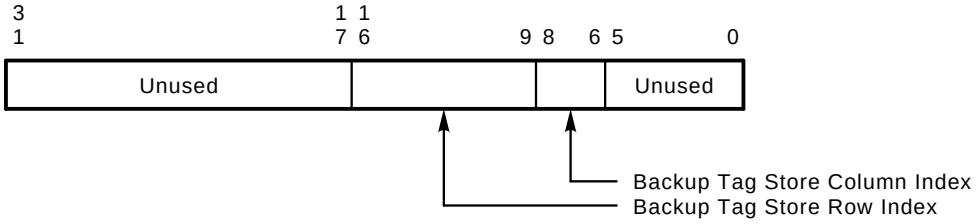


Figure 4–18 Backup Cache Index Register as Used for Backup Cache Tag Store

Table 4–8 Backup Cache Index Register as used for Backup Cache Tag

Data Bit	Name	Description
<31:17>	Unused	Read as 0. Writes ignored.
<16:9>	Backup tag store row index	This field is used as the backup tag store row index during an EPR read (MFPR) or an EPR write (MTPR) of the backup tag store (through the use of BCBTS (EPR 113)).
<8:6>	Backup tag store column index	This field is used as the backup tag store column index during an EPR read (MFPR) or an EPR write (MTPR) of the backup tag store (though the use of BCBTS (EPR 113)).
<5:0>	Unused	Read as 0. Writes ignored.

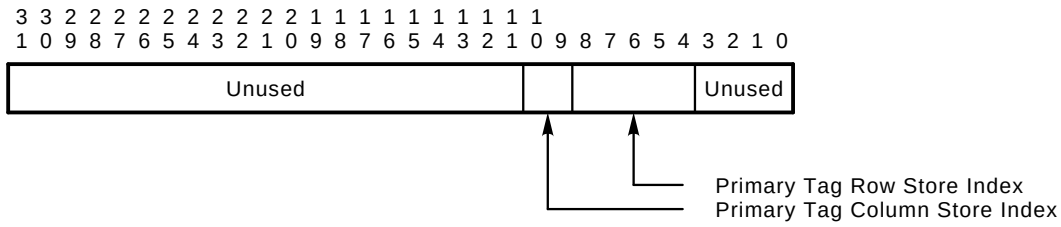


Figure 4–19 Backup Cache Index Register as Used for Primary Cache Tag Store

Table 4–9 Backup Cache Index Register as Used for Primary Cache

Data Bit	Name	Description
<31:11>	Unused	Read as 0. Writes ignored.
<10:9>	Primary tag store column index	This field is used as the primary tag store column index during an EPR read (MFPR) or an EPR write (MTPR) of the backup tag store (through the use of BCP1TS or BCP2TS).
<8:4>	Primary tag store row index	This field is used as the primary tag store row index during an EPR read (MFPR) or an EPR write (MTPR) of the backup tag store (through the use of BCP1TS or BCP2TS).
<3:0>	Unused	Read as 0. Writes ignored.

4.1.3.5.5 Backup Cache Status Register (BCSTS)—EPR 118

The backup cache status (BCSTS) register may be read using an EPR read (MFPR). All bits are writable only by hardware, with the exception of status_lock (BCSTS<0>) which may be cleared using an EPR write (MTPR). Figure 4–20 shows the format for the BCSTS register. Table 4–10 lists bit descriptions.

During normal operation, the BCSTS register is loaded during every memory read or memory write RDAL transaction, when the backup tag store is accessed and parity is calculated. The BCSTS register is loaded during DMA transactions recognized by the C-chip—specifically, DMA cache fill and memory write. In addition, the BCSTS register is loaded during every microcycle used to service an invalidate bus (I bus) request.

The BCSTS register load is disabled when the status_lock (BCSTS<0>) bit is set. In addition, the error address register load is disabled and both tag stores are disabled when the status_lock bit is set. The status_lock bit is set if one of the tag stores produces a parity error or if a RDAL bus error occurs. This allows the CPU to examine the state the C-chip was in when the error occurred. The status_lock (BCSTS<0>) bit is only cleared through an EPR write (MTPR) of the BCSTS register.

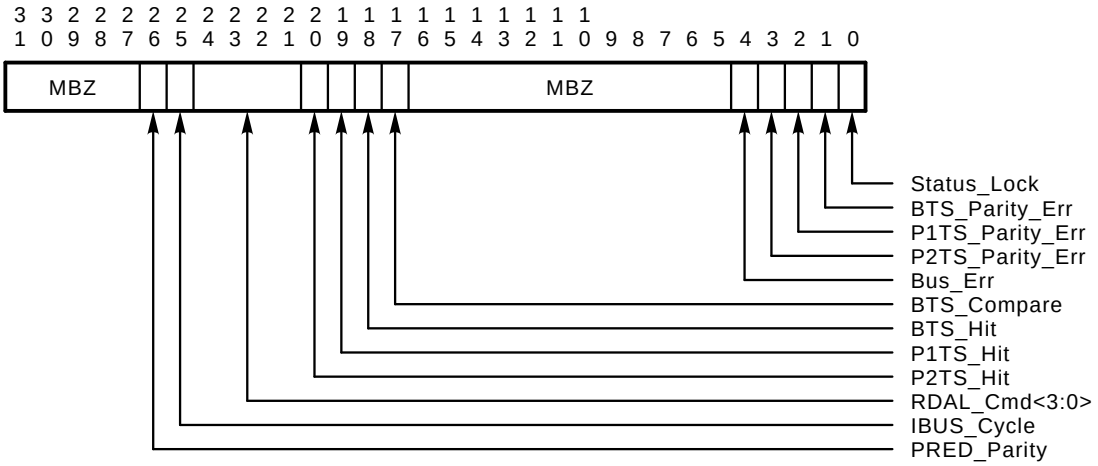


Figure 4–20 Backup Cache Status Register (BCSTS)—(EPR 118_{10} 76_{16})

Table 4–10 Backup Cache Status Register Bits

Data Bit	Name	Description
<31:27>	MBZ	Read as 0. Writes ignored.
<26>	Pred_parity	Predicted parity. The output of the predicted parity generator is loaded into this bit whenever the BCSTS register is loaded.
<25>	Ibus_cycle	Invalidate bus cycle. This bit is set when the BCSTS register is loaded during a microcycle dedicated to servicing an invalidate bus (I-bus) request.
<24:21>	RDAL_cmd<3:0>	RDAL bus command type. This field stores the last non-EPR RDAL command. The field is unpredictable if the IBUS_CYCLE (<25>) bit is set.
<20>	P2TS_hit	Primary tag store 2nd bank hit. This field stores the result of the hit calculation from the last access of the second half of the primary tag store.
<19>	P1TS_hit	Primary tag store 1st bank hit. This field stores the result of the hit calculation from the last access of the first half of the primary tag store.
<18>	BTS_hit	Backup tag store hit. This field stores the result of the backup tag store hit calculation from the last backup tag store access.
<17>	BTS_compare	Backup tag store. This field stores the result of the tag comparison from the last backup tag store access.
<16:5>	MBZ	Read as 0. Writes ignored.

Table 4–10 (Cont.) Backup Cache Status Register Bits

Data Bit	Name	Description
<4>	BUS_ERR	RDAL bus error. This bit is set when an error occurs on the RDAL that may corrupt the cache RAM data. These errors may occur during read miss, write, or fill transactions. Such an error would not happen during a read to I/O space. See the section on Errors for more detail. When bus_err is set, status_lock (BCSTS<0>) is also set, which disables the backup tag store and the primary tag store copy. bus_err is cleared when the status_lock bit is cleared through an EPR write (MTPR), and also during reset.
<3>	P2TS_parity_err	Primary tag store 2nd bank parity error. This bit is set when a parity error occurs in the second half of the primary tag store. The bit is not loaded into the status register unless enable_PTS (BCCTL<2>) is set. When P2TS_parity_err is set, status_lock (BCSTS<0>) is also set. P2TS_parity_err is cleared when the STATUS_LOCK bit is cleared through an EPR write (MTPR), and also during reset.
<2>	P1TS_parity_err	Primary tag store 1st bank parity error. This bit is set when a parity error occurs in the first half of the primary tag store. The bit is not loaded into the status register unless enable_PTS (BCCTL<2>) is set. When P1TS_parity_err is set, status_lock (BCSTS<0>) is also set. P1TS_parity_err is cleared when the STATUS_LOCK bit is cleared through an EPR write (MTPR), and also during reset.
<1>	BTS_parity_err	Backup tag store parity error. This bit contains the result of the last access of the backup tag store. The bit is set when a parity error occurs in the backup tag store. The bit is not loaded into the BCSTS register, unless enable_BTS (BCCTL<1>) is set and force_Bhit (BCCTL<0>) is not set. When BTS_parity_err is set, status_lock (BCSTS<0>) is also set. BTS_parity_err is cleared when the status_lock bit is cleared through an EPR write (MTPR), and also during reset.
<0>	Status_lock	This bit is set by hardware when a parity error occurs in either the backup tag store or the primary tag store, or when an RDAL bus error occurs. Setting this bit locks the BCSTS register and the backup cache error address register (BCERR) against further modification until status_lock is cleared. Both tag stores are disabled when the status_lock bit is set. The bit is cleared by an EPR write (MTPR) of the BCSTS register, using the format shown in Figure 4–20. A 1 must be written to the status_lock location in order to clear this bit. The status_lock bit is the only externally-writable bit in the register. When the status_lock bit is cleared, bus_err, BTS_parity_err, P1TS_parity_err and P2TS_parity_err are cleared as well. status_lock is cleared during reset.

The following list provides some information on interpreting the contents of the status register:

- **Ibus_cycle bit is set:** The RDAL command field (BCSTS<24:21>) is unpredictable since an RDAL command is not necessarily being processed during an invalidate bus cycle. The hit and error fields show the results from the access of the backup and primary tag stores.
- **RDAL_cmd is a memory read or write:** The results of the backup cache tag store access are given by BTS_parity_err, BTS_hit, and BTS_compare. The results of the primary tag store access are given by P1TS_hit and P2TS_hit. The primary tag store error bits have no meaning and are 0, because parity is not calculated on the contents of the primary tag store copy during these transactions; thus, the primary tag store error bits are not loaded.
- **RDAL_cmd is a DMA cache fill or memory write with DMG asserted:** The results of the read of both tag stores are contained in the status bits.

  Note that it is not possible to tell if DMG was asserted using the contents of the BCSTS register.

Table 4–11 summarizes which bits are loaded during each C-chip transaction.

Table 4–11 Status Bits Loaded in BCSTS During C-Chip Transactions

Cycle Type Loaded	BTS Error Bit Loaded	PTS Error Bit Loaded	All Other Bits
Read	Yes	No	Yes
Write	Yes	No	Yes
DMA fill	Yes	Yes	Yes
DMA write	Yes	Yes	Yes
I-bus	Yes	Yes	Yes
EPR read/write	No	No	No

4.1.3.5.6 Backup Cache Control Register (BCCTL)—EPR 119

The backup cache control register (BCCTL) contains several control bits that allow logic external to the C-chip to control the actions of the C-chip. The register may be read using an EPR read (MFPR). The register may be written using an EPR write (MTPR). All the bits are written at once, so all bits must contain valid data when the EPR write is issued.

Three bits of the register are hardware-writable: enable_BTS (BCCTL<1>), enable_ PTS(BCCTL<2>), and force_Bhit(BCCTL<0>). The bits are only written by hardware when RESET_L is asserted; they are written as shown in Figure 4–21. Table 4–12 lists the bit descriptions.

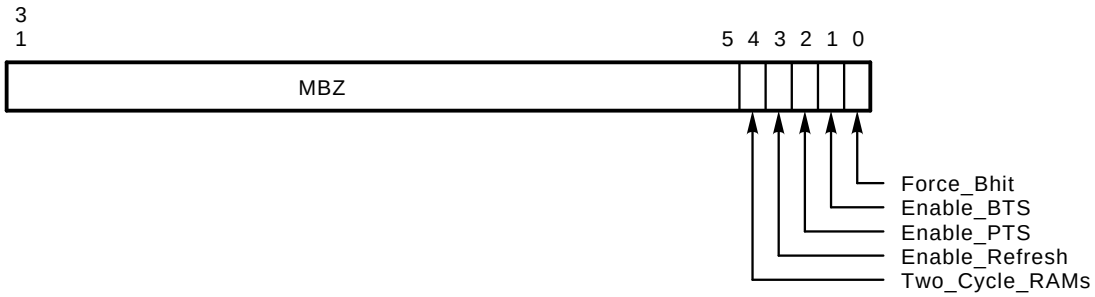


Figure 4–21 Backup Cache Control Register (BCCTL)—(EPR 119 $_{10}$ 77$_{16}$)

Table 4–12 Backup Cache Control Register Bits

Data Bit	Name	Description
<31:5>	MBZ	Read as 0. Write as 0.
<4>	Two_cycle_RAMs	This bit indicates the speed of the cache RAMs, so the C-chip knows how many microcycles are needed to access the RAMs. For the KA670, the console macrocode should set this bit to 0, which means it does not take extra microcycles to access the backup cache RAMs. For example, no slip cycles are needed. When the backup cache is enabled, two_cycle_RAMs should be the same in the C-chip control register and the memory interface. (See Section 4.2.1.3.6)
<3>	Enable_refresh	When this bit is 1, the automatic refresh proceeds normally; each time a refresh is done, the refresh counter is incremented. When the bit is 0, automatic refreshing of the tag stores is disabled, and the backup cache refresh register (BCRFR) incrementer is disabled. The refresh register may drive the invalidate address bus (IA bus, internal to the C-chip), but the refresh counter will never be incremented. This enables explicit control of the BCRFR register through EPR writes (MTPR). Beware that tag store data may be corrupted if each row of each tag store is not refreshed at least once every millisecond. In addition, if enable_refresh (BCCTL<3>) is not set, the BCRFR register is used instead of the backup cache index (BCIDX) register during EPR accesses of the primary tag store and the backup tag store. This feature allows testing of the path from the BCRFR to the tag stores.

Table 4–12 (Cont.) Backup Cache Control Register Bits

Data Bit	Name	Description
<2>	Enable_PTS	Enable primary tag store. When this bit is clear, the C-chip copy of the primary tag store is disabled. Primary cache tag store parity errors are not loaded into the backup cache status (BCSTS) register and do not cause the assertion of SERR_IRQ_L as normal. Enable_PTS is cleared by hardware only when RESET_L is asserted. Enable_PTS must be set when the CPU's primary cache is enabled, to ensure proper invalidate filtering. While the primary tag store is disabled, its contents *may change* as a result of RDAL operations. If the primary tag store has been disabled, it must be flushed through an EPR write (MTPR) of the backup cache flush primary tag state (BCFPTS) register before it is reenabled, to ensure correct operation. The CPU primary cache must also be flushed.
<1>	Enable_BTS	Enable backup tag store. When this bit is clear, the backup cache is disabled. All reads produce a cache miss; no writes are done. When the bit is clear, the backup tag store does not contribute to the calculation of an invalidate hit on the I-bus; in other words, only the access of the primary cache produces an invalidate hit. Backup cache tag store parity errors are not loaded into the backup cache status (BCSTS) register and do not cause the assertion of SERR_IRQ_L as normal. Enable_BTS is cleared (reset to 0) by hardware only when RESET_L is asserted. Enable_BTS should not be reset to 0 during normal operation. If force_Bhit (BCCTL<0>) is set and enable_BTS (BCCTL<1>) is clear, the response of the backup tag store is unpredictable. While the backup tag store is disabled, its contents *may change* as a result of RDAL operations. If the backup tag store has been disabled, it must be flushed through an EPR write (MTPR) of the backup cache flush backup tag store (BCFBTS) before it is reenabled, to ensure correct operation.

Table 4–12 (Cont.) Backup Cache Control Register Bits

Data Bit	Name	Description
<0>	Force_Bhit	Force backup hit. When this bit is set, all non-I/O space backup tag store accesses produce a cache hit, including read_lock accesses. Backup cache tag store parity errors are not reported. All I-bus requests result in an invalidate hit, regardless of the contents of the backup tag store. If enable_BTS (BCCTL<1>) is clear and force_Bhit (BCCTL<0>) is set, the backup tag store response is unpredictable. If a primary tag store parity error occurs, causing status_lock (BCSTS<0>) to be set, the backup cache is not disabled as normal; the force_Bhit condition overrides the status_lock condition. Similarly, the backup cache is not disabled if a bus_err (BCCTL<4>) occurs, causing status_lock to be set; the force_Bhit condition overrides the status_lock condition. When the C-chip is in force_Bhit mode, the cache RAM data is written for each non-I/O space memory write and read on every non-I/O space memory read. The state of the backup tag store, however, is unpredictable; it must be flushed before it is returned to normal mode. The backup tag store must also be initialized, if this has not occurred yet. Force_Bhit is cleared during reset and should be set to 1 by diagnostics only.

4.1.3.6 Maintaining Primary Cache Consistency

Any state change to the primary cache must be reflected in the C-chip copy of the primary tag store. During normal operation, this is done automatically by the C-chip when a cacheable read occurs. If the CPU copy of the primary cache is flushed, the C-chip copy of the primary tag store should also be flushed.

When the primary cache is turned off, the state of the C-chip copy of the primary tag store is irrelevant. If the C-chip copy is enabled, some I-bus requests may generate invalidates on the RDAL as a result of valid bits that were set in the C-chip copy of the primary tag store. Those invalidates are inconsequential to the CPU, since the primary cache is turned off and will be flushed when turned back on.

If the C-chip copy is disabled, accessing the primary tag store copy never causes an I-bus invalidate hit. As a result, the memory interface does not generate any invalidates for the primary cache on the RDAL. Therefore, the state of the C-chip copy of the primary tag store is irrelevant when the primary cache is turned off, although less RDAL traffic is generated if the primary cache copy is also disabled.

Table 4–13 is a matrix showing the proper sequence of events for reenabling a disabled tag store. The matrix assumes that each tag store has been properly initialized. It also assumes that status_lock (BCSTS<0>) is not set. If status_lock is set, the sequence in Chapter 8 should be followed.

Table 4–13 Reenabling a Turned-Off Tag Store

Bits <2:1> in the BCCTL	CPU Primary Cache Off	CPU Primary Cache On
^Enable_BTS, ^Enable_PTS	Everything is off: Flush backup tag store. Flush primary tag store. Write enable_BTS, enable_PTS. Flush and turn on primary cache.	**Illegal** if the I-bus is being used. Primary tag store must be on if the primary cache is on. If the I-bus is not being used, take the actions in the following box.
^Enable_BTS, Enable_PTS	Backup tag store and primary cache are off: Flush backup tag store. Flush primary tag store. Write enable_BTS. Flush and turn on primary cache.	Backup tag store is off: Flush the backup tag store. Write enable_BTS.
Enable_BTS, Enable_PTS	Primary tag store and primary cache are off: Flush primary tag store. Write enable_PTS. Flush and turn on primary cache.	**Illegal** if the I-bus is being used. Primary tag store must be on if the primary cache is on. If the I-bus is not being used, take the actions in the previous box.
Enable_BTS, Enable_PTS	Primary tag store is on, and primary cache is off: Flush primary tag store. Flush and turn on primary cache.	Normal state.

4.1.3.6.1 Backup Cache Error Address Register (BCERR)—EPR 120

The backup cache error address register (BCERR) is a read-only register. It is loaded by hardware every time the backup cache status (BCSTS) reigster is loaded. The BCERR register contains the address of the current transaction. The first error causes the status_lock (BCSTS<0>) bit to be set; this action locks the BCERR register against further writes, regardless of subsequent errors, until the status_lock bit is cleared.

The error address register may be read using an EPR read (MFPR) according to the format shown in Figure 4–22. Table 4–14 lists the bit descriptions.

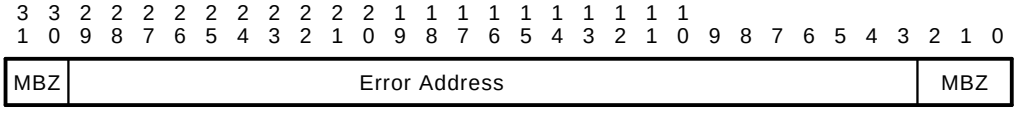


Figure 4–22 Backup Cache C-Chip Error Address Register (EPR 120_{10} 78_{16})

Table 4–14 Backup Cache C-Chip Error Address Register Bits

Data Bit	Name	Description
<31:30>	MBZ	Read as 0.
<29:3>	Error address	This field contains the physical address of the current transaction.
<2:0>	MBZ	Read as 0.

When the BCERR register is loaded during an I-bus transaction, bits BCERR<29> and BCERR<3> are both 0s. This is because the I-bus only uses bits <28:4> of the physical address. The other bits are 0 by default.

The BCERR register is not microcode-writable. If an EPR write (MTPR) of the BCERR register is attempted, the RDAL cycle completes as normal but does not write the register. For example, writes of the BCERR register are ignored.

The address contained in the BCERR register when a BUS_ERR (BCSTS<4>) occurs is unpredictable. The RDAL error may occur several cycles after the address corresponding to the transaction was driven onto the bus. In the meantime, the BCERR register may have been overwritten by an I-bus transaction.

4.1.3.6.2 Backup Cache Flush Backup Tag Store Register (BCFBTS)—EPR 121

The backup cache flush backup tag store (BCFBTS) register is a write-only register. Figure 4–23 shows the register's format.

An EPR write (MTPR) of the BCFBTS register clears all the valid bits in the backup tag store. The C-chip ignores the contents of the RDAL data bus during the transaction. The write to the register causes an immediate flush of all the valid bits in the backup tag store.

An EPR read of the BCFTS register causes the C-chip to complete the normal RDAL cycle for an EPR read (MFPR). For example, reads of the BCFBTS register return unpredictable data.

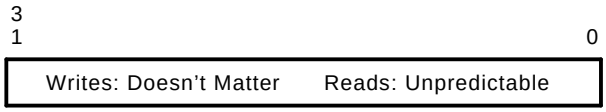


Figure 4–23 Backup Cache Flush Backup Tag Store Register (EPR 121 $_{10}$79$_{16}$)

4.1.3.6.3 Backup Cache Flush Primary Tag Store Register (BCFPTS)—EPR 122

The backup cache flush primary tag store (BCFPTS) register is a write-only register. Figure 4–24 shows the format.

An EPR write (MTPR) of the BCFPTS register clears all the valid bits in the primary tag store. The C-chip ignores the contents of the RDAL data bus during the transaction. The write to the register causes an immediate flush of all the valid bits in the C-chip's copy of the primary cache tag store.

An EPR read of the BCFPTS register causes the C-chip to complete the RDAL cycle as normal for an EPR read (MFPR). For example, reads of the BCFPTS register return unpredictable data.

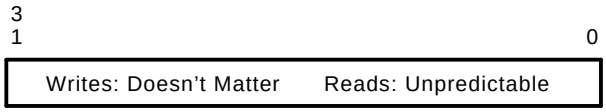


Figure 4–24 Backup Cache Flush Primary Tag Store Register (EPR 122_{10} $7A_{16}$)

4.1.3.7 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.

4.1.3.7.1 Control of the Cache

Normal operational control of the backup cache and primary tag store in the C-chip is provided through writes to the backup cache control (BCCTL) register. 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 the BCFBTS and BCFPTS registers, respectively.

4.1.3.7.2 Error Recovery

When the C-chip detects an error, the C-chip latches error information. This information is available by reading the BCSTS and BCERR registers. 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. The new entry is created with a write to the BCIDX register, followed by a write to the BCBTS, BCP1TS, or BCP2TS register.

See Chapter 8 for a detailed discussion of error recovery procedures.

4.1.3.7.3 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 the BCIDX register, followed by a write to the BCBTS, BCP1TS, or BCP2TS register.

As part of cache initialization, cache refresh must be enabled, and the cache RAM speed must be specified by writing to the backup cache control (BCCTL) register. The console macrocode sets the RAM speed for 1 cycle.

4.1.3.7.4 Diagnostics

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

4.2 KA670 Main Memory System

The KA670 includes a main memory controller implemented as part of a VLSI chip called the G-chip. The KA670 main memory controller communicates with the MS670 memory boards over the MS670 memory interconnect, which uses the G-chip memory interconnect (GMI) for the address, control, and data lines. The contoller supports up to four MS670 memory boards.

4.2.1 G-Chip Memory Controller

As a two-port memory controller, the G-chip interfaces the RDAL bus and the CP bus to a memory subsystem over a private interconnect, the GMI. It also serves as an adapter between the RDAL bus and the CP bus.

4.2.1.1 G-Chip Port

The G-chip port interfaces with the CPU, the C-chip, and the backup cache. The G-chip port also supports the defined synchronous protocols for DMA. The following sections describe the main features of the port.

4.2.1.1.1 G-Chip CPU Port Addressing

The G-chip regards all addresses from the RDAL bus with bit<29> equal to 0 and a non-EPR read or write command, as memory addresses. The G-chip responds to all I/O addresses from the RDAL bus. Transactions with address bit <29> equal to 1 are transferred to the CP bus by the G-chip if the address does not correspond to any of its internal registers.

4.2.1.1.2 G-Chip EPR decoder

The G-chip supports EPR reads and writes to the system support chip (SSC) on the CP bus. These are the only EPRs the G-chip responds to. For EPR addresses that are not in the SSC set, G-chip implements a timeout function. The G-chip decodes SSC EPR numbers from the RDAL address bus for the TOY clock register, the I/O reset register, the console storage registers, and the console registers; it performs the corresponding operations on the CP bus.

If the EPR address passed to the CP bus is not available, the EPR transaction will timeout on the CP bus and the RDAL error signal is asserted to abort the CPU transaction. For this exception, no error flags are set and no addresses are saved. The G-chip supports EPRs 27 to 35 and 55_{10} on the CP bus.

4.2.1.2 G-Chip Write Buffers

The G-chip improves write performance of the RDAL bus with a write buffer or queue. The queue consists of a 4-quadword element ring buffer, each with an address tag. Each element stores valid data that corresponds to the valid byte masks.

The address tags are content-addressable memories (CAMs). The content of CAMS is used to look up and compare with a memory read address, to determine if the data to be read is an element in the queue. If the address hits in the queue, then all the elements that matched are flushed to memory before the read of memory. No CPU-to-CPU-memory write transaction data packing is supported by the queue, because the GMI continuously scans the queue for elements to retire.

Data is loaded sequentially into the queue and is unloaded by the GMI port in the same order. To ensure coherent operation of the system, the queue is flushed under the following circumstances:

- A clear write buffer transaction by the CPU (P-chip)

- A read lock by a device on the CP bus
- An I/O write to an address on the CP bus
- An EPR write to a register on the CP bus
- A memory read address that hits in the queue
- An interrupt vector read from a device on the CP bus

The queue is cleared when RESETL asserts. For example, all the valid entries are invalidated.

4.2.1.3 G-Chip Registers

The G-chip has control and status registers (CSRs) that can be read or written only from the port (by the CPU). They are all initialized on power-up reset, unless otherwise mentioned. This is the only type of reset that the G-chip responds to. It does not respond to I/O_RESET (EPR 55) writes to invoke an internal reset. Table 4–15 lists the register names, desciptions, and addresses.

Table 4–15 G-Chip Registers

Register/s	Description	Address
MEMCSR32	Error status register	2008 0180
MEMCSR33	Memory error address register	2008 0184
MEMCSR34	I/O error address register	2008 0188
MEMCSR35	DMA memory error register	2008 018C
MEMCSR36	Mode control and diagnostic register	2008 0190

4.2.1.3.1 G-Chip Register Addressing

Because there is one G-chip for each CPU, the addresses for all MEMCSRs are fixed. Write operations to read-only registers do not cause a CPU machine check and are responded to as a normal operation. However, the operation does not alter the contents of any G-chip registers.

4.2.1.3.2 G-Chip System Error Status Register (MEMCSR32)

The G-chip reports error information in the MEMCSR32 register. The error flags are cleared by writing a 1 to the respective bits in MEMCSR32. MEMCSR32 is initialized only during power-up reset. Figure 4–25 shows the format. Table 4–16 lists the bit descriptions.

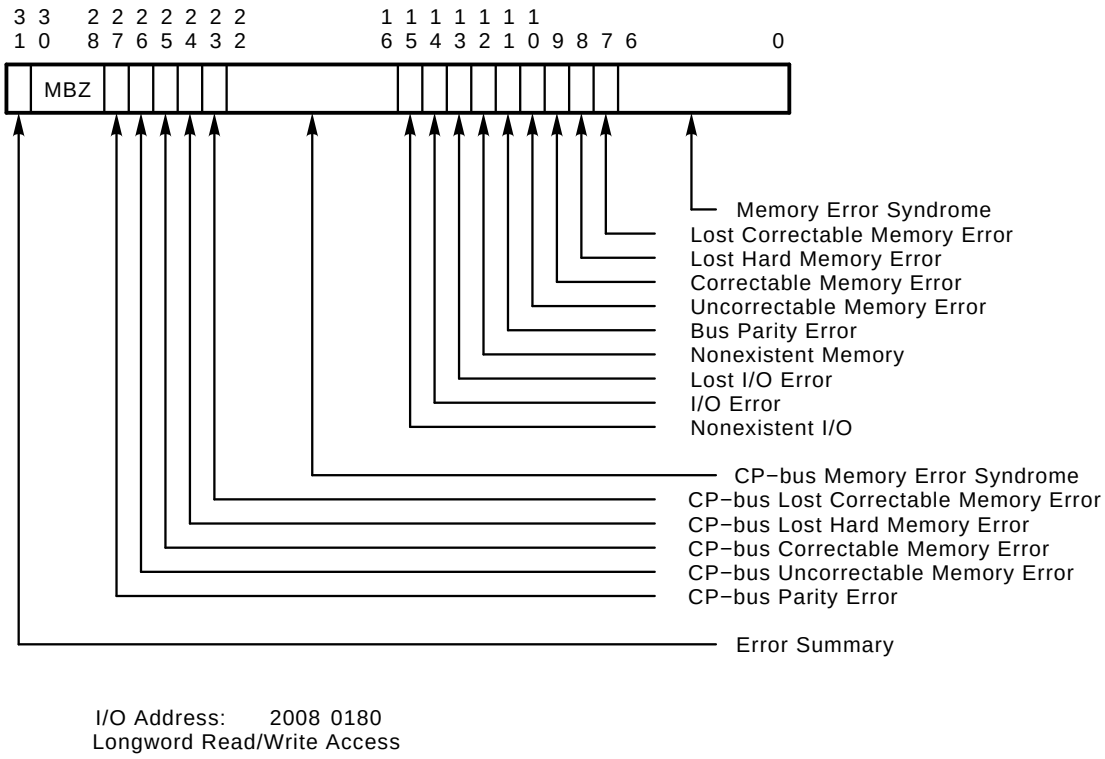


Figure 4–25 G-Chip System Error Status Register (MEMCSR32)

Table 4–16 G-Chip System Error Status Register Bits

MEMCSR32 Data Bit	Name	Description
<31>	Error summary	This read-only bit is set when any error is detected and logged in this register by the G-chip. A 0 is returned when this bit is read, if all the other error bits in this register are 0.
<30:28>	MBZ	Read as 0. Writes have no effect.
<27>	CP bus parity error	This read/write bit is set when a CP bus DAL parity error is detected on a CP bus DMA memory write transaction, if the error address can be saved in MEMCSR35. This bit is cleared by writing a 1. **NOTE** **The CQBIC is the only CP bus DMA device that does not generate or check parity on the CP bus.**

Table 4–16 (Cont.) G-Chip System Error Status Register Bits

MEMCSR32 Data Bit	Name	Description
<26>	CP bus uncorrectable memory error	This read/write bit is set to 1 by an uncorrectable ECC error that occurs during a CP bus DMA memory read or masked write transaction , if the error address can be saved in MEMCSR35. An octaword read is always performed in response to a DMA read request, and this bit may set even if the data is not returned to the CP bus. This bit is cleared by writing a 1.
<25>	CP bus correctable memory error	This read/write bit is set to 1 when a correctable (single-bit) error is detected during a CP bus DMA memory read or masked write transaction, if the error address can be saved in MEMCSR35 and if MEMCSR36<11> is set. This bit is cleared by writing a 1.
<24>	CP bus lost hard memory error	This read/write bit is set to 1 when an uncorrectable ECC error or a CP bus DMA parity error occurs on a transaction initiated by a CP bus DMA master while either <27,26> was set (indicating that MEMCSR35 could not be used). This read/write bit is cleared by writing a 1. When this bit is set, the error and the address of the error are lost.
<23>	CP bus lost correctable memory error	This read/write bit is set to 1 when a correctable ECC error occurs on a transaction initiated by a CP bus DMA master while <25> was set and MEMCSR36<11> was set or MEMCSR36<11> was cleared (indicating that MEMCSR35 could not be used). This read/write bit is cleared by writing a 1. When this bit is set, the error and the address of the error are lost. **NOTE** **Only one of MEMCSR32 bits 27, 26, and 25 can be set at any time, since MEMCSR35 can save only the first error address.**
<22:16>	CP bus memory error syndrome	This read-only field stores the memory error syndrome. The field is loaded when an ECC memory error is detected from CP bus initiated transactions. The priority for logging the syndrome is first error-logged. Subsequent memory error syndromes are not logged until the associated error bits are cleared. This field contains valid data while a correctable or uncorrectable CP bus error bit is set. Writes to this field have no effect.
<15>	G-chip nonexistent I/O	This read/write bit is set if <14> is cleared for G-chip originated I/O transactions to the CP bus which do *not* respond (and hence signal timeout abort errors after the G-chip internal timer overflows). The error address is saved in MEMCSR34. This bit is cleared by writing a 1.

Table 4–16 (Cont.) G-Chip System Error Status Register Bits

MEMCSR32 Data Bit	Name	Description
<14>	G-chip I/O ERROR	This read/write bit is set if <15> is cleared for transactions from the G-chip bus to the CP bus which are terminated by the CPERR signal or by a read parity error, and not by a G-chip timeout abort error. The error address is saved in MEMCSR34. This bit is cleared by writing a 1.
<13>	G-chip LOST I/O ERROR	This read/write bit is set when transactions from the G-chip bus to the CP bus terminate in error, while either the G-chip I/O or nonexistent I/O error bits are set (indicating that MEMCSR34 could not be used). This bit is cleared by writing a 1. **NOTE** **Only one of MEMCSR32 bits 15 or 14 can be set at any time, since MEMCSR34 can save only the first error address.**
<12>	G-chip nonexistent memory address	This read/write bit is set if the error address for G-chip bus transactions to invalid memory addresses can be saved in MEMCSR33. This bit is cleared by writing a 1.
<11>	G-chip bus parity error	This read/write bit is set if the error address for a RDAL parity error from a G-chip memory write transaction can be saved in MEMCSR33. RDAL parity errors are not reported for I/O and external processor register (EPR) write transactions. This bit is cleared by writing a 1.
<10>	G-chip uncorrectable memory error	This read/write bit is set if the error address for an uncorrectable ECC error from a memory read or masked write transaction initiated from the G-chip bus can be saved in MEMCSR33. This bit is cleared by writing a 1.
<9>	G-chip correctable memory error	This read/write bit is set if the error address can be saved in MEMCSR33 and if MEMCSR36<11> is set for a correctable (single-bit) error from a memory read or masked write transaction initiated from the G-chip bus. This bit is cleared by writing a 1.
<8>	G-chip lost hard memory error	This read/write bit is set when either a nonexistent, bus parity, or uncorrectable ECC error occurs as a result of a G-chip bus-initiated transaction while <12, 11, or 10> was set. This read/write bit is cleared by writing a 1. If this bit is set, the address of the error could not be saved in MEMCSR33.

Table 4–16 (Cont.) G-Chip System Error Status Register Bits

MEMCSR32 Data Bit	Name	Description
<7>	G-chip lost correctable memory error	This read/write bit is set when a correctable ECC error occurs from a bus-initiated transaction while <9> and MEMCSR36<11> was set, or while MEMCSR36<11> was cleared (indicating that MEMCSR33 could not be used). This read/write bit is cleared by writing a 1. If this bit is set, the address of the error could not be saved in MEMCSR33. **NOTE** **Only one of MEMCSR32 bits 12, 11, 10, and 9 can be set at any time, since MEMCSR33 can save only the first error address.**
<6:0>	G-chip error syndrome	This read-only field stores the error syndrome and is loaded when a G-chip memory error is detected. The priority for logging the syndrome is first error-logged. Subsequent memory error syndromes are not logged until the associated error bits are cleared. This field contains valid data only when a correctable or uncorrectable G-chip bus error bit is set. Writes to this field have no effect.

4.2.1.3.3 Memory Error Address Register (MEMCSR 33)

MEMCSR33 contains the octaword error address from bus-initiated memory transactions. The address is loaded by the first memory error and is not changed until that error bit is cleared in MEMCSR32. This register is read-only and has valid content only while a corresponding error bit (<12:9>) is set. Figure 4–26 shows the format of the register. Table 4–17 lists the bit descriptions.

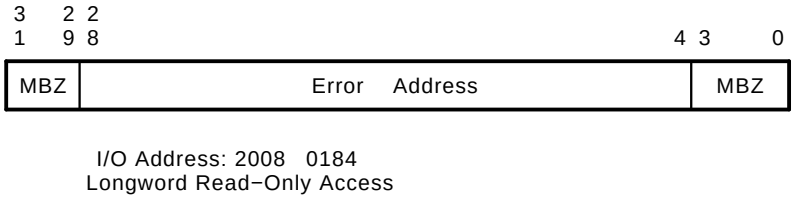


Figure 4–26 G-chip Memory Error Address Register (MEMCSR33)

Table 4–17 Memory Error Address Register Bits

MEMCSR33 Data Bit	Name	Description
<31:29>	MBZ	Read as 0. Writes have no effect.
<28:4>	Error address	Octaword address of the first memory error.
<3:0>	MBZ	Read as 0. Writes have no effect.

4.2.1.3.4 I/O Error Address Register (MEMCSR 34)

MEMCSR34 contains the longword error address of initiated I/O transactions. The address is loaded by the first I/O or nonexistent I/O error and is not changed until that error bit is cleared. This register is read-only and has valid content only while a corresponding error bit (<15:14>) is set.

Note, that since the address is in I/O space, address bit <29> is 1, even though MEMCSR34's bit <29> does not reflect this. Figure 4–27 shows the format. Table 4–18 lists the bit descriptions.

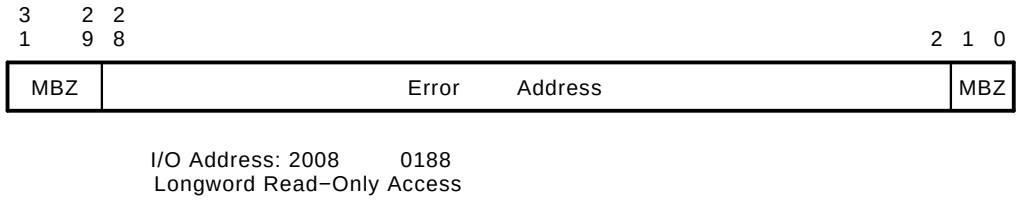


Figure 4–27 G-Chip I/0 Error Address Register (MEMCSR 34)

Table 4–18 G-Chip I/0 Error Address Register Bits

MEMCSR34 Data Bit	Name	Description
<31:29>	MBZ	Read as 0. Writes have no effect.
<28:2>	Error address	Longword address of first initiated I/O error.
<1:0>	MBZ	Read as 0. Writes have no effect.

4.2.1.3.5 CP bus Error Address Register (MEMCSR 35)

MEMCSR35 contains the octaword error address of DMA-initiated transactions from the CP bus. The address is loaded by the first memory error. This address is not changed until that error bit is cleared and another error is logged. This register is read-only and has valid content only while a corresponding error bit (<27:25>) is set. Figure 4–28 shows the format of the register. Table 4–19 lists the bit descriptions.

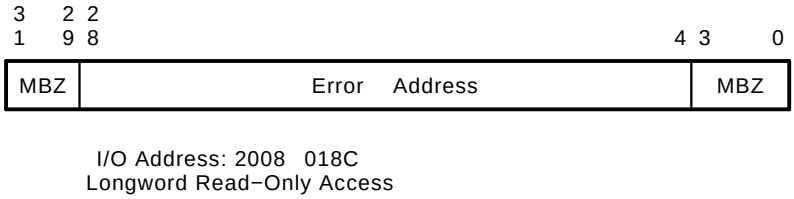


Figure 4–28 CP bus Error Address Register (MEMCSR 35)

Table 4–19 CP Bus Error Address Register Bits

MEMCSR35 Data Bit	Name	Description
<31:29>	MBZ	Read as 0. Writes have no effect.

Table 4–19 (Cont.) CP Bus Error Address Register Bits

MEMCSR35 Data Bit	Name	Description
<28:4>	Error address	Octaword address of first DMA-initiated memory error.
<3:0>	MBZ	Read as 0. Writes have no effect.

4.2.1.3.6 G-Chip Mode Control and Diagnostic Status Register (MEMCSR 36)

The bits in this register control G-chip operating modes. This register also stores diagnostic status information. The MEMCSR36 bits are read/write and are cleared asynchronously with the assertion of RESETL at power-up. Figure 4–29 shows the format of the register. Table 4–20 lists the bit descriptions.

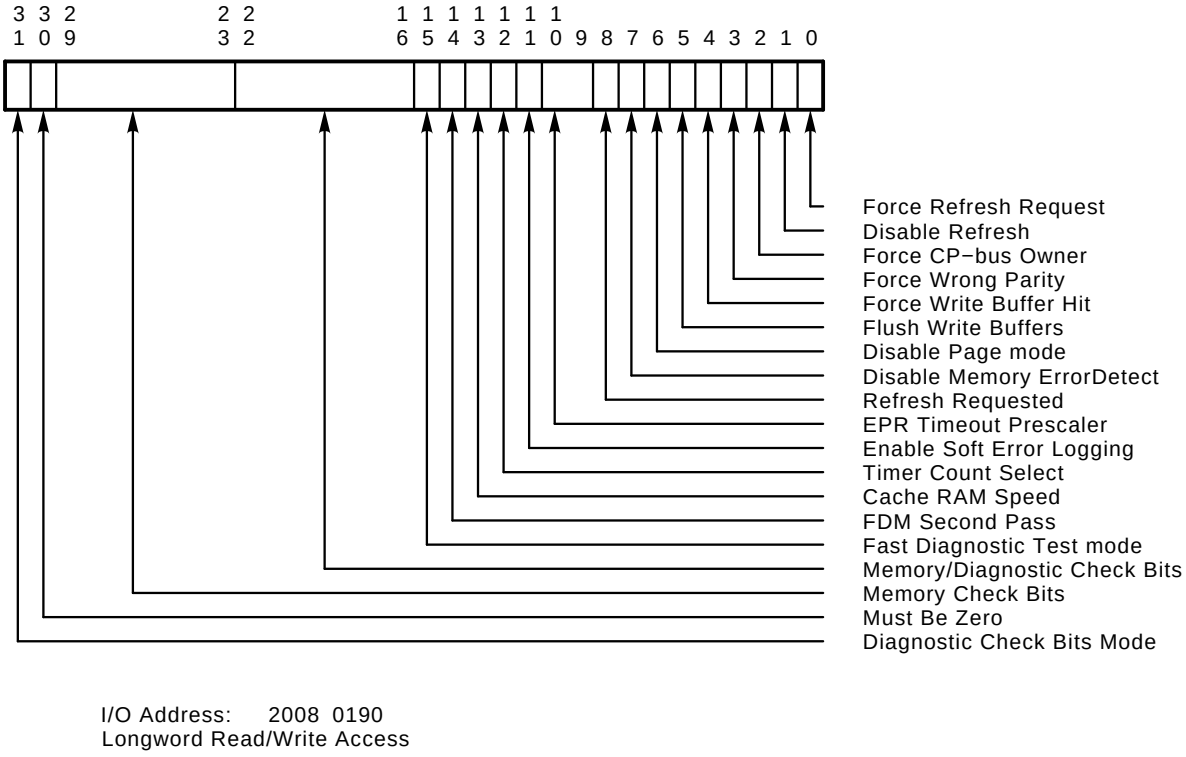


Figure 4–29 G-Chip Mode Control and Diagnostic Status Register (MEMCSR 36)

Table 4–20 G-Chip Mode Control and Diagnostic Status Register Bits

MEMCSR36 Data Bit	Name	Description
<31>	Diagnostic check mode	When set to 1 by a write, this read/write bit enables the contents of MEMCSR36<22:16> to be passed as check bits during a memory write transaction, instead of the normal ECC check bits. This is true unless an RDAL parity error occurred on the write. If an RDAL parity error occurred, the low three check bits of this field are inverted as they are written to memory. When this bit is a 0, the contents of MEMCSR36<22:16> are ignored during memory write transactions. MEMCSR36<22:16> should be written along with this bit.
<30>	Must be zero	Read as 0. Writes have no effect.
<29:23>	Memory check bits	Regardless of the diagnostic check mode bit, the contents of MEMCSR36<29:23> are loaded from ECC check bits for the unaligned longword during a G-chip memory read or a second signature read transaction prior to a MEMCSR36 read. The read check bits for a masked memory write transaction are not latched. When loaded, this bit field is held until the register is read. These bits are read-only and are undefined until a second signature read or any memory read transaction is complete.
<22:16>	Memory/diagnostic check bits	When diagnostic check mode is enabled, this write field substitutes for the check bits generated by the ECC generation logic during memory masked or unmasked write transactions. If a RDAL parity error occurs, the low three check bits are inverted as they are written to memory. If diagnostic check mode is not enabled, the contents of MEMCSR36<22:16> are ignored during memory write transactions. This read field is loaded with the check bits from the ECC check bits for the requested aligned longword of the requested quadword, or for the first of two signature read transactions following a read of MEMCSR36. When loaded the bits are held until the register is read. **NOTE** **The two fields MEMCSR36<29:23> and MEMCSR36<22:16> have a load control pointer that is initialized to point to MEMCSR36<22:16> by a chip reset or by a MEMCSR36 read. The pointer is incremented by the internal memory sequencer for a signature read, or for each returned longword of a G-chip memory octaword read (not a masked write). Therefore, the programmer is responsible for alignment of this pointer during memory diagnostics.**

Table 4–20 (Cont.) G-Chip Mode Control and Diagnostic Status Register Bits

MEMCSR36 Data Bit	Name	Description
<15>	Fast diagnostic test mode	This read/write bit provides a mechanism for speeding up the initial diagnostic testing of memory. Writing a 1 to this bit causes the G-chip to set the MODESEL<1> GMI port output signal, indicating to the GMX that it is in fast diagnostic test mode.
<14>	FDM second pass	In systems with more than four bank pairs of memory per module, the memory test in fast diagnostic mode has to be done in two passes. This read/write bit (cleared at power-up) should be set, and a second pass of the test should be run. This enables testing of modules with more than four banks. This bit has no effect unless MEMCSR36<15> is set.
<13>	Cache RAM speed	On the KA670, this bit should be set to 0, indicating that no extra cycles are needed when accessing the backup cache. When cleared, this bit indicates that the system is using fast, one-cycle cache RAMs. When this bit is a 1, it indicates the system is using two-cycle RAMs. This bit is cleared on power-up. G-chip's interface alters its response behavior by one cycle, depending on the state of this bit.
<12>	Timer count select	This read/write bit enables the timers in the chip to be used over a G-chip clock cycle range of 20 ns to 40 ns. When set, this bit increases the count value of all the G-chip interface timers. The CP bus timers are not affected. When the cycle time is 28 ns or less, this bit should be a 1. For a cycle time greater than 28 ns, this bit should be cleared to 0.
<11>	Enable soft error logging	When this read/write bit is 0, correctable (single-bit) errors are corrected by the ECC logic, but the SERRIRQL output is not asserted, the associated error addresses are not logged in MEMCSR33 or MEMCSR35, the error syndrome fields of MEMCSR32 are not loaded, and <25, 9> are not set. <23, 7> is set by correctable errors to signal these lost correctable errors. When this bit is a 1, correctable errors are corrected by the ECC logic and reported on the SERRIRQL output. Correctable as well as other error addresses and syndromes are logged in MEMCSR33 or MEMCSR35. Also, <25, 9> are set when errors are detected. This makes it easier to reserve the error-logging information for uncorrectable, NXM, or parity errors when soft error reporting is disabled.
<10:9>	EPR timeout prescaler	On the KA670, this field should be set to 11_2. This field scales the EPR timeout counter up to to make it easier to access slow-access C-chip EPR registers. Field values: 11 = 1.5 µs, 10 = 12 µs, 01 =32 µs, 00 = 910 µs.

Table 4–20 (Cont.) G-Chip Mode Control and Diagnostic Status Register Bits

MEMCSR36 Data Bit	Name	Description
<8>	Refresh requested	This read-only flag is set when a refresh transaction is selected as the current operation. This implies that the refresh interval counter has counted to the overflow condition. This flag is cleared by reading MEMCSR36.
<7>	Disable memory error detection	When this bit is a 1, memory error detection and correction are disabled. All memory-related error logging in MEMCSR32, MEMCSR33, and MEMCSR35 is disabled. No memory-related error reporting occurs by asserting the ERRL, CPERRL, HERRIRQL or SERRIRQL output pins.
<6>	Disable page-mode	When set, this bit disables page-mode memory transactions. It causes the G-chip to deassert RASTIME after every memory transaction. This function is for test purposes only. If this bit is set during normal system operation, memory performance is degraded.
<5>	Flush write buffers	When written to a 1, this read/write bit initiates a flush of the QUEUE, the CPQUE, and hence the invalidate QUE. The G-chip delays the assertion of the G-chip ready signal for this MEMCSR36 write until the flush completes. When this bit is set, subsequent writes to MEMCSR36 result in the stall of the ready signal until all queues are flushed. A write of 0 clears this bit and disables the stall conditions.
<4>	Force write buffer hit	When set, this read/write bit forces a memory read address from a corresponding G-chip bus or CP bus port to hit any valid element in the write queues, regardless of the address tag. This ensures that all write queue elements and associated invalidate hit addresses are retired to memory prior to the completion of the pending read. This bit should be set only for diagnostic purposes. If this bit is set during normal system transactions, there is a performance degradation on memory reads that follow memory writes.
<3>	Force wrong parity	When set to 1, this read/write bit forces the result of the CP bus and the G-chip bus parity checkers to be inverted. This results in a parity check failure. This action is used to emulate an RDAL or a CP DAL parity error during memory read or write transactions and G-chip to CP bus transactions. DAL parity is ignored for G-chip bus I/O transactions. This bit is for test purposes only and should not be set during normal system operation.

Table 4–20 (Cont.) G-Chip Mode Control and Diagnostic Status Register Bits

MEMCSR36 Data Bit	Name	Description
<2>	Force CP bus owner	When set to 1, this read/write bit forces the G-chip to request CP bus mastership by asserting CPDMRL. The write that sets this bit is stalled on the G-chip bus until CPDMGIL is received. G-chip gives up the mastership of the CP bus when this bit is cleared. This bit is set to 1 when RESET L asserts, so the G-chip attempts to be the owner of the CP bus following a power-up reset.
<1>	Disable refresh	When set, this read/write bit disables memory refresh (regardless of the state of MEMCSR36<0>) and clears the refresh address counter and interval counter to 0. This function is for test purposes only. The bit should not be set during normal transactions or while the force refresh request bit is set.
<0>	Force refresh request	When cleared, this read/write bit allows the refresh control logic to operate normally. This bit is set at power-up only if the TESTMODE pin is asserted as RESET negates. When set on power-up or by writing a 1 with the TESTMODE pin asserted, this bit forces the G-chip to do continuous memory refresh transactions, incrementing the refresh address on each transaction. A pending memory operation takes precedence over the continuous refresh transactions. If the TESTMODE pin is negated, the G-chip ignores the state of this bit and behaves as if this bit were cleared. The bit can be cleared by writing a 0 and by deasserting the TESTMODE pin at power-up. This behavior facilitates a power-up functional test for probing. **RESTRICTION** **This bit should be used for test purposes only. If TESTMODE is selected and this bit is set during normal system operation, memory operations result in severe performance degradation. Also, memory array power consumption increases. This bit should never be set while the disable refresh bit is set.**

4.2.1.4 Bus Timeout and Nonexistent Addresses

The G-chip prevents the bus from hanging if a nonexistent device is addressed in the following ways, depending on the type of transaction:

- On CPU-to-memory read transactions to nonexistent or invalid locations, the G-chip responds with ERRL, sets the nonexistent memory bit <12>, and logs the address in the memory error address register (MEMCSR33).
- On CPU memory write transactions to nonexistent or invalid locations, the G-chip asserts HERRIRQL, sets the nonexistent memory bit <12>, and logs the address in the memory error address register (MEMCSR33).

- EPR transactions, read interrupt vector transactions, and I/O read/write transactions that are recognized as for G-chip bus to CP bus adapter reaction are transferred to CP bus transactions. If no device responds to these CP bus master transactions, the CP bus master times out, aborts, and informs the slave of the exception.

  For the read transaction exceptions (EPR, I/O, or interrupt), the G-chip responds with ERRL. For I/O (not EPR) write transaction exceptions, the G-chip asserts HERRIRQL. The I/O error bit <14> is set on I/O reads and I/O writes that time out; the address is loaded in the I/O error address register (MEMCSR34). The G-chip does not log any information for EPR or interrupt vector transaction exceptions.

- In all the above cases, the G-chip terminates the transaction by asserting RDYL or ERRL, thus preventing the system from hanging.

  The G-chip does not recognize EPR transactions as for transfer to the CP bus. EPR transactions are timed by the G-chip, according to the time limit established by MEMCSR36<10:9>. The timeout mechanism is the nonexistent EPR timeout counter, which serves to terminate EPR transactions that have not been responded to by a bus device. The G-chip aborts the transaction by asserting ERRL, but does not log any information. This timeout counter starts counting with the assertion of ASL. The counter is is cleared with the assertion of RDYL, ERRL, or RTYL.

4.2.1.5 Peripheral Port (CP Port)

The CP port of the G-chip interfaces with the G-chip port and the GMI port. Sections 4.2.1.5.1 through 4.2.1.5.4 describe the main features of the CP port.

4.2.1.5.1 Addressing

The G-chip, as bus slave, does not respond to I/O transactions initiated from peripheral bus (CP bus) DMA devices. Any transaction whose I/O or memory address does not match the programmed and validated values in the G-chip shadow registers is regarded as a no-operation request. The SSC timeout counter is assumed to cause this transaction to abort, so the G-chip does not respond.

4.2.1.5.2 Multiple-Transfer Transactions and Address Alignment

The CP port supports longword (2-word), quadword (4-word), hexword (6-word), and octaword (8-word) memory transactions; and only longword I/O transactions. It maintains quadword alignment on quadword transactions, and octaword alignment on hexaword and octaword transactions. Quadword alignment is preserved by complementing bit<2> of the address for accessing the second longword. Octaword alignment is preserved by incrementing (modulo-4) bits <3:2> of the address for accessing subsequent longwords.

4.2.1.5.3 Write Buffers

The G-chip improves write performance of the CP bus with the help of a write buffer, called the CPQUE. The CPQUE consists of two octaword buffer elements. Each element has an address tag and can store up to an octaword of data with the corresponding byte masks. The address tags are CAMs; a tag compare (lookup) occurs on CP bus to memory reads, to check if the data to be read is a CPQUE element. If the address compares (hits) in the CPQUE, then both elements are flushed to memory before the memory read is performed. Write data is loaded sequentially into the CPQUE and unloaded by the GMI port in the same order.

To ensure correct operation of the system, the CPQUE is flushed under the following circumstances:

- Read lock on the G-chip bus
- CPU I/O or EPR read to an address on the CP bus

- CPU read interrupt vector transaction
- CP bus memory read address that hits in the CPQUE

The CPQUE is cleared when RESET L asserts during power-up. For example, all the valid elements are invalidated.

4.2.1.5.4 CP Bus Timeout

The G-chip provides a timeout mechanism on the CP bus, to prevent G-chip initiated CP bus transactions to nonexistent I/O or EPR addresses, and to prevent interrupt vectors from hanging the bus. This is done with a CP bus cycle counter that has a fixed cycle count whose absolute time scales with the CP bus clock. The timeout is triggered by the assertion of the CP bus data strobe (CPDSL); it is cleared by the assertion of the CP bus ready or error signals (CPRDYL or CPERRL), or by the negated state of the no response abort (CPNRA) signal after the counter overflows.

The CPNRA signal is a NOR function of the "not me" signals of the DMA devices on the CP bus. If this signal is deasserted, it indicates that one of the DMA devices is going to respond to the current transaction.

If there is no response on the CP bus and the counter overflows, the G-chip looks at the state of CPNRA and reacts as follows:

- If CPNRA is asserted, terminate the transaction by deasserting CPDSL and CPASL. If the aborted transaction was a read (I/O read, EPR read, or an interrupt vector read), return ERRL to the CPU. If the aborted transaction was a write (I/O write), assert HERRIRQL. On I/O read and write transactions that are aborted by timer overflow, the G-chip sets the nonexistent I/O bit <15> and logs the address in MEMCSR34. On EPR reads/writes and interrupt vector reads, the G-chip does not log any information.
- If CPNRA is deasserted, the G-chip waits for CPRDYL or CPERRL from the CP bus. In the extreme case that a device deasserts its "not me" signal and fails to respond, the SSC's CP bus timeout counter should overflow and abort the transaction, thus preventing the system from hanging. This counter can be set to a very high value, about 15 ms.

4.2.1.6 GMI Port

The GMI port of G-chip supports up to 32 banks of memory. The port provides 7-bit error checking and correction (ECC) for a 32-bit memory data bus.

4.2.1.6.1 Memory Addressing

The G-chip can control up to 32 banks (16-bank pairs) of DRAM, with each bank consisting of 32 data bits and 7 bits of ECC code. These banks are addressed as follows :

- Each bank pair has a base address register value resident in the G-chip and CP (shadow register) ports, with either 4 or 6 significant bits (depending on the bank's RAM size).
- Bit <29> of the address is a 0 (memory address space).
- When a validated base address register value matches the address from the address bus or the CP bus, the bank pair at that address is selected for either reading or writing. Two banks are enabled for every base address match. Bit <2> of the address further selects one of the two enabled bank pairs.
  - If the RAM size is 1 megabyte, the base address maps to bits <28:23> of the address, the row address maps to bits <22:13> of the address and the column address maps to bits <12:3> of the bus address.

– If the RAM size is 4 megabytes, the base address maps to bits <28:25>, the row address to bits <24,22:13>, and the column address to bits <23,12:3> of the bus address.

4.2.1.6.2 Support for Pagemode

The GMI port of G-chip supports extended pagemode to improve the GMI bandwidth. Addresses within the same physical memory page—for example, addresses whose bits <28:13> are the same—can be accessed at a faster rate than addresses that are not in the same page. This is done by keeping the row address the same and changing the column address only.

The GMI port provides a timeout counter for pagemode, since DRAMs have a restriction on the time that transactions can be done in page mode. In order to keep the pagemode timeout interval from varying with the G-chip clock cycle times, the count value can be changed by setting the timer count select bit MEMCSR36<12>. This bit should be set at cycle times greater than 28 ns and cleared at cycle times less than or equal to 28 ns. Except for refresh, all transactions are done in page mode as long as the previous and current bank and row address match, and the page mode timeout counter has not overflowed. The page mode timeout counter overflows after 8 µs.

4.2.1.6.3 Memory Error Detection and Correction

On memory write transactions, the source of the memory data comes from the the corresponding write buffer, together with 7 check bits generated from an ECC generator. On memory read transactions, ECC is generated from the memory data inputs and compared to the check bits. The ECC logic uses a 32-bit modified Hamming code to encode the 32-bit data longword into seven check bits.

When an error is detected, the syndrome is loaded into <22:16> or <6:0>, depending on whether the transaction was requested by the G-chip port or the CP port. The G-chip ECC logic detects and corrects single-bit errors in the memory data. Single-bit errors in the check bit field are detected and reported. Double-bit errors are detected and reported, but not corrected.

Modified Hamming Code

Figure 4–30 shows the modified Hamming code. The data bits marked with an X in each row are Exclusive-ORed together to generate the corresponding check bit. In a memory read transaction, a non-zero syndrome indicates an error. If the syndrome generated matches a column of X bits, the error is correctable and the column number corresponds to the corrected bit. If a syndrome value does not match any value in Figure 4–30, it indicates an uncorrectable error. Table 4–21 shows the syndromes from Figure 4–30 that can be read from <22:16> or <6:0>.

Syndrome	G−Chip Data Bus <31:0>				G−Chip Data Bus <32:38>						
	Byte 3	Byte 2	Byte 1	Byte 0	Generated Check Bits						
	31 24	23 16	15 8	7 0	32	C1 33	C2 34	C4 35	C8 36	C16 37	C32 38	CT
S1	XXXX	XXXX	XXXX	XXXX	X							
S2	XXXX	XXXX	XXXX	XXXX		X						
S4	X XX X	X XX X	X XX X	X XX X			X					
S8	XXXXXXXX		XXXXXXXX	XXXXXXXX				X				
S16	XXXXXXXX	XXXXXXXX		XXXXXXXX					X			
S32	XXXXXXXX	XXXXXXXX	XXXXXXXX							X		
ST	X X XX	XX X X	XX X X	XX X X								X

Even Parity − C1, C2, CT

Odd Parity − C4, C8, C16, C32

Error_Syndrome<N> = (Generated CB<N> XOR Memory CB<N>)

Figure 4–30 32-Bit Modified Hamming Code

Table 4–21 Syndrome Examples

MEMCSR32<22:16> MEMCSR32<6:0>	Bit Position in Error
0000000	No error detected.
	Data Bits (0 to 31_{10})
1011000	0
0011100	1
0011010	2
1011110	3
0011111	4
1011011	5
1011101	6
0011001	7
1101000	8
0101100	9
0101010	10
1101110	11
0101111	12

Table 4–21 (Cont.) Syndrome Examples

MEMCSR32<22:16> MEMCSR32<6:0>	Bit Position in Error
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 to 38_{10})
0000001	32
0000010	33
0000100	34
0001000	35
0100000	37
0000111	Result of incorrect check bits written on detection of a RDAL or CP bus parity error.
All others	Multibit errors

Forcing Incorrect Check Bits

When a data parity error is detected from the RDAL during a memory write transaction, incorrect check bits are generated and loaded into memory to force an uncorrectable error for detection on a subsequent memory read. The algorithm for generating incorrect check bits is to complement the generated check bits<2:0> output and pass the generated check bits<6:3> unchanged. This would generate an error syndrome of 0000111.

4.2.1.6.4 Memory Refresh

The G-chip GMI controls DRAMs that must be refreshed at a fixed interval. The G-chip has an internal refresh interval timer. The timer initiates a refresh transaction every 480 cycles if the timer count select bit in MEMCSR36 is cleared (at cycle times less than or equal to 28 ns), and every 336 cycles if the timer count select bit in MEMCSR36 is set (at cycle times greater than 28 ns).

4.2.1.6.5 GMI priority

The GMI port has to arbitrate between the CP port and the G-chip port for memory accesses. The GMI port has a priority-based arbitration scheme to help sustain CPU and I/O performance and latency. The GMI port gives a higher priority to the CP port than the G-chip port. However, when a G-chip read or write request is pending, the GMI port services a maximum of three consecutive CP write requests before servicing one pending G-chip request. After the pending G-chip request is serviced, the "three CP transactions" counter is reset until the condition of CP write service and pending G-chip request occurs. Then the "three CP transactions" counter begins.

From the CP port's perspective, five consecutive writes may occur before the G-chip service interruption is observed. This is a result of the two write buffers and one GMI operation buffer. If a G-chip port request is not pending, there is no restriction on the number of consecutive CP writes serviced by the GMI. CP reads are always given the highest priority, unless the read address matches the address of a buffered CP write. In that case, the write is completed before the read is serviced. Table 4–22 indicates the GMI priority based on the three consecutive CP writes serviced counter while a G-chip port request is pending.

Table 4–22 GMI Port Priority

GMI Priority	GMI Transactions Number of CP Writes < 3	GMI Transactions Number of CP Writes = 3
1	Refresh	Refresh
2	Signature Read	Signature Read
3	CP Port Read	CP Port Read
4	CP Port Write	G-chip Port Read
5	G-chip Port Read	G-chip Port Write
6	G-chip Port Write	CP Port Write

4.2.1.7 Transactions and Port Interactions

This section describes how the three ports of the G-chip interact with each other.

4.2.1.7.1 Support for Cache Invalidates

Because DMA is done on the CP bus, invisible to the G-chip bus, the G-chip provides a mechanism to invalidate cache entries that have been written to by DMA devices. Cache entries are invalidated by doing an octaword DMA write protocol on the G-chip bus. The G-chip supports octaword cache invalidates only; it does not support quadword invalidates. The G-chip, with some external CP bus address latches (I-latch), provides a mechanism to reduce the number of invalidates that have to be done on the G-chip bus by doing an address lookup on a separate invalidate lookup bus.

Invalidate Lookup

In order to support the invalidate lookup protocol, the G-chip requires the module to have external latches of the CP bus address that drive the C-chip invalidate lookup bus. The CP bus address is latched by the I-latch whenever a transaction is initiated on the CP bus. If the transaction is a write and the address is valid (and the CPQUE is not full, then the address is loaded into the CPQUE. At the same time, the G-chip asserts a lookup request signal to the C-chip, indicating that the lookup address is valid. The CP bus write transaction does not complete until the result of the lookup is received from the C-chip. If the address hits in one or both of the cache tag stores, an invalidate hit bit is set in the corresponding CPQUE element; this indicates the address has to be invalidated on the G-chip bus when the data is retired to memory.

There is an additional constraint if cache lookups are initiated when a memory read transaction is in progress on the G-chip bus—the result of the lookup may be misleading if the lookup address is in the same octaword block as the current G-chip read. The G-chip does not stall CP bus writes to avoid this problem. Invalidate lookups take place as usual. However, if a CP bus write occurs simultaneously with a G-chip port memory read, the invalidate hit tags in the CPQUE are set forcibly until the read and its cache fill complete. This action ensures that even if the data returned on the RDAL is not up-to-date and the result of the lookup for that address was a miss, but the G-chip fill just caused it be validated in the cache, an address will be invalidated as the write data is written to memory.

Invalidate Hits

Addresses that hit in either the primary cache tag store or the backup cache tag store have to be invalidated on the G-chip bus. The G-chip has two invalidate hit address buffers that are loaded by the GMI port when the current address marked as having hit is taken from the CPQUE. As soon as one of these buffers is loaded, the G-chip requests the G-chip bus by asserting DMRL. The write transaction on the GMI does not complete until the address is loaded in an invalidate hit buffer. The following transactions are not allowed to complete until both invalidate hit buffers are flushed :

- Memory read
- Memory read lock
- CP bus EPR or I/O read
- Read interrupt vector

The G-chip may retry the following G-chip bus transactions to perform invalidates that prevent deadlocks :

- I/O write to G-chip MEMCSR
- Memory write, SSC EPR write, or an I/O write to the CP bus, that are stalled for any reason

4.2.1.7.2 I/O Transactions

On an I/O read or write transaction initiated by the CPU, the G-chip decodes the address. If the address does not match any of its internal MEMCSRs, the G-chip does that read or write on the CP bus. The G-chip generates a longword address from the quadword address and byte masks provided on the G-chip bus. All I/O transactions are either byte, word, or longword. On an EPR read or write transaction, the G-chip decodes the EPR number; if the number corresponds to an SSC EPR number, the G-chip does the read or write on the CP bus.

The G-chip port to CP port interface is made up of an address, data, and operation buffer. The G-chip port loads information about the transaction into this buffer. The CP port master continually monitors and unloads the buffer when the buffer has an operation in it. If the buffer is full when the G-chip port needs to load an operation, that transaction stalls on the G-chip bus. This buffer is used for all transactions initiated by the CPU and performed on the CP bus:

- I/O read
- I/O read lock
- I/O write
- I/O write unlock
- EPR read
- EPR write
- Interrupt vector read
- Memory read lock
- Memory write unlock

On read transactions (except the memory read lock) where the G-chip port waits for a response from the CP bus, the G-chip slave controller monitors the state of the data buffer for valid data.

4.2.1.7.3 Loading and Unloading Write Queues

The organization of the CPQUE and the QUEUE are different, but their operation is the same. Elements of a queue are loaded and a valid bit is set by the corresponding port controller. Note that transaction-to-transaction data packing is not done by the write queues, since the GMI continuously unloads any valid elements following the previously described operation priority.

If at least one of the buffers in a queue is valid, a write request is made to the GMI port by the corresponding port. The GMI then services the writes according to its priority scheme.

Note that each element in the queue implements a valid bit. If a valid bit is set, the GMI port regards the element as full and does the memory write. The GMI port does not keep data waiting in the buffers in order to fill the buffer or pack longwords. Writes are done whenever the GMI can service them. When all valid bits for the elements of a queue are set, a *full* signal is sent to the port controller. Also, when all valid bits for the elements of a queue are clear, an *empty* signal is sent to the port controller.

The G-chip supports interlocked read transactions from the CPU to memory and the CP bus, and from the CP bus to memory. Any device (CPU or CP bus DMA) that does a locked transaction, has to be master of the CP bus and the Q22-bus (in a Q22-bus system). The address and cycle status code for the lock is broadcasted on the CP bus, allowing the CQBIC (if present) to retry the transaction if it is not master of the Q22-bus. The read from memory takes place only after there are no more retries from the CQBIC.

The read lock is regarded as successful if there are no uncorrectable errors in the requested read data. Under normal circumstances, when there are no DAL parity errors on the returned data, the G-chip expects that the next transaction on the bus (that initiated the read lock) is a write unlock. The lock is regarded as completed when another transaction is initiated on that bus. If the transaction is not a write unlock, it is assumed that write unlock is lost and will not happen.

If the read lock is initiated on the G-chip bus, a lost write unlock causes the G-chip to do a dummy write unlock on the CP bus. This unlocks the Q22-bus and clears the lock.

If the read lock is initiated on the CP bus, then any transaction on the CP bus – even a G-chip master transaction – can clear the lock.

4.2.1.7.4 Interrupts

The G-chip interrupts the CPU with one hard error interrupt and one soft error interrupt. The G-chip does not have any vectored interrupts; however, it does support reading interrupt vectors from the CP bus. All interrupt vector read transactions from the CPU are transferred through the G-chip to CP bus interface. The vector that is read from the interrupting device is provided to the CPU on the RDAL, without any modifications. The G-chip does ensure that the CPQUE and the invalidate hit buffer addresses are flushed before the vector is returned on the RDAL.

4.2.1.7.5 Transaction Summary

Table 4–23 indicates whether the write buffers or invalidate hit buffers are flushed on various G-chip bus and CP bus transactions, before the transactions complete.

Table 4–23 System Requirements for Buffered Writes and Invalidates

	Buffered Writes				
Transactions	**Stall Until Retired**	**G-Chip Port**	**CP Port**	**Invali-dates**	**Remarks**
G-chip memory read (no lock)	Yes	No	No	Yes	All the CP writes that have been retired to memory have to be invalidated before the read completes.
G-chip memory read (lock)	-	No	Yes	Yes	It is important to retire CP writes here, so the CPU gets the most current data from I/O devices.
G-chip I/O read (no lock and lock)	-	No	Yes	Yes	It is important to retire CP writes here, so the CPU gets the most current data from I/O devices.
G-chip memory write	No	No	No	No	
G-chip I/O write	-	Yes	No	No	The I/O device should get the data written by the CPU. Here the CPU communicates with the I/O device through CSRs.
G-chip IAK	-	No	Yes	Yes	On interrupts, the CPU issues a clear write buffer command, and G-chip writes can be flushed at that command. The CP writes and their invalidates have to be flushed.
G-chip EPR read/write	-	Yes (write)	Yes (read)	Yes (read)	

Table 4–23 (Cont.) System Requirements for Buffered Writes and Invalidates

Transactions	Buffered Writes: Stall Until Retired	Buffered Writes: G-Chip Port	CP Port	Invalidates	Remarks
G-chip clear write buffer	-	Yes	No	No	No CPU transaction should be allowed to happen until the G-chip write buffers are flushed.
CP read lock	Yes	Yes	No	No	The I/O device should get up-to-date data.
CP memory read (no lock)	Yes	No	No	No	Stall until hit element is retired
CP memory write	No	No	No	No	

4.2.1.8 Exceptions

The G-chip responds to exceptions and errors by terminating transactions with an error signal on either bus and/or by interrupting the CPU.

Exception	G-Chip Response
G-chip memory write transactions with RDAL parity errors	The G-chip interrupts the CPU by asserting HERRIRQL. The G-chip does the write to memory, but forces an uncorrectable memory error in that location by complementing the three least significant check bits. The G-chip bus parity error bit is set in MEMCSR32<11>, and the octaword address is logged in MEMCSR33.
An uncorrectable memory error on the read portion of a masked write from the QUEUE	The G-chip asserts HERRIRQL. The G-chip uncorrectable memory error bit is logged in MEMCSR32<10>, and the octaword address of that location is loaded in MEMCSR33. In this case, the write is not completed.
G-chip memory reads with uncorrectable memory errors in the first quadword of data	The G-chip terminates the transaction with error. On G-chip quadword memory reads with uncorrectable memory errors in the second (unrequested) quadword, the G-chip does not do a cache fill. In all cases, the G-chip logs the G-chip uncorrectable memory error bit in MEMCSR32<10>, and the octaword address in MEMCSR33.
G-chip memory transactions with invalid memory addresses	The G-chip asserts ERRL (on memory reads) or HERRIRQL (on memory writes), sets the G-chip nonexistent memory bit in MEMCSR32<12>, and logs the octaword address in MEMCSR33.
G-chip I/O read transactions that terminate in an error on the CP bus	The G-chip asserts ERRL, logs the G-chip I/O error bit in MEMCSR32<14>, and logs the longword I/O address in MEMCSR34.
G-chip I/O write transactions that terminate in an error on the CP bus	The G-chip asserts HERRIRQL, logs the G-chip I/O error bit in MEMCSR32<14>, and logs the longword address of the error in MEMCSR34.
G-chip I/O read/write transactions that time out on the CP bus	The G-chip asserts ERRL (on reads) or HERRIRQL (on writes), and logs the nonexistent I/O bit in MEMCSR32<15>.

Exception	G-Chip Response
G-chip interrupt vector reads or EPR reads that time out on the CP bus	The G-chip asserts ERRL, but does not log any error bits in MEMCSR32 or addresses in in MEMCSR34.
G-chip EPR writes that timeout on the CP bus	The G-chip does not notify the CPU by asserting HERRIRQL, and no errors are logged.
CP bus initiated memory read transactions with uncorrectable memory errors	The G-chip responds by terminating the transaction with CPERRL. Multiple-transfer read transactions (CP bus quad, hexa, or octa) are aborted on uncorrectable errors in the earlier transfers. For example, if a CP bus octaword read has an uncorrectable error in the second transfer, the third and fourth transfers are aborted by the G-chip and the G-chip expects the master device to terminate the transaction. If there is an uncorrectable memory error in an unrequested longword, the G-chip does not interrupt the CPU. In all the cases, the G-chip sets the CP bus memory correctable error bit in MEMCSR32<25> or the CP bus uncorrectable error bit in MEMCSR32<26>, and logs the octaword address of the error in MEMCSR35.
CP bus memory write transactions with DAL parity errors	The G-chip interrupts the CPU by asserting a HERRIRQL. The G-chip does the write to memory, but forces an uncorrectable memory error in that location by complementing the three least significant check bits. The CP bus parity error bit is set in MEMCSR32<27>, and the octaword address is logged in MEMCSR35.
An uncorrectable memory error occurs on the read portion of a masked write from the CPQUE	The G-chip asserts HERRIRQL. The CP bus uncorrectable memory error bit is logged in MEMCSR32<26>, and the octaword address of that location is loaded in MEMCSR35. In this case, G-chip does not do the write.

If there is a correctable error on any memory read or masked memory write transaction, the G-chip:

1. Asserts SERRIRQL.
2. Logs the CRD error bit corresponding to the port (G-chip or CP) that requested the transaction.
3. Logs the address in the corresponding memory error address register, MEMCSR33 (if the error occurs on a G-chip transaction) or MEMCSR35 (if the error occurs on a CP bus transaction).
4. Writes the correct data back to main memory.

5
The Console Line, TOY Clock, and Bus System

This chapter describes the console serial line and the time-of-year (TOY) clock. The chapter also provides an overview of the KA670 bus system.

5.1 KA670 Console Serial Line

The console serial line provides the KA670 processor with a full-duplex, RS-423 EIA, serial line interface that 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 the *VAX Architecture Reference Manual* .

5.1.1 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 to 35. Table 5–1. lists the registers.

Table 5–1 Console Registers

IPR Number		Register Name	Mnemonic
Decimal	**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

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

The console receiver control/status register (RXCS), internal processor register 32, is used to control and report the status of incoming data on the console serial line. Figure 5–1 shows the format of the register. Table 5–2 lists the bit descriptions.

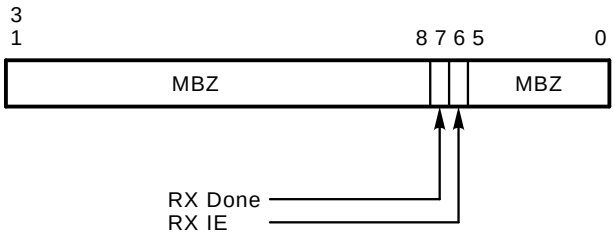


Figure 5–1 Console Receiver Control/Status Register—(IPR 32 $_{10}$ 20$_{16}$)

Table 5–2 Console Receiver Control/Status Register Bits

Data Bit	Name	Description
<31:8>	MBZ	These bits read as 0s. 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. The bit is also cleared on power-up or the negation of DCOK.
<6>	RX IE	Receiver interrupt Enable (read/write). When set, this bit causes an interrupt to be requested at IPL14 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 or the negation of DCOK.
<5:0>	Unused	These bits read as 0s. Writes have no effect.

5.1.1.2 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. Figure 5–2 shows the format of the register. Table 5–3 lists the bit descriptions.

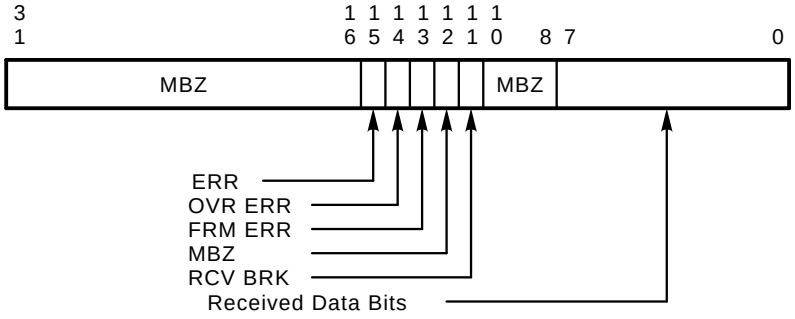


Figure 5–2 Console Receiver Data Buffer – (IPR 33$_{10}$ 21$_{16}$)

Table 5–3 Console Receiver Data Buffer Bits

Data Bit	Name	Description
<31:16>	MBZ	These bits always read as 0. Writes have no effect.
<15>	ERR	Error (read-only). Writes have no effect. This bit is set if RBUF <14> or <13> is set. The bit is clear if these two bits are clear. This bit cannot generate a program interrupt. The bit is cleared on power-up or the negation of DCOK.
<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. The bit is cleared by reading the RXDB, on power-up or 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. The bit is cleared by reading the RXDB, on power-up or the negation of DCOK. **Error conditions are updated when the character is received, and it remains present until the character is read. At that point, the error bits are cleared.**
<12>	MBZ	This bit always reads as 0. 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. The bit is cleared by reading the RXDB register, power-up, or the negation of DCOK.
<10:8>	MBZ	These bits always read a as 0. Writes have no effect.
<7:0>	Received data bits (read-only). Writes have no effect. These bits contain the last received character.	

5.1.1.3 Console Transmitter Control/Status Register (IPR 34)

The console transmitter control/status register (TXCS), internal processor register 34, is used to control and report the status of outgoing data on the console serial line. Figure 5–3 shows the format of the register. Table 5–4 lists the bit descriptions.

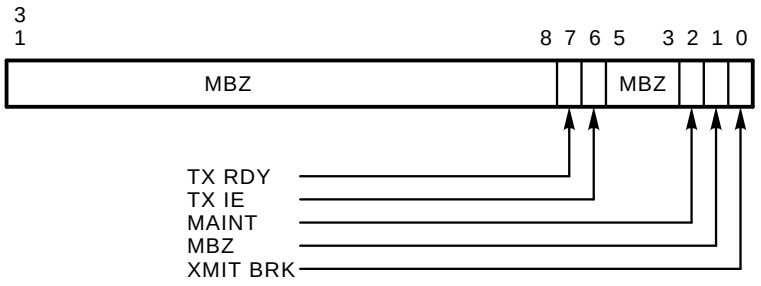


Figure 5–3 Console Transmitter Control/Status Register (IPR 34_{10} 22_{16})

Table 5–4 Console Transmitter Data Buffer

Data Bit	Name	Description
<31:8>	MBZ	These bits read as 0s. 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 or the negation of DCOK.
<6>	TX IE	Transmitter interrupt enable (read/write). When set, this bit causes an interrupt request at IPL14 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 or the negation of DCOK.
<5:3>	MBZ	Read as 0s. 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 or the negation of DCOK.
<1>	Unused	This bit reads as 0. 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 operates normally, but the output line remains low. Thus, software can transmit dummy characters to time the break. This bit is cleared on power-up.

5.1.1.4 Console Transmitter Data Buffer—(IPR 35)

The console transmitter data buffer (TXDB), internal processor register 35, is used to buffer outgoing data on the serial line. Figure 5–4 shows the format of the register. Table 5–5 lists the bit descriptions.

Figure 5–4 Console Transmitter Data Buffer—(IPR 35_{10} 23_{16})

Table 5–5 Console Transmitter Data Buffer Bits

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

5.1.2 Break Response

The console serial line unit recognizes a break condition that 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 KA670 halts and transfers program control to UVROM 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.

5.1.3 Baud Rate

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

The user selects the desired baud rate through the baud rate select signals that are received from an external 8-position switch mounted on the console module (H3604). The KA670 firmware reads this code from boot and diagnostic register bits <6:4>, complements and loads the code into SSC configuration register bits <14:12>.

Table 5–6 lists the baud rate selections, the corresponding codes 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 5–6 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

5.1.4 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}.

5.2 KA670 TOY Clock and Timers

The KA670 clocks include the time-of-year clock (TODR), a subset interval clock (subset ICCS), as defined in the *VAX Architecture Reference Manual*, and two additional programmable timers modeled after the VAX standard interval clock.

5.2.1 Time-of-Year Clock (TODR)—EPR 27

The KA670 time-of-year clock (TODR) forms an unsigned 32-bit binary counter that is driven from a 100 Hz oscillator. The least significant bit of the clock represents a resolution of 10 milliseconds, with less than 0.0025 percent error. The register counts only when it contains a nonzero value. This register is implemented in the SSC chip. Figure 5–5 shows the format.

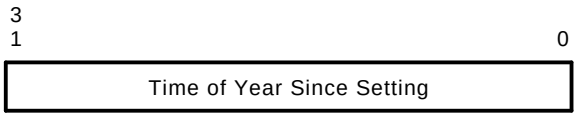


Figure 5–5 Time-of-Year Clock (TODR) – (EPR 27_{10} $1B_{16}$)

During a power failure, the time-of-year clock is maintained by battery backup circuitry that interfaces through the external connector to a set of batteries mounted on the CPU console module. The clock remains valid for greater than 162 hours when using the NiCad battery pack (3 batteries in series) mounted on the I/O distribution insert panel .

The SSC configuration register contains a battery low (BLO) bit. If this bit is set after initialization, the TODR is cleared remains at 0 until software writes a nonzero value into it.

NOTE
After writing a nonzero value into the TODR, software should clear the BLO bit by writing a 1 to it.

5.2.2 Interval Timer (ICCS)—EPR 24

The KA670 interval timer (ICCS), internal processor register 24, is implemented according to the *VAX Architecture Reference Manual*. The interval clock control/status (ICCS) register is implemented as the standard subset of the standard VAX ICCS in the CPU chip. NICR and ICR are not implemented. Figure 5–6 shows the format or the ICCS register. Table 5–7 lists the bit descriptions.

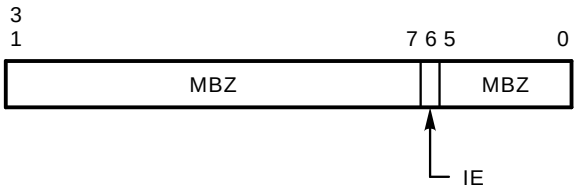


Figure 5–6 Interval Timer (ICCS) – (EPR 24_{10} 18_{16})

Table 5–7 Interval Timer Bits

Data Bit	Name	Description
<31:7>	MBZ	Read as 0s. Must be written as 0s.
<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 0.01 percent. When the bit is clear, interval timer interrupts are disabled. This bit is cleared on power-up.
<5:0>	MBZ	Read as 0s. Must be written as 0s.

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

5.2.3 Programmable Timers

The KA670 features two programmable timers. Although modeled after the VAX standard interval clock, the timers are accessed as I/O space registers rather than as internal processor registers. Also, an added control bit stops the timer upon overflow. If so enabled, the timers will interrupt at IPL 14 upon overflow. The interrupt vectors are programmable, and are set to 78 and 7C by the firmware.

Each timer is composed of four registers:

Timer *n* control register
Timer *n* interval register
Timer *n* next interval register
Timer *n* interrupt vector register

n represents the timer number (0 or 1).

5.2.3.1 Timer Control Registers (TCR0 and TCR1)

The KA670 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 5–7 shows the format. Table 5–8 lists the bit descriptions.

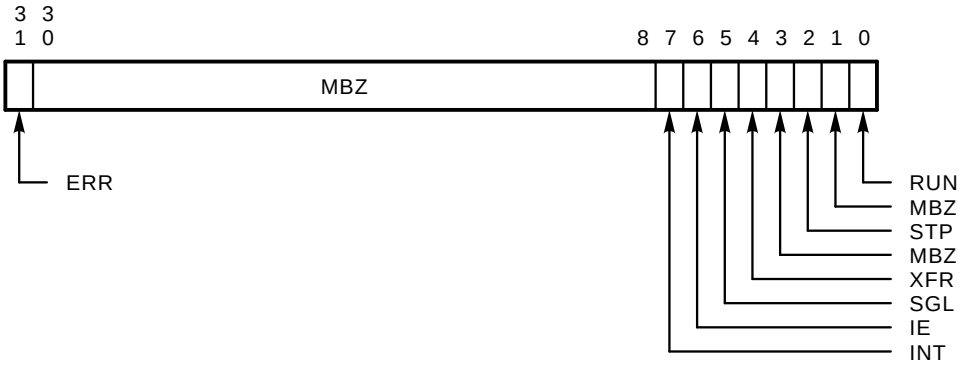


Figure 5–7 Timer Control Registers (TCR0 and TCR1)

Table 5–8 Timer Control Register Bits

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 1 to this bit clears the bit. ERR is cleared on power-up.
<30:8>	MBZ	Read as 0s. Must be written as 0s.
<7>	INT	Interrupt (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 the bit. INT is cleared on power-up.
<6>	IE	Interrupt enable (read/write). When this bit is set, the timer will interrupt at IPL 14 when the INT bit is set. IE is cleared on power-up.
<5>	SGL	Read/write. Setting this bit causes the timer interval register to be incremented by 1 if the RUN bit is cleared. If the RUN bit is set, then writes to the SGL bit are ignored. SGL is always read as 0. SGL is cleared on power-up.
<4>	XFR	Transfer (read/write). Setting this bit causes the timer next interval register to be copied into the timer interval register. XFR is always read as 0. XFR is cleared on power-up.
<3>	MBZ	Read as 0s. Must be written as 0s.
<2>	STP	Stop (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. STP is cleared on power-up.
<1>	MBZ	Read as 0s. Must be written as 0s.

Table 5–8 (Cont.) Timer Control Register Bits

Date Bit	Name	Description
<0>	RUN	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. RUN is cleared on power-up.

5.2.3.2 Timer Interval Registers (TIR0 and TIR1)

The KA670 has two timer interval registers—one for timer 0 (TIR0), and one for timer 1 (TIR1). TIR0 is accessible at address 2014 0104_{16}, and TIR1 is accessible at 2014 0114_{16}.

The timer interval register is a read-only register containing the interval count. When the RUN bit is 0, writing a 1 increments the register. When the RUN bit is 1, the register is incremented once every microsecond.

When the counter overflows, the INT bit is set; 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 5–8 shows the format of the registers.

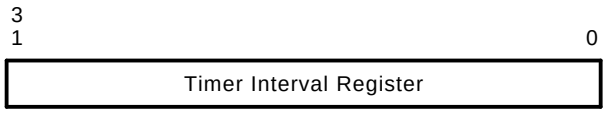


Figure 5–8 Timer Interval Registers (TIR0 and TIR1)

5.2.3.3 Timer Next Interval Registers (TNIR0 and TNIR1)

The KA670 has two timer next interval registers—one for timer 0 (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. Figure 5–9 shows the format of the registers.

These read/write registers contain the value written into the timer interval register after overflow or in response to a 1 written to the XFR bit. The timer next interval registers are cleared on power-up.

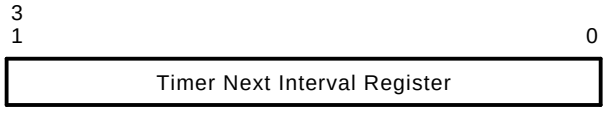


Figure 5–9 Timer Next Interval Registers (TNIR0 and TNIR1)

5.2.3.4 Timer Interrupt Vector Registers (TIVR0 and TIVR1)

The KA670 has two timer interrupt vector registers—one for timer 0 (TIVR0), and one for timer 1 (TIVR1). TIVR0 is accessible at address 2014 $010C_{16}$, and TIVR1 is accessible at 2014 $011C_{16}$. These registers are implemented in the SSC chip. The resident firmware sets TIVR0 to 78_{16} and TIVR1 to $7C_{16}$. Figure 5–10 shows the format.

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

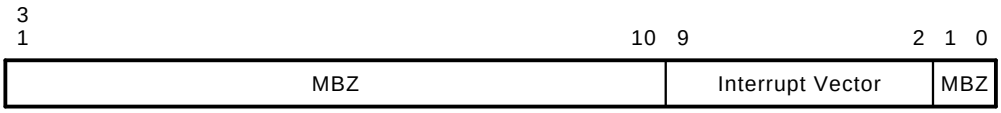


Figure 5–10 Timer Interrupt Vector Registers (TIVR0 and TIVR1)

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

5.3 KA670 Bus Overview

The KA670 has three major buses:

- Data address lines (RDAL)
- Peripheral (CP)
- G-chip memory interconnect (GMI)

5.3.1 RDAL Bus

The RDAL bus connects the CPU, FPA, and backup cache chip to the memory controller. The KA670 supports the following components on the RDAL bus:

- Four of the five core chips (plus memory controller):
  - —CPU chip (P-chip)
  - —Clock chip (CLK-chip)
  - —Floating point accelerator chip (F-chip)
  - —Backup cache controller chip (C-chip)
  - —Memory controller chip (G-chip)

The KA670 does not support the following components on the RDAL bus:

- System support chip (SSC)
- Any other peripheral components

5.3.2 The CP Bus

The CP bus connects the I/O subsystem to the memory controller. The KA670 depends on and supports the following components on the CP bus:

- Clock chip (CCLOCK DC509)
- Q22-bus adapter chip (CQBIC DC527)
- Second-generation Ethernet controller chip (SGEC DC541)
- Single host adapter chip (SHAC DC542)
- System support chip (SSC DC511)
- CP bus arbiter (ARB chip)

The KA670 does not support the following components on the CP bus:

- 90 ns memory controller (CMCTL DC357)
- 60 ns memory controller (CMCTL DC557)
- CPU chip (DC341)
- Graphics and system support chip (GSSC)

5.3.2.1 The CCLOCK Chip

This chip generates the precision MOS clock signals needed to operate the the G-chip and other core peripheral chips in synchronization with the CP bus. In addition, the CCLOCK chip provides two synchronizers for synchronizing asynchronous DMA functions to the CP bus.

5.3.2.2 CP Bus Arbiter

The CP bus arbiter (ARB chip) controls which peripheral device is granted CP bus mastership. The CP bus does not support DMA grant daisy-chaining, so the ARB chip receives separate requests from each device and issues a separate grant to each device. The arbiter must give the CQBIC the highest priority. Then the G-chip must be given the second highest priority. The third highest priority goes to the SGEC. The arbiter then uses a round-robin priority mechanism for the two SHACs.

5.3.3 GMI Bus

The GMI bus creates a path between the memory controller and main memory. There are two chips that support the memory subsystem:

- Memory controller chip (G-chip DC561)
- G-chip memory interface chip (GMX DC562)

6
KA670 Boot and Diagnostic Facility

The KA670 boot and diagnostic facility features two registers, 256 kilobytes of erasable programmable read only memory (EPROM) and 1 kilobyte of battery backed up RAM. The EPROM and battery backed up RAM may be accessed with longword, word, or byte references.

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

6.1 Boot and Diagnostic Register (BDR)

The boot and diagnostic register (BDR) is a longword-wide register, located in the VAX I/O page at physical addresses 2008 4000 to 2008 $407C_{16}$. The register is implemented uniquely on the KA670. The register can be accessed by KA670 software, but not by external Q22-bus devices. The BDR allows the boot and diagnostic firmware as well as the operating system to read various KA670 configuration bits.

The low byte and upper word of the BDR present the same information in each of the 32 successive longwords. The second byte (bits <15:8>) provides a byte of the LAN station address in each successive longword. Note that only the first 8 bytes contain the station address. The next 24 bytes are for testing purposes. Figure 6–1 shows the format for the boot and diagnostic register. Table 6–1 lists the bit descriptions.

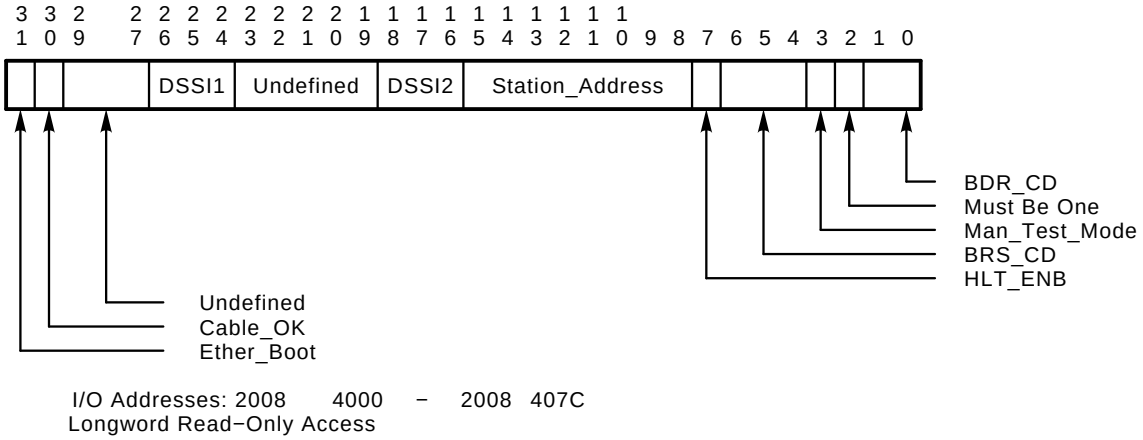


Figure 6–1 Boot and Diagnostic Register (BDR)

Table 6–1 Boot and Diagnostic Register Bits

Data Bit	Name	Description
<31>	Ether_boot	Enable Ethernet remote boot. This bit reflects the current setting of the enable Ethernet remote boot jumper on the console module (H3604). If the setting is 0, remote Ethernet boots are *enabled*. If this bit is 1, remote Ethernet boots requests are ignored.
<30>	Cable_OK	Console module cable okay. When this bit is 0, there is a high probability that the console module cable is functioning correctly. If this bit is 1, the console module cable is either malfunctioning or plugged in the wrong orientation. This bit is determined by sending a signal to the console module over one path and reading it back over another path on the cable.
<29:27>	Undefined	Should not be read or written.
<26:24>	DSSI1	This field contains the DSSI node number for the external DSSI bus (the bus that is accessed through the console module).
<23:19>	Undefined	Should not be read or written.
<18:16>	DSSI2	This field contains the DSSI node number for the internal DSSI bus (the bus that is accessed through the backplane connector).
<15:8>	Station_address	The KA670's hardware LAN station address EPROM is accessed by reading the BDR several times at successive addresses. The encoding for the station address is as follows: BDR + 00: SA byte 0 BDR + 04: SA byte 1 BDR + 08: SA byte 2 BDR + 0C: SA byte 3 BDR + 10: SA byte 4 BDR + 14: SA byte 5 BDR + 18: Checksum byte 0 BDR + 1C: Checksum byte 1 The last 24 bytes are for testing purposes.
<7>	HLT ENB	Halt enable (read-only). Writes have no effect. This bit reflects the state of the BREAK ENABLE switch on the console module (H3604). When asserted, this signal enables the halting of the CPU upon detection of a console break condition. On a power-up, the KA670 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). When a HALT instruction is executed in kernel mode, the 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).

Table 6–1 (Cont.) Boot and Diagnostic Register Bits

Data Bit	Name	Description
<6:4>	BRS CD	Baud rate select (read-only). Writes have no effect. These three bits originate from the console module's (H3604) baud rate select switch. They reflect the baud rate setting, as listed in the following table:

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

<3>	Man_test_mode	Manufacturing test mode (read-only). Writes have no effect. When set, the KA670 is in normal run mode. When set (by grounding a test point on the backplane), the KA670 is in manufacturing test mode. In this mode, special diagnostic test script on run.
<2>	MBO	Must be one (read-only). Writes have no effect.
<1:0>	BDG_CD	Boot and diagnostic code (read-only). Writes have no effect. This 2-bit field reflects the setting of the power-up mode switch on the console module (H3604). The KA670 firmware programs use BDG_CD <1:0> to determine the power-up mode, as listed in the following table:

BDR<1:0>	Power-Up Mode
11	Run
10	Language inquiry
01	Test
00	Unused

6.2 Diagnostic LED Register (DLEDR)

The diagnostic LED register (DLEDR), address 2014 0030_{16}, is implemented in the SSC chip. The register contains four read/write bits that control the external LED display. A 0 in a bit turns on the corresponding LED. All four bits are cleared on power-up or the negation of DCOK, to provide a power-up lamp test. Figure 6–2 shows the format of the register. Table 6–2 lists the bit descriptions.

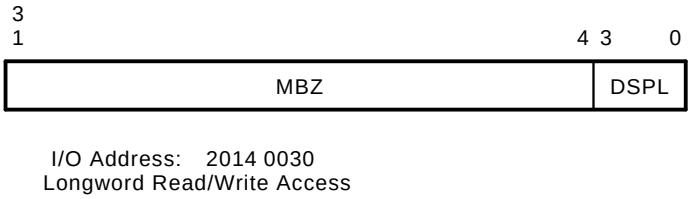


Figure 6–2 Diagnostic LED Register (DLEDR)

Table 6–2 Diagnostic LED Register Bits

Data Bit	Name	Description
<31:4>	MBZ	Read as 0s. Must be written as 0s.
<3:0>	DSPL	Display (read/write). These four bits update an external LED display. Writing a 0 to a bit turns on the corresponding LED. Writing a 1 to a bit turns the LED off. The display bits are cleared (all LEDs are turned on) on power-up or the negation of DCOK.

6.3 EPROM Memory

The KA670 has 256 kilobytes of EPROM memory for storing code for board initialization, VAX standard console emulation, board self-tests, and boot code. EPROM memory may be accessed through byte, word, and longword references. EPROM read accesses take 250 ns. The EPROM is organized as a 128K × 8-bit array. CP 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-kilobyte block of the local EPROM space. (2004 0000 to 2004 $1FFF_{16}$)

6.3.1 EPROM Address Space

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

NOTE
There is no concept of halt unprotect space on the KA670 (as used on previous Q22-bus MicroVAX systems).

Any I-stream read from the EPROM space places the KA670 in halt mode. The Q22-bus SRUN signal is deasserted, which turns off the front panel RUN light. The CPU is protected from further halts.

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

NOTE
The KA670 logic that controls halt mode/run mode cannot detect I-stream read references that hit the primary cache. Therefore, halt mode/run mode is unaffected by these cache hits.

6.3.2 KA670 Resident Firmware Operation

The KA670 CPU module's 256-kilobyte EPROM contains the resident firmware. The firmware can be entered by transferring program control to location 2004 0000_{16}.

Section 9.3.1 lists the various halt conditions that cause the KA670 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 that allows the user to examine and alter the state of the processor
- Diagnostic tests run at power-up to check out the CPU, the memory system, and the Q22-bus map
- Support of video or hardcopy terminals as the console terminal

6.3.2.1 Power-Up Modes

The boot and diagnostic EPROM programs use boot and diagnostic code <1:0> (Section 9.9) to determine the power-up modes listed in Table 6–3.

Table 6–3 Power-Up Modes

Code	Power-Up Mode	Description
11	Run (factory setting)	If the console terminal supports the DEC multinational character set, the user is prompted for a language if the time-of-year clock battery backup has failed, or SSC RAM is corrupted or unintialized (first power-up). Full startup diagnostics are run.
01	Language inquiry	If the console terminal supports the DEc multinational character set, the user is prompted for a language on every power-up and restart. Full startup diagnostics are run.
10	Test	EPROM programs run wraparound serial line unit (SLU) tests.
00	Unused.	

6.4 Battery Backed-Up RAM

The KA670 contains 1 kilobyte of battery backed-up static RAM (found in the SSC), for use as a console scratchpad. This RAM supports byte, word, and longword references. Read operations take 700 ns to complete. Write operations require 600 ns. The RAM is organized as a 256 × 32-bit (one-longword) array. The array appears in a 1-kilobyte block of the VAX I/O page, at addresses 2014 0400 to 2014 $07FF_{16}$. This array is not protected by parity; CP bus parity is neither checked nor generated on reads or writes to this RAM.

6.5 KA670 Initialization

The VAX architecture defines three kinds of hardware initialization:

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

6.5.1 Power-Up Initialization

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

6.5.2 Hardware Reset

A KA670 hardware reset occurs on power-up or the negation of DCOK. A hardware reset initiates the hardware halt procedure (Section 3.1.6.6) with a halt code of 03. The hardware reset also initializes some IPRs and most I/O page registers to a known state. Those IPRs affected by a hardware reset are noted in Section 3.1.1.3. The description for each I/O space register describes the effect of a hardware reset on that register.

6.5.3 I/O Bus Initialization

An I/O bus initialization occurs on power-up, the negation of DCOK, or as the result of an MTPR to IPR 55 (IORESET) or console UNJAM command. An I/O bus initialization clears the interprocessor communication (IPCR) and DMA system error (DSER) registers. It also causes the Q22-bus interface to acquire both the CP bus and Q22-bus, then assert the Q22-bus BINIT signal. The assertion of BINIT on the Q22-bus does not effect the KA670.

6.5.3.1 I/O Bus Reset Register (IPR 55)

The I/O bus reset register (IORESET), IPR 55_{10} is implemented in the SSC chip. An MTPR of any value to the IORESET register causes an I/O bus initialization. Note that the second generation Ethernet controller chip (SGEC) and single host adapter chip (SHAC) are *not* reset by MTPRs to IPR 55.

6.5.4 Processor Initialization

A processor initialization occurs

- On power-up
- On the negation of DCOK
- As the result of a console INITIALIZE command
- After a halt caused by an error condition

In addition to initializing those registers defined in the *VAX Architecture Reference Manual,* the KA670 firmware must also configure main memory, the local I/O page, and the Q22-bus map during a processor initialization.

6.5.4.1 Configuring the Local I/O Page

The following registers control the configuration of the KA670 local I/O page. They are unique to CPU designs that use the system support chip (SSC), 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
- CP bus timeout register

6.5.5 SSC Base Address Register (SSCBR)

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

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

This read/write register is set to 2014 0000_{16} on power-up or the negation of DCOK. Bits SSCBR<31:30,10:0> are unused. They read as 0s, and must be written as 0s. SSCBR<29> is read as 1 and must be written as 1. This register should also be set to 2014 0000_{16} by firmware during processor initialization. Figure 6–3 shows the format of the SSCBR register.

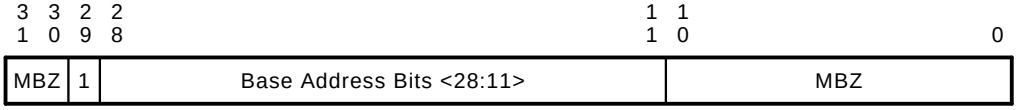


Figure 6–3 SSC Base Address Register (SSCBR)

6.5.6 BDR Address Decode Match Register (BDMTR)

The BDR address decode match register, address 2014 0140_{16}, controls the base address of the BDR. This read/write register is cleared on power-up or the negation of DCOK. BDMTR<31:30,1:0> are unused. They read as 0s, and must be written as 0s. This register should be set to 2008 4000_{16} by firmware during processor initialization. Figure 6–4 shows the format of the BDMTR register.

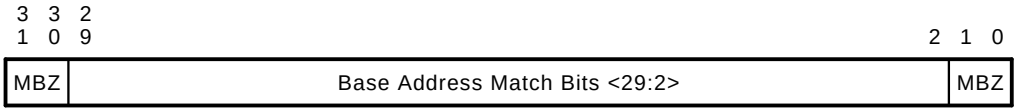


Figure 6–4 BDR Address Decode Match Register (BDMTR)

6.5.7 BDR Address Decode Mask Register (BDMKR)

The BDR address decode mask register, address 2014 0144 $_{16}$, controls the range of addresses that 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 or the negation of DCOK. Bits BDMKR<31:30,1:0> are unused. They read as 0s, and must be written as 0s. This register should should be set to 0000 007C $_{16}$ (32 copies of the BDR) by firmware during processor initialization , because successive bytes of the KA670's LAN station address are read using the BDR. Figure 6–5 shows the format of the BDMKR register.

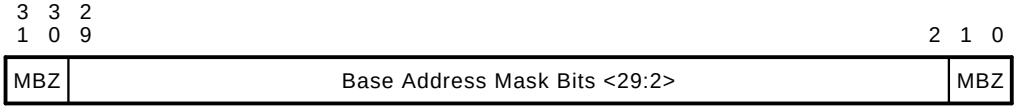


Figure 6–5 BDR Address Decode Mask Register (BDMKR)

NOTE
The KA670 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; they should not be accessed, because they could cause unpredictable behavior.

6.5.8 SSC Configuration Register (SSCCR)

The SSC configuration register, address 2014 0010 $_{16}$, controls the setup parameters for the console serial line, programmable timers, EPROM, TOY clock and BDR register. Figure 6–6 shows the format of the SSC configuration register. Table 6–4 lists the bit descriptions.

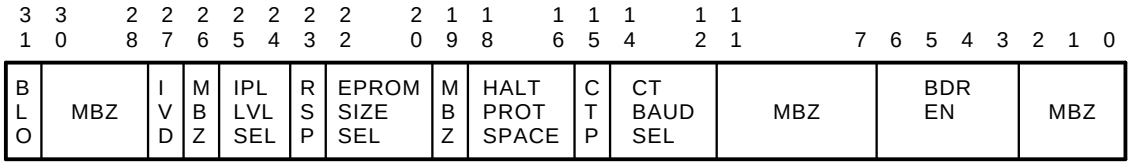


Figure 6–6 SSC Configuration Register (SSCCR)

Table 6–4 SSC Configuration Register Bits

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 1. If this bit is set, then the TOY clock will be cleared by power-up or the negation of DCOK.
<30:28>	MBZ	Read as 0s. Must be written as 0s.
<27>	IVD	Interrupt vector disable (read/write). When this bit is set, the console serial line and programmable timers do not respond to interrupt acknowledge cycles. IVD is cleared on power-up, the negation of DCOK, or a processor initialization.
<26>	MBZ	Read as 0s. Must be written as 0s.
<25:24>	IPL_ LVL_SEL	IPL level select (read/write). These bits 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_2) in order 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, the negation of DCOK, or a processor initialization.
<23>	RSP	ROM speed (read/write). This bit selects the EPROM access time. This bit must be set for the KA670 EPROMs to run at maximum speed. This bit is cleared on power-up or the negation of DCOK. The bit must be set to 1 by a processor initialization.
<22:20>	ROM_ SIZE_ SEL	EPROM address space size select (read/write). These bits control the size of the range of addresses that the EPROM responds to. These bits must be set to 101_2 because the KA670 contains 256 Kbytes of EPROM, yielding an address range of 256 Kbytes (2004 0000 to 2007 $FFFF_{16}$). These bits are cleared on power-up or the negation of DCOK, yielding an address range of 8 Kbytes (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 110_2 because the KA670's 256 Kbyte EPROM yields a halt mode address range of 256 Kbytes (2004 0000–2007 $FFFF_{16}$). These bits are cleared on power-up or the negation of DCOK. These bits must be set to the proper value by a processor initialization. Note that any instruction fetch from the EPROM puts the KA670 in halt protect mode.
<15>	CTP	Control P enable (read/write). When this bit is set, typing Ctrl P at the console will halt the CPU if halts are enabled (BDR<7> set). When this bit is cleared, typing Break at the console will halt the CPU if halts are enabled (BDR<7> set). CTP is cleared on power-up or the negation of DCOK.

Table 6–4 (Cont.) SSC Configuration Register Bits

Data Bit	Name	Description
<14:12>	CT BAUD SELECT	Console terminal baud rate select (read/write). These bits select the baud rate of the console terminal serial line. They are cleared on power-up or the negation of DCOK. They should be loaded from the complement of BDR<6:4> by the processor initialization code. The codes correspond to selected baud rates, as listed 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

<11:7>	MBZ	Read as 0. Must be written as 0.
<6:4>	BDR EN	BDR enable (read/write). These bits enable the BDR. To enable the BDR, these bits must be set to 111_2 by a processor initialization.They are cleared on power-up or the negation of DCOK.
<3:0>	MBZ	Read as 0. Must be written as 0.

NOTE
The SSC baud clock runs about 1.7 percent fast, within the VAX standard mandated accuracy. This is due to the accuracy of the crystal oscillator.

6.6 CP Bus Timeout Control Register (CBTCR)

The CP bus timeout register, address 2014 0020_{16}, controls the amount of time allowed to elapse before a CP bus cycle is aborted by the SSC. Note that the G-chip also has a CP bus timeout mechanism that will prevent *most* bus-initiated CP bus transactions to nonexistent I/O or EPR addresses, or to interrupt vectors, from hanging the bus.

The G-chip's timer uses a NOR of all the CP bus DMA devices *"not me"* as an indicator of a pending NXM address. In an extreme case of a broken CP bus DMA device that says it *will* respond (does not assert the CP bus *"not me"*) but does not respond, the CBTCR overflows; this causes a machine check, which prevents the system from hanging. Figure 6–7 shows the CBTCR format. Table 6–5 lists the bit descriptions.

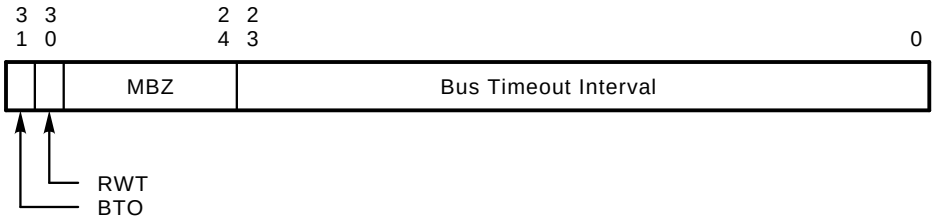


Figure 6–7 CP Bus Timeout Control Register (CBTCR)

Table 6–5 CP Bus Timeout Control Register Bits

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. BTO is cleared by writing a 1, by a power-up, or by 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 1, by a power-up, or by the negation of DCOK.
<29:22>	MBZ	Read as 0s. Must be written as 0s.
<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 1 microsecond to 16.77 seconds, in 1-microsecond increments. Writing a 0 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 or the negation of DCOK. This register should be loaded with 0000 4000_{16} on a processor initialization, for a timeout value of 15 milliseconds.

7
Interface Subsystems

The KA670 module has interfaces for the Q22-bus, the Ethernet, and a mass storage bus. This chapter describes the three interfaces.

7.1 KA670 Q22-bus Interface

The KA670 includes a Q22-bus interface implemented with a single VLSI chip called the CQBIC. The chip contains a CP 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 CP addresses. This function 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 CP 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 to the Q22-bus interface registers. Longword reads and writes of the local Q22-bus memory space are buffered and translated into 2-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 16-word, block mode writes from the Q22-bus to main memory. These words are buffered, then transferred to main memory by using two asynchronous DMA octaword transfers. For block mode writes of less than 16 words, the words are buffered and transferred to main memory by 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/s.

  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, on block mode reads, the next three words of the block mode read can be delivered without any additional CP cycles. The maximum read bandwidth for Q22-bus block mode references is 2.4 Mbytes/s. 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 to BR4 and translates them into CPU interrupts at levels 17 to 14.

The Q22-bus interface detects Q22-bus NOSACK timeouts, Q22-bus interrupt acknowledge timeouts, Q22-bus nonexistent memory timeouts, main memory errors on DMA accesses from the Q22-bus, and Q22-bus device parity errors.

7.1.1 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 7–1.) This translation process is performed by the Q22-bus interface, using the Q22-bus map. This map contains 8192 mapping registers, one for each page in the Q22-bus memory space. Each of these registers 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 7–1 shows how Q22-bus addresses are translated into main memory addresses.

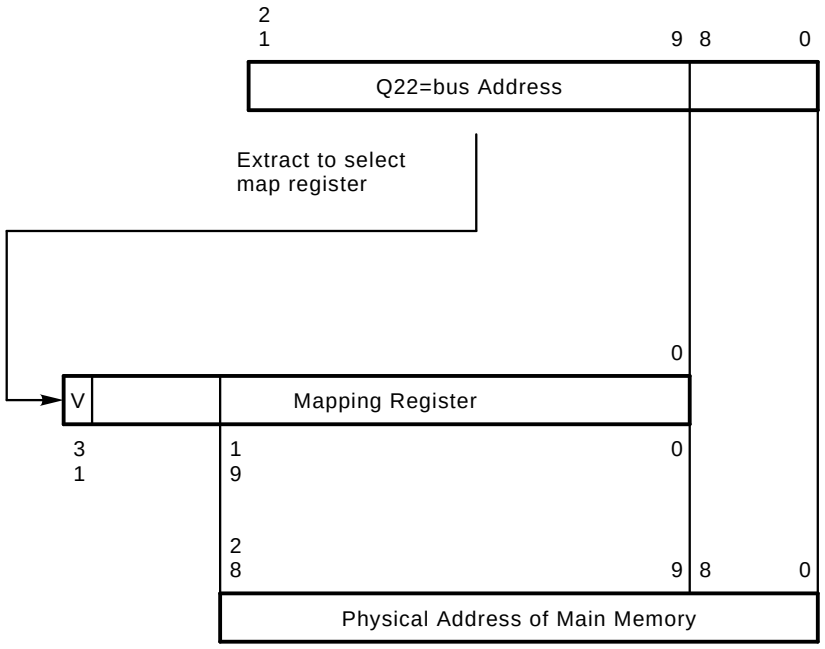


Figure 7–1 Q22-bus Address Translation

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

1. The interprocessor communication register's LM EAE bit is set.
2. The valid bit of the selected mapping register is set.
3. During read operations, the mapping register must map into existent main memory, or a Q22-bus timeout occurs. (During write operations, the Q22-bus interface returns Q22-bus BRPLY before checking for existent local memory. The response depends only on conditions 1 and 2 above). If the location pointed to by a valid MAP entry does not exist, MEMERR on the CP bus is asserted to cause an interrupt at IPL 1D.

NOTE

In the case of local-miss, global-hit transactions, the state of the LM EAE bit is ignored. A local-miss, global-hit is defined as follows. A CPU access of Q22-memory is mapped to main memory (global hit). However, the map entry for the Q22 address is not stored in the CQBIC's map cache (local miss). As a result, the map entry is read in memory before the original access can complete.

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

7.1.1.1 Q22-bus Map Registers (QMR)

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-kilobyte block of main memory, but are accessed through the CQBIC chip by using 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 that the register maps. Table 7–1 lists the register addresses.

Table 7–1 Q22-bus Map Register Addresses

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

Figure 7–2 shows the format of the Q22-bus map registers (QMRs). Table 7–2 lists the bit descriptions.

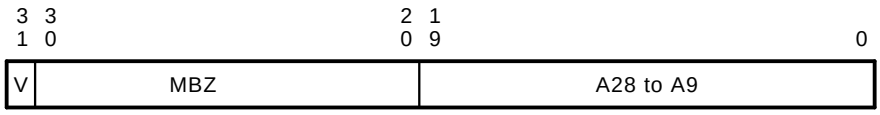


Figure 7–2 Q22-bus Map Register Format

Table 7–2 Q22-bus Map Register Bits

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; 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; the Q22-bus interface does not respond to addresses within that page. This bit is undefined on power-up or the negation of DCOK.
<30:20>	Unused	These bits always read as 0 and must be written as 0.
<19:0>	A28 to A9	Address bits <28:9> (read/write). When a Q22-bus map register is selected by a Q22-bus address, and 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 or the negation of DCOK.

7.1.1.2 Accessing the Q22-bus Map Registers

Although the CPU accesses the Q22-bus map registers by using aligned longword references to the local I/O page (addresses 2008 8000 to 2008 FFFC $_{16}$), the map actually resides in a 32-kilobyte 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 make the mapping function work incorrectly.

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 to 2008 FFFC$_{16}$). If the register is in the Q22-bus map cache, the Q22-bus interface internally resolves any conflicts between CPU and Q22-bus transactions (if both are trying to access the Q22-bus map cache entries at the same time), then returns the data.

If the map register is not in the map cache, the Q22-bus interface will force the CPU to retry, acquire the CP bus, and perform an asynchronous DMA read of the map register. When the read is complete, 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 first latches the data. On the completion of the CPU write, the interface acquires the CP 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 is 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.

7.1.1.3 The Q22-bus Map Cache

To speed up the process of translating Q22-bus addresses to main memory addresses, the Q22-bus interface uses a fully associative, 16-entry, Q22-bus map cache implemented in the CQBIC chip.

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 7–3 shows the format of a Q22-bus map cache entry. Table 7–3 lists the bit descriptions.

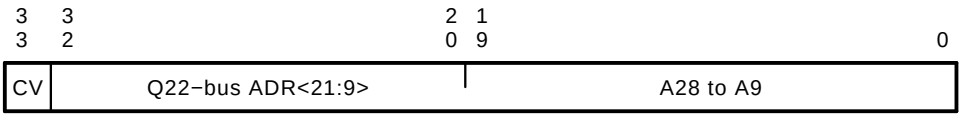


Figure 7–3 Q22-bus Map Cache Entry Format

Table 7–3 Q22-bus Map Cache Entry Bit Description

Data Bit	Name	Description
<33>	CAMValid	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 • By the negation of DCOK • By setting the Q22-bus map cache invalidate all (QMCIA) bit in the interprocessor communication register • On writes to IPR 55 (IORESET) • By a write to the Q22-bus map base register • By writing to the QMR 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 to A9	If a mapping register's CAMValid bit is set and the register is selected by a Q22-bus address, then these 20 bits are used as main memory address bits 28 to 9. Q22-bus address bits 8 to 0 are used as local memory address bits 8 to 0. These bits are undefined on power-up.

7.1.2 CP to Q22-bus Address Translation

CP bus addresses within the Local Q22-bus I/O space, addresses 2000 0000 to 2000 $1FFF_{16}$, are translated into Q22-bus I/O space addresses by using bits <12:0> of the CP bus address as bits <12:0> of the Q22-bus address and asserting BBS7. Q22-bus address bits <21:13> are driven as 0s.

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

7.1.3 Interprocessor Communications Facility

The KA670 can only be configured as a Q22-bus arbiter.

The KA670 interprocessor communication facility allows other processors on the Q22-bus to request program interrupts from the KA670 without using the Q22-bus interrupt request lines. It also controls external access to local memory (through the Q22-bus map).

7.1.3.1 Interprocessor Communication Register (IPCR)

The interprocessor communication register at address 2000 $1F40_{16}$ is a 16-bit register that resides in the Q22-bus I/O page address space. This register can be accessed by any device that can become Q22-bus master (including the KA670 itself). The IPCR is implemented in the CQBIC chip and is byte-accessible, which means a write byte instruction can write to either the low or high byte without affecting the other byte. Figure 7–4 shows the format of the IPCR register. Table 7–4 lists the bit descriptions.

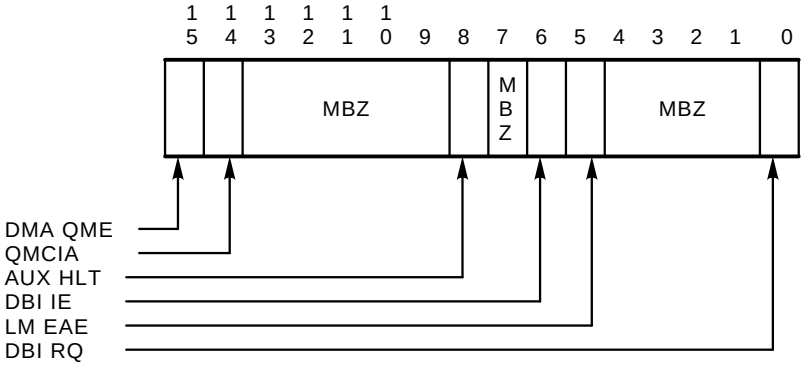


Figure 7–4 Interprocessor Communication Register (IPCR)

Table 7–4 Interprocessor Communication Register Bits

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 while a Q22-bus device was attempting to read main memory. The bit is set if DMA system error register bit DSER<4> (main memory error) is set, or the CP timer expires. The main memory error bit indicates that an uncorrectable error occurred when an external device (or CPU) was accessing the KA670 local memory. The CP timer expiring indicates that the memory controller did not respond when the Q22-bus interface initiated a DMA transfer. This bit is cleared by writing a 1 to it, on power-up, by the negation of DCOK, by writes to IPR 55 (IORESET), and whenever DSER<4> is cleared.
<14>	QMCIA	Q22-bus map cache invalidate all (write-only). Writing a 1 to this bit clears the CAMValid bits in the cached copy of the map. This bit always reads as 0. Writing a 0 has no effect.
<13:09>	Unused	Read as 0s. Must be written as 0s.
<8>	AUX HLT	Auxiliary halt (read-only). When set, this bit has no effect on the operation of the onboard CPU. This bit is cleared on power-up, by the negation of DCOK, and 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 0. Must be written as 0.

Table 7–4 (Cont.) Interprocessor Communication Register Bits

Data Bit	Name	Description
<6>	DBI IE	Doorbell interrupt enable. Read/write when the KA670 is Q22-bus master. Read-only when another device is Q22-bus master. When set, this bit enables interprocessor doorbell interrupt requests through IPCR<0>. This bit is cleared on power-up, the negation of DCOK, or writes to IPR 55 (IORESET).
<5>	LM EAE	Local memory external access enable. Read/write when the KA670 is Q22-bus master. Read-only when another device is Q22-bus master. When set, this bit enables external access to local memory (using the Q22-bus map). This bit is cleared on power-up or the negation of DCOK.
<4:1>	Unused	Read as 0s. Must be written as 0s.
<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 or the negation of DCOK.

7.1.3.2 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 1 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 KA670 CPU sets the IPL to 14. The IPL is set to 17 for external Q22-bus BR4 interrupts.

7.1.4 Q22-bus Interrupt Handling

The KA670 responds to interrupt requests BR7 to BR4 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 will interrupt at IPL 14. They have priority over all Q22-bus BR4 interrupt requests. After responding to any interrupt request BR7 to BR4, the CPU sets the processor priority to IPL 17. All BR7 to BR4 interrupt requests are disabled, unless software lowers the interrupt priority level.

Interrupt requests from the KA670 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.

7.1.5 Configuring the Q22-bus Map

The KA670 implements the Q22-bus map in an 8K longword (32-kilobyte) block of main memory. This map must be configured by the KA670 firmware during a processor initialization. The map is configured by writing the base address of the uppermost 32-kilobyte block of good main memory into the Q22-bus map base register. The base of this map must be located on a 32-kilobyte boundary.

NOTE
This 32-kilobyte 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 to 2008 FFFC $_{16}$.

7.1.5.1 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-kilobyte 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 0;they will return 0 when read. Figure 7–5 shows the format of the register.

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

The contents of this register are undefined on power-up or the negation of DCOK. The contents are not affected by the assertion of BINIT on the Q22-bus.

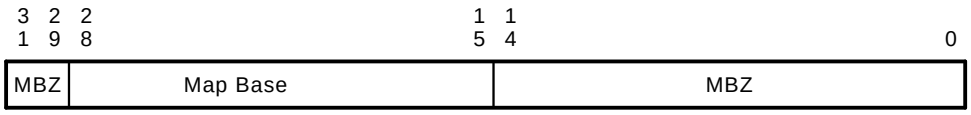


Figure 7–5 Q22-bus Map Base Address Register (QBMBR)

7.1.6 System Configuration Register (SCR)

The system configuration register, address 2008 0000 $_{16}$, contains the processor number that determines the address of the IPCR register, a BHALT enable bit, a power okay flag and an auxiliary flag. Figure 7–6 shows the format of the register. Table 7–5 lists the bit descriptions.

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

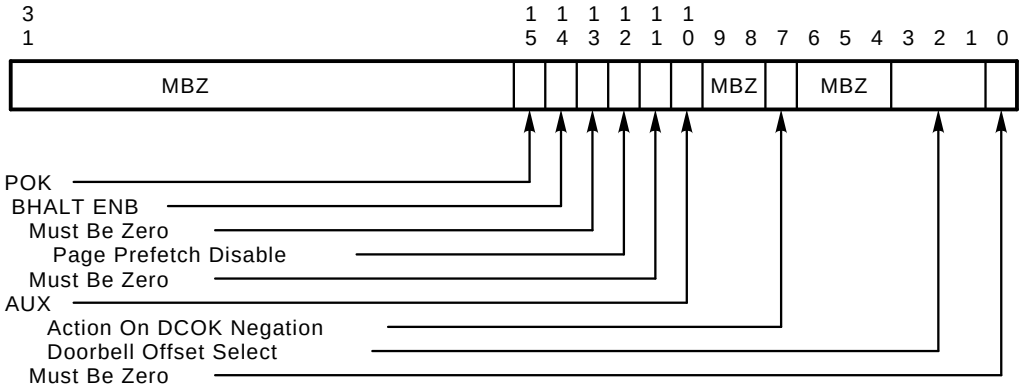


Figure 7–6 System Configuration Register (SCR)

Table 7–5 System Configuration Register Bits

Data Bit	Name	Description
<31:16>	Unused	Read as 0. Must be written as 0.
<15>	POK	Power okay (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 or the negation of DCOK.
<14>	BHALT EN	BHALT enable (read/write). This bit controls the effect of Q22-bus BHALT signal on the CPU. When the bit is set, asserting the Q22-bus BHALT signal halts the CPU and assert DSER<15>. When the bit is cleared, the Q22-bus BHALT signal has no effect. This bit is cleared on power-up or the negation of DCOK.
<13>	Unused	Read as 0. Must be written as 0.
<12>	Page Prefetch Disable	Read/write. This bit should be set on the KA670. When set, this bit prohibits the CQBIC from prefetching the map when a Q22-bus transaction address reaches a page boundary. Stopping MAP prefetching buys back some needed CP bus bandwidth and lowers the CP devices latency. This bit is cleared on power-up or the negation of DCOK.
<11>	Unused	Read as 0. Must be written as 0.
<10>	AUX	Auxiliary (read-only). Writes have no effect. This bit defines the auxiliary and arbiter modes of operation of the KA670. When read as a 0, arbiter mode is selected. When read as a 1, auxiliary mode is selected. Because the KA670 can only be configured as an arbiter, this bit should always read as 0.
<9:8>	Unused	Read as 0. Must be written as 0.
<7>	Action on DCOK Negation	Read/write. If DCOK is negated on the Q22-bus, clearing this bit causes the Q22-bus interface to assert SYSRESET. This action causes a hardware reset of the board; control is passed to the resident firmware, using the hardware halt procedure with a halt code of 3. If DCOK is negated on the Q22-bus, setting this bit causes the Q22-bus interface to assert HALCYON. This action passes control to the resident firmware, using the hardware halt procedure with a halt code of 2. This bit is cleared on power-up or the negation of DCOK.
<6:4>	Unused	Read as 0. Must be written as 0.
<3:1>	Reserved	Reserved for use by Digital.
<0>	Unused	Read as 0. Must be written as 0.

7.1.7 Error-Reporting Registers

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

- DMA system error register (DSER)
- Q22-bus error address register (QBEAR)
- DMA error address register (DBEAR)

These registers are in the local VAX I/O address space. They can only be accessed by the local processor.

The **DSER** is implemented in the CQBIC chip. This register logs main memory errors on DMA transfers, Q22-bus parity errors, Q22-bus nonexistent memory errors, and a Q22-bus no grant condition.

The **QBEAR** contains the address of the page in Q22-bus space that caused a parity error during an access by the local processor.

The **DBEAR** contains the address of the page in local memory that caused a memory error during an access by an external device or the processor in a local-miss, global-hit transaction. Any access by the local processor that the Q22-bus interface maps into main memory will provide the procesor with (1) error status when the processor does a retry for a read local-miss, global hit, or (2) an interrupt in the case of a local-miss global-hit write.

7.1.7.1 DMA System Error Register (DSER)

The DSER, address 2008 0004_{16}, is a longword, word, or byte-accessible register available to the local processor. The bits in this read/write register are cleared to 0 on power-up, the negation of DCOK or writes to IPR 55 (IORESET). All bits are set to 1, to record the occurrence of an event. They are cleared by writing a 1. Writing 0s has no effect.

Figure 7–7 shows the format of the register. Table 7–6 lists the bit descriptions.

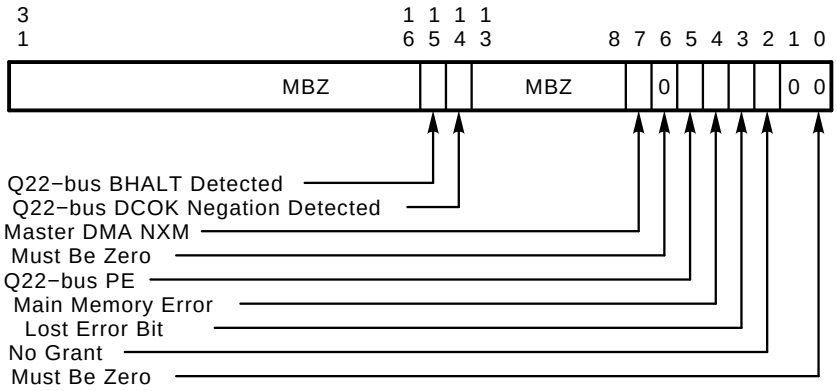


Figure 7–7 DMA System Error Register (DSER)

Table 7–6 DMA System Error Register Bits

Data Bit	Name	Description
<31:16>	Unused	Read as 0. Must be written as 0.
<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. The bit is cleared by writing a 1, writes to IPR 55 (IORESET), power-up, or 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. This bit is cleared by writing a 1, writes to IPR 55 (IORESET), power-up, or the negation of DCOK.

Table 7–6 (Cont.) DMA System Error Register Bits

Data Bit	Name	Description
<13:8>	Unused	Read as 0. Must be written as 0.
<7>	MASTER DMA NXM	Read/Write to clear. This bit is set when the CPU performs a demand Q22-bus read cycle or write cycle that does not reply after 10 µs. During interrupt acknowledge cycles or request read cycles, this bit is not set. The bit is cleared by writing a 1, writes to IPR 55 (IORESET), power-up, or the negation of DCOK.
<6>	Unused	Read as 0. Must be written as 0.
<5>	Q22-bus parity error	Read/Write to clear. This bit is set when the CPU performs a Q22-bus demand read cycle that returns a parity error. This bit is not set during interrupt acknowledge cycles, or request read cycles. The bit is cleared by writing a 1, writes to IPR 55 (IORESET), power-up, or the negation of DCOK.
<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. This bit is cleared by writing a 1, writes to IPR 55 (IORESET), power-up, or the negation of DCOK.
<3>	Lost error	Read/write to clear. This bit indicates that an error address was lost because DSER<7,5,4,0> was previously set and a subsequent error of either type occurred that normally would have captured an address and set either DSER<7,5,4,0> flag. The bit is cleared by writing a 1, writes to IPR 55 (IORESET), power-up, or the negation of DCOK.
<2>	No grant timeout	Read/write to clear. This bit is set if the Q22-bus does not return a bus grant within 10 ms of the bus request from a CPU demand read cycle or write cycle. This bit is not set during interrupt acknowledge or request read cycles . The bit is cleared by writing a 1, writes to IPR 55 (IORESET), power-up, or the negation of DCOK.
<1:0>	Unused	Read as 0. Must be written as 0.

7.1.7.2 Q22-bus Error Address Register (QBEAR)

The Q22-bus error address register, address 2008 0008 $_{16}$, is a read-only, longword-accessible register 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. Figure 7–8 shows the format of the register.

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

The QBEAR contains the address of the page in Q22-bus space that caused

- A parity error during an access by the onboard CPU, which set DSER<5>
- A master timeout that set DSER<7>

Q22-bus address bits <21:9> are loaded into QBEAR bits <12:0>. QBEAR bits <31:13> always read as 0s.

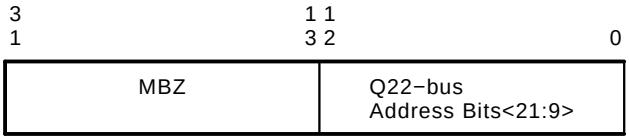


Figure 7–8 Q22-bus Error Address Register (QBEAR)

NOTE
This is a read-only register. Attempts to write will generate a hard error (IPL 1D).

7.1.7.3 DMA Error Address Register (DBEAR)

The DMA error address register, address 2008 000C $_{16}$, is a read-only, longword-accessible register implemented in the CQBIC chip. The register contains valid information only when DSER<4> (main memory error) is set. Reading this register when DSER<4> is clear will return undefined data. Figure 7–9 shows the format of the register.

The DBEAR contains the map-translated address of the page in local memory that caused a memory error or nonexistent memory error during an access by:

- An external device
- 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> is set. Additional main memory errors or nonexistent memory errors have no effect on the DBEAR until software clears DSER<4> .

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

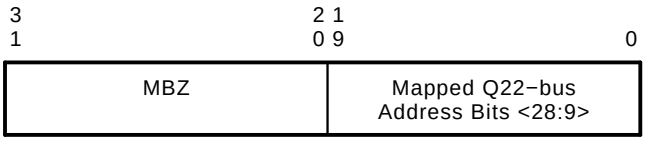


Figure 7–9 DMA Error Address Register (DBEAR)

NOTE
This is a read-only register. Attempting a write will generate a hard error (IPL 1D).

7.1.8 Error Handling

Parity

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

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

If a parity error is detected by another Q22-bus device on a CPU demand read reference to Q22-bus memory or I/O space, then 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, DSER<5> is set, and the address of the Q22-bus page being accessed is captured in QBEAR<12:0>. However, no machine check is generated.

Memory and I/O Space

The Q22-bus interface checks all CPU references to Q22-bus memory and I/O spaces to ensure only masked and unmasked longword accesses are attempted. Any other type of reference initiates a machine check abort.

Timers

The Q22-bus interface maintains several timers to prevent incomplete accesses from hanging the system indefinitely. They include a 10µs nonexistent 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 ms NO GRANT timer for acquiring the Q22-bus.

Nonexistent Memory

If there is a nonexistent memory (NXM) error (10 µs timeout) while accessing the Q22-bus on a demand read reference, 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 a NXM error on a prefetch read or an interrupt acknowledge vector read, then the prefetch or interrupt acknowledge reference is aborted. However, no information is captured and no machine check occurs.

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

Bus Grants

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

If the Q22-bus interface tries and fails to obtain Q22-bus mastership within 10 ms on a CPU demand read reference, DSER<2> is set and a machine check abort is initiated.

Power Failures

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 power-fail trap, until DCOK is negated.

7.2 KA670 Network Interface

The KA670 includes a network interface , implemented through the second-generation Ethernet controller chip (SGEC). When used in conjunction with the H3604 cover panel, this interface allows the KA670 to be connected to either a ThinWire or standard Ethernet network. The interface supports the Ethernet data link layer as specified in the *VAX Architecture Reference Manual*. The SGEC also supports CP bus parity protection.

7.2.1 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/second in variable-length packets. Each packet has the format shown in Figure 7–10.

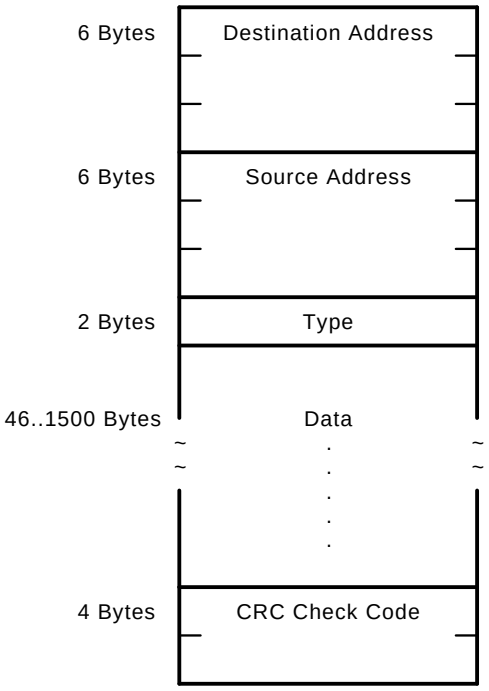


Figure 7–10 Ethernet Packet Format

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 called 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).

- When the bus is clear, all devices that want to access the bus have equal priority (multiaccess), 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.

- *Physical address:* The unique address associated with a particular station on the Ethernet. This address should be distinct from the physical address of any other station on any other Ethernet.
- *Multicast address:* A multidestination 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 that denotes the set of all 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 forty-eight 1s is always treated as the broadcast address.

The hardware address of the KA670 module is determined at the time of manufacture. The address is stored in the network interface station address (NISA) ROM. Because every device that connects to an Ethernet network must have a unique physical address, the bit pattern blasted into the NISA ROM must be unique for each KA670 . The multicast addresses that the KA670 will respond to are determined by the multicast address filter mask in the network interface initialization block.

7.2.2 NI Station Address ROM (NISA ROM)

The network interface includes a byte-wide, 32-byte, socketed ROM called the network interface station address ROM. One byte of this ROM appears in the second byte of each of 32 consecutive longwords in the address range 2008 4000 to 2008 $407C_{16}$. Bytes one, three, and four of each longword are defined in the boot and diagnostic register (Section 6.1). The second byte of the first six longwords contain the 48-bit network physical address (NPA) of the KA670 . The low-order byte in the remaining 26 longwords is for testing. This address range is read-only. Writes to this address range will complete with no effect.

7.3 Programming the Ethernet Controller Chip (SGEC)

The operation of the second-generation Ethernet controller chip (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
- *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. The host memory resident structures handle the actions and statuses related to buffer management.

7.3.1 Programming Overview

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
- Suspended

State transitions occur as a result of port driver commands writing to a NICSR, or various external events. Some of the port driver commands require the referenced process to be in a specific state.

Here is a summary of a simple programming sequence of the chip:

1. After power-up or reset, verify the self-test completed successfully.
2. Write NICSRs to set major parameters such as the system base register, interrupt vector, address filtering mode, and so on.
3. Create the transmit and receive lists in memory, and write the NICSRs to identify them to the SGEC.
4. Place a setup frame in the transmit list, to load the internal reception address-filtering table.
5. Start the reception and transmission processes, placing them in the running state.
6. Wait for SGEC interrupts. NICSR5 contains all the global interrupt status bits.
7. If either the reception or transmission process enters the suspended state, correct the cause of the suspension:
   - Issue a TX POLL DEMAND command to return the transmission process to the running state.
   - If desired, issue an RX POLL DEMAND command 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.

7.3.2 Command and Status Registers

The SGEC contains 16 command and status registers that the host can acces.

7.3.3 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 NICSR*x* is the base address plus 4*x* 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 specifed with several access modes. Table 7–7 lists the different access modes for bits.

Table 7–7 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/write.
W	Write-only. Unpredictable on read.
R/W1	Read, or clear by writing a 1. Writing with a 0 has no effect.

In order to save chip space, but not tie up the host bus for extended periods of time, the 16 NICSRs are divided into two groups:

1. Physical NICSRs–0–to 7, 15.
2. Virtual NICSRs–8–to 14.

The group that a NICSR belongs to determines the way the host accesses that NICSR.

7.3.3.1 Physical NICSRs

These registers are physically present in the chip. The host accesses these NICSRs 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.

7.3.3.2 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 may take up to 20 µs. To avoid tieing up the host bus, virtual NICSR access requires several steps by the host.

The NICSR5<DN> signal is used to synchronize access to the virtual NICSRs. After accessing the first virtual NICSR access, the SGEC deasserts NICSR5<DN> until it completes the action.

NOTE

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

7.3.3.2.1 Virtual NICSR Write

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

1. Issues a write NICSR instruction. The instruction completes immediately, but the data is not yet copied by the SGEC.
2. Waits for NICSR5<DN>. *The host cannot access any SGEC virtual NICSR before NICSR5<DN> asserts.*

7.3.3.2.2 Virtual NICSR Read

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

1. Issues a read NICSR instruction. The instruction completes immediately, but no valid data is sent to the host.
2. Waits for NICSR5<DN>. *The host cannot access any SGEC virtual NICSR before NICSR5<DN> asserts.*
3. Reissues a read NICSR instruction to the *same* NICSR as in step 1. The host receives valid data.

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

During host writes to NICSRs, the SGEC may generate an interrupt on parity errors. For this reason, the NICSR0 register must be the first one written by the host. Figure 7–11 shows the format of the register. Table 7–8 lists the bit descriptions.

Parity Errors

A parity error during a NICSR0 host write may cause a host system crash, due to an erroneous interrupt vector. To prevent such an error, NICSR0 must be written as follows while the SGEC's assigned IPL is disabled:

1. Write NICSR0.
2. Read NICSR0.
3. Compare the value read to the value written. If the values do not match, return to step 1.
4. Read NICSR5 and examine NICSR5<ME> for a pending parity interrupt. If an interrupt is pending, write NICSR5 to clear it.

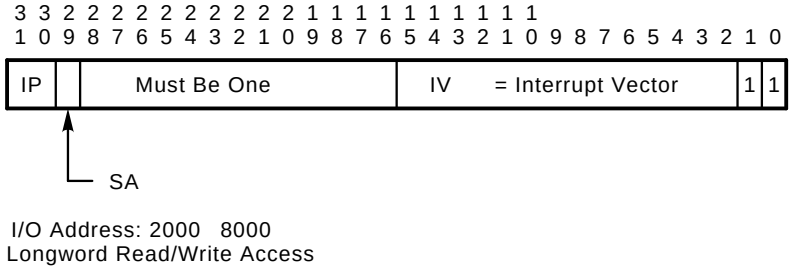


Figure 7–11 Vector Address, IPL, Sync/Asynch (NICSR0)

Table 7–8 NICSR0 Bits

Bit	Name	Access	Description
<31:30>	IP	R/W	Interrupt Priority. These bits indicate the VAX interrupt priority level that the SGEC will respond to, as follows: **IP** / **IPL_{16}** 00 / 14 01 / 15 10 / 16 11 / 17 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 correspond to the IPL level that the pin is wired to.
<29>	SA	R/W	Synchronous/asynchronous. This bit determines the SGEC's operating mode when it is the bus master. When set, the SGEC operates as a synchronous device. When clear, the SGEC operates as an asynchronous device.
<15:00>	IV	R/W	Interrupt vector. During an interrupt acknowledge cycle for an SGEC interrupt, these bits contain the value that the SGEC will drive on the host bus CDAL<31:0> pins. (CDAL pins <1:0> and <31:16> are set to 0.) Bits <1:0> are ignored when NICSR0 is written, and set to 1 when read.

NICSR0 Access

Value after reset	$1FFF0003_{16}$
Read access rules	None.
Write access rules	The SGEC's assigned IPL must be disabled.

7.3.5 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. Figure 7–12 shows the format of the register. Table 7–9 lists the bit descriptions.

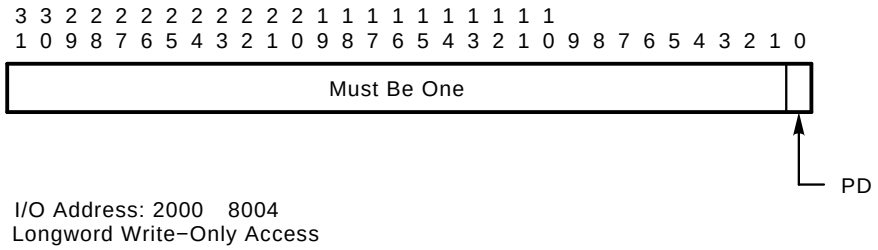


Figure 7–12 Transmit Polling Demand (NICSR1)

Table 7–9 NICSR1 Bits

Bit	Name	Access	Description
<31:01>	MBZ		Must be one. 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.

NICSR1 Access

Value after reset	Not applicable.
Read access rules	None.
Write access rules	Transmission process suspended.

7.3.6 Receive Polling Demand (NICSR2)

The receive polling demand NICSR (NICSR2) is used by the port driver to tell the SGEC that it put a packet on the receive list. Figure 7–13 shows the format of the register. Table 7–10 lists the bit descriptions.

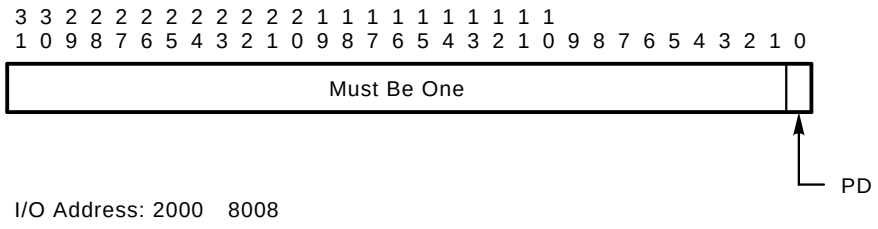


Figure 7–13 NICSR2 Format

Table 7–10 NICSR2 Bits

Bit	Name	Access	Description
<31:01>	MBZ		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.

NICSR2 Access

Value after reset	Not applicable.
Read access rules	None.
Write access rules	Receive process suspended.

7.3.7 Descriptor List Addresses (NICSR3, NICSR4)

The two descriptor list address registers are identical in function. One is for the transmit buffer descriptors, and one is for the receive buffer descriptors. In both cases, the registers serve 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.

For best performance, it is recommended that the descriptor lists 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* (Section 7.3.9). Otherwise, the respective process will remain in the stopped state. New list 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 tries to read any of these registers before ever writing to them, the SGEC responds with unpredictable values. Figure 7–14 shows the format of the registers. Table 7–11 lists the bit descriptions.

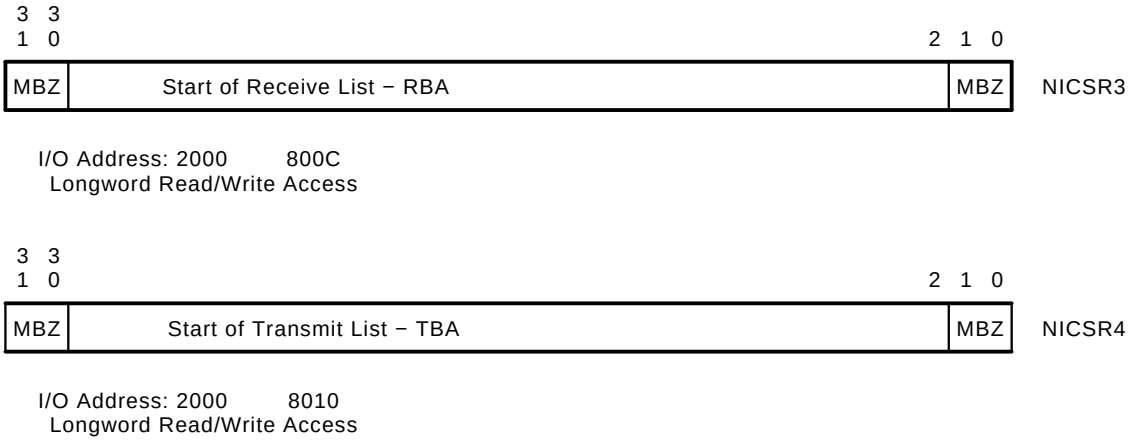


Figure 7–14 Descriptor List Addresses Format

Table 7–11 Descriptor List Address Bits

Bit	Name	Access	Description
<31:30>	MBZ		Must be 1. Ignored on writes. Read as 0.
<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.

NICSR3 Access

Value after reset:	Unpredictable.
Read access rules:	None.
Write access rules:	Receive process stopped or suspended.

NICSR4 Access

Value after reset	Unpredictable.
Read access rules	None.
Write access rules	Transmit process stopped or suspended.

After NICSR3 or NICSR4 is 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 the right condition later*.

7.3.8 Status Register (NICSR5)

This register contains all the status bits the SGEC reports to the host. Figure 7–15 shows format of the register format. Table 7–12 lists the bit descriptions.

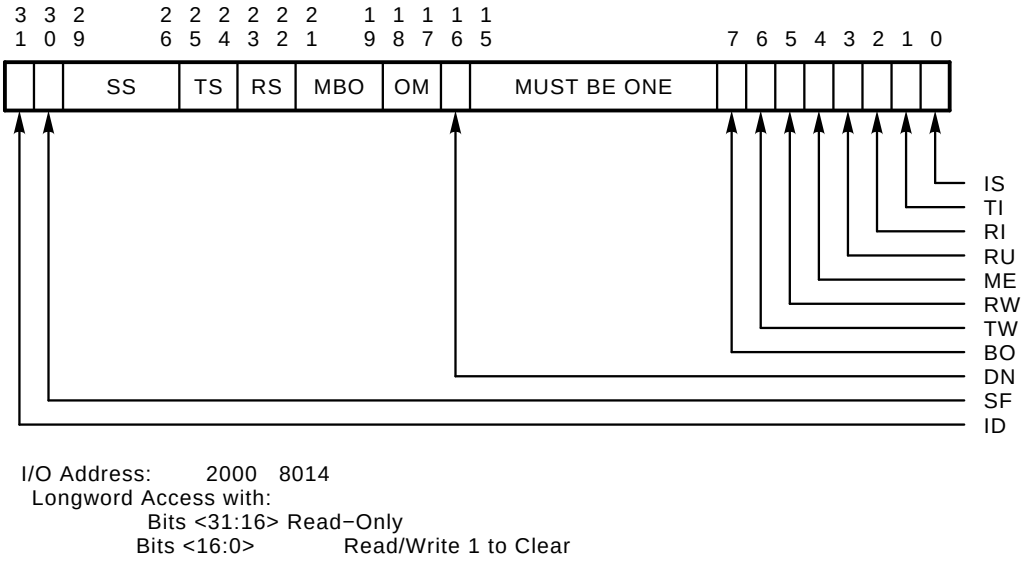


Figure 7–15 NICSR5 Format

Table 7–12 NICSR5 Bits

Bit	Name	Access	Description
<31>	ID	R	Initialization done. When set, this bit indicates the SGEC has completed the initialization (reset and self-test) sequences and is ready for further commands. When clear, this bit 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, this bit indicates the SGEC self-test has failed. The self-test completion code bits indicate the failure type.

Table 7–12 (Cont.) NICSR5 Bits

Bit	Name	Access	Description
<29:26>	SS	R	Self-test status. This field provides the self-test completion code, according to the following table. the code is 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

The self-test takes 25 ms to complete after the hardware or software reset.

<25:24> TS R Transmission process state. This field indicates the current state of the Transmission process, as follows:

Value	Meaning
00	Stopped
01	Running
10	Suspended

Section 7.3.24 explains the transmission process operation and state transitions.

<23:22> RS R Reception process state. This field indicates the current state of the Reception process, as follows:

Value	Meaning
00	Stopped
01	Running
10	Suspended

Section 7.3.23 explains the reception process operation and state transitions.

Table 7–12 (Cont.) NICSR5 Bits

Bit	Name	Access	Description
<18:17>	OM	R	Operating mode. These bits indicate the current SGEC operating mode, as follows:

Value	Meaning
00	Normal operating mode.
01	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 7.3.25 explains this mode of operation.
10	External Loopback—Indicates the SGEC is working in full-duplex mode. Frames from the transmit list are transmitted on the Ethernet wire and looped back to the receive list, subject to address filtering. Section 7.3.25 explains this mode of operation.
11	Reserved for diagnostics.

<16>	DN	R	Done. When set, this bit indicates the SGEC has completed a requested virtual NICSR access. After a reset, this bit is set.
<15:8>	MBO		Must be 1. This field is reserved. Read as 1. Writes are ignored.
<7>	BO	R/W1	Boot message. When set, this bit indicates that the SGEC has detected a boot_message on the serial line and has set the external pin BOOT_L.
<6>	TW	R/W1	Transmit watchdog timer interrupt. When set, this bit 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. This is also reported into the transmission descriptor status TDES0<TO> flag.

Table 7–12 (Cont.) NICSR5 Bits

Bit	Name	Access	Description
<5>	RW	R/W1	Receive watchdog timer interrupt. When set, this bit 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> is also set. The reception process remains in the running state.
<4>	ME	R/W1	Memory error. This bit isset when any of the followings occur: • SGEC is the CP bus master, and the ERR_L pin is asserted by external logic (generally indicating a memory problem). • A parity error was detected on a 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, the port driver must issue a reset command and rewrite all NICSRs.**
<3>	RU	R/W1	Receive buffer unavailable. When set, this bit 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 7.3.23 explains the reception process state transitions. After being 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 flp 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, this bit indicates that a frame has been placed on the receive list. Frame-specifi status information was posted in the descriptor. The reception process remains in the running state.

Table 7–12 (Cont.) NICSR5 Bits

Bit	Name	Access	Description
<1>	TI	R/W1	Transmit interrupt. When set, this bit 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 7.3.24 explains the transmission process state transitions. To resume processing transmit descriptors, the port driver must issue the TX 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 to 6.

NICSR5 Access

Value after reset	$0039FF00_{16}$.
Read access rules	None.
Write access rules	NICSR5<07:01> bits cleared by 1, others bits not writeable.

7.3.8.1 NICSR5 Status Report

The NICSR5 status register is divided into two words:

- High word—Contains the global status of the SGEC (as the initialization status), the DMA and operation mode, and the receive and transmit process states.
- Low word—Contains the status related to the receive and transmit frames.

Any change of the NICSR5 bits <ID>, <SF>, <OM>, or <DN> is always the result of a host command. These changes are 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 two cases above, the driver must poll on NICSR5 to get acknowledgement of its command—for example, polling on <ID, SF> after a 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-bit NICSR5 register in two phases:

- The high word is modified first.
- Then the low word is written, which generates an interrupt to the host.

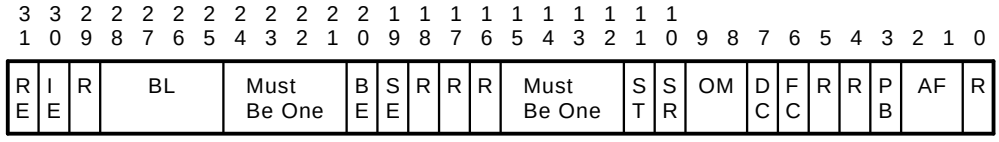
In this case, the driver must scan the NICSR5 low word to get the interrupt status, then scan the NICSR high word to get the related process state. For example, <TI> interrupt with <TS> = SUSPENDED reports an end of transmission due to a Tx descriptor unavailable.

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: 4 × *Tcycles* of the host clock

7.3.9 Command and Mode Register (NICSR6)

This register serves to establish operating modes and issue port driver commands. Figure 7–16 shows the format of the register. Table 7–14 lists the bit descriptions.



I/O Address: 2000 8018
Longword Read/Write Access

R = reserved.

Figure 7–16 NICSR6 Format

Table 7–14 NICSR6 Bits

Bit	Name	Access	Description
<31>	RE	R/W	Reset command. When this bit is set, the SGEC will abort all processes and start 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 reads after a hardware reset.**
<30>	IE	R/W	Interrupt enable mode. When this bit is set, setting NICSR5 bits 1 to 6 will generate an interrupt.
<29>	R		Reserved.

Table 7–14 (Cont.) NICSR6 Bits

Bit	Name	Access	Description
<28:25>	BL	R/W	Burst limit mode. This field 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, and 8. When SE is set, the only permissible values are 1 and 4. Values of 2 or 8 are forced to 1 or 4, respectively. After initialization, the burst limit is set to 1.
<24:21>	MBO		This field is reserved. Writes are ignored. Read as 1.
<20>	BE	R/W	Boot message enable mode. When set, this bit 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 this bit is 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 1.

Table 7–14 (Cont.) NICSR6 Bits

Bit	Name	Access	Description
<11>	ST	R/W	Start/stop transmission command. When this bit is 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 transmission 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, the NICSR4 must already been written to. Otherwise, the transmission process remains in the stopped state. When this bit is 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. See Section 7.3.24 for more information.

Table 7–14 (Cont.) NICSR6 Bits

Bit	Name	Access	Description
<10>	SR	R/W	Start/stop reception command. When this bit is set, the reception process is placed in the running state, the SGEC tires to acquire a descriptor from the receive list and process incoming frames. Descriptor acquisition is attempted from the *current* position in the list the address set by NICSR3 or the position retained when the reception 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. The first time this command is issued, NICSR3 must already have been written to. Otherwise, the reception process remains in the stopped state. When this bit is cleared, the reception process is placed in the stopped state after completing reception of the current frame. The next descriptor position in the receive list is saved and becomes the *current* position after reception is restarted. The stop reception command is honored only when the reception process is in the running or suspended states. See Section 7.3.23 for more information.

Table 7–14 (Cont.) NICSR6 Bits

Bit	Name	Access	Description
<9:8>	OM	R/W	Operating mode. These bits determine the SGEC's main operating mode.

Value	Meaning
00	Normal operating mode.
01	Internal loopback—The SGEC will loop back 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, subjecting it to the address filtering and validity check process.
10	External loopback—The SGEC transmits normally and enables its receive logic to receive its own transmissions. The receive logic treats the looped frame as it would any other frame, subjecting it to the address filtering and validity check process.
11	Reserved for diagnostics.

<7>	DC	R/W	Disable data chaining mode—When this bit is set, no data chaining occurs in receptions. Frames that are longer than the current receive buffer are truncated. RDES0<FS,LS> will always be set. The frame length returned in RDES0<FL> will be the *true* length of the nontruncated frame, while RDES0<BO> will indicate that the frame has been truncated due to buffer overflow. When this bit is clear, frames that are too long for the current receive buffer are transferred to the next buffer(s) in the receive list.
<6>	FC	R/W	Force collision mode—This bit allows the collision logic to be tested. The chip must be in **internal loopback** mode for FC to be valid. If this bit is set, a collision is forced during the next transmission attempt. The collision results in 16 transmission attempts, with excessive collision reported in the transmit descriptor.

Table 7–14 (Cont.) NICSR6 Bits

Bit	Name	Access	Description
<5:4>	MBO		Must be 1. This field is reserved. Writes are ignored. read as 1.
<3>	PB	R/W	Pass bad frames mode. When this bit is set, the SGEC passes frames that have been damaged by collisions or that are too short due to premature reception termination. Both events should have occurred within the collision window (64 bytes). Otherwise, other errors will be reported. When this bit is clear, these frames are discarded and never show up in the host receive buffers. **NOTE** **Pass bad frames mode is subject to the address filtering mode. For example, to monitor the network, this mode must be set together with the promiscuous value of address filtering mode.**
<2:1>	AF	R/W	Address filtering mode. These bits define the way incoming frames will be address-filtered:
<0>	MBO		Must be 1. This field is reserved. Writes are ignored. Read as 1.

Value	Meaning
00	Normal—Incoming frames are filtered according to the values of the <HP> and <IF> bits of the setup frame descriptor.
01	Promiscuous—All incoming frames are passed to the host, regardless of the <HP> bit value.
10	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.
11	Unused—Reserved.

NICSR6 Access

Value after reset	$83E0F000_{16}$ or $03E0F000_{16}$.
Read access rules	None.
Write access rules	
• <RE, IE, BE>	Unconditional.
• <BL, SE, OM>	Reception and transmission processes stopped.
• <FC>	Reception and transmission processes stopped, internal loopback mode
• <DC, PB, AF>	Reception process stopped.
• Start receive <SR>=1	Reception process stopped and NICSR3 initialized.
• Start transmit <ST>=1	Transmission stopped and NICSR4 initialized.
• Stop receive <SR>=0	Reception process running or suspended.
• Stop transmit <ST>=0	Transmission 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 the right condition later*.

7.3.10 System Base Register (NICSR7)

This NICSR contains the physical starting address of the VAX system page table. The host software must load this register before any address translation occurs, so that memory is not corrupted. Figure 7–17 shows the format of the register. Table 7–15 lists the bit descriptons.

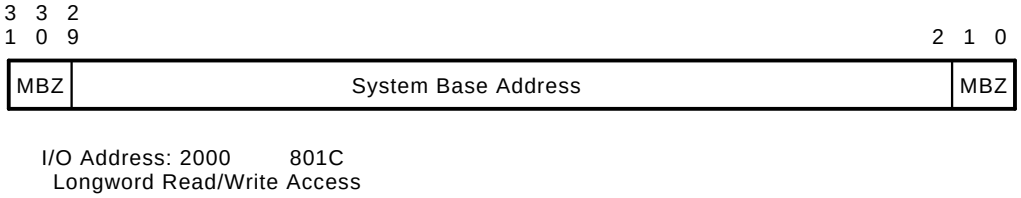


Figure 7–17 NICSR7 Format

Table 7–15 NICSR7 Bits

Bit	Name	Access	Description
<31:30>	MBZ		Must be 0. Read as 0. Writes are ignored.
<29:00>	SB	R/W	System base address. The physical starting address of the VAX system page table. Not used if virtual addressing (VA) is cleared in all descriptors. **This register should be loaded only one time after a reset. Subsequent modifications of this register may cause unpredictable results.**

NICSR7 Access

Value after reset	Unpredictable.
Read access rules	None.
Write access rules	Writing once after initialization.

7.3.11 Reserved Register (NICSR8)

This register is reserved.

7.3.12 Watchdog Timers (NICSR9)

The SGEC has two timers that restrict the length of time in which the chip can receive or transmit. Figure 7–18 shows the format of the register. Table 7–16 lists the bit descriptions.

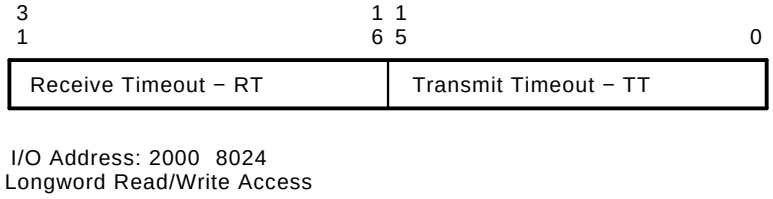


Figure 7–18 NICSR9 Format

Table 7–16 NICSR9 Bits

Bit	Name	Access	Description
<31:16>	RT	R/W	Receive watchdog timeout. 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 will cut off reception and set the NICSR5<RW> bit. If the timer is set to 0, it will never time out. The value of RT is an unsigned integer. With a 10 Mhz serial clock, this provides a range of 72 µs to 100 ms. The default RT value is 1250, corresponding to 2 ms. The Rx watchdog timer is programmed only while the reception process is in the stopped state. **NOTE** **A receive watchdog value between 1 and 44 is forced to the minimum timeout value of 45 (72 µs).**

Table 7–16 (Cont.) NICSR9 Bits

Bit	Name	Access	Description
<15:00>	TT	R/W	Transmit watchdog timeout. The transmit watchdog timer protects the network against babbling SGEC transmissions, in addition to any such circuitry present in tranceivers. If the transmitter stays on for *TT* × 16 cycles of the serial clock, the SGEC will cut off the transmitter and set the NICSR5<TW> bit. If the timer is set to 0, it will never time out. The value of TT is an unsigned integer. With a 10 Mhz serial clock, this provides a range of 72 µs to 100ms. The default TT value is 1250, corresponding to 2 ms. The transmit watchdog timer is programmed only while the transmission process is in the stopped state. **NOTE** **A transmit watchdog value between 1 and 44 is forced to the minimum timeout value of 45 (72 µs).**

NICSR9 Access

Value after reset	00000000_{16}.
Read access rules	None.
Write access rules	
• Receive watchdog timer	Reception process stopped.
• Tranmit watchdog timer	Transmission process stopped.

The transmit and receive watchdog timers are enabled by default. These timers are set to their default values after hardware or software resets.

7.3.13 Revision Number and Missed-Frame Count (NICSR10)

This register contains a missed-frame counter and SGEC identification information. Figure 7–19 shows the register format. Table 7–17 lists the bit descriptions.

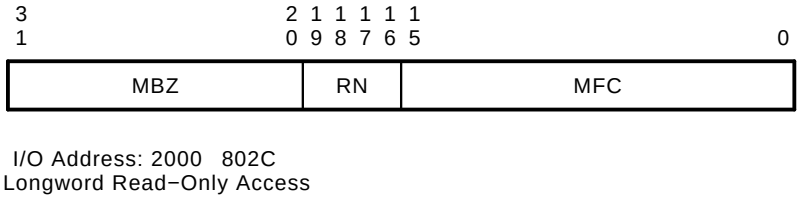


Figure 7–19 NICSR10 Format

Table 7–17 NICSR10 Bits

Bit	Name	Access	Description
<31:21>	MBZ		Must be 0. Read as 0. Writes are ignored.
<20:16>	RN	R	Chip revision number. This field stores the revision number for this particular SGEC.
<15:00>	MFC	R	Missed frame count. This field 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.

NICSR10 Access

Value after reset	00030000_{16}.
Read access rules	Missed-frame counter cleared by read.
Write access rules	Not applicable.

7.3.14 Boot Message (NICSR11, 12, 13)

These registers contain the boot message verification and processor fields. Figure 7–20 shows the format of the registers. Table 7–18 lists the bit descriptions.

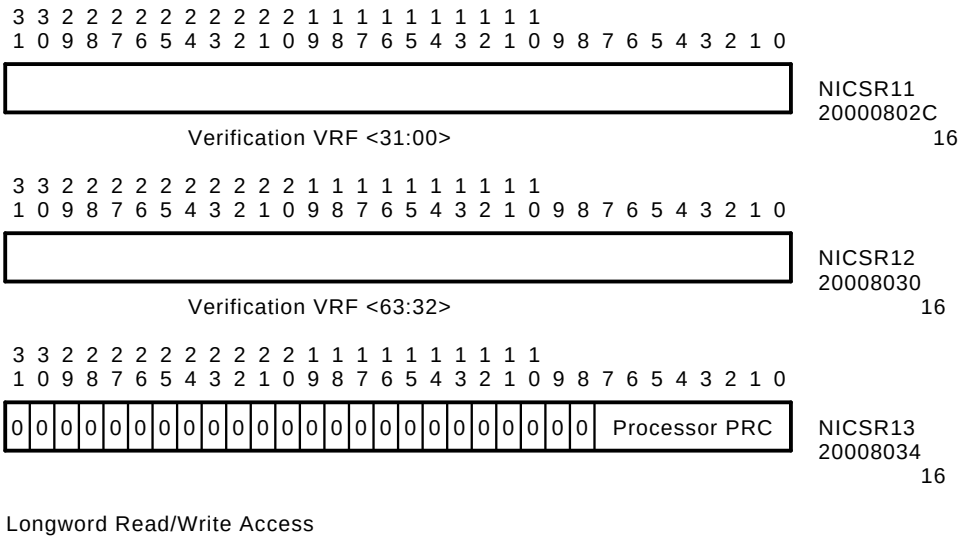


Figure 7–20 Boot Message

Table 7–18 NICSR11,12,13 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.

NICSR11,12,13 Access

Value after reset	00000000_{16} for each register—NICSR11, NICSR12, and NICSR13.
Read access rules	None.
Write access rules	Boot message disabled (<NICSR6<BE> = 0.)

7.3.15 Diagnostic Registers (NICSR14,15)

These registers are reserved for diagnostic features.

7.3.15.1 Diagnostic Breakpoint Address Register (NICSR14)

This register is a virtual CSR. It contains the breakpoint address that causes the internal CPU to jump to a patch address. Figure 7–21 shows the format of the register. Table 7–19 lists the bit descriptions. This register can be loaded only in diagnostic mode (NICSR6 <OM>=<11>).

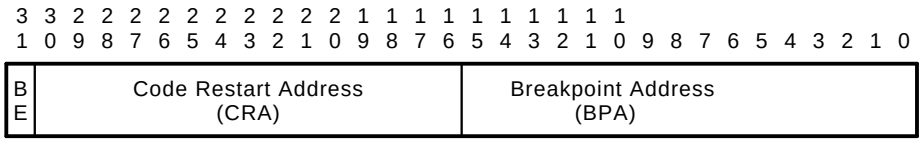


Figure 7–21 NICSR14 Format

Table 7–19 NICSR14 Bits

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

Table 7–19 (Cont.) NICSR14 Bits

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

NOTE
This register works in conjunction with the diagnostic descriptors to allow software patches.

NICSR14 Access

Value after reset	00000000_{16}.
Read access rules	None.
Write access rules	Diagnostic mode.
Violation	Addressing NICSR14 while NICSR5<DN> is deasserted.

7.3.15.2 Monitor Command Register (NICSR15)

This register is a physical CSR. It contains the bits that select the internal test block operation mode. Figure 7–22 shows the format of the register. Table 7–20 lists the bit descriptions.

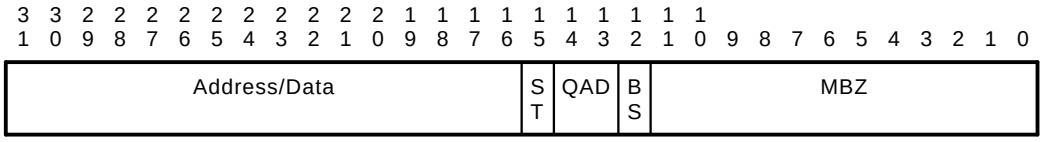


Figure 7–22 NICSR15 Format

Table 7–20 NICSR15 Bits

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

Table 7–20 (Cont.) NICSR15 Bits

Bit	Name	Type	Description
<14:13>	QAD	W	Quad select bits. This field defines the specific four bits of the internal data bus or address bus that are monitored on the external test pins BM_L/TEST<3:0>. This field is meaningful only in test mode (TSM=1). The 2-bit code is interpreted as follows:
<12>	BS	W	Bus select. When reset, the internal data bus is monitored on the external test pins BM_L/TEST<3:0>. When set, the monitoring is applied on the internal address bus. This bit is meaningful only in test mode (TSM=1).
<11:0>	MBZ	-	Must be 0.

QAD	Data	Address
00	<03:00>	<03:00>
01	<07:04>	<07:04>
10	<11:08>	<11:08>
11	<15:12>	0, IOP_WR_L,<13:12>

NICSR15 Access

Value after reset	$00000FFF_{16}$.
Read access rules	None.
Write access rules	Reserved for debugging.
Violation	Setting <ST> with arandom SGEC internal address.

7.3.16 Descriptors and Buffers—Format

The SGEC transfers frame data to and from receive and transmit buffers in host memory. These buffers are pointed to by descriptors that 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 (implicitly or explicitly) forward-linked list of descriptors. The last entry may point back to the first entry, thus creating a ring structure. Explicit chaining descriptors, through setting xDES1<CA> 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 (*x*DES0) of each descriptor. Unused bits in *x*DES0 are written as 0. Unused bits in *x*DES1 to *x*DES3 may be used by the port driver, and the SGEC will never disturb them.

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

NOTES

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

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

7.3.17 Receive Descriptors

Figure 7–23 shows the format of receive descriptors. The following sections describe the words in the descriptor.

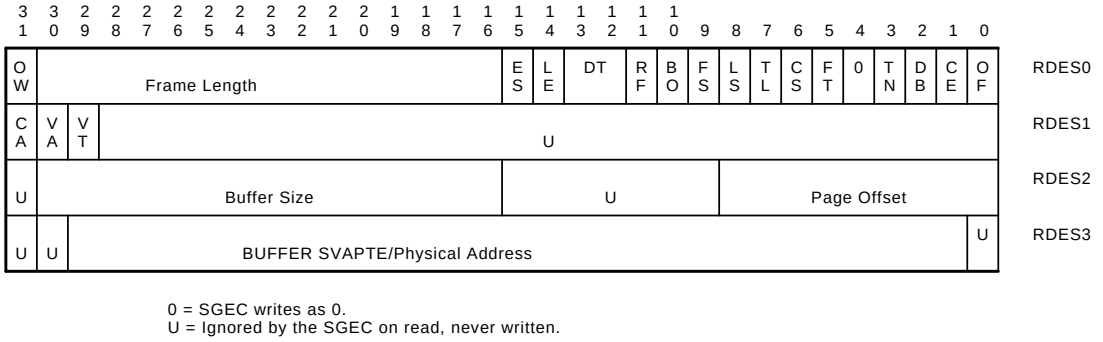


Figure 7–23 Receive Descriptor Format

7.3.17.1 RDES0 Word

The RDES0 word contains the status and length of the received frame. This word also indicates who owns the descriptor—the host or the SGEC. Table 7–21 list the RDES0 bit descriptions.

Table 7–21 RDES0 Bits

Bit	Name	Description
<31>	OW	Owner bit. When set, this bit indicates the descriptor is owned by the SGEC. When cleared, this bit indicates the descriptor is owned by the host. The SGEC clears this bit after completing processing of the descriptor and its associated buffer.
<30:16>	FL	Frame length. This field indicates the length in bytes of the received frame. The field is meaningless if RDES0<LE> is set.
<15>	ES	Error summary. This bit is the logical OR of RDES0 bits OF,CE,TN,CS,TL,LE,RF.

Table 7–21 (Cont.) RDES0 Bits

Bit	Name	Description
<14>	LE	Length error. When set, this bit 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.
<13:12>	DT	Data type. Indicates the type of frame the buffer contains, as follows:

Value	Meaning
00	Serial received frame
01	Internally looped back frame
10	Externally looped back frame, serial received frame.

<11>	RF	Runt frame. When set, this bit indicates the frame was damaged by a collision or premature termination before the collision window had passed. Runt frames are passed to the host only if (NICSR6<PB>) is set. This bit is meaningless if RDES0<OF> is set.
<10>	BO	Buffer overflow. When set, this bit indicates that the frame has been truncated due to a buffer that was too small to fit the frame size. This bit may be set only if data chaining is disabled (NICSR6<DC> = 1).
<09>	FS	First segment. When set, this bit indicates that this buffer contains the first segment of a frame.
<08>	LS	Last segment. When set, this bit indicates that this buffer contains the last segment of a frame and status information is valid.
<07>	TL	Frame too long. When set, this bit indicates the frame length exceeds the maximum Ethernet specified size of 1518 bytes. **NOTE** **The frame too long bit is only a frame length indication and does not cause any frame truncation.**
<06>	CS	Collision seen. When set, this bit indicates the frame was damaged by a collision that occurred after the 64 bytes following the SFD.
<05>	FT	Frame type. When set, this bit indicates the frame is an Ethernet type frame (frame length field is > 1500). When clear, this bit indicates the frame is an IEEE 802.3 type frame. This bit is meaningless for Runt frames < 14 bytes.
<4>	0	

Table 7–21 (Cont.) RDES0 Bits

Bit	Name	Description
<03>	TN	Translation not valid. When set, this bit indicates that a translation error occurred when the SGEC was translating a VAX virtual buffer address. This bit will set only if RDES1<VA> was set. The reception process remains in the running state and tries to acquire the next descriptor.
<02>	DB	Dribbling bits. When set, this bit indicates the frame contained a noninteger multiple of 8 bits. This error is reported only if the number of dribbling bits in the last byte is greater than 2. This bit is meaningless if RDES0<CS> or RDES0<RF> are set. The CRC check is performed independent of this error. However, only whole bytes are run through the CRC logic. **Consequently, received frames with up to 6 dribbling bits will have this bit set. But if <CE> (or another error indicator) is not set, these frames should be considered valid:**
<01>	CE	CRC error. When set, this bit indicates that a CRC error has occurred on the received frame.
<00>	OF	Overflow. When set, this bit indicates that received data in this descriptor's buffer was corrupted due to internal FIFO overflow. This action generally occurs if SGEC DMA requests are not granted before the internal receive FIFO fills up.

CE	DB	Error
0	0	None
0	1	None
1	0	CRC error
1	1	Alignment error

7.3.17.2 RDES1 Word

The RDES1 word contains a chain address and virtual addressing bit that affect the RDES3 word. Table 7–22 lists the RDES1 bit descriptions.

Table 7–22 RDES1 Bits

Bit	Name	Description
<31>	CA	Chain address. When this bit is set, RDES3 is interpreted as another descriptor's VAX physical address. This allows the SGEC to process multiple, noncontiguous descriptor lists and explicitly chain the lists together. Note that contiguous descriptors are implicitly chained. In contrast to what is done for a receive buffer descriptor, the SGEC clears neither the ownership bit RDES0<OW> 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 **owned by the host**, regardless of the ownership bit state.
<30>	VA	Virtual addressing. When this bit is 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 this bit is clear, RDES3 is interpreted as the actual physical address of the buffer:

VA	VT	Addressing mode
0	x	Physical
1	0	Virtual—SVAPTE type
1	1	Virtual—PAPTE type

<29>	VT	Virtual type. If virtual addressing (RDES1<VA> = 1) is used, this bit indicates the type of virtual address translation. When this bit is set, the buffer address RDES3 is interpreted as a system virtual address of the page table entry (SVAPTE). When this bit is clear, the buffer address is interpreted as a physical address of the page table entry (PAPTE). This bit is meaningful only if RDES1<VA> is set.
<28:0>	u	

7.3.17.3 RDES2 Word

This word contains the buffer size of the data buffer, as well as the byte offset of buffer within the page. Table 7–23 lists the RDES2 bit descriptions.

Table 7–23 RDES2 Bits

Bit	Name	Description
<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 buffers size must be an even number of bytes.**

Table 7–23 (Cont.) RDES2 Bits

Bit	Name	Description
<15:9>	U	Unused. Ignored by the SGEC on reads. Never written.
<08:00>	PO	Page offset. The byte offset of the buffer within the page. This field is meaningful only if RDES1<VA> is set. **NOTE** **Receive buffers must be word-aligned.**

7.3.17.4 RDES3 Word

The RDES3 word is interpreted as the address of either the page table entry or the the buffer, depending on the setting of the RDES1 word. Table 7–24 lists the bit descriptions.

Table 7–24 RDES3 Bits

Bit	Name	Description
<31:00>	SV/PV/PA	SVAPTE/PAPTE/Physical address. If RDES1<VA> is set, RDES3 is interpreted as the address of the page table entry and used in the virtual address translation process. The setting of RDES1<VT> determines the type of the address—system virtual address (SVAPTE) or physical address (PAPTE). If 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.**

7.3.17.5 Receive Descriptor Status Validity

Table 7–25 summarizes the validity of the receive descriptor status bits regarding the Reception completion status:

Table 7–25 Receive Descriptor Status Validity

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

V = valid.
M = meaningless.

7.3.18 Transmit Descriptors

Figure 7–24 shows the format of the transmit descriptors. The following sections describe each word in the descriptor.

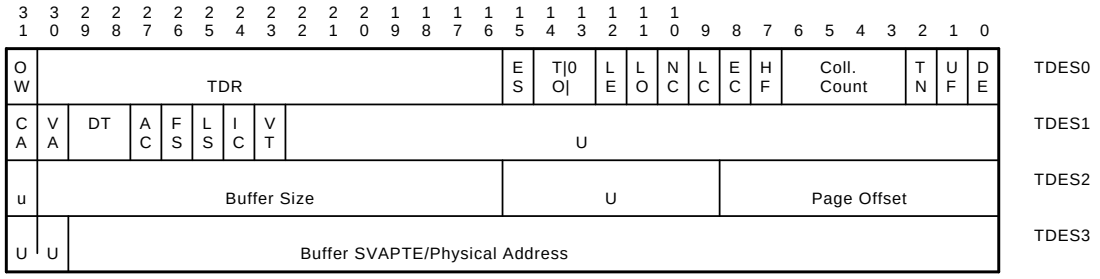


Figure 7–24 Transmit Descriptor Format

7.3.18.1 TDES0 Word

The TDES0 word contains the status of the transmitted frame. TDES0 also indicates who owns the descriptor—the SGEC or the host. Table 7–26 lists the bit descriptions.

Table 7–26 TDES0 Bits

Bit	Name	Description
<31>	OW	Owner bit. When set, this bit indicates the descriptor is owned by the SGEC. When cleared, this bit indicates the descriptor is owned by the host. The SGEC clears this bit upon completing processing of the descriptor and its associated buffer.
<29:16>	TDR	Time domain reflectometer. This field is a count of bit time. The count is useful for locating a fault on the cable, using the velocity of propagation on the cable. This field is valid only if TDES0<EC> is also set. Two excessive collisions in a row and with the same or similar TDR values (within 20) indicate a possible cable open.
<15>	ES	Error summary. This bit is the logical OR of UF, TN, EC, LC, NC, LO, LE and TO.
<14>	TO	Transmit watchdog timeout. If this bit is set, 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	

Table 7–26 (Cont.) TDES0 Bits

Bit	Name	Description
<12>	LE	Length error. When set, this bit 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, this bit indicates a loss of carrier during transmission (possible short circuit in the Ethernet cable). This bit is meaningless in internal loopback mode (NICSR5<OM>=1).
<10>	NC	No carrier. When set, this bit indicates the carrier signal from the transceiver was not present during transmission (possible problem in the transceiver or transceiver cable). This bit is meaningless in internal loopback mode (NICSR5<OM>=1).
<09>	LC	Late collision. When set, this bit indicates frame transmission was aborted due to a late collision. This bit is meaningless if TDES0<UF>.
<08>	EC	Excessive collisions. When set, this bit indicates that the transmission was aborted because 16 successive collisions occurred while attempting to transmit the current frame.
<07>	HF	Heartbeat fail. When set, this bit indicates a heartbeat collision check failure. The transceiver failed to return a collision pulse as a check after the transmission. Some tranceivers do not generate heartbeat, so they always have this bit set. If the transceiver does support heartbeat, this bit indicates a transceiver failure. The bit is meaningless if TDES0<UF>.
<06:03>	CC	Collision count. This is a 4-bit counter, indicating the number of collisions that occurred before the transmission attempt succeeded or failed. This bit is meaningless when TDES0<EC> is also set.
<02>	TN	Translation not valid. When set, this bit indicates that a translation error occurred when the SGEC was translating a VAX virtual buffer address. TN may only set if TDES1<VA> was set. The transmission process enters the suspended state and sets NICSR5<TI>.

Table 7–26 (Cont.) TDES0 Bits

Bit	Name	Description
<01>	UF	Underflow error. When set, this bit indicates that the transmitter has truncated a message due to data being late from memory. UF indicates that the SGEC encountered an empty transmit FIFO while transmitting a frame. The transmission process enters the suspended state and sets NICSR5<TI>.
<00>	DE	Deferred. When set, this bit 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.

7.3.18.2 TDES1 Word

The TDES1 word contains a chain address and virtual addressing bit that affect the TDES3 word. Table 7–27 lists the TDES1 bit descriptions.

Table 7–27 TDES1 Bits

Bit	Name	Description
<31>	CA	Chain address. When this bit is set, TDES3 is interpreted as another descriptor's VAX physical address. This allows the SGEC to process multiple, noncontiguous descriptor lists and explicitly chain the lists. Note that contiguous descriptors are implicitly chained. In contrast to what is done for a receive buffer descriptor, the SGEC does not clear the ownership bit TDES0<OW> or one of the other bits of the TDES0 chain descriptor after processing. To protect against an infinite loop, a chain descriptor that points back to itself is seen as **owned by the host**, regardless of the setting of the ownership bit.
<30>	VA	Virtual addressing. When thsi bit is set, TDES3 is interpreted as a virtual address. The TDES1<VT> bit determines the type of virtual address translation. The SGEC uses TDES3 and TDES2<page offset> to perform a VAX virtual address translation process and obtain the physical address of the buffer. When clear, TDES3 is interpreted as the actual physical address of the buffer.

VA	VT	Addressing Mode
0	x	Physical
1	0	Virtual—SVAPTE type
1	1	Virtual—PAPTE type

Table 7–27 (Cont.) TDES1 Bits

Bit	Name	Description
<29:28>	DT	Data type. This field Indicates the type of data the buffer contains, according to the following table:

Value	Meaning
00	Normal transmit frame data
10	Setup frame (Explained in Section 7.3.19.)
11	Diagnostic frame

<27>	AC	Add CRC disable. When this bit is 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.
		NOTE **If the transmitted frame is shorter than 64 bytes, the SGEC adds the padding field and the CRC regardless of the <AC> flag.**
<26>	FS	First segment. When set, this bit indicates the buffer contains the first segment of a frame.
<25>	LS	Last segment. When set, this bit indicates the buffer contains the last segment of a frame.
<24>	IC	Interrupt on completion. When the bit is 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. If virtual addressing is used (TDES1<VA> = 1), this bit indicates the type of virtual address translation. When this bit is set, the buffer address TDES3 is interpreted as a system virtual address of the page table entry (SVAPTE). When this bit is clear, the buffer address is interpreted as a physical address of the page table entry (PAPTE). This bit is meaningful only if TDES1<VA> is set.
<22:0>	U	

7.3.18.3 TDES2 Word

This word contains the buffer size of the data buffer, as well as the byte offset of buffer within the page. Table 7–28 lists the TDES2 bit descriptions.

Table 7–28 TDES2 Bits

Bit	Name	Description
<31>	U	
<30:16>	BS	Buffer size. The size, in bytes, of the data buffer. If this field is 0, the SGEC ignores this buffer. The frame size is the sum of all BS fields of the frame segments (between and including the descriptors that have TDES1<FS> and TDES1<LS> set.) **NOTE** **If the port driver wants to suppress transmission of a frame, this field must be set to 0 in all descriptors that make up the frame, before the SGEC acquires them. If this rule is not adhered to, corrupted frames may be transmitted.**
<08:00>	PO	Page offset. This field is the byte offset of the buffer within the page. Only meaningful if TDES1<VA> is set. **NOTE** **Transmit buffers may start on arbitrary byte boundaries.**

7.3.18.4 TDES3 word

The TDES3 word is interpreted as the address of either the page table entry or the the buffer, depending on the setting of the TDES1 word. Table 7–29 lists the bit descriptions.

Table 7–29 TDES3 Bits

Bit	Name	Description
<31:00>	SV/PV/PA	SVAPTE/PAPTE/Physical address. If TDES1<VA> is set, TDES3 is interpreted as the address of the page table entry and used in the virtual address translation process. The setting of TDES1<VT> determines the type of the address—system virtual address (SVAPTE) or physical address (PAPTE). If TDES1<VA> is clear, TDES3 is interpreted as the physical address of the buffer. When TDES1<CA> is set, RDES3 is interpreted as the VAX physical address of another descriptor. **NOTE** **Transmit buffers may start on arbitrary byte boundaries.**

7.3.18.5 Transmit Descriptor Status Validity

Table 7–30 summarizes the validity of the transmit descriptor status bits regarding the transmission completion status:

Table 7–30 Transmit Descriptor Status Validity

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

V = valid.
M = meaningless.

7.3.19 Setup Frame

A setup frame defines SGEC Ethernet destination addresses. These addresses are to filter all incoming frames. The setup frame is *never* transmitted over the Ethernet or 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 + one physical address

7.3.19.1 First Setup Frame

A setup frame must be *queued* (placed in the transmit list with SGEC ownership) to the SGEC before the reception process starts. The only exception is when the SGEC operates in promiscuous reception mode.

NOTE

The self-test completes with the SGEC address filtering table fully set to 0. A reception process that starts before a setup frame is loaded will reject all the incoming frames except those with a destination physical address of 000000h .

7.3.19.2 Subsequent Setup Frame

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

The setup frame does not affect the reception process state. However, while the setup frame is being processed, the SGEC is disengaged from the Ethernet wire.

7.3.19.3 Setup Frame Descriptor

Figure 7–25 shows the format of the setup frame descriptor. Table 7–31 lists the bit descriptions. The following sections describe each word of the descriptor.

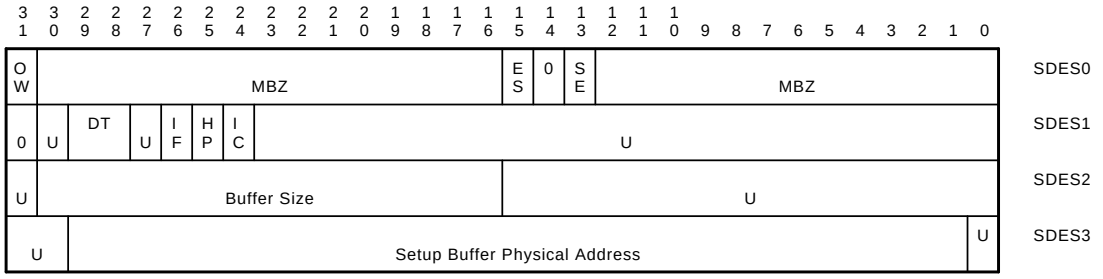


Figure 7–25 Setup Frame Descriptor Format

Table 7–31 Setup Frame Descriptor Bits

Word	Bit	Name	Description
SDES0	<13>	SE	Setup error. When set, this bit indicates the setup frame's buffer size is not 128 bytes.
	<15>	ES	Error summary. This bit is set when SE is set.
	<31>	OW	Owner bit. When set, this bit indicates the descriptor is owned by the SGEC. When cleared, this bit indicates the descriptor is owned by the host. The SGEC clears this bit upon completing processing of the descriptor and its associated buffer.
SDES1	<24>	IC	Interrupt on completion. When this bit is set, the SGEC sets NICSR5<TI> after this setup frame is processed.
	<25>	HP	Hash/Perfect filtering mode. When this bit is 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 this bit is clear, the SGEC does a perfect address filter of incoming frames according to the addresses specified in the setup frame.

Table 7–31 (Cont.) Setup Frame Descriptor Bits

Word	Bit	Name	Description
	<26>	IF	Inverse filtering. When this bit is set, the SGEC does an inverse filtering. That is, the SGEC accepts the incoming frames with a destination address not matching the perfect addresses, and rejects the frames with destination address matching one of the perfect addresses. This bit is 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. The physical address of the setup buffer. **NOTE** **The setup buffer must be word-aligned.**

7.3.19.4 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-bit Ethernet destination addresses. It compares the addresses of any incoming frame to these stored addresses and rejects frames based on the status of Inverse_Filtering flag SDES1<IF>:

- If SDES1<IF> = 0, reject addresses that do not match.
- If SDES1<IF> = 1, reject addresses that do match.

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. Figure 7–26 shows the format for addresses.

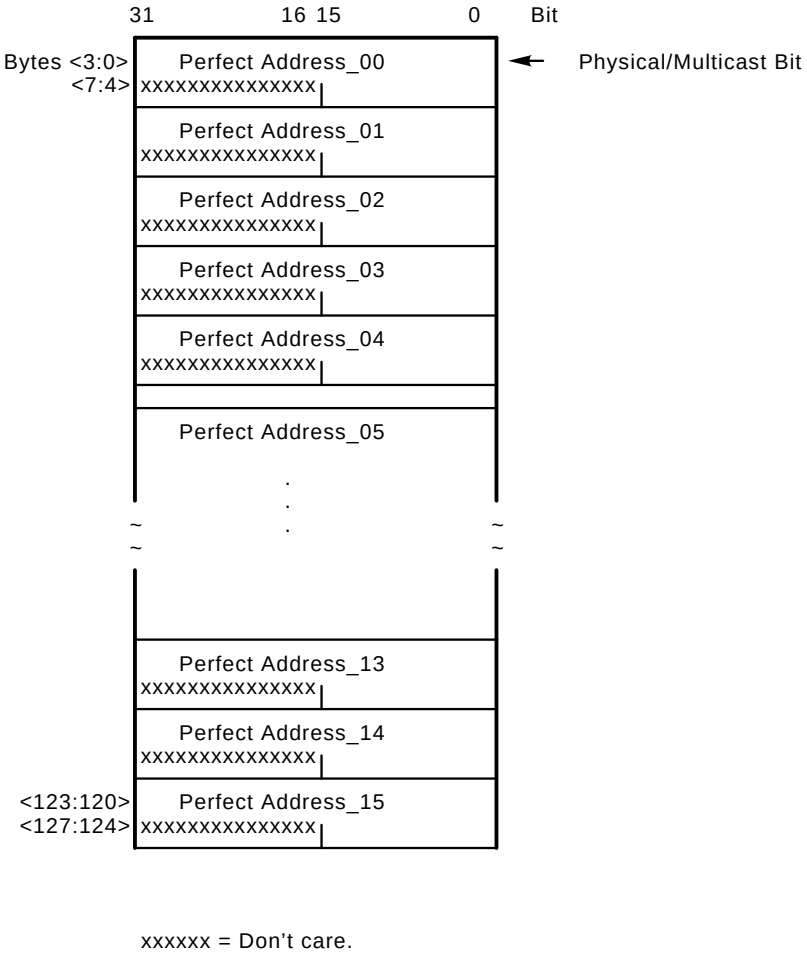


Figure 7–26 Perfect Filtering Setup Frame Buffer Format

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

Example 7–1 shows a fragment of a perfect filtering setup buffer.

```
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 for address display.

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

Example 7–1 Perfect Filtering Buffer

7.3.19.5 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.

Multicast Address For any incoming frame with a multicast destination address, the SGEC applies the standard Ethernet CRC function to the first 6 bytes containing the destination address. Then the SGEC uses the most significant 9 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 the filtering still reduces the number of frames present to the host.

Figure 7–27 shows the format for the hash table and the physical address.

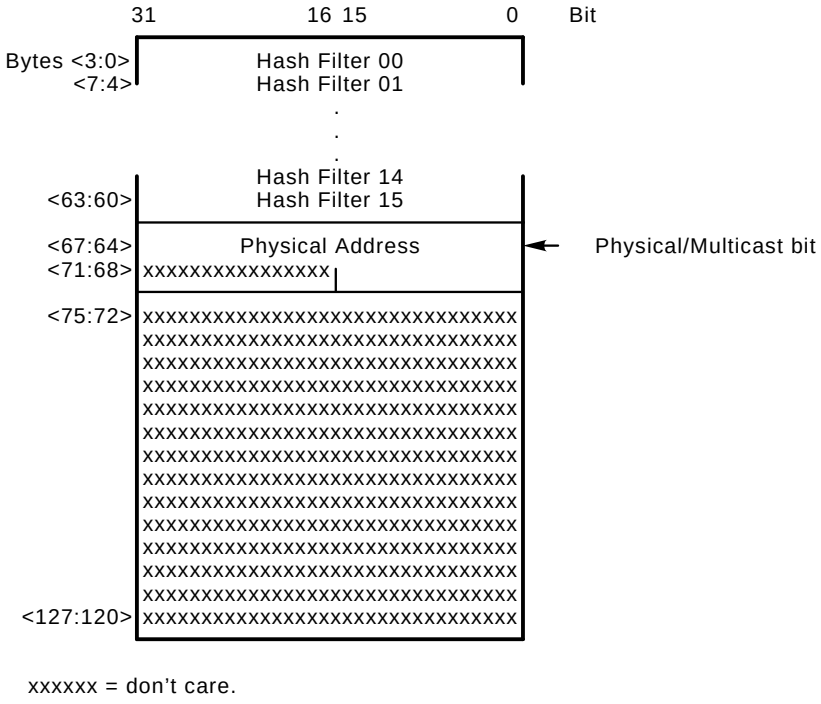


Figure 7–27 Imperfect Filtering Setup Frame Format

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

Example 7–2 shows 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 7–2 (Cont.) Imperfect Filtering Buffer

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

❶ Ethernet Multicast addresses written according to the DEC STD 134 specification 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 7–2 Imperfect Filtering Buffer

Example 7–3 shows a C program to compute the hash bucket heads and create the resulting setup frame buffer.

```
#include <stdio>

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

main()
{
    int i, hash;
/*             */
/* This program accepts 48-bit 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.     */
/* Nonmulticast addresses are ignored.          */
/*              */
/* Input is terminated by typing CTRL/Z. The program then 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 7–3 (Cont.) Creating an Imperfect Filtering Setup Frame Buffer (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],addressY3vMfgyY^"vbw1m{

/* 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 = crc  ^ mean;
  crc[5] = crc[5] ^ mean;
  crc[3] = crc[3] ^ mean;
  crc[2] = crc[2] ^ mean;
  crc[1] = mean;
```

Example 7–3 Creating an Imperfect Filtering Setup Frame Buffer (C Program)

7.3.20 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 0, except where noted.*

NOTE
The SGEC does not explicitly disown any owned descriptor, so the owned bit of descriptor may be left in a state indicating SGEC ownership.

Table 7–32 lists the NICSR fields that are not set to 0 after a reset.

Table 7–32 NICSR Field Values After Reset

Field	Value
NICSR3	Unpredictable
NICSR4	Unpredictable
NICSR5<DN>	1
NICSR6<BL>	1
NICSR6<RE>	Hardware reset: unpredictable
	Software reset: 1
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 of the self-test routine, the test is aborted and the self-test failure NICSR5<SF> bit is set together with the self-test error status NICSR5<SS> bit. SS indicates the reason for the failure.

NOTE
The self-test takes 25 ms 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 (hardware or software) may be issued. The only restriction is that SGEC NICSRs should not be accessed during a 1-microsecond period following the reset. Access during this period will produce a CP bus timeout error. Access to SGEC NICSRs during the self-test are permitted; however, only NICSR5 reads should be performed.

7.3.21 Interrupts

Interrupts are generated as a result of various events. NICSR5 contains all the status bits that can cause an interrupt, provided NICSR6<IE> is set. The port driver must clear the interrupt bits (by writing a 1 to the bit position) to enable further interrupts from the same source.

Interrupts are *not queued*. 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 descriptor.

An interrupt is generated only *once* for simultaneous, multiple interrupting events. It is the port driver's 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> may 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 clears *all* NICSR5 set interrupt bits before the interrupt has been acknowledged, the interrupt is suppressed.

7.3.22 Startup Procedure

The port driver must perform the follwoing sequence of checks and commands in order to prepare the SGEC for operation:

1. Wait for the SGEC to complete its initialization sequence by polling on NICSR5<ID> and NICSR5<SF> (Section 7.3.8).
2. Examine NICSR5<SF> to find out whether the SGEC passed its self-test. If it did not, it should be replaced (Section 7.3.8).
3. Write NICSR0 to establish system configuration dependent parameters (Section 7.3.4).
4. If the port driver intends to use VAX virtual addresses, NICSR7 must be written to identify the system page table to the SGEC (Section 7.3.10).
5. If the port driver wants to change the default settings of the watchdog timers, it must write to NICSR9 (Section 7.3.12).
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 (Section 7.3.7).
7. Write NICSR6 to set global operating parameters and start the transmission and reception processes. Both processes enter the running state, then try to acquire descriptors from the respective descriptor lists and begin processing incoming and outgoing frames (Section 7.3.9). The reception and transmission processes are independent of each other, so they can be started and stopped separately.

   CAUTION
   If address filtering (either perfect or imperfect) is desired, the reception process should start only 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.

7.3.23 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. 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 by setting NICSR6<SR>
- When the SGEC begins writing frame data to a data buffer pointed to by the current descriptor
- At the last acquired descriptor chained (RDES1<CA> = 1) to another descriptor
- When a virtual translation error is encountered RDES0<TN> while the SGEC is 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, the SGEC performs address filtering according to NICSR6 fields AF, HP, and the SGEC's internal filtering table. If the frame fails the address filtering, the frame is ignored and purged from the FIFO. Frames 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 are 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 any 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 (RDES0<OW> bit cleared) as their data buffers fill up or after the last segment of a frame is 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 process sets NICSR5<RU> and enters the suspended state. The position in the receive list is retained.

Any incoming frames received in this state cause the SGEC to fetch the current descriptor in the host memory. If the descriptor is now owned by the SGEC, the reception process 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 7–33 summarizes the reception process state transitions and resulting actions:

Table 7–33 Reception Process State Transitions

From State	Event	To State	Action
Stopped	Start reception.	Running	Receive polling begins from the last list position or from the 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 tries to acquire a descriptor owned by the host.	Suspended	NICSR5<RU> is set when the last acquired descriptor buffer is consumed. The position in the list is retained.
Running	Stop reception.	Stopped	The reception process is stopped after the current frame (if any) is completely transferred to data buffer(s). The position in the list is retained.
Running	A memory or host bus parity error is encountered.	Stopped	Reception is cut off and NICSR5<ME> is set.
Running	Reset.	Stopped	Reception is cut off.
Suspended	Receive poll demand or incoming frame and available descriptor.	Running	Receive polling resumes from the last list position or from the list head (if NICSR3 was modified by the port driver).
Suspended	Stop reception.	Stopped	None.
Suspended	Reset.	Stopped	None.

7.3.24 Transmission Process

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. After completing frame transmission (or giving up), status information is written to the TDES0 words. When polling starts, it continues (in sequential or descriptor-chained order) until the SGEC encounters either 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 that 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.

As the transmission process starts in the running state, it first expects to find a descriptor with TDES1<FS> set. Frame data transfer from the host buffer to the internal FIFO is initiated.

At the same time, if the current frame had TDES1<LS> clear, the transmission process tries to acquire the next descriptor. The process expects either 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 is 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.
- 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.

Suspended State

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> will be 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. In all the cases listed, the position in the transmit list 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 transmit descriptor list. The port driver *must* explicitly issue a transmit poll demand command after rectifying the suspension cause.

Table 7–34 summarizes the transmission process state transitions:

Table 7–34 Transmission Process State Transitions

From State	Event	To State	Action
Stopped	Start transmission.	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	The SGEC tries to acquire a descriptor owned by the host.	Suspended	NICSR5<TI> is set. The position in the list is retained.

Table 7–34 (Cont.) Transmission Process State Transitions

From State	Event	To State	Action
Running	Out-of-order delimiting flag (TDES0<FS> or TDES0<LS>) encountered.	Suspended	TDES0<LE> and NICSR5<TI> are set. The position in the list is retained.
Running	The frame transmission aborts due to a locally induced error.	Suspended	Appropriate TDES0 and NICSR5<TI> bits are set. The position in the list is retained.
Running	Stop transmission.	Stopped	The transmission process is stopped after the current frame, if any, is transmitted. The position in the list is retained.
Running	Transmit watchdog expires.	Stopped	Transmission is cut off and NICSR5<TW> , TDES0<TO> are set. The position in the list is retained.
Running	Memory or host bus parity error encountered.	Stopped	Transmission is cut off, and NICSR5<ME> is set.
Running	Reset.	Stopped	Transmission is cut off.
Suspended	Transmit poll demand.	Running	Transmit polling resumes from last list position or from the list head (if NICSR4 was modified by the port driver).
Suspended	Stop transmission.	Stopped	None.
Suspended	Reset.	Stopped	None.

7.3.25 Loopback Operations

The SGEC supports two loopback modes:

- **Internal mode**

  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. In this mode, the SGEC is disengaged from the Ethernet wire.

- **External loopback**

  This mode is generally used to verify correct operations up to the Ethernet cable. 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 that carries its own transmissions and places incoming frames in the receive list.

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

In either of these modes, all 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. New lists must be identified to the SGEC by appropriately writing NICSR3 and NICSR4.
3. Write to NICSR6<OM> according to the desired loopback mode. Use start commands to place the transmission and reception processes in the running state.
4. Respond and process any SGEC interrupts, as in normal processing.

To restore normal operations, the port driver must execute step 1 above, then write the OM field in NICSR6 with 00.

7.3.26 Support for DNA CSMA/CD Counters and Events

Table 7–35 describes the SGEC features that support the port driver in implementing and reporting the specified counters and events.

Table 7–35 CSMA/CD Counters

Counter	SGEC Feature
Time since counter creation	Supported by the host driver.
Bytes received	The port driver must add up the RDES0<FL> fields of all successfully received frames.
Bytes sent	The port driver must add up the TDES2<BS> fields of all successfully transmitted buffers.
Frames received	The port driver must count the successfully received frames in the receive descriptor list.
Frames sent	The port driver must count the successfully transmitted frames in the transmit descriptor list.
Multicast bytes received	The port driver must add up the RDES0<FL> fields of all successfully received frames with multicast address destinations.
Multicast frames received	The port driver must count the successfully received frames with multicast address destinations.
Frames sent, initially deferred	The port driver must count the successfully transmitted frames with TDES0<DE> set.
Frames sent, single collision	The port driver must count the successfully transmitted frames with TDES0<CC> equal to 1.

Table 7–35 (Cont.) CSMA/CD Counters

Counter	SGEC Feature
Frames sent, multiple collisions	The port driver must count the successfully transmitted frames with TDES0<CC> greater than 1.
Send failure–excessive collisions	The port driver must count the transmit descriptors having TDES0<EC> set.
Send failure–carrier check failed	The 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> are set, indicating a short circuit.
Send failure–open circuit	Two successive transmit descriptors with the excessive collisions flag TDES0<EC> are set with the same time domain reflectometer value TDES0<TDR>, indicating 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	The port driver must count the receive descriptors having RDES0<CE> set with RDES0<DB> cleared.
Receive failure–framing error	The port driver must count the receive descriptors having both RDES0<CE> and RDES0<DB> set.
Receive failure–frame too long	The port driver must count the receive descriptors having RDES0<TL> set.
Unrecognized frame destination	Not applicable.
Data overrun	The port driver must count the receive descriptors having RDES0<OF> set.
System buffer unavailable	Reported in the Missed-frame counter NICSR10<MFC>. (See Table 7–17.)
User buffer unavailable	Not applicable.
Collision detect check failed	The 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>.

7.4 KA670 Mass Storage Interface

The KA670 contains two DSSI bus interfaces that are implemented with the two single host adapter chips (SHACs). These interfaces allow the KA670 to transmit packets of data to, and receive packets of data from, up to 14 other DSSI devices (typically RF-type disk drives and TF-type streaming tape drives). The two DSSI buses are distinct from each other, with each supporting seven devices. The SHACs support CP bus parity protection.

7.4.1 SHAC Overview

The single host adapter chip (SHAC) is a single-chip, VLSI version of Digital's systems communications architecture (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 systems in clusters. DSSI has adopted the same interface, so the same VMS 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, the SHAC provides a solution to the following problems:

- Interfacing a group of mass storage device controllers (MSDCs) to a VAX
- Interfacing several VAX systems to a common group of MSDCs and, if higher level protocols support this option, to one another

Where two or more VAX systems 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 systems communicate at high speed, with a 4 to 5 Mbyte/s burst transfer rate. The SHAC handles the problem of providing effective, efficient, and reliable interfacing between this DSSI bus and the CPU that has direct host memory access (DMA) over the host's 32-bit wide, 16 Mbyte/s CP bus. All communications between those connected to the DSSI will 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.
- Up to four SHACs can be installed on a single host bus.
- Because there must be a host, there can be up to seven MSDCs on a single DSSI.

The SHAC provides a small amount of buffering (1.2 kilobytes) on the chip to improve bus utilization on both sides, but SHAC is designed to pass data through from one bus to the other as rapidly as the two buses 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 reduced intruction set 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 runtime. This capability allows the RAM to read in new code and data from the main memory, so it can adapt its behavior to new circumstances. This feature permits inexpensive upgrades of SHACs after they are installed in the field. It also allow the SHAC to store infrequently accessed code in main memory, providing more capability than could be included in on-chip ROM.

SCA Communications Architecture

The SHAC works under Digital's systems communications architecture (SCA). This architecture defines four layers (Figure 7–28). The architecture can be realized in a variety of ways. Two realizations at the lowest two levels, in the diagram are the computer interconnect (CI) and Digital storage system interconnect (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 can be realized at both the board and chip level, and products at both levels are in design within Digital.

The SHAC is a chip-level product that connects the host bus to the DSSI bus. The SHAC is controlled by the CPU through a CI port driver, accepting and delivering CI-defined packets over the DSSI bus. Layers above the port driver are invisible to SHAC.

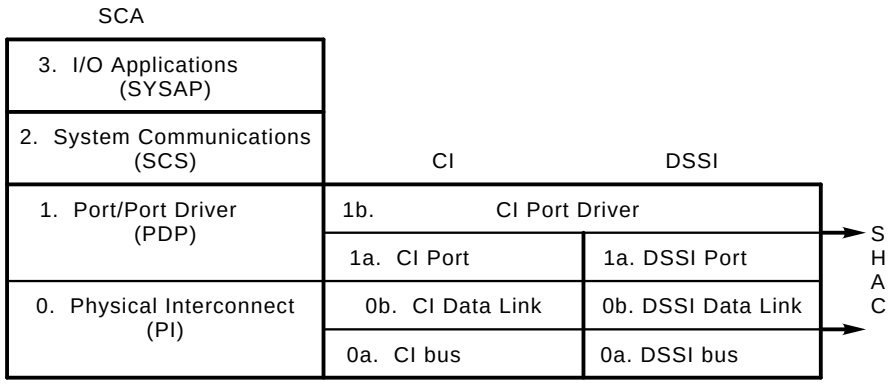


Figure 7–28 Relationship of the DSSI to SCA and CI

The port driver maintains a set of seven queues in its system space. Four of these contain commands for the SHAC to execute. Command priority is determined by the queue a command 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 use by the host and SHAC, to stuff with commands and responses and then queue 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. The header on each block includes a pair of queue pointers (for a doubly linked queue) and various standard identifiers that specify the contents of 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. After a block is removed from a queue, the block is visible only to the entity that 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 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 1).

The following 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.

7.4.2 CI-DSSI Overview

At start-up, the host provides the SHAC with a number of pointers to internal host structures. One of these structures is the port queue block (PQB), which 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, SHAC goes to the highest command queue that has something enqueued. Choices are CMDQ0 to CMDQ3, with 3 being most urgent. SHAC dequeues an entry from the queue and examines the entry's header to see what it must do with the entry. The entry 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. A deleivery item could be a datagram or a message.

A *datagram* is a one-sided communication—one which is sent without any assurance of either receipt or reply. One obvious application for a datagram 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, the host uses a datagram. All datagrams are of a length 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 exists. After 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 ...*). All messages are of a length 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. The SHAC supports retries if a message fails to get through. If the command is not delivered before the retry limit is reached, the SHAC returns the command to the host, marking it as *undeliverable*; then the SHAC breaks the virtual circuit to that node.

Sample Transaction

A full transaction might go something like this:

1. The host queues a message for node 3 (for example, a disk controller) to copy a block of 16 kilobytes 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
   - Making the last previous 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 [*], 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. If successful in obtaining the bus, the SHAC specifies node 3 as the target. These steps are called *arbitration* and *selection*.

[*] 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 the element is.

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 the type of message, then transmits 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 that permits simultaneous, buffered reading and writing on the two buses connected to the SHAC. After the transmission is successfully completed, node 3 responds with a 1-byte acknowledgment of success (parity and checksum proper, and no other errors).

4. The SHAC is still holding the only pointer to the message block in host memory. The SHAC 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 will look for responses.
   - Alternatively, the SHAC simply puts the block back on the MFREEQ that holds the standard envelopes for messages. At this point, the single message has been delivered, and the message envelope is back in circulation.

5. Node 3 processes the message, then contends for the bus. After obtaining the bus, the node selects the SHAC as its target. The node then sends a standard CI message as above, telling the SHAC to transmit the required data.

   In general, the SHAC does not send the data immediately, because it is obliged to handle traffic according to position in the queue and according to queue priority. Instead, the SHAC takes an empty envelope from MFREEQ, writes the message into the envelope, and puts the envelope 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 again and carries out the command, possibly interleaving other transmissions of higher priority to this node or any priority to other nodes, until the last byte is sent. The SHAC uses transmissions of 4 kilobytes whenever possible. A 4-kilobyte transmission takes about 1 ms on the DSSI bus. After 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. The node again contends for the bus and upon obtaining it specifies the SHAC as its target. Then the node 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 the SHAC queues the envelope 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 part 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. The SHAC only fetches an envelope from the free queues when its own supply is gone, and it only returns them to the free queues when it has a full supply (four of a type). By this and other efforts at traffic conservation, the SHAC attempts to optimize its rate of doing useful work.

7.4.3 SHAC Registers

The P-chip communicates directly with the two SHACs through a set of device registers in each of the SHACs. For each SHAC, these registers occupy a 1-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 the SHAC software chip reset (SSWCR) register 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

Conventions Used in This Section

The KA670 has two SHACs, one for the internal DSSI bus and one for the external DSSI. The internal bus is the bus brought out through the backplane connector. The external bus is the bus brought out through the console module.

For simplicity, the following sections provide a single description of each register for both SHACs, with I/O addresses given for each SHAC. In these sections:

- **SHAC1** is the SHAC controlling the internal DSSI bus.

  SHAC1 registers fall in the I/O addresses range of 2000 4000 to 2000 $41FF_{16}$

- **SHAC2** is the SHAC controlling the external DSSI bus.

  SHAC2 registers fall in the range of 2000 4200 to 2000 $43FF_{16}$.

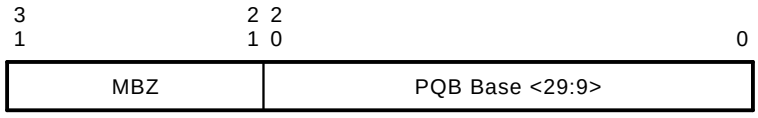
7.4.3.1 CI Port Registers

The following registers are based on the CI port architecture.

7.4.3.1.1 Port Queue Block Base Register (PQBBR)

The port queue block base register (PQBBR) contains the uppermost bits of the physical address for 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 7–29 shows the format of the register. Table 7–36 lists the bit descriptions.

PQBBR is writable only when the port is in the disabled or disabled/maintenance state. The register is readable anytime except during chip intialization.



SHAC1 I/O Address: 2000 4048
SHAC2 I/O Address: 2000 4248
Longword Read/Write Access.

Figure 7–29 Port Queue Block Base Register (PQBBR)

Table 7–36 Port Queue Block Base Address Register (PQBBR) Bits

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

Following a chip reset, PQBBR contains the configuration shown in Figure 7–30. Table 7–37 lists the bit descriptions.

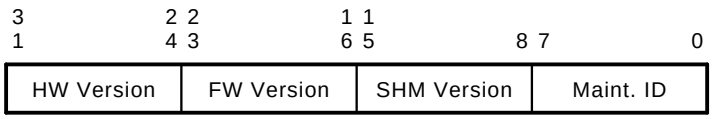


Figure 7–30 Port Queue Block Base Register (PQBBR) After Reset

Table 7–37 Port Queue BLock Base Address Register Bits

Data Bit	Name	Description
<31:24>	HW version	This field contains the SHAC's hardware version, which is greater than 0.
<23:16>	FW version.	This field contains the SHAC's firmware version, which is greater than 0.
<15:8>	SHW Version	This field contains the SHAC's shared host memory version. This value is 0 until the shared host memory data area has been read in; thereafter, the value is greater than 0.
<7:0>	Maint. ID	This field contains the CI port maintenance ID, which should always be 22_{16}.

7.4.3.1.2 Port Status Register (PSR)

The port status register (PSR) contains a status report. If interrupts are enabled (for example, (PMCSR<2>) set), the port interrupts the CPU each time that it writes to this register. After an interrupt is requested by the port, the value of PSR is fixed does not change until the CPU releases it by writing the port status release control register (PSRCR). Figure 7–31 shows the format of the port status register. Table 7–38 lists the bit descriptions.

The PSR register is read-only and may be read anytime by the port driver, except during chip initialization. The value of PSR following a write to it is unpredictable.

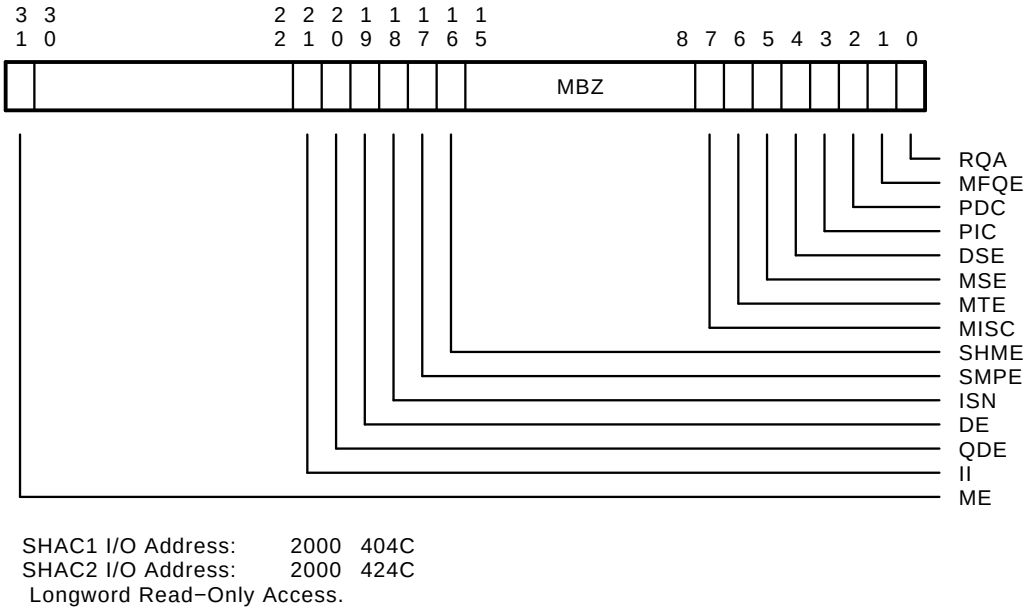


Figure 7–31 Port Status Register (PSR) Bits

Table 7–38 Port Status Register Bits

Data Bit	Name	Description
<31>	ME	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. When this bit is set, the port is in the *uninitialized* state (not functional). Maintenance errors normally indicate a severe SHAC hardware or software failure.
<30:22>	MBZ	Read as 0. 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 (not 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 (not functional).
<19>	DE	Diagnostic error. When this bit is 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 (not functional).
<18>	ISN	Illegal segment number. When set, this bit this indicates a SHAC internal error in which the SHAC attempted to load a nonexistent external segment from the SHAC shared host memory. This causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (not functional).

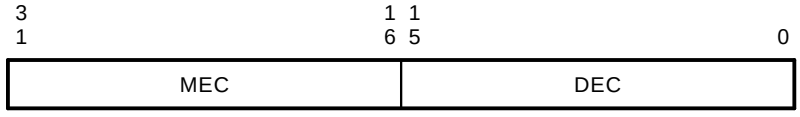
Table 7–38 (Cont.) Port Status Register Bits

Data Bit	Name	Description
<17>	SMPE	Slave mode parity error. This bit is set by the occurrence of a parity error during a CPU access of a SHAC device register. This causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (not functional).
<16>	SHME	Share host memory error. This bit is set by the occurrence of an error involving the SHAC shared host memory. This causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (not functional).
<15:8>	MBZ	Read as 0. Writes have no 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 this bit is set, the maintenance timer has expired. The port is in the *uninitialized*/*maintenance* state.
<5>	MSE	Memory system error. When this bit is set, the port has encountered an uncorrectable data or nonexistent memory error in referencing memory. The port is in the *disabled* or *disabled*/*maintenance* state. See Section 7.4.3.1.4 for more information.
<4>	DSE	Data structure error. When this bit is set, the port has encountered an error in a port data structure (for example, queue entry, PQB, BDT, or page table). The port is in the *disabled* or *disabled*/*maintenance* state. See Sections 7.4.3.1.3 and 7.4.3.1.4for more information. Note that errors in queue structures leave the queues locked.
<3>	PIC	port initializtion complete. When this bit is set, the port has completed internal initialization. The port is in the *disabled* or *disabled*/*maintenance* state.
<2>	PDC	Port disable complete. When this bit is set, the port is in the *disabled* or *disabled*/*maintenance* state.
<1>	MFQE	Message free queue is empty. When set, this bit indicates the port tried to remove an entry from the Message Free Queue (MFREEQ) and found the queue empty. The port can continue to process commands, so the MFREEQ may not be empty at the time the port driver gets control.
<0>	RQA	Response queue available. When set, this bit indicates the port has inserted an entry on an empty response queue.

7.4.3.1.3 Port Error Status Register (PESR)

The port error status register (PESR) indicates the type of error that caused a port status register error of DSE (PSR<4>) or an MISC (PSR<7>) error. Figure 7–32 shows the format of the PESR register. Table 7–39 lists the bit descriptions.

PESR is read only by the CPU. The register is valid only after a DSE or MISC error, or after certain ME (PSR<31>) and DE (PSR<19>) errors. The register's value at any other time, including after a write to it, is unpredictable.



SHAC1 I/O Address: 2000 4050
SHAC2 I/O Address: 2000 4250
Longword Read–Only Access

Figure 7–32 Port Error Status Register (PESR) Bits

Table 7–39 Port Error Status Register (PESR) Bits

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.

7.4.3.1.4 Port Failing Address Register (PFAR)

The port failing address register (PFAR) contains the memory address where one of the following failures occurred: a DSE, MSE, ME, or DE error (as indicated by PSR), or after a response with buffer memory system error status. The address may be

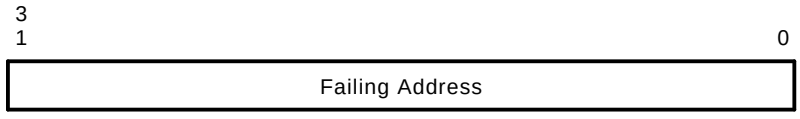
- The exact failing address
- An address in the same page as the exact failing address
- An address in some part of the data structure (for a DSE error)

For a DSE error, PFAR contains a virtual address or offset. For MSE interrupts and buffer memory system errors, the PFAR contains a physical address. For ME errors, 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.

Figure 7–33 shows the format for the port failing address register.

PFAR is read only by the CPU. The register is readable after a DSE, MSE, ME, or DE error; or after a response with buffer memory system error status. At any other time, including after a write to the register, the value is unpredictable.



SHAC1 I/O Address: 2000 4054
SHAC2 I/O Address: 2000 4254
Longword Read–Only Access

Figure 7–33 Port Failing Address Register (PFAR)

7.4.3.1.5 Port Parameter Register (PPR)

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. The PPR value at any other time, including after a to it, is unpredictable. PPR is read only by the CPU. Figure 7–34 shows the format of the register. Table 7–40 lists the bit descriptions.

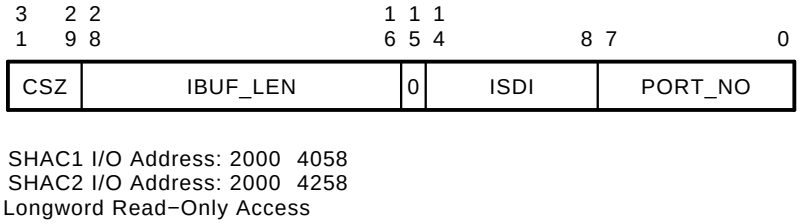


Figure 7–34 Port Parameter Register (PPR)

Table 7–40 Port Parameter Register (PPR) Bits

Data Bit	Name	Description
<31:29>	CSZ	Cluster size. For SHAC, this value is always 0, indicating a maximum of 16 ports on the DSSI bus. Note that the DSSI architecture only allows 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 0, 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 0.
<7:0>	Port_NO	Port number. This is the same as the SHAC's DSSI ID.

7.4.3.1.6 Port Control Registers

The port control registers are 32-bit registers which are write-only by the CPU. To invoke the function provided by any of the control registers, the CPU writes a 1 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. Figure 7–35 shows the format of the port control registers.

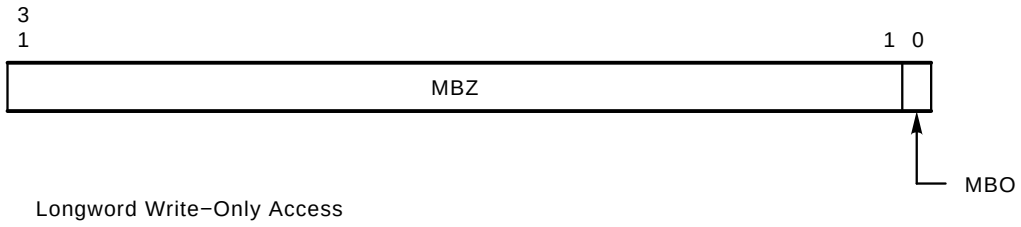


Figure 7–35 Port Control Registers

7.4.3.1.6.1 Port Command Queue 0 Control Register (PCQ0CR)

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.

SHAC1 I/O Address: 2000 4080_{16}
SHAC2 I/O Address: 2000 4280_{16}

7.4.3.1.6.2 Port Command Queue 1 Control Register (PCQ1CR)

This register is the same as the PCQ0CR register, but refers to CMDQ1.

SHAC1 I/O Address: 2000 4084_{16}
SHAC2 I/O Address: is 2000 4284_{16}

7.4.3.1.6.3 Port Command Queue 2 Control Register (PCQ2CR)

This register is the same as PCQ0CR, but refers to CMDQ2.

SHAC1 I/O Address: 2000 4088_{16}
SHAC2 I/O Address: 2000 4288_{16}

7.4.3.1.6.4 Port Command Queue 3 Control Register (PCQ3CR)

This register is the same as PCQ0CR, but refers to CMDQ3.

SHAC1 I/O Address: is 2000 $408C_{16}$
SHAC2 I/O Address: is 2000 $428C_{16}$

7.4.3.1.6.5 Port Datagram Free Queue Control Register (PDFQCR)

If the port driver inserts an entry on the DFREEQ when it is empty, the port driver writes the PDFQCR register 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.

SHAC1 I/O Address: is 2000 4090_{16}
SHAC2 I/O Address: is 2000 4290_{16}

7.4.3.1.6.6 Port Message Free Queue Control Register (PMFQCR)

This register is the same as PDFQCR, but refers to MFREEQ.

SHAC1 I/O Address: 2000 4094_{16}
SHAC2 I/O Address: 2000 4294_{16}

7.4.3.1.6.7 Port Status Release Control Register (PSRCR)

After the port driver has received an interrupt and read the PSR register, it returns the PSR to the port by writing the port status release control register (PSRCR).

SHAC1 I/O Address: 2000 4098_{16}
SHAC2 I/O Address: 2000 4298_{16}

7.4.3.1.6.8 Port Enable Control Register (PECR)

The port driver enables the port by writing the port enable control register (PECR). PECR is ignored if the port is in the *uninitialized* , *uninitialized/maintenance* , *enabled* , or *enabled/maintenance* state.

SHAC1 I/O Address: 2000 $409C_{16}$
SHAC2 I/O Address: 2000 $429C_{16}$

7.4.3.1.6.9 Port Disable Control Register (PDCR)

The port driver disables the port by writing the poprt disable control register (PDCR). When disabled, the port sets PDC (PSR <2>) and requests an interrupt, if interrupts are enabled. PDCR is ignored if the port is in the *uninitialized*, *uninitialized*/*maintenance*, *disabled*, or *disabled*/*maintenance* state.

SHAC1 I/O Address: 2000 $40A0_{16}$
SHAC2 I/O Address: 2000 $42A0_{16}$

7.4.3.1.6.10 Port Initialize Control Register (PICR)

The port driver initializes the port by writing the port initialize control register (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.

SHAC1 I/O Address: 2000 $40A4_{16}$
SHAC2 I/O Address: 2000 $42A4_{16}$

7.4.3.1.6.11 Port Maintenance Timer Control Register (PMTCR)

The port driver forces the maintenance timer to reset its expiration time by writing the port maintenance timer control register (PMTCR). If the PMTCR is not written again before the expiration time, the port enters the *uninitialized*/*maintenance* state, setting MTE (PSR <6>) and requesting an interrupt if interrupts are enabled. PMTCR is ignored if the maintenance timer is not running.

SHAC1 I/O Address: 2000 $40A8_{16}$
SHAC2 I/O Address: 2000 $42A8_{16}$

7.4.3.1.6.12 Port Maintenance Timer Expiration Control Register (PMTECR)

The port driver forces a maintenance timer expiration interrupt by writing the port maintenance timer expiration control register (PMTECR). This register may be written only while the maintenance timer is enabled and the port is in the *enabled*, *enabled*/*maintenance*, *disabled*, or *disabled*/*maintenance* state.

SHAC1 I/O Address: is 2000 $40AC_{16}$
SHAC2 I/O Address: is 2000 $42AC_{16}$

7.4.3.1.7 Port Maintenance Control and Status Register (PMCSR)

The port maintenance control and status register (PMCSR) is for maintenance-level control and status reporting. The CI port architecture defines all but the two least significant bits. Figure 7–36 shows the format of the PMCSR register. Table 7–41 lists the bit descriptions.

The bits can be divided into two categories:

- Status bits—The port sets these bits 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—These bits are read and written by the port driver only. They are cleared by a reset.

  There are two types of function control bits:

  – Init—This type of bit invokes a function (for example, initialization) by setting it. The bit always reads as 0, except while the function is active.

– 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.

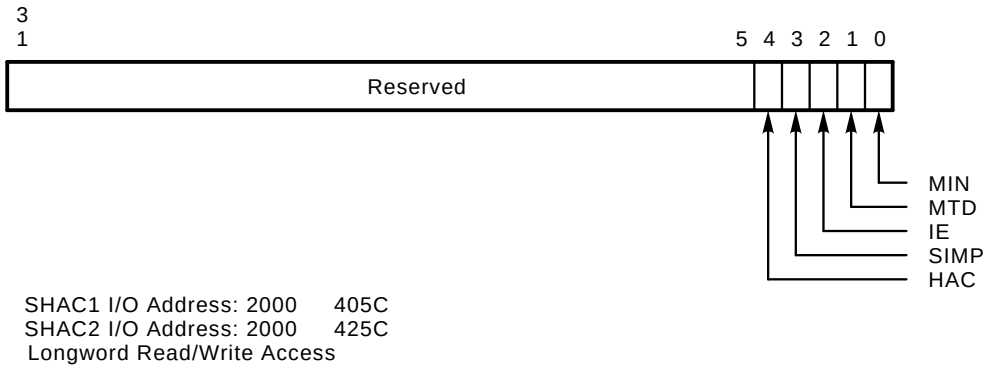


Figure 7–36 Port Maintenance Control and Status Register (PMCSR)

Table 7–41 Port Maintenance Control and Status Register (PMCSR) Bits

Data Bit	Name	Description
<31:5>	Reserved	These reserved bits should not be written. Reads return unpredictable results.
<4>	HAC	Host access feature. This bit must be 0, except for diagnostic purposes. This is an enable/disable class control bit.
<3>	SIMP	Simple SHAC mode. This bit must be 0, except for diagnostic purposes. This is an enable/disable class control bit.
<2>	IE	Interrupt enable. When this bit is set, interrupts from the port to the CPU are enabled. The power-up state is cleared (interrupts disabled). This is an enable/disable class control bit.
<1>	MTD	Maintenance timer disable. This bit is read and written by CPU. If the bit is set, the maintenance timer is turned off. The timer is set to the initial value and suspended. If the bit is clear, the timer works normally. The power-up state is cleared (timer enabled). This is an enable/disable class control bit.
<0>	MIN	Maintenance init. Writing a 1 to this bit resets the port. Upon completion, the port is in the *uninitialized* state and the MIN bit is clear. Writing a 0 to this bit has no effect. It always reads as 0, except while the reset function is active. Although MIN resets the port, this action is is not equivalent to a write to the SHAC software chip reset register. In particular, MIN does not reset the SHAC shared host memory address.

7.4.3.2 SHAC-Specific Registers

The following registers are used for additional maintenance level control. They are not defined in the CI port architecture.

7.4.3.2.1 SHAC Software Chip Reset (SSWCR)

When the CPU writes FFFF FFFF$_{16}$ to the SHAC software chip reset (SSWCR) register, a chip reset is performed. The result is equivalent to that of the hardware chip reset following system power-up. On completion, all device registers are reset to their power-up state, and the port is in the *uninitialized* state. Figure 7–37 shows the format of the SSWCR register.

SSWCR is write only by the CPU and may be written to at any time. The register's value when read is unpredictable. If anything other than FFFF FFFF$_{16}$ is written to SSWCR, the result is undefined.

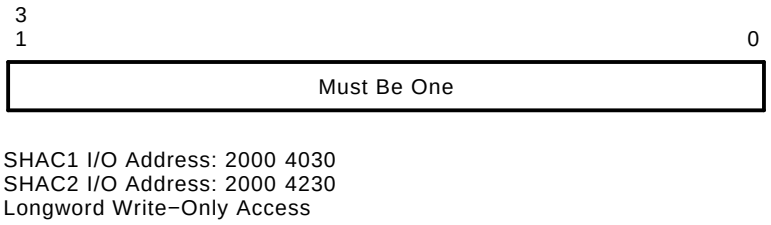


Figure 7–37 SHAC Software Chip Reset (SSWCR)

7.4.3.2.2 SHAC Shared Host Memory Address (SSHMA)

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

SSHMA is read and written by the 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.

Figure 7–38 shows the format for the SHAC shared host memory address.

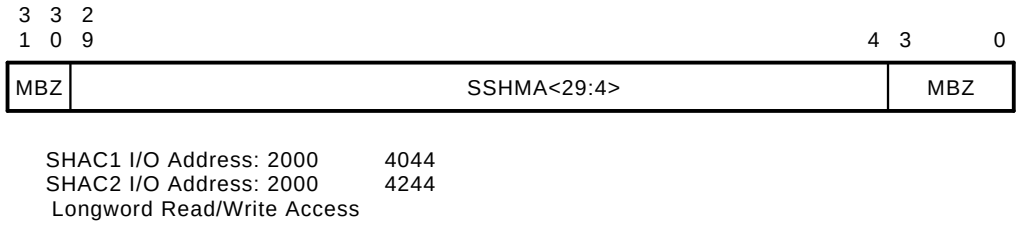


Figure 7–38 SHAC Shared Host Memory Address (SSHMA)

8
KA670 Error Handling

This chapter describes unexpected KA670 system error exceptions and interrupts, as seen from the macrocoder's point of view. The chapter is organized with respect to the system control block (SCB) entry points—vectors pointing to service routines. All error notifications pass thorugh these entry points.

The chapter describes several primary SCB entry points in detail, in order to explain KA670-specific information. This information can help the operating system interface macrocode programmer determine exact errors, console HALT codes, or interrupt/exceptions.

Table 8–1 lists the CPU's internally generated SCB entry points and highlights the specific points covered in this chapter. The chapter also offers recommendations from the module and chip designers for error recovery strategies. Section 3.1.6 describes exceptions and interrupts that are a result of normal system operation.

The chapter provides information on:

- How to discern what error(s) happened, given the SCB point through which the error was dispatched
- What parameters are pushed on the stack
- What the failure codes are for halt and machine check
- What information exists for each error
- How to clean up the error after determining its cause
- How to restore the state of the machine, and what level of recovery is possible

Table 8–1 CPU Internally Generated SCB Entry Points

Mnemonic	SCB Index	Description
SCB_MACHCHK *	004_{16}	Machine check
SCB_KSNV *	008	Kernel stack not valid
SCB_PWRFL *	00C	Power fail
SCB_RESPRIV	010	Reserved/privileged instruction
SCB_XFC	014	XFC instruction
SCB_RESOP	018	Reserved operand
SCB_RESADD	01C	Reserved addressing mode
SCB_ACV	020	Access control violation
SCB_TNV	024	Translation not valid
SCB_TP	028	Trace pending
SCB_BPT	02C	Breakpoint trace fault
SCB_ARITH	034	Arithmetic fault
SCB_CHMK	040	Change mode to kernel
SCB_CHME	044	Change mode to executive
SCB_CHMS	048	Change mode to supervisor
SCB_CHMU	04C	Change mode to user
SCB_SMERR *	054	Soft error interrupt
SCB_HMERR *	060	Hard error interrupt
SCB_IPLSOFT	080-0BC	Software interrupt levels
SCB_INTTIM	0C0	Interval timer interrupt
SCB_EMULATE	0C8	Emulated instruction trap (PSL<FPD>=0)
SCB_EMULFPD	0CC	Emulated instruction fault (PSL<FPD>=1)

*This entry-point vector is described in detail in this chapter.

8.1 Error Handling—SCB Entry Points

This section provides an overview of the entry points for all levels of hardware-detected errors, in the order of their severity. Following sections provide details on each error type.

- **Console error halt**—A halt to console mode is caused by one of several errors such as interrupt stack not valid (Table 8–2). For certain halt conditions, the console prompts for a command and waits for operator input. For other halt conditions, the console may try to restart or bootstrap the system, as defined by the *VAX Architecture Manual*.
- **Machine check**—A hardware error occurred synchronously with the CPU execution of instructions. Instruction-level recovery and retry may be possible.
- **Power fail**—The power supply deasserted the power OK module signal. Software has 20 milliseconds to save processor state.
- **Hard error interrupt** —A hardware error occurred asynchronously with the CPU execution of instructions. Usually, this means data was lost or the state was corrupted, so instruction-level recovery is not possible.
- **Soft error interrupt**—A hardware error occurred that was not fatal to the process or system. System error software should be able to recover and continue.

- **I/O device interrupt**—An error occurred while an I/O device was performing DMA to or from main memory. There are other causes for these interrupts. Therefore, an I/O device interrupt does not necessarily mean that a hardware error occurred. System error software should be able to recover and continue.
- **Kernel stack not valid**—During exception processing, a memory management exception was encountered while trying to push information on the kernel stack.

8.1.1 Error Categories for SCB Entry Points

Table 8–2 lists the various categories of errors, organized by SCB entry point. The section also describes the basic steps in error handlings and recovery. Separate sections provide details on how to distinguish errors within each category.

Table 8–2 Error Summary Based on SCB Entry Points

SCB Index	Entry Point	Error Categories
–	Console halt	Interrupt stack not valid Kernel-mode halt Double errors Illegal SCB vector
04	Machine check	Floating point processor-related errors Memory management errors Microcode/CPU errors Primary cache read errors • Tag parity errors (D-stream only) • parity errors (D-stream only) Backup cache read errors • Data parity errors Main memory read errors • RDAL data parity errors (on nonmasked bytes) • Uncorrectable memory errors (D-stream only) • Main memory NXM I/O read errors • CP bus data parity errors • CP bus NXM/timeouts (D-stream only) • Q22-bus NXM/NOSACK errors • Q22-bus NOGRANT errors • Q22-bus device parity errors
0C	Power fail	

Table 8–2 (Cont.) Error Summary Based on SCB Entry Points

SCB Index	Entry Point	Error Categories
54	Soft error interrupt	Primary cache read errors • Tag parity errors (I-stream only) • Data parity errors (I-stream only) Primary cache write errors • Tag parity errors Backup cache read errors • Tag parity errors Main memory read errors • RDAL data parity errors (on masked bytes or I-stream) • Correctable main memory errors • Uncorrectable main memory errors (I-stream only) Main memory write errors • Correctable main memory errors I/O write errors • CP bus NXM/timeouts (I-stream only)
60	Hard error interrupt	Main memory write errors • RDAL data parity errors • Main memory NXM • Uncorrectable main memory errors (masked writes only) I/O write errors • CP bus NXM/timeouts • Q22-bus NXM/NOSACK errors • Q22-bus NOGRANT errors
08	Kernel stack not valid	

8.1.2 Macrocode Error Handling and Recovery

This section covers the basic steps in error handling and recovery. All errors (except those leading to a console halt) go through SCB vector entry points and are handled by service routines provided by the operating system. A console halt transfers macrocode execution control directly to the console firmware code.

Error handling and recovery can be divided into the following steps:

- State collection

- Analysis
- Recovery
- Retry

8.1.2.1 Error State Collection

Before error analysis can begin, all relevant states must be collected. The stack frame provides the program counter/program status longword (PC/PSL) pair for all exceptions and interrupts. For machine checks, the stack frame also provides details about the error.

In addition to the stack frame, machine checks and hard and soft error interrupts usually require analysis of other registers. In these cases, it is strongly suggested that all of the following states be read and saved:

PCSTS	Primary cache status register
PCERR	Primary cache error address register
BCSTS	C-chip status register
BCCTL	C-chip control register
BCERR	C-chip error address register
MEMCSR32	G-chip system error status register
MEMCSR33	G-chip memory error address register
MEMCSR34	G-chip I
MEMCSR35	G-chip CP bus error address register
DSER	CQBIC DMA system error register
QBEAR	CQBIC Q22-bus error address register
DEAR	CQBIC DMA error address register
SGEC CSR5	SGEC status register
PSR	SHAC(s) port status register
PESR	SHAC(s) error status register
PFARS	SHAC(s) port falling address register
SSCBTR	SSC bus timeout register

For the purposes of the following discussion, assume that each of these registers is saved in a variable whose name is constructed by prefixing “s_” to the register name. For example, the BCERR register would be saved in the variable s_bcerr.

8.1.2.2 Error Analysis

After obtaining the error state in the collection process, the error condition can be analyzed. Analysis of machine checks and hard and soft error interrupts should be guided by the parse trees shown in the appropriate sections.

NOTE

If an errors is detected in or by one of the two caches, the cache is usually disabled automatically. However, to minimize the possibility of nested errors, it is suggested that error analysis and recovery for memory or cache-related errors be performed with both caches disabled. To maintain cache coherency, the primary cache must be disabled before the primary tag store copy in the C-chip.

In some cases, a single error is reported in two ways. For example, primary cache tag parity errors are reported as soft error interrupts and also as machine checks for D-stream read hits. Software must be prepared to handle error interrupts for which there is no apparent cause. In the primary cache tag parity error example, the machine check handler error recovery phase will clean up the error condition, so the error interrupt handler will not find any error bits set.

8.1.2.3 Error Recovery

Recovering from errors consists of clearing any latched error state and restoring the system to normal operation. There are special considerations involved in recovering from cache or memory errors, discussed in the next section.

In some instances, it may be desirable to stop using hardware that is the source of a large number of errors. For example, if a cache reports a large number of errors, it may be better to disable the cache. A suggestion is to have software maintain error counts, which should be compared against error thresholds on every error report. If the count (per unit time) exceeds the threshold, disable the hardware.

8.1.2.4 Special Considerations for Cache and Memory Errors

Cache and memory error recovery requires special consideration:

- Cache and memory error recovery should always be done with both caches disabled:

  ```
  PCSTS := ENABLE_REFRESH+^ENABLE_PTS+^FORCE_HIT;
  BCCTL := ENABLE_REFRESH+^ENABLE_PTS+^ENABLE_BTS+
           ^FORCE_BHIT;
  ```

  To maintain cache coherency, the primary cache must always be disabled before the primary tag store copy in the C-chip. The refresh enable bit should always remain set.

- The error recovery process should start with the most distant component and work toward the CPU. In the KA670 system, SHAC, SGEC, and CQBIC errors should be processed first, followed by SSC errors, C-chip errors, and, finally, primary cache errors.

- G-chip errors are cleared by writing the write-one-to-clear bits in CSR32 to 35. The suggested way to do this is to write the values saved during error state collection back to the registers.

- SSC errors are cleared by writing the write-one-to-clear bits in the SSCBTR register. The suggested way to do this is to write the value saved during error state collection back to the register.

- C-chip backup tag store parity errors are recovered by rewriting the tag using the error address register:

  ```
  IF s_bcsts<BTS_PERR> THEN
    BEGIN
    BCIDX := s_bcerr;
    BCBTS := %x20000000; /* Good Parity, Not Valid */
    END;
  ```

  C-chip primary tag store parity errors are recovered by rewriting the tag, using the error address register:

```
IF s_bcsts<P1TS_PERR> THEN
  BEGIN
  BCIDX := s_bcerr;
  BCP1TS := %x20000000;  /* Good Parity, Not Valid */
  END;

IF s_bcsts<P2TS_PERR> THEN
  BEGIN
  BCIDX := s_bcerr;
  BCP2TS := %x20000000;  /* Good Parity, Not Valid */
  END;
```

C-chip errors are cleared by writing the write-one-to-clear bits in the BCSTS register. The suggested way to do this is to write the value saved during error state collection back to the register.

- Primary cache tag parity errors are recovered by rewriting all tags:

```
IF s_pcsts<TAG_PARITY_ERROR> THEN
  FOR i := 0 to 255 DO
    BEGIN
    PCIDX := i * 8;
    PCTAG := %x40000000;   /* Good Parity, Not Valid */
    END;
```

Primary cache errors are cleared as part of the process of reenabling the cache, described in the following paragraphs.

- The primary cache and primary tag store copy in the C-chip must always be in the same state. If the primary cache is disabled, the C-chip primary tag store should also be disabled. Conversely, if the C-chip primary tag store is disabled, the primary cache should also be disabled.

  To bring both caches back to normal operation, the following sequence is required:

```
BCFBTS := 0;
BCFPTS := 0;
BCCTL := ENABLE_REFRESH+ENABLE_PTS+ENABLE_BTS;
PCSTS := s_pcsts OR
                (ENABLE_REFRESH+ENABLE_PTS+FLUSH_CACHE)
                 AND NOT FORCE_HIT;
```

  Note that either cache may be disabled by clearing the appropriate enable bits in BCCTL and PCSTS while performing the sequence above. In any case, BCCTL<ENABLE_PTS> and PCSTS<ENABLE_PTS> must always be in the same state.

If one or both caches are disabled, the system operates at reduced efficiency:

Configuration	Efficiency
Both caches on	100%
Primary cache off	70%
Backup cache off	50%
Both caches off	12%

8.1.2.5 Error Retry

Error retries are a function of the error type (machine check or error interrupt) and the error state. The individual sections in this chapter specify the conditions under which the instruction stream may be restarted for different errors.

Before attempting a retry, the stack must be trimmed of all parameters except the PC/PSL pair. This is necessary only for machine checks, because error interrupts do not provide any additional parameters on the stack. An REI then restarts the instruction stream and retries the error. Some form of software loop control should be provided to limit the possibility of an error loop.

If a retry is not attempted, software must determine if the error was fatal to the current process, the processor, or the entire system, and take the appropriate action.

8.2 Console Halt and Halt Interrupt

A console halt is not an exception, but a transfer of control by the CPU microcode directly into the boot ROM's console macrocode, at address 2004 0000_{16}. Console halts are initiated at power-up by certain microcode-detected double-error conditions, and by the assertion of a halt signal. Table 8–3 lists the codes and their meanings.

A halt interrupt is generated when the SSC asserts the CPU's HALT_L pin due to one of the following actions:

- Pressing $\overline{\text{Break}}$ on an unsecured console terminal
- Asserting the SSC's HALT_IN signal

There is no exception stack frame associated with a console halt. Instead, SAVPC and SAVPSL provide the necessary information (including the halt code). See Section 3.1.6 for the formats of SAVPC and SAVPSL.

Table 8–3 Console Halt Codes

Code (Hex)	Mnemonic	Meaning
02	ERR_HLTPIN	HALT_L asserted (break, or external halt).
03	ERR_PWRUP	Initial power-up.
04	ERR_INTSTK	Interrupt stack not valid during exception processing.
05	ERR_DOUBLE	Machine check during exception processing.
06	ERR_HLTINS	HALT instruction executed in kernel mode.
07	ERR_ILLVEC	SCB vector bits <1:0> = 11.
08	ERR_WCSVEC	SCB vector bits <1:0> = 10.
0A	ERR_CHMFI	CHMx instruction executed while on the interrupt stack.
10	ERR_MCHK_ACV_TNV	ACV/TNV during machine check processing.
11	ERR_KSNV_ACV_TNV	ACV/TNV during kernel-stack-not-valid processing.
12	ERR_MCHK_MCHK	Machine check during machine check processing.
13	ERR_KSNV_MCHK	Machine check during kernel-stack-not-valid processing.
19	ERR_IE_PSL26_24_101	PSL<26:24> = 101 during interrupt or exception.
1A	ERR_IE_PSL26_24_110	PSL<26:24> = 110 during interrupt or exception.

Table 8–3 (Cont.) Console Halt Codes

Code (Hex)	Mnemonic	Meaning
1B	ERR_IE_PSL26_24_111	PSL<26:24> = 111 during interrupt or exception.
1D	ERR_REI_PSL26_24_101	PSL<26:24> = 101 during REI.
1E	ERR_REI_PSL26_24_110	PSL<26:24> = 110 during REI.
1F	ERR_REI_PSL26_24_111	PSL<26:24> = 111 during REI.
3F	ERR_SELFTEST_ FAILED	(Microcoded) power-up self-test failed in the CPU.

NOTE
The halt code value is packed into the SAVPSL longword (bits <13:8>) before passing control to the boot ROM console macrocode.

8.3 Machine Check Exception

The machine check exception indicates a serious system error. Under certain circumstances, the error may be recoverable by restarting the instruction. The ability to recover depends on the machine check code, the VAX restart bit (R) in the machine check stack frame, the state of PSL's first part done bit <FPD>, and the state of the double-error bit (PCSTS<trap2>).

A machine check results from an internally detected consistency error. For example, the microcode reaches an impossible state or an externally detected hardware error such as a memory parity error occurs.

A machine check is technically a macro instruction ABORT. The CPU microcode tries to convert the condition to a FAULT by unwinding the current instruction, but there is no guarantee that the instruction can be properly restarted. As much diagnostic information as possible is pushed on the stack (Section 8.3.1), and the rest of the error parsing is left to the operating system.

When the software machine check handler receives control, it must explicitly acknowledge receipt of the machine check with the following instruction:

```
MTPR    #0, #PR$_MCESR          ; PR$_MCESR=38
```

This acknowledgement should be done early in the software machine check handler to clear the internal machine-check-in-progress flag.

8.3.1 Machine Check Stack Frame

Information in the machine check stack frame (Figure 8–1) is parsed by the error-handling macrocode to determine exactly what caused the machine check.

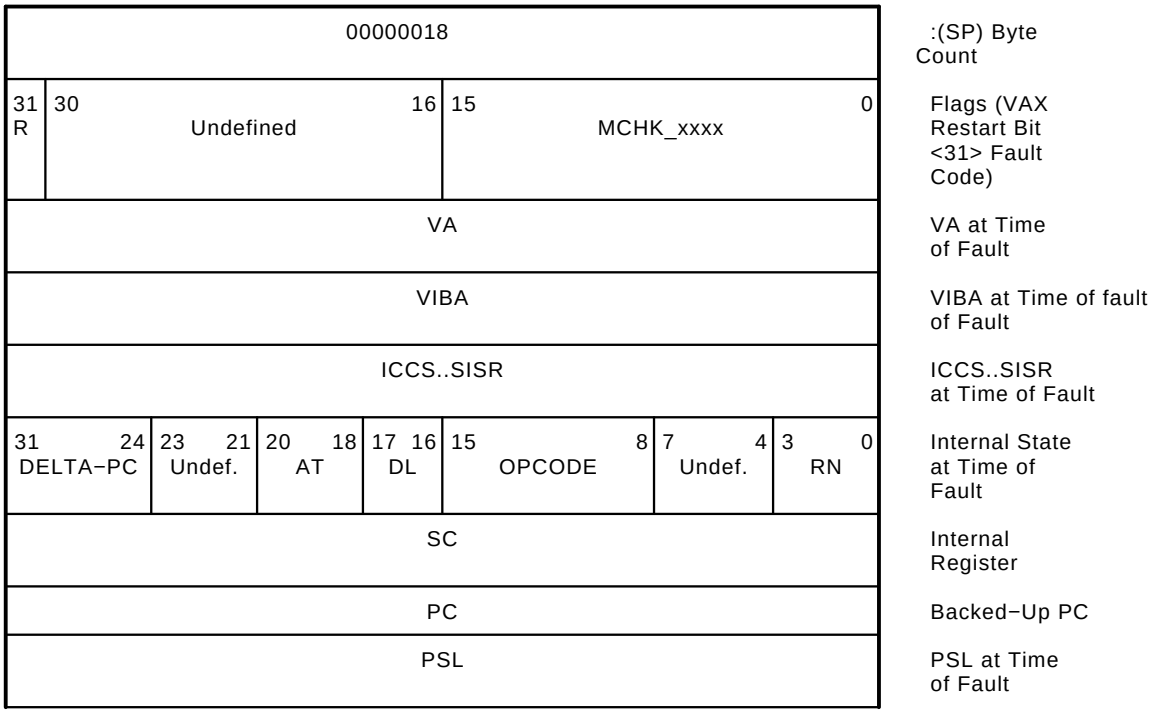


Figure 8–1 Stack Frame for Machine Check Exception

- **Byte count –** The size of the stack frame in bytes is 18_{16} bytes, not including PSL, PC, and the byte count longword. Stack frame PC and PSL values should always be referenced using this count as an offset from the stack pointer.
- **R (VAX restart bit) –** A flag from the hardware and microcode to the operating system, used in the software equation to determine whether or not the current macroinstruction is restartable after error cleanup. Other terms include PSL<FPD>, and PCSTS<trap2> (the primary cache double-error bit).
- **Fault code –** The type of machine check (Figure 8–2).
- **VA –** The address being processed by the CPU. This address is not necessarily relevant; the error handler should check the specific error address corresponding to the device or mechanism that signaled the error.
- **VIBA –** The CPU prefetch virtual instruction buffer address at the time of the fault.
- **ICCS..SISR –** The interrupt state information format (Table 8–4).

Table 8–4 Interrupt State Format

Bits	Contents
<22>	ICCS<6>
<15:1>	SISR<15:1>

- **Delta-PC –** Difference in the values of the current incremented PC (at the time the machine check was detected) and the PC of the instruction opcode. The exact

interpretation of this field requires a detailed knowledge of the internal pipeline operation of the CPU. This field should not be used by software to make recovery decisions.

- **AT –** The current setting of the CPU's E-box address type latch, possibly relating to the last (or upcoming) memory reference. Table 8–5 lists the values and interpretation.

Table 8–5 AT (Address-Type) Codes

Value (Binary)	Interpretation
000	Read
001	Write
010	Modify
011	Unassigned, CPU chip error
100	Unassigned, CPU chip error
101	Address
110	Variable bit
111	Branch

- **DL –** The current setting of the CPU's E-box data length latch, possibly relating to the last (or approaching) memory reference. Table 8–6 lists the values and interpretation.

Table 8–6 Data Length (DL) Codes

Value (Binary)	Interpretation
00	BYTE
01	WORD
10	LONG, F_Floating
11	QUAD, D_Floating, G_Floating

- **Opcode –** The opcode byte value of the instruction being processed at the time of the fault. For a 2-byte opcode, the value is the second byte.
- **RN –** The value of the CPU's E-box RN register at the time of the fault, possibly indicating the last GPR referenced by the E-box during specifier or instruction flows.
- **SC –** Internal microcode-accessible register.
- **PC, PSL –** Standard exception stack frame program counter and program status longword at the time of the fault.

The machine check fault code from the stack frame specifies the type of error and the conditions under which restart is possible. Table 8–7 lists the possible fault codes.

Table 8–7 Machine Check Fault Codes

Code (Hex)	Mnemonic	Meaning	
01	MCHK_FP_PROTOCOL_ ERROR	Protocol error during FPU operand/result transfer.	(R=1).(FPD=0)
02	MCHK_FP_ILLEGAL_OPCODE	Illegal opcode detected by FPU.	(R=1).(FPD=0)
03	MCHK_FP_OPERAND_PARITY	Operand parity error detected by FPU.	(R=1).(FPD=0)
04	MCHK_FP_UNKNOWN_ STATUS	Unknown status returned by FPU.	(R=1).(FPD=0)
05	MCHK_FP_RESULT_PARITY	Returned FPU result parity error.	(R=1).(FPD=0)
08	MCHK_TBM_ACV_TNV	TB miss status generated in ACV/TNV microflow.	((R=1)+(FPD=1))
09	MCHK_TBH_ACV_TNV	TB hit status generated in ACV/TNV microflow.	((R=1)+(FPD=1))
0A	MCHK_INT_ID_VALUE	Undefined INT.ID value during interrupt service.	((R=1)+(FPD=1))
0B	MCHK_MOVC_STATUS	Undefined state bit combination in MOVCx.	(FPD=1) [see description]
0C	MCHK_UNKNOWN_IBOX_ TRAP	Undefined trap code produced by the I-box.	(R=1).(FPD=0)
0D	MCHK_UNKNOWN_CS_ADDR	Undefined control store address reached.	((R=1)+(FPD=1))
10	MCHK_BUSERR_READ_ PCACHE	Primary cache tag or data parity error during read.	((R=1)+(FPD=1)).(TR2=0)
11	MCHK_BUSERR_READ_DAL	DAL bus or data parity error during read.	((R=1)+(FPD=1)).(TR2=0)
12	MCHK_BUSERR_WRITE_DAL	DAL bus error on write or clear write buffer.	No
13	MCHK_UNKNOWN_BUSERR_ TRAP	Undefined bus error microtrap.	No

R = the VAX restart bit in the machine check stack frame.
FPD = the CPU PSL<FPD> first part done bit.
TR2 = the CPU PCSTS<trap2> double-error bit.
. = the logical AND operation.
+ = the logical OR operation.

8.3.2 Machine Check Parse Tree

Figure 8–2 shows the machine check parse tree.

An error parse tree is a diagram that shows how the progression of an error can be tracked. Each horizontal line represents a signal. Moving from left to right, each vertical line represents a deeper level of error. The most nested errors are toward the right side of the diagram. The least nested errors are toward the left.

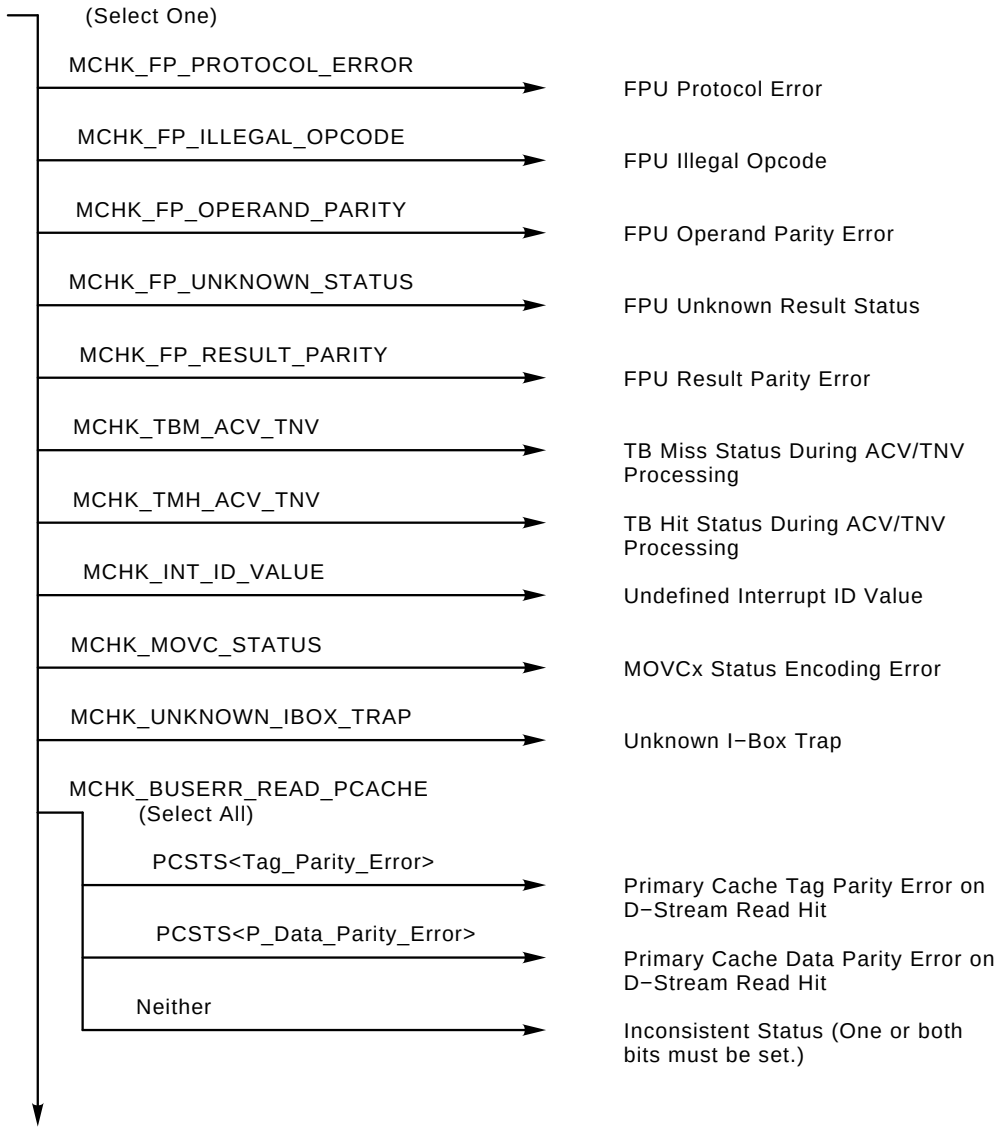


Figure 8–2 (Cont.) Machine Check Parse Tree

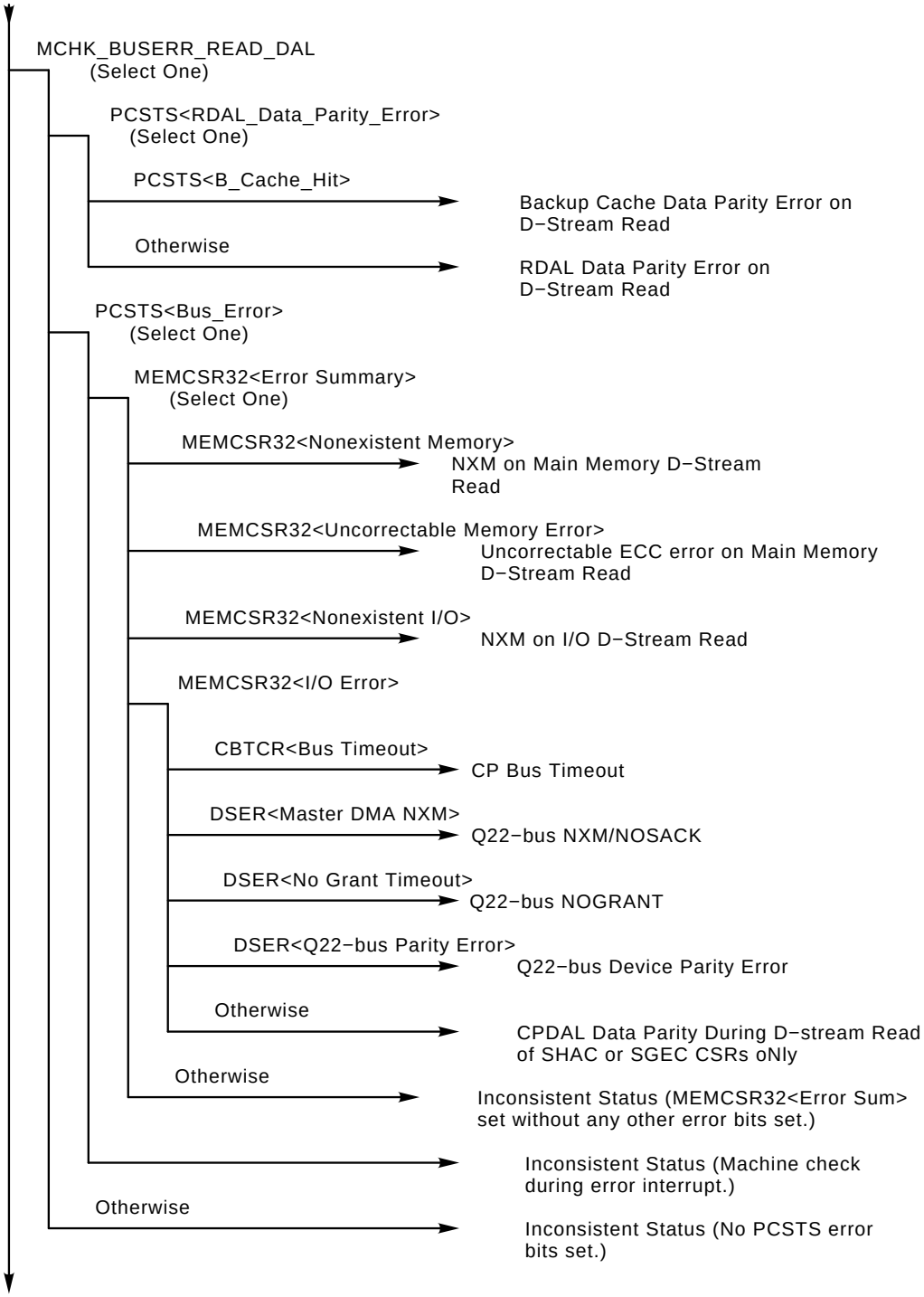
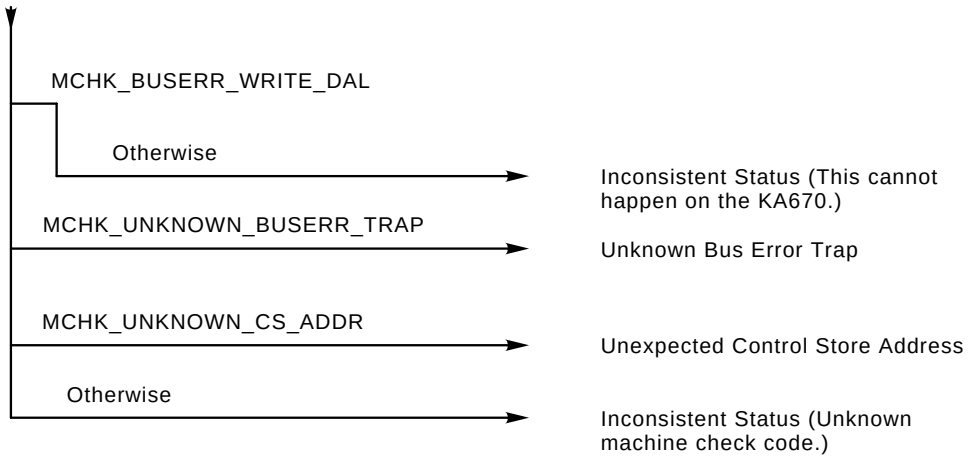


Figure 8–2 (Cont.) Machine Check Parse Tree



Key:

Select One	− Exactly one case must be true. If zero or more than one is true, the status is inconsistent.
Select All	− More than one case may be true.
Otherwise	− Fall−through case for Select One, if no other options are true.
Neither	− Fall−through case for Select All, if no other options are true.

Figure 8–2 Machine Check Parse Tree

8.3.3 MCHK_FP_PROTOCOL_ERROR

Description: The CPU or FPU detected a protocol error during an operand/result transfer. During a result return, the cases listed in Table 8–8 cause this machine check.

Table 8–8 MCHK_FP_PROTOCOL_ERROR

CPSTA	CPDAT<2:0>	Detected by
00	xxx	CPU
11	xxx	CPU
10	000	FPU

This error is probably due to a bit flipped on the CPSTA or CPDAT lines during an opcode, operand, or result transfer.

Recovery procedures: No explicit error recovery is required. If the error reoccurs, disable the FPU by writing a 0 to ACCS<1>.

Restart condition: Because this error is detected during the execution flow of an FPU instruction, R should always be 1 and PSL<FPD> should always be 0.

Retry if:

```
(R=1) AND (PSL<FPD>=0)
```

8.3.4 MCHK_FP_ILLEGAL_OPCODE

Description: An illegal opcode was detected by the FPU and reported during result return. The probably cause is a bit flipped on the CPSTA or CPDAT lines during an opcode transfer to the FPU.
Recovery procedures: No explicit error recovery is required. If the error reoccurs, disable the FPU by writing a 0 to ACCS<1>.

Restart condition: Because this error is detected during the execution flow of an FPU instruction, R should always be 1 and PSL<FPD> should always be 0.

Retry if:

```
(R=1) AND (PSL<FPD>=0)
```

8.3.5 MCHK_FP_OPERAND_PARITY

Description: The FPU detected a parity error during an operand transfer and reported the error during a result return. Note that the CPU should also have detected backup cache or memory parity errors, which would have resulted in a MCHK_BUSERR_READ_DAL machine check instead.

The MCHK_FP_OPERAND_PARITY machine check indicates that the parity error was detected only by the FPU. This implies that the CPU generated bad parity, or the CPU and FPU saw different operands (or parity) from the backup cache or memory. The error is a function of the data source listed in Table 8–9 (which cannot be determined from the machine check).

Table 8–9 MCHK_FP_OPERAND_PARITY

Data Source	Possible Causes
GPR	CPU parity generation error, FPU parity<P> checking error, RDAL bus error during operand transfer
I-stream	CPU parity generation error, FPU parity<P> checking error, RDAL bus error during operand transfer
Primary cache	CPU parity checking error, FPU parity<P> checking error, RDAL bus error during operand transfer
Backup cache	CPU parity checking error, FPU parity<P> checking error, RDAL bus error during operand transfer
Memory	CPU parity checking error, FPU parity<P> checking error, RDAL bus error during operand transfer

NOTE
It is possible to get this machine check if an FPU operand is read from an I/O space location. CPU parity checking is disabled for I/O space reads, but the FPU checks parity for all operands.
Recovery procedures: No explicit error recovery is required. If the error reoccurs, disable the FPU by writing a 0 to ACCS<1>.

Restart condition: Because this error is detected during the execution flow of an FPU instruction, R should always be 1 and PSL<FPD> should always be 0.

Retry if:

```
(R=1) AND (PSL<FPD>=0)
```

8.3.6 MCHK_FP_UNKNOWN_STATUS

Description: The FPU returned an unassigned status code. This error occurs when CPSTA=10 and CPDAT<2:0>=111 appear with the returned result (from the FPU to CPU). The probable cause is a bit flipped on the CPSTA or CPDAT lines during the result transfer to the CPU.
Recovery procedures: No explicit error recovery is required. If the error reoccurs, disable the FPU by writing a 0 to ACCS<1>.

Restart condition: Because this error is detected during the execution flow of an FPU instruction, R should always be 1 and PSL<FPD> should always be 0.

Retry if:

```
(R=1) AND (PSL<FPD>=0)
```

8.3.7 MCHK_FP_RESULT_PARITY

Description: The CPU detected a result data parity error during an FPU result transfer. The probable cause is a bit flipped on the RDAL bus or parity lines.

Recovery procedures: No explicit error recovery is required. If the error reoccurs, disable the FPU by writing a 0 to ACCS<1>.

Restart condition: Because this error is detected during the execution flow of an FPU instruction, R should always be 1, PSL<FPD> should always be 0.

Retry if:

```
(R=1) AND (PSL<FPD>=0)
```

8.3.8 MCHK_TBM_ACV_TNV

Description: During ACV/TNV microcode processing, the MMGT.STATUS bits specified a TB-miss status (which should not be possible during ACV/TNV processing). The probablle cause is an internal error in the memory management hardware or microbranch logic.
Recovery procedures: No explicit error recovery is required in response to this error.

Restart condition: This error can happen during the microcode processing of an ACV/TNV exception on any virtual memory reference.

Retry if:

```
((R=1) OR (PSL<FPD>=1))
```

8.3.9 MCHK_TBH_ACV_TNV

Description: During ACV/TNV microcode processing, the MMGT.STATUS bits specified a TB-hit status (which should not be possible during ACV/TNV processing). The probable cause is an internal error in the memory management hardware or the microbranch logic.
Recovery procedures: No explicit error recovery is required.

Restart condition: This error can happen during the microcode processing of an ACV/TNV exception on any virtual memory reference.

Retry if:

```
(R=1) OR (PSL<FPD>=1)
```

8.3.10 MCHK_INT_ID_VALUE

Description: During interrupt processing, the microbranch on the contents of the INT.ID register resulted in an unexpected interrupt ID. The probable cause is a failure in the interrupt encoding logic or microbranch logic.

Recovery procedures: No explicit error recovery is required.

Restart condition: Because interrupts can only occur between instructions or in the middle of interruptable instructions, R should always be a 1 unless PSL<FPD> is a 1. If PSL<FPD> is a 1, PC should point to MOVCx, CMPCx, LOCC, SKPC, SCANC or SPANC.

Retry if:

```
(R=1) OR (PSL<FPD>=1)
```

8.3.11 MCHK_MOVC_STATUS

Description: During the execution of MOVCx, the two state bits that encode the state of the move (forward, backward, fill) were found set to the fourth (illegal) combination. The probable cause is a failure in the state bit logic or microbranch logic.

Recovery procedures: No explicit error recovery is required.

Restart condition: Because the state bits encode the operation, the instruction cannot be restarted in the middle of the MOVCx. If software can determine that no specifiers were overwritten (MOVCx destroys R0 to R5 and memory, due to string writes), the instruction may be restarted from the beginning by clearing PSL<FPD>. This should be done only if the source and destination strings do not overlap and if:

```
(PSL<FPD>=1)
```

8.3.12 MCHK_UNKNOWN_IBOX_TRAP

Description: The I-box requested a microtrap to report an illegal instruction or a reserved operand fault, but the bits that encode the reason specified an illegal value. The probable cause is a failure in the I-box/E-box interface, or in the microsequencer trap logic.

Recovery procedures: No explicit error recovery is required.

Restart condition: Because this microtrap can only occur at an instruction boundary, R should be a 1 and PSL<FPD> should be a 0.

Retry if:

```
(R=1) AND (PSL<FPD>=0)
```

8.3.13 MCHK_BUSERR_READ_PCACHE

This code indicates that one of two errors was detected during a D-stream read that hit in the primary cache. To get either of these machine checks, the primary cache must be enabled.

PCSTS<tag_parity_error> and PCSTS<P_data_parity_error> distinguish the cases. If neither bit is set, the status is inconsistent, and the error should not be retried. In both cases, PCERR contains the physical address of the error.

8.3.13.1 Primary Cache Tag Parity Error on D-Stream Read Hit

Description: A primary cache tag parity error was detected on a D-stream read hit. PCSTS<trap1>, PCSTS<interrupt>, and PCSTS<tag_parity_error> should all be set. This error is also reported by a soft error interrupt.)
Recovery procedures: Write all primary cache tags with good parity and cleared valid bits. Then perform the full memory error recovery procedures (Section 8.1.2.3). If the error reoccurs, disable both the primary cache and the primary tag store copy in the C-chip.

Restart condition:

Retry if:

```
((R=1) OR (PSL<FPD>=1)) AND (PCSTS<trap2>=0).
```

8.3.13.2 Primary Cache Data Parity Error on D-Stream Read Hit

Description: A Primary cache data parity error was detected on a D-stream read hit. PCSTS<trap1> and PCSTS<P_data_parity_error> should both be set.
Recovery procedures: Perform the full memory error recovery procedures (Section 8.1.2.3). If the error reoccurs, disable both the primary cache and the primary tag store copy in the C-chip.

Restart condition:

Retry if:

```
((R=1) OR (PSL<FPD>=1)) AND (PCSTS<trap2>=0)
```

8.3.14 MCHK_BUSERR_READ_DAL

This code indicates that one of two classes of errors was detected during a D-stream read. PCSTS<RDAL_data_parity_error> and PCSTS<bus_error> distinguish the two classes. If neither or both bits are set, the status is inconsistent and the error should not be retried.

8.3.14.1 Data Parity Error on D-Stream Read

A data parity error was detected during a D-stream read. The source of the data parity error is either the backup cache or memory, distinguished by PCSTS<B_cache_hit>. In both cases, PCERR contains the physical address of the error.

8.3.14.1.1 Backup Cache Data Parity Error on D-Stream Read

Description: A data parity error was detected during a D-stream read hit in the backup cache. PCSTS<TRAP1>, PCSTS<RDAL_data_parity_error>, and PCSTS<B_cache_hit> should all be set.
Recovery procedures: Perform the full memory error recovery procedures (Section 8.1.2.3). If the error reoccurs, disable the backup cache.

Restart condition:

Retry if:

```
((R=1) OR (PSL<FPD>=1)) AND (PCSTS<trap2>=0).
```

8.3.14.1.2 Memory Data Parity Error on D-Stream Read

Description: A data parity error was detected during a D-stream read from memory. Note that an actual memory parity error would have been reported as a bus error (next section). This error implies that the parity went bad between the G-chip and the CPU. PCSTS<trap1> and PCSTS<RDAL_data_parity_error> should both be set. PCSTS<B_cache_hit> should be cleared.

Recovery procedures: Perform the full memory error recovery procedures (Section 8.1.2.3).

Restart condition:

Retry if:

```
((R=1) OR (PSL<FPD>=1)) AND (PCSTS<trap2>=0).
```

8.3.14.2 Bus Error on D-Stream Read

An RDAL D-stream read transaction was terminated with ERR_L. In order for the BCSTS register to log this error, the backup cache must be on and the reference must be to a non-I/O space address.

8.3.14.2.1 Memory Error on Requested Quadword of D-stream Read

Description: The G-chip detected an error on a D-stream read. MEMCSR33 must match (to the closest octaword) PCERR. Otherwise, the status is inconsistent, and the error should not be retried. MEMCSR33 and PCERR contain the physical address of the error.

The source of the error is distinguished by bits in MEMCSR32, as follows:

- **MEMCSR32<error summary> –** This bit must be set, indicating that an error has been logged by the G-chip.
- **MEMCSR32<nonexistent memory> –** The non-I/O octaword address logged in MEMCSR33 is not mapped by the G-chip, so the address is nonexistent. PCSTS<trap1> should be set. If this is a real NXM, a retry will not succeed and should not be attempted if NXMs are expected. If NXMs are not expected and a retry is desired, do so under the conditions stated above.
- **MEMCSR32<uncorrectable memory error> –** The G-chip detected an uncorrectable ECC error within an octaword at the address logged in MEMCSR33. PCSTS<trap1> should be set.
- **MEMCSR32<nonexistent I/O> –** The I/O longword address logged in MEMCSR34 was not responded to by any of the CP bus I/O devices, so the address is considered nonexistent. PCSTS<trap1> should be set. If this is a real NXM, a retry will not succeed and should not be attempted if NXMs are expected. If NXMs are not expected and a retry is desired, do so under the conditions stated above.
- **MEMCSR32<I/O> –** The I/O longword read whose address is logged in MEMCSR34 had an error. There are five types of CP bus error that can cause this error:
  - **CBTCR<bus timeout> –** This is a CP bus timeout. This bit indicates that a CP bus I/O device is in an inconsistent state—the device deasserts its NOT_ME line (indicating the address *is* for it), but never completes the CP bus cycle with RDY. The SSC's 15-millisecond watch dog timer expires, terminating the cycle with ERR. PCSTS<trap1> should be set.
  - **DSER<master DMA NXM> –** This is a Q22-bus NXM/NOSACK error. The Q22-bus address is stored in DMEAR.

- **DSER<no grant> –** This a Q22-bus NOGRANT error, indicating that the CQBIC's 10-microsecond NOGRANT timer has expired.
- **DSER<Q22-bus parity error> –** This a Q22-bus device parity error. The Q22-bus address is stored in DMEAR.
- **None of the above bits set –** This indicates that a CP bus parity error occurred while reading either the SHAC's or SGEC's internal registers.

Recovery procedures: Perform the full memory error recovery procedures (Section 8.1.2.3).

Restart condition: Unless otherwise stated in the error descriptions, retry if:

```
((R=1) OR (PSL<FPD>=1)) AND (PCSTS<trap2>=0),
```

8.3.15 MCHK_BUSERR_WRITE_DAL

Description: An RDAL write or clear write buffer transaction was terminated with the ERR_L terminator. For the BCSTS register to log the error, the backup cache must be on and the reference must be to a non-I/O space address. The G-chip in the KA670 never does this, so the error is considered serious and unrecoverable.

8.3.16 MCHK_UNKNOWN_BUSERR_TRAP

Description: The CPU's BIU requested a microtrap to report a cache or bus error, but the bits that encode the reason specified an illegal value. The probable cause is a failure in the BIU or microsequencer trap logic.

Recovery procedures: No explicit error recovery is required.

Restart condition: Because this error may be masking a write error, a retry should not be attempted.

8.3.17 MCHK_UNKNOWN_CS_ADDR

Description: An unexpected address was reached in the CPU's control store. The probable cause is a failure in the microsequencer logic or a microcode bug.

Recovery procedures: No explicit error recovery is required.

Restart conditions:

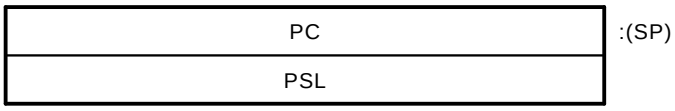
Retry if:

```
(R=1) OR (PSL<FPD>=1)
```

8.4 Power-Fail Interrupt

Power-fail interrupts are requested to report imminent loss of power to the module. Power-fail interrupts are requested at IPL $1E_{16}$ and dispatched through SCB vector $0C_{16}$.

The stack frame for a power fail interrupt is as follows:

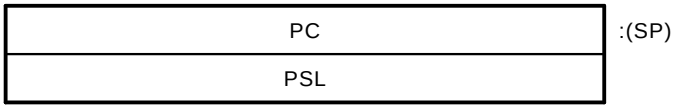


The KA670 system supports the standard Q22-bus time of 20 milliseconds to execute the software needed to save the processor state.

8.5 Hard Error Interrupts

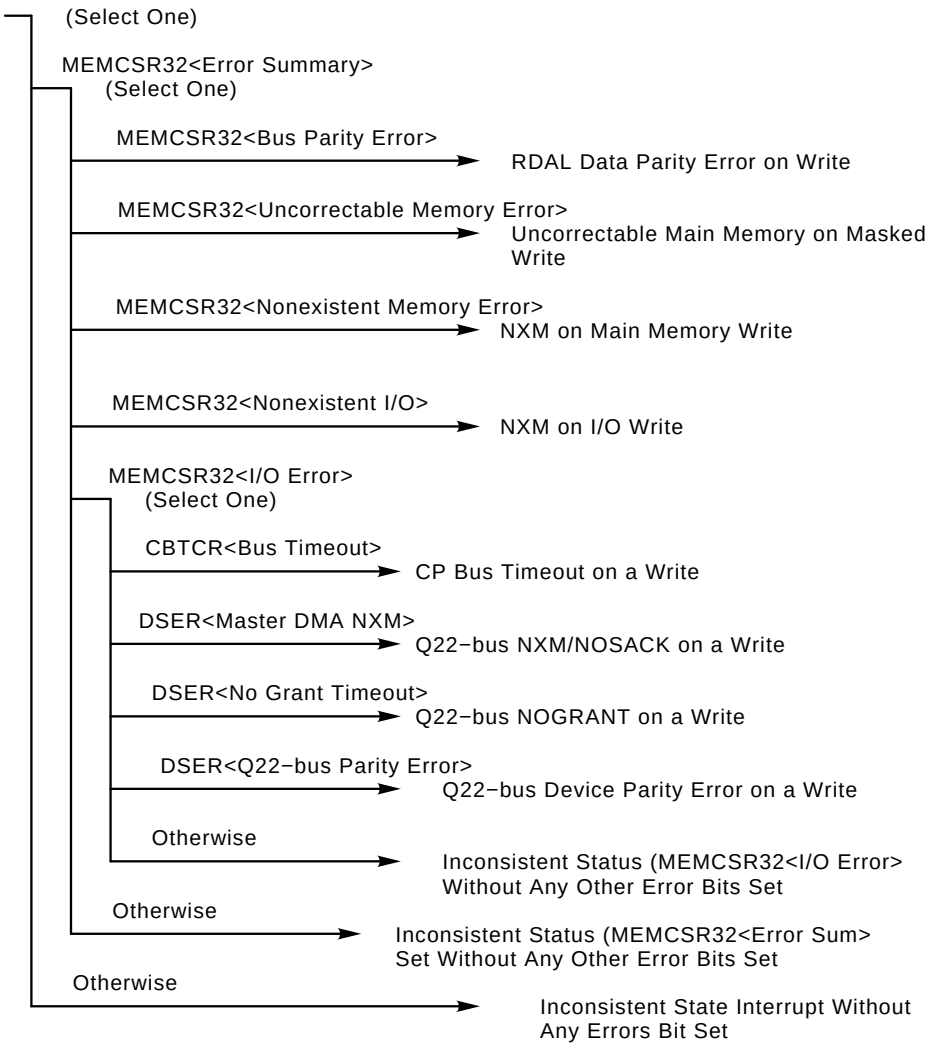
Hard error interrupts are requested to report any error detected asynchronously with instruction execution. This results in an interrupt at IPL $1D_{16}$, dispatched through SCB vector 60_{16}. Typically, these errors indicate that machine state is corrupted and that retry is not possible.

The stack frame for a hard error interrupt is as follows:



8.5.1 Parse Tree for a Hard Error Interrupt

Figure 8–3 shows the parse tree for a hard error interrupt.



Key:

Select One	− Exactly one case must be true. If zero or more than one is true, the status is inconsistent.
Select All	− More than one case may be true.
Otherwise	− Fall−through case for Select One, if no other options are true.

Figure 8–3 Parse Tree for a Hard Error Interrupts

8.5.2 RDAL Data Parity Error on Memory Write

Description: The G-chip detected a parity error on the RDAL data for a memory write from the CPU. The probable cause is a bit flipped on the D-bus, or an intentional or unintentional bad parity generated by the CPU. The GMI is completed with bad ECC.

Recovery procedures: Clear all error bits in MEMCSR32.

Restart conditions: Because this error indicates a failed write, no retry is possible.

8.5.3 Uncorrectable Main Memory Error on Masked Write

Description: The G-chip detected an uncorrectable error in the read part of a read-modify-write transaction on a masked memory write from the CPU.The GMI is completed with bad ECC.

Recovery procedures: Clear all error bits in MEMCSR32.

Restart conditions: Because this error indicates a failed write, no retry is possible.

8.5.4 Main Memory Nonexistent Write

Description: The G-chip detected a NXM error on the RDAL data for a memory write from the CPU.

Recovery procedures: Clear all error bits in MEMCSR32.

Restart conditions: Because this error indicates a failed write, no retry is possible.

8.5.5 I/O Nonexistent Write

Description: The G-chip detected a NXM error for a I/O write from the CPU.

Recovery procedures: Clear all error bits in MEMCSR32.

Restart conditions: Because this error indicates a failed write, no retry is possible.

8.5.6 CP Bus Timeout on a Write

Description: The SSC's 15-millisecond CP bus watchdog has expired on an I/O write from the CPU.

Recovery procedures: Clear all error bits in MEMCSR32.

Restart conditions: Because this error indicates a failed write, no retry is possible.

8.5.7 Q22-bus NXM/NOSACK on a Write

Description: The CQBIC has failed while trying to complete a write on the Q22-bus.

Recovery procedures: Clear all error bits in MEMCSR32 and DSER.

Restart conditions: Because this error indicates a failed write, no retry is possible.

8.5.8 Q22-bus NOGRANT on a Write

Description: The CQBIC has failed to be granted the Q22-bus while trying to complete a write.

Recovery procedures: Clear all error bits in MEMCSR32 and DSER.

Restart conditions: Because this error indicates a failed write, no retry is possible.

8.5.9 Q22-bus Device Parity Error on a Write

Description: The CQBIC has been notified of a Q22-bus data parity error while trying to complete a write.

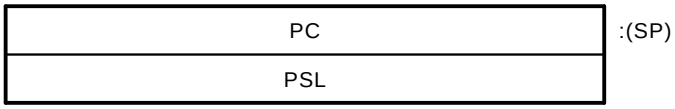
Recovery procedures: Clear all error bits in MEMCSR32 and DSER.

Restart conditions: Because this error indicates a failed write, no retry is possible.

8.6 Soft Error Interrupts

Soft error interrupts are requested to report errors that were detected but did not affect instruction execution. This results in an interrupt at IPL $1A_{16}$, dispatched through SCB vector 54_{16}.

The stack frame for a soft error interrupt is as follows:



8.6.1 Parse Tree for Soft Error Interrupts

Figure 8–4 shows the parse tree for soft error interrupts.

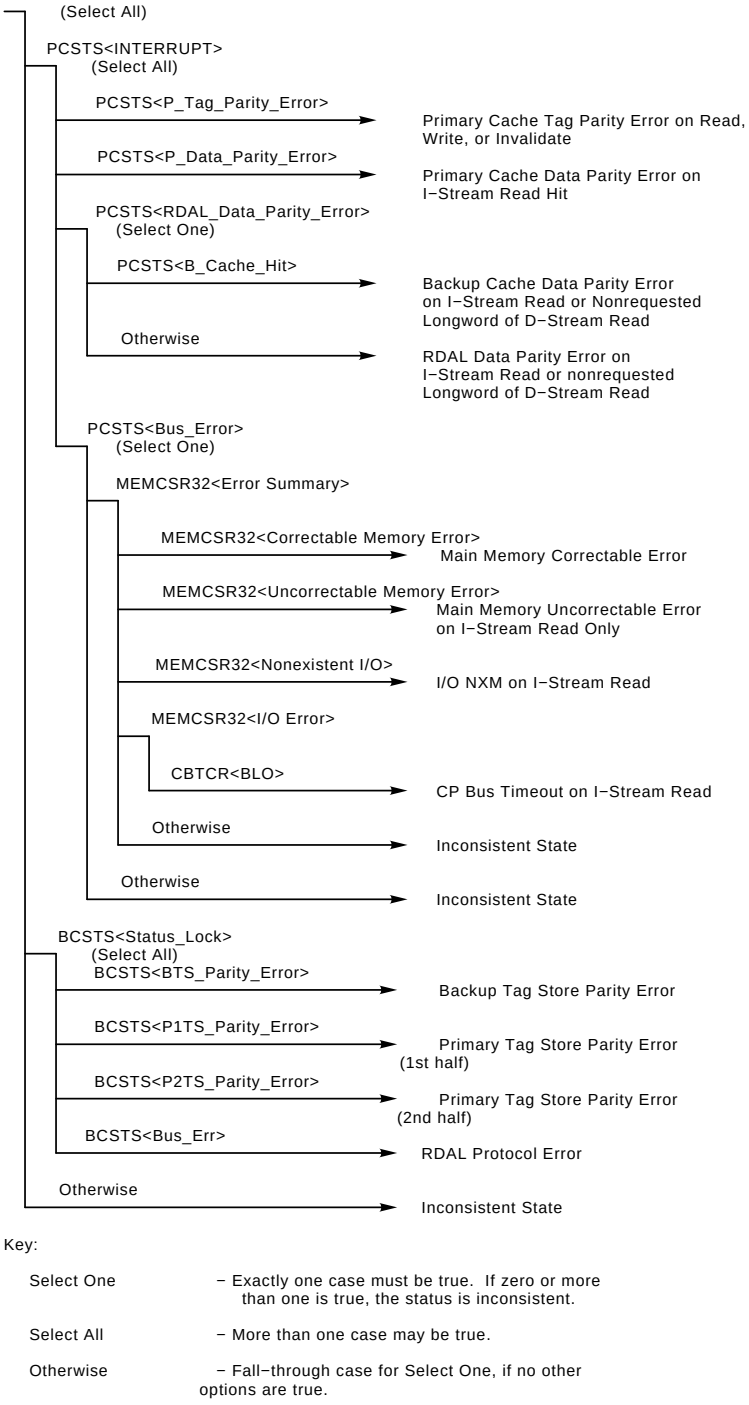


Figure 8–4 Soft Error Interrupt Parse Tree

All errors reported by this interrupt are soft errors. A retry is always possible after recovery, unless otherwise stated in the description of an error.

8.6.2 Cache or Memory Errors

A primary cache error, data parity error, or bus error is detected during a memory reference. PCSTS<interrupt> distinguishes this class from others. At least one of PCSTS<P_tag_parity_error>, PCSTS<P_data_parity_error>, PCSTS<RDAL_data_parity_error>, or PCSTS<bus_error> should be set. PCERR does not contain the error address in this class of errors.
The CPU microcode automatically retries all I-stream errors, using D-stream reads. If the error is hard, the D-stream read is reported as a machine check with code MCHK_BUSERR_READ_DAL. If the error is transient, it is reported as a soft error interrupt; this indicates the retry was successful.

8.6.2.1 Primary Cache Errors

Two types of primary cache errors can be detected during an I-stream read, write, or invalidate. The primary cache must be on to get either error. PCSTS<P_tag_parity_error> and PCSTS<P_data_parity_error> distinguish the two cases.

8.6.2.1.1 Primary Cache Tag Parity Error

Description: A primary cache tag parity error was detected during an I-stream read, a D-stream read miss, a write, or an invalidate. If the error had been detected during a D-stream read hit, the error would have been reported as a machine check with code MCHK_BUSERR_READ_PCACHE. PCSTS<P_tag_parity_error> and PCSTS<interrupt> should both be set.

Recovery procedures: Write all primary cache tags with good parity and cleared valid bits. Then perform the full memory error recovery procedures (Section 8.1.2.3). If the error reoccurs, disable both the primary cache and the primary tag store copy in the C-chip.

8.6.2.1.2 Primary Cache Data Parity Error on I-Stream Read Hit

Description: A primary cache data parity error was detected during an I-stream read hit. If the error had been detected during a D-stream read hit, the error would have been reported as a machine check with code MCHK_BUSERR_READ_PCACHE. PCSTS<P_data_parity_error> and PCSTS<interrupt> should both be set.

Recovery procedures: Perform the full memory error recovery procedures (Section 8.1.2.3). If the error reoccurs, disable both the primary cache and the primary tag store copy in the C-chip.

8.6.2.2 RDAL Data Parity Errors

An RDAL data parity error can be detected during an I-stream read or on the nonrequested longword of a D-stream read. If the error had been detected in the requested longword of a D-stream read, the error would have been reported as a machine check with code MCHK_BUSERR_READ_DAL. The source of the data parity error is either the backup cache or memory, distinguished by PCSTS<B_cache_hit>.

8.6.2.2.1 Backup Cache Data Parity Error

Description: A data parity error was detected during an I-stream read or in the nonrequested longword of a D-stream read that hit in the backup cache. PCSTS<interrupt>, PCSTS<RDAL_data_parity_error>, and PCSTS<B_cache_hit> should all be set.

Recovery procedures: Perform the full memory error recovery procedures (Section 8.1.2.3). If the error reoccurs, disable the backup cache.

8.6.2.2.2 Memory Data Parity Error

Description: A data parity error was detected during an I-stream read or in the nonrequested longword of a D-stream read from memory. Note that an actual memory parity error would have been detected by the G-chip and reported as a memory read error (Section 8.6.2.3.1). This error implies that the parity went bad between the G-chip and the CPU. PCSTS<interrupt> and PCSTS<RDAL_data_parity_error> should both be set. PCSTS<B_cache_hit> should be cleared.

Recovery procedures: Perform the full memory error recovery procedures (Section 8.1.2.3).

8.6.2.3 Bus Error on I-Stream Read

An RDAL I-stream read transaction was terminated with ERR_L. If the error had been detected during a D-stream read, the error would have been reported as a machine check with code MCHK_BUSERR_READ_DAL. For the BCSTS register to log the error, the backup cache must be on and the reference must be to a non-I/O space address.

8.6.2.3.1 Memory Error I-Stream Read

Description: The G-chip detected an error on an I-stream read. Depending on the type of error, either MEMCSR33 or MEMCSR34 contains the physical address of the error.

The source of the error is distinguished by bits in MEMCSR32, as follows:

- **MEMCSR32<uncorrectable memory error> –** An uncorrectable ECC error was found in the naturally aligned octaword around the I-stream read. PCSTS<bus_error>, PCSTS<INTERRUPT>, BCSTS<Bus_err>, and BCSTS<status_lock> should all be set.
- **MEMCSR32<nonexistent I/O> –** The I-stream read from I/O space is nonexistent. The G-chip detects an I/O NXM. This differs from a CP bus timeout, where the SSC's watchdog timer detects the timeout. PCSTS<bus_error> and PCSTS<interrupt>, should also be set.
- **MEMCSR32<I/O error> –** The I-stream read from I/O space was timed out by the SSC. The SSC's watchdog timer detected a CP bus timeout. This differs from an I/O NXM, where the G-chip detects the I/O NXM. PCSTS<bus_error>, PCSTS<interrupt>, and CBTCR<BTO> should all be set.

Recovery procedures: Perform the full memory error recovery (Section 8.1.2.3).

8.6.3 Cache Fill Errors on the Nonrequested Quadword of a Read

Description: The C-chip detected an RDAL data parity error on the nonrequested fill quadword of a D-stream or I-stream read.

The source of the error is distinguished by the PCSTS<B_cache_hit> bit. If PCSTS<B_cache_hit> is set, then the source of the data was the backup cache RAMs. If PCSTS<B_cache_hit> is clear, then the source on the data is the G-chip.

Recovery procedures: Perform the full memory error recovery procedures (Section 8.1.2.3).

8.6.4 C-Chip Errors

The C-chip detected a tag parity error during a tag store access, or a bus protocol error during an RDAL transaction. The C-chip must be on to detect either of these errors.

8.6.4.1 C-Chip Backup Tag Store Parity Error

Description: A tag store parity error was detected in the backup tag store during a read, write, fill, invalidate, or I_bus access. BCSTS<BTS_parity_err> and BCSTS<status_lock> should both be set. BCERR contains the physical address of the error.

Recovery procedures: Use the BCERR address to rewrite the tag with good parity and a cleared valid bit. Then perform the full memory error recovery procedures (Section 8.1.2.3). If the error reoccurs, disable the backup cache.

8.6.4.2 C-Chip Primary Tag Store Parity Error

Description: A tag store parity error was detected in one of the primary tag stores during a fill, invalidate, or I_bus access. Either BCSTS<P1TS_parity_err>, BCSTS<P2TS_parity_err>, or both should be set, along with BCSTS<status_lock>. BCERR contains the physical address of the error.

Recovery procedures: Use the BCERR address to rewrite the tag with good parity and a cleared valid bit. Then perform the full memory error recovery procedures (Section 8.1.2.4). If the error reoccurs, disable both the primary cache and the primary tag store copy in the C-chip.

8.6.4.3 C-Chip Bus Protocol Error

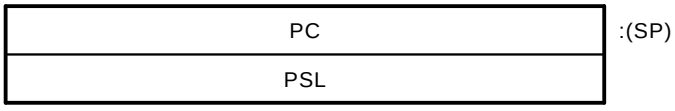
Description: If the cache RAM speed in the C-chip (BCCTL<two_cycle_RAMs>) and the G-chip (MEMCSR36<cache ram speed>) are different, the C-chip may detect a protocol error during memory writes or cache fill operations. BCSTS<bus_err> and BCSTS<status_lock> should both be set.

Recovery procedures: Check the cache RAM speeds in the C-chip and G-chip to make sure that they agree. Then perform the full memory error recovery procedures (Section 8.1.2.3).

8.7 Kernel Stack Not Valid Exception

A kernel stack not valid exception occurs when a memory management exception is detected while attempting to push information on the kernel stack during microcode processing of another exception. A console halt with an error code of ERR_INTSTK occurs if a memory management exception is encountered while attempting to push information on the interrupt stack.

The kernel stack not valid exception is dispatched through SCB vector 08_{16}. The stack frame is as follows:



No additional information is provided for this exception.

8.8 Errors Without Notification

There are errors that do not produce explicit notification. These errors only set the error status bits. The next time the software inspects the register (for other reasons), it may want to check or log these bits.

8.8.1 Parity Generation and Detection Philosophy

The following list summarizes the parity generation and check characteristics of the KA670:

- The CPU generates parity on write data and checks parity on read data (for memory transactions). The CPU does not generate parity on command/address information.
- A CPU I-stream parity error is reported as an interrupt, with the appropriate bits set in the primary cache status register. The microcode then tries to recover with a D-stream read, which results in a machine check (MCHK_BUSERR_READ_DAL) if the error is hard.
- The primary cache (contained in the CPU) supports parity on both the tag and data store.
- The backup cache supports parity on both the tag and data store. On cache fills and writes, parity is stored and then checked by the CPU during reads.
- The G-chip detects RDAL data parity errors on writes.
- The FPU generates parity for FPU results and checks parity on RDAL data bus floating operands.
- The two DSSI interface chips (SHACs) generate parity for all CP bus DMA writes. The SHACs check parity on DMA reads and internal register reads.
- The network interface chip (SGEC) generates parity for all CP bus DMA writes. The SGEC checks parity on DMA reads and internal register reads.
- The SSC does not support parity, so the 1-kilobyte of internal battery backed-up RAM and the internal registers are not protected.
- The CQBIC does not support parity on the CP bus, so the bus's DMA activity and internal registers are not protected by parity. Note, the CQBIC *does* support parity on the Q22 bus.

8.8.2 Microcode-Detected Error Summary

The following list shows errors detected (triggered) by microcode checks. All the errors listed are described in this chapter.

- Console halt
  - —ERR_INTSTK
  - —ERR_DOUBLE
  - —ERR_HLTINS
  - —ERR_ILLVEC
  - —ERR_WCSVEC
  - —ERR_CHMFI
  - —ERR_MCHK_ACV_TNV

—ERR_KSNV_ACV_TNV

—ERR_MCHK_MCHK

—ERR_KSNV_MCHK

—ERR_IE_PSL26_24_101

—ERR_IE_PSL26_24_110

—ERR_IE_PSL26_24_111

—ERR_REI_PSL26_24_101

—ERR_REI_PSL26_24_110

—ERR_REI_PSL26_24_111

—ERR_SELFTEST_FAILED

- Machine check

  —MCHK_FP_UNKNOWN_STATUS

  —MCHK_TBM_ACV_TNV

  —MCHK_TBH_ACV_TNV

  —MCHK_INT_ID_VALUE

  —MCHK_MOVC_STATUS

  —MCHK_UNKNOWN_IBOX_TRAP

  —MCHK_UNKNOWN_BUSERR_TRAP

  —MCHK_UNKNOWN_VECTOR_STATUS

- Kernel stack not valid

8.8.3 Errors Detected by Self-Tests

There are two levels of self-test errors—power-up CPU microcode self-test, and boot ROM macrocode self-test.

- A failing microcode self-test produces a halt- to- console error code of ERR_ SELFTEST_FAILED.
- A failing macrocode self-test produces an error message printed at the console. The system is left at the >>> prompt.

Firmware

- Chapter 9, Firmware

9
Firmware

This chapter describes the KA670 functional firmware. The firmware is VAX-11 code that resides in EPROM on the KA670 module. The chapter covers the following major topics:

- Firmware capabilities
- Halt entry, halt exit, and halt dispatch
- Power-up
- Operating system bootstrap
- Console service
- Console commands
- Diagnostics

Typically KA670 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 KA670 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.

A more detailed description of the major components of the KA670 is provided in Section 9.1 and a structural diagram of the KA670 firmware is given in Figure 9–1.

Terms in This Chapter

Firmware
A generic term describing all program code in the KA670 EPROM. Sometimes, firmware is referred to as either the *boot ROM*, *diagnostics ROM*, or *console ROM*, depending on context.

Virtual memory boot (VMB) or primary bootstrap
The boot program.

Diagnostic or self-test
The ROM-based diagnostic program.

Console or console program
The operator interface.

9.1 Firmware Capabilities

The KA670 firmware provides the following services:

- Diagnostics that test all components on the board and verify 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 following sections describes in detail the functions and external characteristics of the KA670 firmware.

9.2 Firmware Overview

The KA670 firmware comprises several major functional blocks of code, as shown in Figure 9–1.

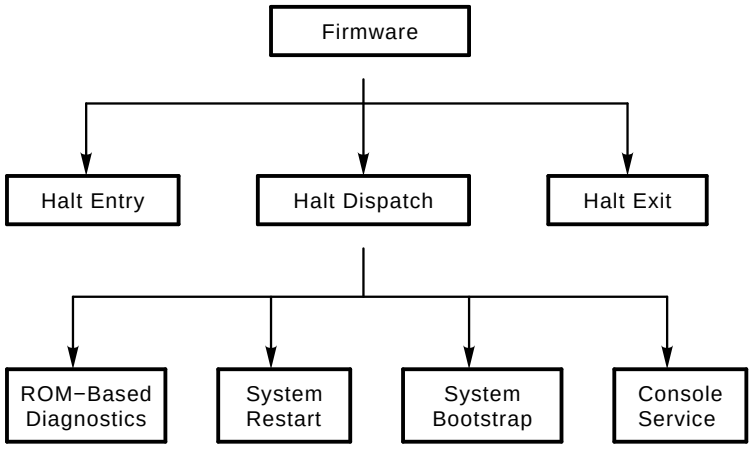


Figure 9–1 KA670 Firmware Structural Components

The halt entry code is entered following system halts, resets, or severe errors. Basically, this code is responsible for saving the machine state and transferring control to the firmware dispatcher. The halt dispatcher determines the nature of the halt, then transfers control to the appropriate code. The halt exit code is entered whenever a transition is desired from a halted state to the running state. The halt exit code performs a restoration of the saved context prior to the transition. Section 9.3 describes the halt codes.

The ROM-based diagnostics consist of functional component diagnostics invoked by a diagnostic executive at power-up or by the TEST command from the console. These functions are described in Sections 9.4 (power-up) and 9.9 on diagnostics.

Depending on the nature of the halt and the hardware context, the firmware attempts either an operating system restart (Section 9.6), a bootstrap operation (Section 9.5), or transitions to console I/O mode (Section 9.7).

9.3 Halt Entry, Exit, and Dispatch

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.

9.3.1 Halt Entry—Saving Processor State

The entry code, at physical address 20040000, is executed whenever a halt occurs. The processor may halt for a variety of reasons. Table J–1 provides a complete list of the halt reasons and the associated messages.

PR$_SAVPSL<13:8>(restart_code), IPR 43 stores the reason for the halt. PR$_SAVPC, IPR 42, contains the value of the PC when the processor was halted. On a powerup, PR$_SAVPC is undefined.

One of the first actions of the firmware after a halt is to save the current LED code. Then the firmware writes an "E" to the diagnostic LEDs. This action occurs within the first several instructions after entering the firmware. The purpose of this action is to let the user know that at least some instructions have been successfully executed.

The KA670 firmware unconditionally saves 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 firmware uses the timers, so the previous time context is lost.

Several registers are unconditionally set to predetermined values by the firmware on any halt, processor initialization, or bootstrap. This action ensures that the firmware can run and protects the board from physical damage.

The following registers fall into this category:

- SSCCR, the SSC configuration register
- ADxMCH & ADxMSK, the SSC address match and mask registers
- CBTCR, the CDAL bus timeout control register
- TIVRx, the SSC timer interrupt vector registers

On every halt entry, the firmware sets the console serial line baud rate based on the value read from the BDR register.

9.3.2 Halt Dispatch

The action that the firmware takes on a halt depends primarily on the following information:

- The Break enable switch, BDR<7>(halt_enable).
- 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 the KA670 recognizes a break condition from the console serial line. The switch also determines the default action taken on a power-up or other internal halt condition. If breaks are enabled, the firmware enters the console by default. If breaks are disabled, the firmware attempts a recovery operation.

Operating systems can use the console program mailbox, CPMBX<1:0>(halt_action) (Figure H–2) to override the Break enable switch setting and instruct the firmware to enter the console service, attempt to restart the operating system, or reboot the system following a halt.

The user can also specify a default halt action with the SET HALT console command ((Section 9.8), in case the operating system or user application does not set the console program mailbox. This command allows users to specify autobooting on power-ups, even when breaks are enabled. For HALT instructions and error halt conditions, the SET HALT command is similar in function to the console program mailbox; however, the command has lower precedence and is only used when the console program mailbox is 0.

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 of dispatching collapses, in three categories:

02: External halts
03: Reset/power-up
xx: HALT instruction and all error halts

Table 9–1 summarizes the action taken on all halt conditions except external halts, which are described in Section 9.3.2.1. The actual halt dispatch state machine is described in detail in Section I.1.

Table 9–1 Halt Action Summary

Reset/ Power-Up or Halt	Break Enable Switch	User-Defined Halt Action	Operating System Mailbox Halt Action	Action(s)
T	1	0,1,3	x	Diagnostics, console.

T = a reset or power-up condition.
F = a HALT instruction or error halt condition.
x = don't care.

Table 9–1 (Cont.) Halt Action Summary

Reset/ Power-Up or Halt	Break Enable Switch	User-Defined Halt Action	Operating System Mailbox Halt Action	Action(s)
T	1	2,4	x	Diagnostics. If successful, boot. If either fails, console.
T	0	x	x	Diagnostics. If successful, boot. If either fails, console.
F	1	0	0	Console.
F	0	0	0	Restart. If this fails, boot. If boot fails, console.
F	x	1	0	Restart. If restart fails, console.
F	x	2	0	Boot. If boot fails, console.
F	x	3	0	Console.
F	x	4	0	Restart. If restart fails, boot. If boot fails, console.
F	x	x	1	Restart. If restart fails, console.
F	x	x	2	Boot. If boot fails, console.
F	x	x	3	Console.

T = a reset or power-up condition.
F = a HALT instruction or error halt condition.
x = don't care.

Because the KA670 does not support battery backed-up main memory, an operating system restart operation is not attempted on a power-up.

9.3.2.1 External Halts

The following conditions can trigger an external halt (PR$_SAVPSL<13:8>(restart_code) = 2). Different actions are taken, depending on the condition.

- A break condition on the system console serial line, if the Break enable switch is set to enabled (BDR<7>(halt_enable) = 1). As a result, the console is entered.

  NOTE
  You can use the S SET CONTROLP ENABLE console command to establish Ctrl P as the break condition.

- The assertion of the BHALT line on the Q22-bus, if the SCR<14>(BHALT_ENABLE) bit in the CQBIC is set. As a result, the console is entered.
- Negation of DCOK on the Q22-bus, if the SCR<7>(DCOK_ACTION) bit is set. (By default this bit is clear.) As a result, the console is entered.

- Recognition of a valid MOP BOOT message by an appropriately initialized SGEC, if the remote_boot_enable jumper is in place (BDR<31>(remote_boot_enable) = 1). As a result, a bootstrap is attempted. If the bootstrap 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.

Restart Button

Pushing the Restart button typically initiates a power-up sequence and destroys system state. The Restart button negates DCOK. The negation of DCOK may also be asserted by the DEQNA sanity timer, or any other Q22-bus module that chooses to implement the Q22-bus restart/reboot protocol. Since the SCR<7>(DCOK_ACTION) bit is cleared on power-up, the default action after deasserting DCOK is to generate a processor restart.

9.3.3 Halt Exit—Restoring the Processor State

When the firmware exits, it uses the saved context currently defined. This context is initially determined by what was saved on entry to the firmware. The context may be modified by console commands or automatic operations, such as an automatic bootstrap on power-up.

When restoring the context, the firmware flushes the CPU internal cache (if enabled) and invalidates all translation buffer entries by 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 an REI from that stack. This action implies that the user's ISP is valid before the firmware can exit. This is done automatically on a bootstrap. However, it is suggested that the SP be set to a valid memory location before issuing the START or CONTINUE command. Also, the user should validate PR$_SCBB before executing a NEXT command, since the firmware uses the trace trap vector for this function. At power-up, the user ISP is set to 200_{16} and PR$_SCBB is undefined.

9.4 Power-Up

This section describes the sequence of events which occurs on power-up. On a power-up, the KA670 firmware performs a unique set of actions, including locating and identifying a console device, language query, and the diagnostic countdown. Certain actions depend on the state of the mode switch on the H3604-SA panel. The switch has three settings: Test, Query, and Normal.

9.4.1 Identifying the Console Device

The firmware tries to identify the type of console device present, so the device may be used to display further diagnostic progress. Normally, the console device is the device attached to the console serial line. In this case, the firmware send outs the device attributes escape sequence <ESC>[c on the console serial line to determine 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 hardcopy devices.

After a console device has been identified, the firmware displays the KA670 banner message, similar to the following:

```
KA670-A V3.0, VMB 2.11
```

The banner message contains the processor name, the version of the firmware, and the version of VMB. The letter code in the firmware version indicates whether the firmware is pre-field test (X), field test (T), or an official release (V). The first digit indicates the major release number, and the trailing digit indicates the minor release number.

Next, 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 (Figure 9–2).

After the language query, the firmware invokes the ROM-based diagnostics and eventually displays the console prompt.

9.4.1.1 Mode Switch Set to Test

If the mode switch is set to Test, the console serial line external loopback test is executed. The purpose of this test is to verify that the console serial line connections from the KA670 through the H3604-SA panel are intact.

NOTE

An external loopback connector should be inserted in the serial line connector on the H3604-SA panel before cycling power to invoke this test.

During this test, the firmware toggles between two states—active and passive. Each state lasts a few seconds and displays a different number on the LEDs.

During the active state (about 3 seconds), 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), the LEDs are set to 3.

If at any time the firmware detects an error (parity, framing, overflow, or no characters), the firmware hangs and displays a 6 on the LEDs.

9.4.1.2 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 the language used for displaying critical system messages.

Figure 9–2 shows the language selection menu.

The user may select from one of the 11 supported languages. For those languages that do not have a unique keyboard, the menu displays supported country-specific keyboard variants in parentheses. If no response is received within 30 seconds, the language defaults to English (United States/Canada).

NOTE

The language query occurs only if the console device supports the DEC Multinational Character Set. Devices that do not support the character set (such as the VT100 terminal), default to English (United States/Canada).

After this inquiry, the firmware proceeds as if the mode switch were set to Normal.

```
 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 9–2 Language Selection Menu

9.4.1.3 Mode Switch Set to Normal

If the mode switch is set to Normal, then the next step in the power-up sequence is to execute the bulk of ROM-based diagnostics. In addition to message text, the console displays a countdown to indicate diagnostic test progress. Figure 9–3 shows a successful diagnostic countdown.

```
Performing normal system tests.
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 9–3 Normal Diagnostic Countdown

In the case of diagnostic failures, a diagnostic register dump is performed, similar to the example in Figure 9–4. The remaining diagnostics execute, and the countdown continues. For a detailed description of the register dump, see Section 9.9.

```
Performing normal system tests.
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..

?5F 2 0E FF 0000 0000 02       ; SUBTEST_5F_0E, DE_SGEC.LIS

P1=00000000  P2=00000000  P3=5839FF00  P4=00000000  P5=00000000
P6=00000000  P7=00000000  P8=00000000  P9=0000080A P10=00000003
r0=00000054  r1=20084019  r2=00004206  r3=00000000  r4=00000000
r5=1FFFFFFC  r6=C0000003  r7=20008000  r8=00004000 EPC=00000000
06..05..04..03..
Normal operation not possible.
```

Figure 9–4 Abnormal Diagnostic Countdown

If the diagnostics have successfully completed and halts are enabled, the firmware displays the console prompt and enters console I/O mode.

```
>>>
```

If the diagnostics have successfully completed and halts are disabled, the firmware tries to boot an operating system (Figure 9–5).

```
Loading system software.
No default boot device has been specified.
Devices:
-DIA0 (RF30)
-DIB1 (RF30)
-MUA0 (TK70)
-EZA0 (08-00-2B-03-82-78)
Device? [EZA0]:

(BOOT/R5:0 EZA0)

  2..
-EZA0
```

Figure 9–5 Console Boot Display With No Default Boot Device

9.4.2 LED Codes

In addition to the console diagnostic countdown, the diagnostic LEDs on the KA670 module and the H3604 console module panel display a hexadecimal value. The purpose of the LED display is to improve fault isolation when there is no console terminal, or when the hardware cannot communicate with the console terminal. Table 9–2 lists all

LED codes and the associated actions performed at power-up. The LED code is changed before the corresponding test or action is performed.

Table 9–2 LED Codes

LED Display	Actions
F	Initial state on power-up, no code has executed.
E	Entered ROM, some instructions have executed.
D	Waiting for power to stabilize (POK).
C	SSC RAM, SSC registers, and ROM checksum tests.
B	Primary cache, interval timer, and virtual mode tests.
A	FPA tests.
9	Backup cache, primary cache, and memory tests.
8	G-chip, memory, and I/O interaction tests.
7	CQBIC (Q22-bus) tests.
6	Console loopback tests.
5	SHAC DSSI subsystem tests.
4	SGEC Ethernet subsystem tests.
3	Console I/O mode.
2	Control passed to the VMB.
1	Control passed to the secondary bootstrap.
0	Program I/O mode, control passed to the operating system.

9.5 Operating System Bootstrap

Bootstrapping is the process of loading and transferring control to an operating system. The KA670 supports bootstraping of the following operating systems: VAX/VMS and VAXELN. The KA670 will also boot MDM diagnostics and any user application image that conforms to the boot formats described in this manual.

On the KA670, a bootstrap occurs when (1) a BOOT command is issued at the console, or (2) when the processor halts and the conditions specified in the Table 9–1 for automatic bootstrap are satisfied.

9.5.1 Preparing for the Bootstrap

Before dispatching to the primary bootstrap (VMB), the firmware initializes the system to a known state. The initialization sequence is as follows:

1. Check CPMBX<2>(BIP). If the bit is set, the 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 the 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 command.
   d. 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 command.
   f. Set IPCR<5>(LMEAE).
7. Validate the PFN bitmap. If invalid, rebuild it.
8. Search for a 128-kilobyte contiguous block of good memory, as defined by the PFN bitmap. If a block cannot be found, the bootstrap fails.
9. Initialize the general-purpose registers:

   R0 = address of descriptor of the boot device name, or 0 if no device is specified.
   R2 = length of PFN bitmap in bytes.
   R3 = address of PFN bitmap.
   R4 = time of day from PR$_TODR at power-up.
   R5 = boot flags.
   R10 = halt PC value.
   R11 = halt PSL value (without halt code and map enable).
   AP = halt code.
   SP = base of the 128-kilobyte good memory block + 512.
   PC = base of the 128-kilobyte 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-kilobyte 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. Also, local memory is partitioned as shown in Figure 9–6.

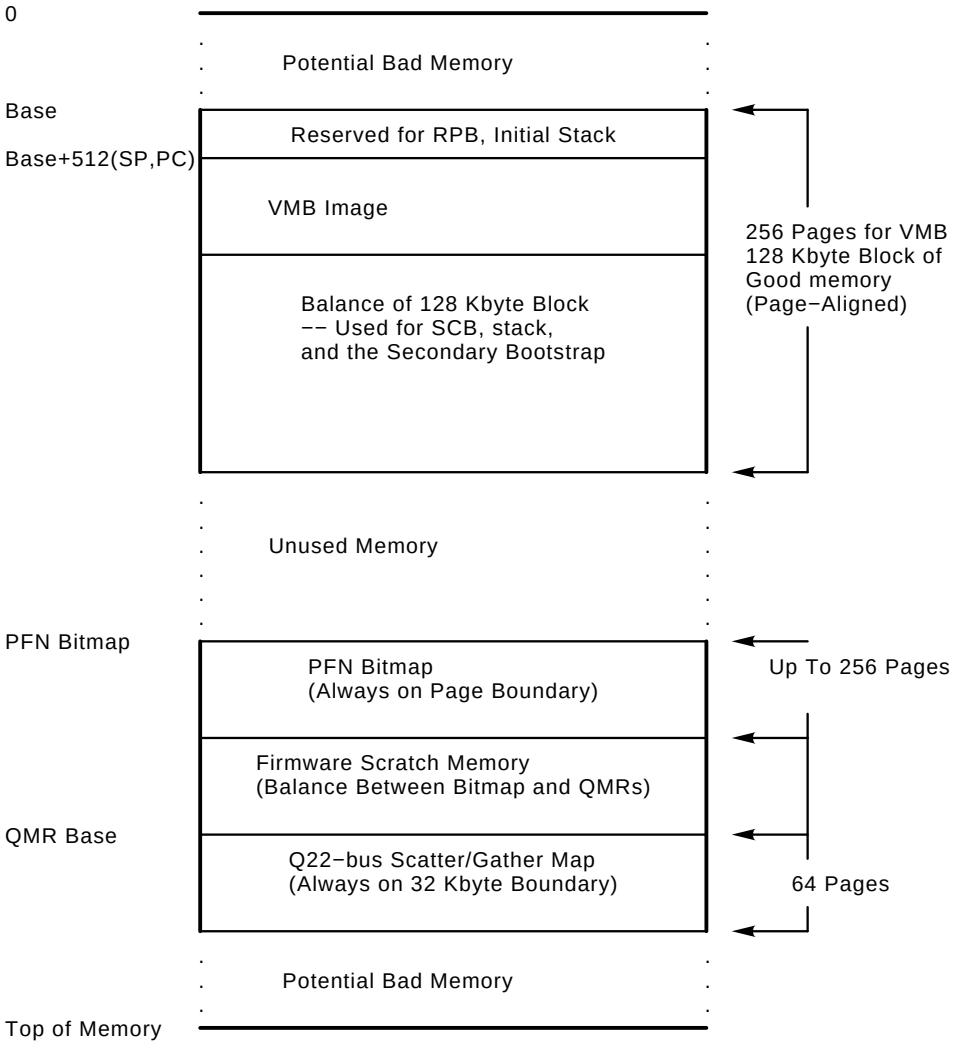


Figure 9–6 Memory Layout Before VMB Entry

9.5.1.1 Boot Devices

The KA670 firmware passes the address of a descriptor of the boot device name to VMB through R0. The device name used for the bootstrap operation is any 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 with 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. For longer strings the console prints an error message. Otherwise, the console makes no attempt at interpreting or validating the device name. The console converts the string to all uppercase and passes to VMB the address of a string descriptor for the device name in R0.

Table 9–3 lists supported devices and their corresponding boot device names used in BOOT commands.

Table 9–3 KA670 Supported Boot Devices

	Boot Name*	Controller Type	Device Type(s)
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	
	MUcn	TQK50 MSCP	TK50
		TQK70 MSCP	TK70
		KFQSA MSCP	
		KLESI	TU81E
Network:			
	EZA0	On-board Ethernet	—
	XQcn	DEQNA	—
		DELQA	—
		DESQA	—
PROM:			
	PRA0	MRV11	—
	PRB0	On-board EPROM	—

* Boot device names consist of at least a two-letter device code, followed by a single character controller letter (A...Z), and ending 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), endingin with a $.

NOTE

Table 9–3 presents a definitive list of boot devices that the KA670 supports. However, the KA670 will likely boot other devices that adhere to the MSCP standards.

9.5.1.2 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 9–7 shows the location of the boot flags used by VMB in the boot flag longword.

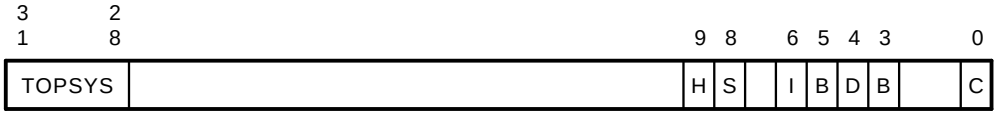


Figure 9–7 VMB Boot Flags (/R5:)

Field	Name	Description
0	RPB$V_ CONV	Conversational bootstrap.
3	RPB$V_ BBLOCK	Secondary bootstrap from bootblock. When this bit is set, VMB reads logical block number 0 of the boot device and tests it for conformance with the bootblock format. If in conformance, the block is executed to continue the bootstrap. No attempt to perform a Files-11 bootstrap is made.
4	RPB$V_DIAG	Diagnostic bootstrap. When this bit is set, the load image requested over the network is [SYS0.SYSMAINT]DIAGBOOT.EXE.
5	RPB$V_ BOOBPT	Bootstrap breakpoint. If this flag is set, a breakpoint instruction is executed in VMB and control is transferred to XDELTA prior to boot.
6	RPB$V_ HEADER	Image header. If this bit is set, VMB transfers control to the address specified by the file's image header. If this bit is not set, VMB transfers control to the first location of the load image.
8	RPB$V_ SOLICT	File name solicit. When this bit is set, VMB prompts the operator for the name of the application image file. A maximum of a 39 character file specification is permitted.
9	RPB$V_ HALT	Halt before transfer. When this bit is set, VMB halts before transffering control to the application image.
31:28	RPB$V_ TOPSYS	This field can be any value from 0 through F. This flag changes the top level directory name for the system disks with multiple operating systems. For example, if TOPSYS is 1, the top level directory name is [SYS1...]. **NOTE** **This does not apply to network bootstraps.**

9.5.2 Primary Bootstrap, Virtual Memory Boot

Virtual memory boot (VMB) is the primary bootstrap for booting VAX processors. On the KA670, VMB is resident in the firmware. VMB is copied into main memory before control is transferred to it. VMB then loads the secondary bootstrap image and transfers control to that image.

NOTE

In certain cases, such as VAXELN systems, 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 list summarizes VMB's operation:

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) (Table I–2).
4. Initialize the secondary bootstrap argument list (Table I–3).
5. If not a PROM boot, locate a minimum of three consecutive valid QMRs.
6. Write `2` to the diagnostic LEDs and display `2..` on the console, to indicate that VMB is searching for the device.
7. Optionally, solicit from the console a Bootfile: name.
8. On the console, write the name of the boot device from which VMB will attempt to boot. 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 as 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.

The following is a sample console display of a successful automatic bootstrap:

```
Loading system software.
(BOOT/R5:0 DUA0)

 2..
-DUA0
 1..0..
```

After a successful bootstrap operation, control is passed to the secondary bootstrap image, with the memory layout as shown in Figure 9–8.

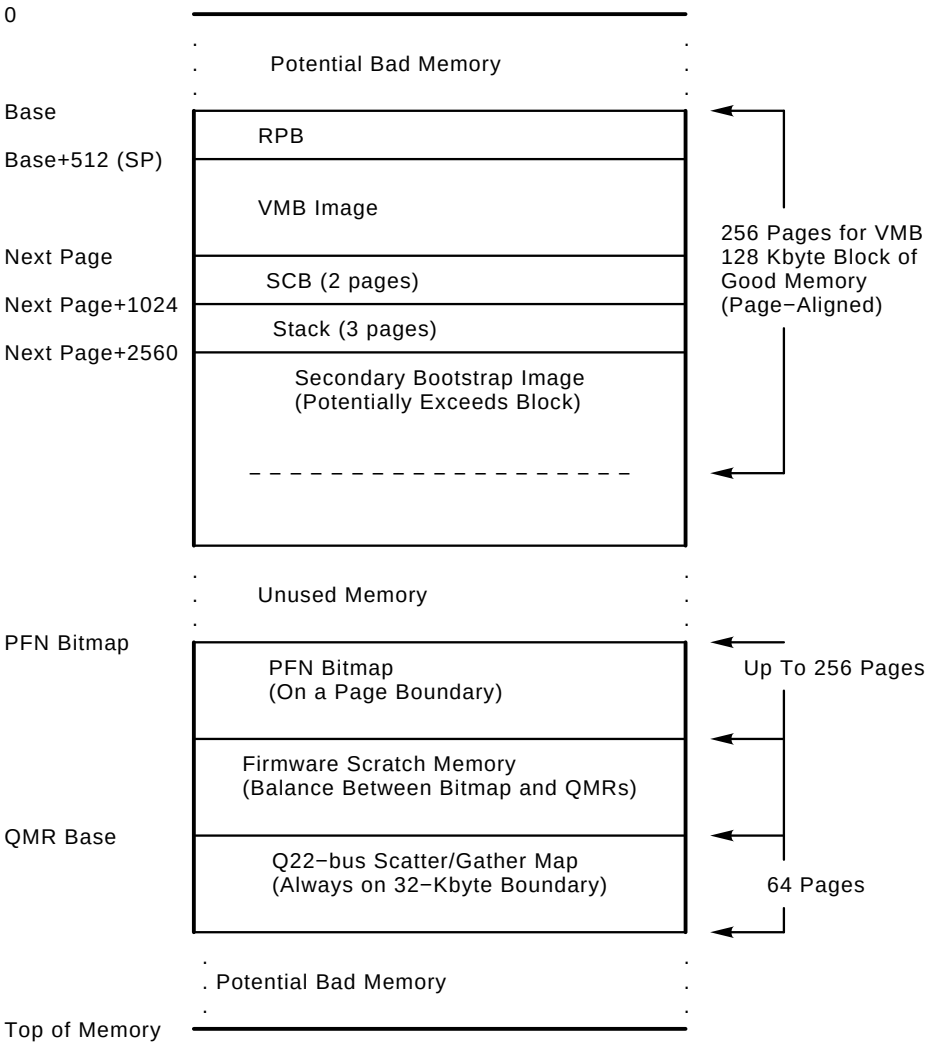


Figure 9–8 Memory Layout at VMB Exit

In the event that an operating system has an extraordinarily large secondary bootstrap that overflows the 128 kilobytes 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.

9.5.3 Device-Dependent Bootstrap Procedures

The KA670 supports bootstrapping from a variety of boot devices. The following sections describe the various device-dependent boot procedures.

9.5.3.1 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 the PROM boot also). Digital's VMS and ELN operating systems use the Files-11 bootstrap procedure, while the ULTRIX-32 operating system 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, VMB 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. See Section 9.5.1.2. 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. Verify that the contents of the boot block conform to the boot block format (Figure 9–9).
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 9–9.

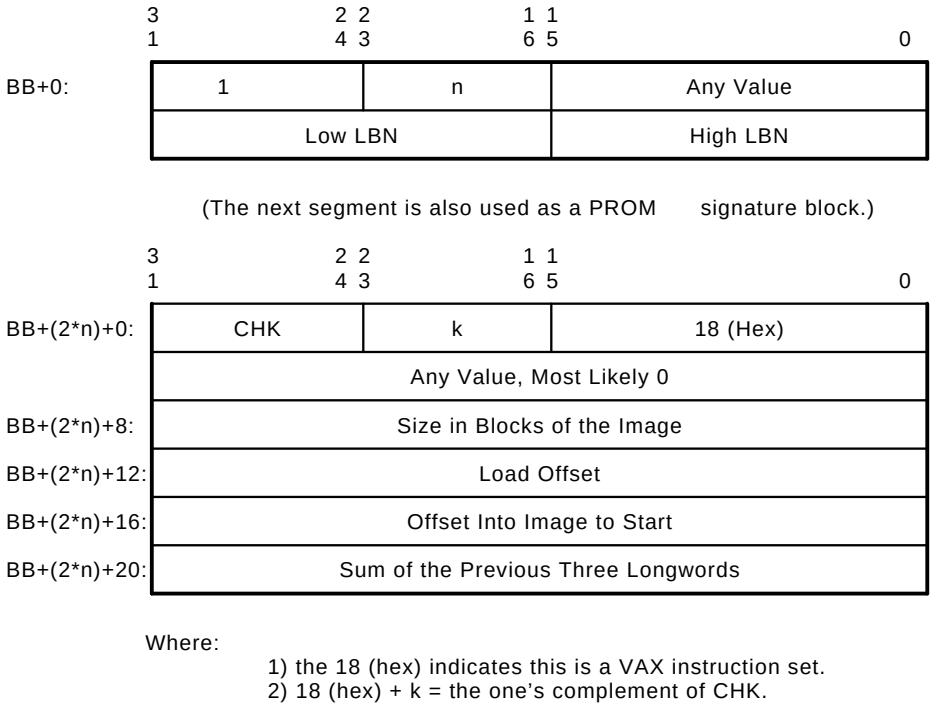


Figure 9–9 Boot Block Format

9.5.3.2 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 9–9. If PRA0 is the selected device, then VMB searches through Q22-bus memory on 16-kilobyte 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 is copied into main memory and VMB transfers control to that image at the offset specified in the PROM boot block. If not, the next page is tested.

NOTE

The boot image does not have to reside in PROM. Any boot image in Q22-bus memory space with a valid signature block on a 16-kilobyte boundary is a candidate. Indeed, the 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-kilobyte block are verified to make sure they are marked good in the PFN bitmap. The PROM must be copied contiguously; if all required pages cannot fit into the memory immediately following the VMB image, the boot fails.

9.5.3.3 Network Bootstrap Procedure

Whenever a network bootstrap is selected on a KA670, VMB makes continuous attempts to boot from the network. VMB uses the DNA maintenance operations protocol (MOP) as the transport protocol for network bootstraps and other network operations. (Refer to Appendix F for a complete description of supported MOP functions during bootstrap.) After a network boot is invoked, VMB turns on the designated network link and repeats load attempts until one of the following occurs:

- A successful boot.
- Fatal controller error.
- VMB is halted from the operator console.

The KA670 supports the loading of a standard operating system, a diagnostic image, or a user-designated program, using network bootstraps. The default image is the standard operating system. However, a user may select an alternate image by setting either the RPB$V_DIAG bit or the boot flag longword R5 in the RPB$V_SOLICT bit. Note that the RPB$V_SOLICT bit has precedence over the RPB$V_DIAG bit. If both bits are set, then the solicited file is requested. (Refer to Figure 9–7 for the use of these bits.)

NOTE

VMB accepts a maximum of a 39-character file specification for solicited boots. If the network server is running VMS, the following defaults apply to the file specification:

- **The directory is MOM$LOAD:**
- **The file extension is .SYS**

When the defaults are used, the 39-character file specification only needs to specify the filename.

The KA670 VMB uses the MOP program load sequence for bootstrapping the module, and the MOP *dump/load* protocol type for message exchanges related to loads. Table F–1 and Table F–2 list the MOP message types used in the exchange.

VMB, the requester, starts by sending a REQ_PROGRAM message in the appropriate envelope (Table F–3) to the MOP dump/load multicast address (Table F–4). 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 acknowledgement that 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 serves 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 acknowledgement to each exchange.

The server resends a packet with a specific load number, until the server sees a request with the load number incremented. The final acknowledgement is sent by the requester, wit a load number equivalent to the load number of the appropriate LOAD_w_XFER message + 1.

During the boot sequence, a response must be made to the REQ_PROGRAM message within the current timeout limit. If not, the timeout limit is increased by 4 seconds, up to a maximum of about 4 minutes. The initial timeout limit is 8 seconds.

9.6 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 process is often called *restart* or *warmstart*, but should not be confused with a processor restart that results in firmware entry.

On the KA670, a restart occurs if the conditions specified in Table 9–1 are satisfied.

To restart a halted operating system, the firmware searches system memory for the restart parameter block (RPB), a data structure constructed by VMB for this purpose. See Table I–2 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, the 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, the restart fails.

5. Check the operating system RPB$L_RSTRTFLG<0>(RIP) flag. If it is set, the restart fails.
6. Write `0` on the diagnostic LEDs.
7. Dispatch to the restart address, RPB$L_RESTART, with the following:

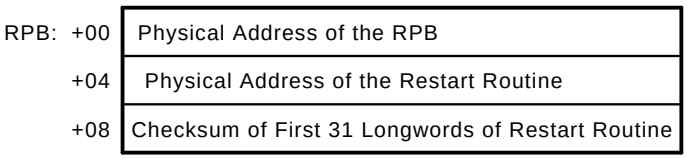
   SP = the physical address of the RPB + 512.
   AP = the halt code.
   PSL = 041F0000.
   PR$_MAPEN = 0.

If the restart is successful, the operating system must clear CPMBX<3>(RIP).

If the restart fails, the firmware prints `Restart failure.` on the system console.

9.6.1 Locating the Restart Parameter Block

The RPB is a page-aligned control block that can be identified by the first three longwords. The following diagram shows the format of the RPB signature. See Table I–2 for a complete description of the RPB.



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 an invalid physical address or 0, return to step 1. The check for 0 is necessary to ensure that a page of 0s does not pass the test for a valid RPB.
3. Calculate the 32-bit two's 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.

9.7 Console Service

The KA670 is, by definition, halted whenever the console program is running and the >>> prompt is displayed on the console terminal. When halted, the firmware provides most of the services of a standard VAX console (*VAX Architecture Reference Manual*) through the designated system console device. The firmware also implements several commands not defined in the *VAX Architecture Reference Manual*.

9.7.1 Console Control Characters

Control characters are typed by holding down the Ctrl key and pressing the second key.

In console I/O mode, several typed characters have special meanings.

Return	Ends a command line. No action is taken on a command until after it is terminated by a carriage return. A null line terminated by a carriage return is treated as a valid, null command. No action is taken, and the console reprompts for input. The carriage return is echoed as carriage return, line feed.
<X (Rubout)	When the operator types a rubout character, the console deletes the previously typed character. What appears on the console terminal depends on whether the terminal is a video terminal or a hardcopy terminal. • For hard copy terminals, the console echoes with a backslash (
Ctrl A or F14	Toggles between insertion/overstrike mode for command line editing. By default, the console powers up in overstrike mode.
Ctrl B or the up arrow (or down arrow)	Recall previous command(s). Command recall works only if sufficient memory is available. This function may then be enabled and disabled using the SET RECALL command.
Ctrl C	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 or left arrow	Moves the cursor left one position.
Ctrl E	Moves the cursor to the end of the line.
Ctrl F or right arrow	Moves the cursor right one position.
Ctrl H or backspace key F12	Moves cursor to the beginning of the line.
Ctrl O	Causes the console to throw away transmissions to the console terminal until the next Ctrl O is entered. Ctrl O is echoed as ^O<CR> when it disables output, but is not echoed when it reenables output. Output is reenabled if the console prints an error message, or 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. Ctrl S also enables output.
Ctrl Q	Resumes output to the console terminal. Additional Ctrl Q sequences are ignored. Ctrl S and Ctrl Q are not echoed.
Ctrl S	Stops output to the console terminal until Ctrl Q is typed. Ctrl S and Ctrl Q are not echoed.
Ctrl U	The console echoes ^U<CR>, and deletes the entire line. If Ctrl U is typed on an empty line, the console echoes ^U<CR> and prompts for another command.
Ctrl R	Causes the console to echo <CR><LF> followed by the current command line. This function can 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 — If the console is in console I/O mode, typing Break is equivalent to typing 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 with an ASCII code less than 32_{10} or between 128 and 159_{10} are unrecognized. If an unrecognized code is typed. it is echoed as a caret (^) followed by the character with ASCII code 64 greater. For example, BEL (ASCII code 7) is echoed as ^G, since capital G is ASCII code 71 (7 + 64 = 71).

When a control character is deleted with rubout, 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 responds with an error message.

9.7.2 Console Command Syntax

The console accepts commands that are up to 80 characters in length. It responds to longer commands with an error message. The count does not include rubbed-out characters or the carriage returna at the end of a command.

You can abbreviate commands. 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 non-whitespace characters that starts with a slash (/, ASCII code 47_{10}).

All numbers (addresses, data, and counts) are in hexadecimal. However, note that symbolic register names number the registers in decimal. The console does not distinguish between upper and lowercase characters in numbers or in commands; both are accepted.

9.7.3 Console Command Keywords

The KA670 firmware implements a variant of the VAX SRM console command set. The only commands defined in the VAX SRM and not supported by the KA670 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 KA670 console is similar to other VAX consoles.

Table 9–4 lists command and qualifier keywords.

Table 9–4 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
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

An asterisk (*) marks the minimal number of characters required to uniquely identify the keyword.

Table 9–6 at the end of the command descriptions provides a complete summary of the console commands.

9.7.4 Console Command Qualifiers

All qualifers 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 9–7 at the end of the command section provides a summary of the qualifers recognized by the KA670 console.

9.7.4.1 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 six address spaces:

Physical memory (/P qualifier)
Virtual memory (/V qualifier)
General-purpose registers (/G qualifier)
Internal processor registers (/I qualifier)
Protected memory (/U qualifier)
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 KA670 console supports symbolic references to addresses. A symbolic reference simultaneously defines the address space for a given symbol. Table 9–5 lists the symbolic addresses supported by the console, grouped according to address space.

Table 9–5 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

All symbolic values in this table are in hexadecimal.

Table 9–5 (Cont.) Console Symbolic Addresses

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
/I - Internal Processor Registers							
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$_sirr	14	pr$_ tbdr	24	—	34
—	05	pr$_sisr	15	pr$_ cadr	25	—	35
—	06	—	16	pr$_ mcesr	26	—	36
—	07	—	17	pr$_ mser	27	pr$_ ioreset	37
pr$_ p0br	08	pr$_iccr	18	pr$_accs	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	pr$_ tbdata	3B
pr$_sbr	0C	—	1C	—	2C	—	3C
pr$_slr	0D	—	1D	—	2D	—	3D
—	0E	—	1E	—	2E	pr$_sid	3E
—	0F	—	1F	pr$_ tbtag	2F	pr$_ tbchk	3F
—	70	pr$_brfr	74	pr$_ bcerr	78	pr$_ pctag	7C
pr$_ bcbts	71	pr$_ bcidx	75	pr$_ bcfbts	79	pr$_ pcidx	7D
pr$_ bcp1ts	72	pr$_ bcsts	76	pr$_ bcfpts	7A	pr$_ pcerr	7E
pr$_ bcp2ts	73	pr$_ bcctl	77	pr$_ vinstr	7B	pr$_ pcsts	7F
/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

Table 9–5 (Cont.) Console Symbolic Addresses

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
/P - Physical (VAX I/O Space)							
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	memcsr18	20080148	memcsr19	2008014c
memcsr20	20080150	memcsr21	20080154	memcsr22	20080158	memcsr23	2008015c
memcsr24	20080160	memcsr25	20080164	memcsr26	20080168	memcsr27	2008016c
memcsr28	20080170	memcsr29	20080174	memcsr30	20080178	memcsr31	2008017c
memcsr32	20080180	memcsr33	20080184	memcsr34	20080188	memcsr35	2008018c
memcsr36	20080190						
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
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
shac1_ sswcr	20004030	shac1_ sshma	20004044	shac1_ pqbbr	20004048	shac1_ psr	2000404c
shac1_ pesr	20004050	shac1_ pfar	20004054	shac1_ ppr	20004058	shac1_ pmcsr	2000405C
shac1_ pcq0cr	20004080	shac1_ pcq1cr	20004084	shac1_ pcq2cr	20004088	shac1_ pcq3cr	2000408C

Table 9–5 (Cont.) Console Symbolic Addresses

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
/P - Physical (VAX I/O Space)							
shac1_ pdfqcr	20004090	shac1_ pmfqcr	20004094	shac1_ psrcr	20004098	shac1_ pecr	2000409C
shac1_ pdcr	200040A0	shac1_ picr	200040A4	shac1_ pmtcr	200040A8	shac1_ pmtecr	200040AC
shac2_ sswcr	20004230	shac2_ sshma	20004244	shac2_ pqbbr	20004248	shac2_ psr	2000424c
shac2_ pesr	20004250	shac2_ pfar	20004254	shac2_ ppr	20004258	shac2_ pmcsr	2000425C
shac2_ pcq0cr	20004280	shac2_ pcq1cr	20004284	shac2_ pcq2cr	20004288	shac2_ pcq3cr	2000428C
shac2_ pdfqcr	20004290	shac2_ pmfqcr	20004294	shac2_ psrcr	20004298	shac2_ pecr	2000429C
shac2_ pdcr	200042A0	shac2_ picr	200042A4	shac2_ pmtcr	200042A8	shac2_ pmtecr	200042AC
shac_ sswcr	20004230	shac_ sshma	20004244	shac_ pqbbr	20004248	shac_ psr	2000424c
shac_ pesr	20004250	shac_ pfar	20004254	shac_ ppr	20004258	shac_ pmcsr	2000425C
shac_ pcq0cr	20004280	shac_ pcq1cr	20004284	shac_ pcq2cr	20004288	shac_ pcq3cr	2000428C
shac_ pdfqcr	20004290	shac_ pmfqcr	20004294	shac_ psrcr	20004298	shac_ pecr	2000429C
shac_ pdcr	200042A0	shac_ picr	200042A4	shac_ pmtcr	200042A8	shac_ pmtecr	200042AC

Any Address Space

* (asterisk)	The last location successfully referenced in an EXAMINE or DEPOSIT command.
+ (plus sign)	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.
- (hyphen)	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.

9.7.5 References to Processor Registers and Memory

The KA670 console is implemented by macrocode executing from EPROM. The console command interpreter cannot modify actual processor registers. When the console is entered, the console saves the processor registers in console memory. All command references to the processor registers are directed to the corresponding saved values, not to the registers themselves.

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

The following registers are saved by the console. Any direct reference to these registers is intercepted by the console, directing access to the saved copies.

R0...R15	General-purpose registers
PR$_IPL	Interrupt priority level register
PR$_SCBB	System control block base register
PR$_ISP	Interrupt stack pointer
PR$MAPEN	Memory management enable register

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

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

9.8 Console Commands

The following sections define the commands accepted by the console, when it is in console I/O mode.

Syntax Conventions

The following conventions are used to describe command syntax:

[ ]	Enclose optional command elements.
{ }	Enclose a command element.
...	Indicates a series of command elements.

The console allows you to override the default radix by using the following commands:

%d	Decimal (For example, %d1234)
%x	Hexadecimal (For eample, %xFEEBFCEA)
%b	Binary (For example, %b1001)
%o	Octal (For example, %o1070)

The following is an example of 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}
The boot flags value is a 32-bit hexadecimal value passed to VMB in R5. The console does not interpret this value. Figure 9–7 lists the bit assignments of R5. To specify a default boot flags longword, use the SET BFLAG command. To display the default setting, use 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 the console to issue a `VAL TOO BIG` error message. Otherwise, the console makes no attempt at interpreting or validating the device name. The console converts the string to uppercase and passes VMB a string descriptor in R0 to this device name.

To specify a default boot device, use the SET BOOT command. To display the name of the default device, use the SHOW BOOT command. The factory default is the Ethernet device, EZA0.

Description

The console initializes the processor and transfers execution to VMB. VMB tries to boot the operating system from the device specified by the BOOT command. If no device is specified, VMB tries to boot from the default device. The console qualifies the bootstrap operation by passing a *boot flags* value to VMB in R5. See Section 9.5 for a detailed description of the bootstrap process and how the default bootstrap device is determined.

If you do not specify a device name or qualifiers with the BOOT command, the default values are used. Explicitly stating the boot flags value or the boot device overrides the current default value for the current boot request, but does not change the default value stored in battery backed-up RAM (BBU RAM).

There are three ways to set the default boot device and and boot flags value:

- The operating system may write a default boot device and flags into the appropriate locations in BBU RAM (Appendix H).
- The user may explicitly set the default boot device and boot flags with the console SET BOOT and SET BFLAG commands.
- Under any of the following conditions, the console prompts the user for the default boot device
  - The power-up mode switch is set to query mode.

- The console detects that the battery failed, which means the contents of BBU RAM 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 (SGEC) or neither of the above two methods has been performed. Simply stated, the console prompts the user for a default boot device at every power-up, until such a request has been satisfied.

If no default boot device is specified in BBU RAM, the console issues a list of potential bootable devices at power-up and 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
```

CONFIGURE

Format

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

Qualifiers

None.

Arguments

None.

Description

The processor begins instruction execution at the address currently contained in the program counter. The processor is not initialized. The console enters program I/O mode. Internally, the CONTINUE command pushes the user's PC and PSL onto the user's ISP, then executes an REI instruction. This action implies that the user's ISP is pointing to some valid memory.

Examples

```
>>>CONTINUE
>>>
```

DEPOSIT

Format

The data size is a word.

/L
The data size is a longword.

/Q
The data size is a quadword.

/G
The address space is the general-purpose register set, R0 to R15. The data size is always long.

/I
The address space is the internal processor registers (IPRs). These are the registers accessible only by the MTPR and MFPR 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 occurs. 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 is not 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 the command 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 with a value of 3 always generate a double-bit error).

Arguments

{address}
A long word address that specifies the first location into which data is deposited. The address can be any legal address specifier as defined in Section 9.7.4.1 and Table 9–5.

{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 0s.

[{data}]
Additional data to be deposited (up to a maximum of six values).

Description

This command deposits the data into the address specified. If you do not specify an address space or data size qualifiers, the defaults are the last address space and data size used in a DEPOSIT, EXAMINE, MOVE or SEARCH command. After processor initialization, the default address space is physical memory, the default data size is a longword, and the default address is 0. 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 a byte.

/W
The data size is a word.

/L
The data size is a longword.

/Q
The data size is a quadword.

/G
The address space is the general-purpose register set, R0 to R15. The data size is always long.

/I
The address space is the internal processor registers (IPRs). These are the registers accessible only by the MTPR and MFPR 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, so it 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 the command 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. If specified, the ECC bits actually read are displayed in parentheses following the data. 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 9.7.4.1 and Table 9–5. If no address is specified, "+" is assumed.

Description

This command examines the contents of the memory location or register specified by the address. If no address is specified, "+" is assumed. The display line consists of a single-character address specifier, the hexadecimal physical address to be examined, and the examined data also in hexadecimal.

EXAMINE uses the same qualifiers as DEPOSIT. However, the /WRONG qualifier causes EXAMINE to ignore ECC errors on reads from physical memory. EXAMINE also supports an /INSTRUCTION qualifier that will disassemble 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 MFPR    S^#2A,B^44(R1)
>>>E/INS                                               ! Look at next instruction.
  P 20040048   DB MFPR    S^#2B,B^48(R1)
>>>
>>>
>>> E 0
  P 00000000
>>> E *
  P 00000000
```

FIND

Format

FIND ***[qualifier-list]***

Qualifiers

/MEMORY

Search memory for a page-aligned block of good memory, 128 kilobytes in length. The search looks only at memory that is deemed usable by the bitmap. This command leaves the contents of memory unchanged.

/RPB

Search all of physical memory for a restart parameter block. The search does not use the bitmap to qualify which pages are looked at. The command leaves the contents of memory unchanged.

Arguments

None.

Description

The FIND command has the console search main memory, starting at address 0, for either a page-aligned, 128-kilobyte 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

Qualifiers

None.

Arguments

None.

Description

The HALT command has no effect. It is included for compatability with other consoles.

Examples

```
>>>HALT                                    ! Pretend to halt.
>>>
```

HELP

Format

Qualifiers

None.

Arguments

None.

Description

The HELP command helps 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 that must be filled in
                   with a syntactically correct value

Valid qualifiers:
    /B /W /L /Q /INSTRUCTION
    /G /I /V /P /M
    /STEP: /N: /NOT
    /WRONG /U

Valid commands:
    DEPOSIT [<qualifiers>] <address> [<datum> [<datum>]]
    EXAMINE [<qualifiers>] [<address>]
    MOVE [<qualifiers>] <address> <address>
    SEARCH [<qualifiers>] <address> <pattern> [<mask>]
    SET BFL(A)G <boot_flags>
    SET BOOT <boot_device>

    SET HALT <halt_action>
    SET RECALL 0|1

    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 <language_number>

    SHOW BFL(A)G
    SHOW BOOT
    SHOW DEVICE

    SHOW DSSI
```

```
HELP

    SHOW ETHERNET
    SHOW LANGUAGE
    SHOW MEMORY [/FULL]

    SHOW HALT

    SHOW RLV12
    SHOW QBUS
    SHOW UQSSP

    SHOW SCSI
    SHOW TRANSLATION <physical_address>

    SHOW VERSION
    HALT
    INITIALIZE
    UNJAM
    CONTINUE
    START <address>
    REPEAT <command>
    X <address> <count>
    FIND [/MEMORY | /RPB]
    TEST [<test_code> [<parameters>]]
    BOOT [/R5:<boot_flags> | /<boot_flags>] [<boot_device>[:]]
    NEXT [count]
    CONFIGURE
    HELP
>>>
```

INITIALIZE

Format

Qualifiers

None.

Arguments

None.

Description

The INITIALIZE command performs a processor initialization. The following registers are initialized, as specified in the *VAX Architecture Reference Manual*:

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.
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 KA670 firmware also performs the following initialization tasks:

Initializes the CDAL bus timer.
Initializes the address decode and match registers.
Initializes the programmable timer interrupt vectors.
Reads the BDR registers to determine the baud rate, then configures the SSCCR register accordingly.
Clears all error status bits.

Examples

```
>>>INIT
>>>
```

MOVE–MOVE_CMD

Format

MOVE ***[qualifier-list] {src_address} {dest_address}***

Qualifiers

/B
The data size is a byte.

/W
The data size is a word.

/L
The data size is a longword.

/Q
The data size is a 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 MOVE commands 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 that occur when accessing data in main memory are ignored. On writes, the ECC bits for this data are forced to the value of 3. ECC bits with a value of 3 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 9.7.4.1 and Table 9–5. If no address is specified, "+" is assumed.

Description

On a MOVE command, the console copies the block of memory that starts 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 the 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, a single step is assumed. The console enters spacebar step mode, as described in the *VAX Architecture Reference Manual*. In this mode, subsequent spacebar strokes initiate single steps. A carriage 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 command. Therefore, the following restrictions apply to the use of the NEXT command:

- If memory management is enabled, the NEXT command works only if the first page in SSC RAM is mapped somewhere in S0 (system) space.
- The NEXT command does not work where time-critical code is being executed.
- The NEXT command elevates the IPL to 31 for long periods of time (milliseconds) while single stepping over commands.
- Unpredictable results occur if the macro instruction being stepped over modifies the SCBB, or the trace trap entry. This means you cannot use the NEXT command with other debuggers. This also implies that the user should validate PR$_SCCB before using the NEXT command.

Examples

```
>>>DEP /L/P 1000 50D650D4                        ! Create a simple program.
>>>DEP /L/P 1004 125005D1
>>>DEP /L/P 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 SP 1000 1000                           ! ...and the stack pointer.
>>>
```

```
>>>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}
A valid console command other than REPEAT.

Description

In a REPEAT command, the console repeatedly displays and executes the specified command. To stop the repeating, type Ctrl C. You can specify any valid console command to repeat, 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 a byte.

/W
The data size is a word.

/L
The data size is a longword.

/Q
The data size is a 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. It 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 9.7.4.1 and Table 9–5. If no address is specified, "+" is assumed.

{pattern}
The target data.

[{mask}]
A longword containing the target bits to be masked out.

Description

The SEARCH command finds all occurrences of a pattern, and reports the addresses where the pattern was found. If you use the /NOT qualifier, the command reports all addresses where the pattern did not match.

The command accepts an optional mask to specify bits whose setting does not matter. For example, to ignore bit 0 in the comparison, specify a mask of 1. If the omit the mask argument, the mask defaults to 0.

Conceptually, a match condition occurs if the following condition is true:
(pattern AND NOT mask) EQUALS (data AND NOT mask)

pattern is the target data.
mask is the optional bit mask (which defaults to 0).
data is the data (byte, word, long, quad) at the current address.

The \NOT qualifier and match condition determine whether or not the SEARCH command reports the address:

/NOT Qualifier Used?	Match Condition	Report the Address?
No	True	Yes
No	False	No
Yes	True	No
Yes	False	Yes

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 occurances...
  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 non-zero...
  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 interpretted as...
>>>                                              ! ...a "spin" (10$: brb 10$).
>>>      ! Note, none found.
```

SET

Format

SET ***{parameter} {value}***

Parameters

BFL(A)G
Set the default R5 boot flags. The value must be a hexadecimal number of up to eight digits.

BOOT
Set the default boot device. The value must be a valid device name as specified in the BOOT command section.

CONTROLP
Sets Ctrl P as the console halt condition, instead of a break. Values of 1 or ENABLED set Ctrl P as the halt condition. Values of 0 or DISABLED set break as the halt condition. In either case, the setting of the break enable switch determines whether or not a halt will occur.

HALT
Sets the user-defined halt action. Acceptable values are 0 to 4 or the keywords DEFAULT, RESTART, REBOOT, HALT, and RESTART_REBOOT. Refer to Table 9–1.

HOST
Invoke the DUP or maintenance driver on the selected node. Only SET HOST /DUP accepts a value parameter.

/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 KA670 DUP driver only supports SEND DATA IMMEDIATE messages, and hence those devices which also support them.

/DSSI node —Select the DSSI node, by number, from 0 to 7.

/UQSSP —Select the Q22-bus device, using one of three methods:

/DISK n —Specify the disk controller number *n*, from 0 to 255. (The resulting fixed address for *n*=0 is 20001468. The floating rank for *n* > 0 is 26.)

/TAPE n —Specify the tape controller number *n*, from 0 to 255. (The resulting fixed address for *n*=0 is 20001940. 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 and 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, 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 the console language and keyboard type. If the current console terminal does not support the DEC Multinational Character Set (MCS), then this command has no effect and the console remains in English message mode. Acceptable values are 1 to 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

Sets the command recall state to either enabled (1) or disabled (0).

Qualifiers

—

Depends on the parameters used.

Arguments

None.

Description

The SET command sets the specified console parameter to the specified value.

Examples

```
>>>
>>>SET BFLAG 220
>>>
>>>SET BOOT DIA0

>>>
>>>SET HALT REBOOT
>>>
```

```
>>>SET HOST /DUP /DSSI 0
Starting DUP server...

DSSI Node 0 (SUSAN)
Copyright © 1988  Digital Equipment Corporation
DRVEXR V1.0  D  5-JUL-1988 15:33:06
DRVTST V1.0  D  5-JUL-1988 15:33:06
HISTRY V1.0  D  5-JUL-1988 15:33:06
ERASE  V1.0  D  5-JUL-1988 15:33:06
PARAMS V1.0  D  5-JUL-1988 15:33:06
DIRECT V1.0  D  5-JUL-1988 15:33:06
End of directory

Task Name? PARAMS
Copyright © 1988  Digital Equipment Corporation

PARAMS> STAT PATH

   ID  Path Block     Remote Node      DGS_S      DGS_R      MSGS_S     MSGS_R
   -- ------------  ---------------  ---------- ---------- ---------- -------
---
    0 PB  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 © 1988  Digital Equipment Corporation

PARAMS> SHOW NODE

Parameter     Current           Default        Type     Radix
--------- ---------------  ---------------  --------  -----
NODENAME          SUSAN             RF30        String  Ascii   B

PARAMS> SHOW ALLCLASS

Parameter     Current           Default        Type     Radix
--------- ---------------  ---------------  --------  -----
ALLCLASS                1                0      Byte    Dec   B

PARAMS> EXIT
Exiting...

Stopping DUP server...
>>>

>>>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}*)

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.

HALT
Show the user-defined halt action: DEFAULT, RESTART, REBOOT, HALT, or RESTART_REBOOT. Refer to Table 9–1 for usage.

DSSI
Show the status of all nodes that can be found on the DSSI bus. For each node found, 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 if a device is bootable.

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. If the node is not running, the console displays an MSCP server and no device information.

ETHERNET
Show the hardware Ethernet address for all Ethernet adapters found. If no Ethernet adapters are found the response is blank.

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 bitmap.

/FULL Show the normally inaccessible areas of memory, such as the PFN bitmap pages, the console scratch memory pages, and the Q22-bus scatter/gather map pages.

QBUS
Show all Q22-bus I/O addresses that respond to an aligned word read. For each address, the console displays the hexadecimal address in the VAX I/O space, the octal address as it would appear in the Q22-bus I/O space, and the hexadecimal word data that was read.

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, ENABLED or DISABLED.

RLV12
Show all RL01 and RL02 disks that appear on the Q22-bus.

SCSI
Show any SCSI devices in the system.

TRANSLATION
Show any virtual addresses that map to the specified physical address. The firmware uses the current values of the 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 on the Q22-bus that support the UQSSP protocol. For each disk or tape, the console displays the controller number, the controller CSR address, the boot name, and the type of each device connected to the controller. The command does not indicate if a device is bootable.

The device information is obtained from the media type field of the MSCP command GET UNIT STATUS. If the node is not running, the console displays an MSCP server with no device information.

VERSION
Show the current version of the firmware.

Qualifiers

–
Depends on the specific parameter.

Arguments

None.

Description

The SHOW command displays the setting of the specified console parameter.

Examples

```
>>>
>>>SHOW BFLAG
00000220
>>>
>>>SHOW BOOT
DIA0

>>>
>>>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)
-DUC4 (RF30)

UQSSP Disk Controller 3 (760344)
-DUD5 (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 Bitmap
-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



KA670-A V3.0, VMB 2.11

>>>
```

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 START command tells the console to start executing instructions at the specified address. If you do not specify an address, 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 command, followed by a CONTINUE command. START does not perform an INITIALIZE command.

Examples

```
>>>START  1000
```

TEST

Format

TEST ***[{test_number} [{test_arguments}]]***

Qualifiers

None.

Arguments

{test_number}
A two-digit hexadecimal number specifying the test to execute.

{test_arguments}
Up to five additional test arguments. The console accepts these arguments, but does not attach any meaning to them. For the interpretation of these arguments, refer to the test specification for each test.

Description

The TEST command tells the console to invoke a diagnostic test program specified by the test number. If you specify test number 0, the power-up script is executed. The console accepts an optional list of up to five additional hexadecimal arguments.

For a detailed explanation of the diagnostics, see Section 9.9.

Examples

```
>>>             ! Execute the power-up diagnostic script
>>>             ! Warning...this has the same affect as a power-up!
>>>
>>>TEST 0


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
_____________________________________________________________________________
    20051000  SCB
    20051F04  De_executive
30  2005A688  Memory_Init_Bitmap *** mark_Hard_SBEs ******
31  2005A4B0  Memory_Setup_CSRs  **********
32  20059F7C  G_Chip_registers   **********
33  20059EF0  G_Chip_powerup     **
34  200535C0  SSC_ROM            *
35  20060DAC  B_Cache_diag_mode  addr_incr wait_time_secs extended_test *******
36  20061B00  B_Cache_w_memory   addr_incr *********
37  20061EA8  P_B_Cache_w_memory addr_incr *********
38  2006219F  G_Chip_timeout     ******
3F  2005C360  Mem_FDM_Addr_shorts *** cont_on_err ******
40  2005CDC4  Memory_count_pages First_board Last_bd Soft_errs_allowed *******
41  2005CF9C  Board_Reset        *
42  2005367C  Chk_for_Interrupts *****
44  20061750  P_Cache_w_memory   addr_incr *********
45  20058A2C  cache_mem_cqbic    start_addr end_addr addr_incr *******
46  200605EC  P_Cache_diag_mode  addr_incr wait_time_secs extended_test *******
47  2005CB6C  Memory_Refresh     start_a end incr cont_on_err time_seconds *****
48  2005BF94  Memory_Addr_shorts start_add end_add * cont_on_err pat2 pat3 ****
49  2005BB38  Memory_FDM         *** cont_on_err ******
4A  2005B80C  Memory_ECC_SBEs    start_add end_add add_incr cont_on_err ******
4B  2005B514  Memory_Byte_Errors start_add end_add add_incr cont_on_err ******
4C  2005B0B0  Memory_ECC_Logic   start_add end_add add_incr cont_on_err ******
4D  2005AF28  Memory_Address     start_add end_add add_incr cont_on_err ******
4E  2005AD60  Memory_Byte        start_add end_add add_incr cont_on_err ******
4F  2005AAB0  Memory_Data        start_add end_add add_incr cont_on_err ******
51  20062419  FPA                *******
52  20053C1E  SSC_Prog_timers    which_timer wait_time_us ***
53  20053EE8  SSC_TOY_Clock      repeat_test_250ms_ea Tolerance ***
54  2005374E  Virtual_Mode       *********
55  20054097  Interval_Timer     *
56  2005EB6C  SHAC_LPBCK         *******
58  2005F378  SHAC_RESET         dssi_bus port_number time_secs
59  2005DCB8  SGEC_LPBCK_ASSIST  time_secs **
5A  20058930  R_G_Chip_RDAL      dont_report_memory_bad repeat_count *
5C  2005E220  SHAC               shac_number *******
5F  2005D06C  SGEC               loopback_type no_ram_tests ******
60  20058564  SSC_Console_SLU    start_BAUD end_BAUD ******
62  200544CC  console_QDSS       mark_not_present selftest_r0 selftest_r1 *****
63  20054648  QDSS_any           input_csr selftest_r0 selftest_r1 ******
80  20057EB0  CQBIC_memory       **********
81  20054133  Qbus_MSCP          IP_csr ******
82  200542F5  Qbus_DELQA         device_num_addr ****
83  20055326  KZQSA_LPBCK1       controller_number ********
84  200569C4  KZQSA_LPBCK2       controller_number *********
85  200547A0  KZQSA_memory       incr test_pattern controller_number *******
86  20054C4C  KZQSA_DMA          Controller_number main_mem_buf ********
87  20057BFC  KZQSA_EXTLPBCK     controller_number ****
90  20053B9F  CQBIC_registers    *
91  20053B38  CQBIC_powerup      **
99  2006262E  Flush_Ena_Caches   dis_flush_primary dis_flush_backup
9A  2005F908  INTERACTION        pass_count disable_device ****
9B  200622ED  Init_memory_8MB    *
9C  20058D78  List_CPU_registers *
9D  20059CD7  Utility            Expnd_err_msg get_mode init_LEDs clr_ps_cnt
9E  20054108  List_diagnostics   *
9F  2006270E  Create_A0_Script   **********
C1  20053271  SSC_RAM_Data       *
C2  20053444  SSC_RAM_Data_Addr  *
C5  20059DEE  SSC_registers      *
C6  200531B8  SSC_powerup        *********
```

```
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

>>>
>>>             ! Show the diagnostic state.
>>>
>>>T FE


 Bitmap=01FF2000, Length=00002000, Checksum=FFFF, Busmap=01FF8000
 Test_number=9E, Subtest=01, Loop_Subtest=00, Error_type=FF
 Error_vector=0000, Last_exception_PC=00000000, Severity=02
 Total_error_count=0000, Led_display=09, Console_display=9E, save_mchk_code=11
 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=0FFFFC00448E
 Return_stack=201406A8, Subtest_pc=20054125, Timeout=00030D40

>>>
>>>             ! Display the CPU registers.
>>>
>>>T 9C


  SBR=01FB8000    SLR=00002021  SAVPC=80000011 SAVPSL=80404174    SCBB=20051000
 P0BR=80000000   P0LR=00100A80   P1BR=7FC45400   P1LR=001FFD6F     SID=0B000003
 TODR=00060BF0   ICCS=00000000   ACCS=00000000  MAPEN=00000000   BDMTR=20084000
 TCR0=00000005   TIR0=1AA01F6F  TNIR0=00000000  TIVR0=00000078   BDMKR=0000007C
 TCR1=00000001   TIR1=1AA5E858  TNIR1=0000000F  TIVR1=0000007C     SCR=0000D000
 RXCS=00000000   RXDB=0000000D   TXCS=00000000   TXDB=00000030    DSER=00000000
PCSTS=0000080A  PCERR=0000B520  PCIDX=000007F8  PCTAG=40000000   QBEAR=0000000F
BCSTS=01800000  BCCTL=0000000E  BCERR=20059238  BCIDX=000007F0    DEAR=00000000
BCBTS=20000000 BCP1TS=20000000 BCP2TS=20000000  BCRFR=00000420   QBMBR=01FF8000
  BDR=3BFA08AF  DLEDR=0000000C  SSCCR=00D55570  CBTCR=00004000   IPCR0=0000
DSSI_1=04  (BUS_1)     PQBBR_1=03060022    PMCSR_1=00000000    SSHMA_1=00008A20
     PSR_1=00000000     PESR_1=00000000     PFAR_1=00000000      PFR_1=00000000
DSSI_2=05  (BUS_0)     PQBBR_2=03060022    PMCSR_2=00000000    SSHMA_2=0000CA20
     PSR_2=00000000     PESR_2=00000000     PFAR_2=00000000      PFR_2=00000000
NICSR0=1FFF0003   3=00004030   4=00004050   5=8039FF00   6=83E0F000  7=00000000
NICSR9=04E204E2  10=00030000  11=00000000  12=00000000  13=00000000 15=0000FFFF
NISA=08-00-2B-06-10-
42     RDES0=00441300   1=00000000   2=05EE0000  3=000046F0
                       TDES0=00008C80   1=07000000   2=00400000  3=000040FA
MEM_FRU 1      MCSR_0=80000002      1=80800002      2=81000002       3=81800002
MEM_FRU 2      MCSR_4=00000006      5=00000006      6=00000006       7=00000006
MEM_FRU 3      MCSR_8=00000006      9=00000006     10=00000006      11=00000006
MEM_FRU 4      MCSR12=00000006     13=00000006     14=00000006      15=00000006
RMESR=00440044  RMEAR=00000000 RIOEAR=00080188   CEAR=00000000   MCDSR=3E391700

>>>
```

UNJAM

Format

Qualifiers

None.

Arguments

None.

Description

The UNJAM command performs an I/O bus reset. This is implemented by writing 1 to IPR 55. The command also performs an explicit software reset on the SGEC and SHAC chips, since 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 (writes to 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 into an 8-bit register initially set to 0. The command characters include the checksum and separating whitespace, but not the terminating carriage return, rubouts, or characters deleted by rubout.

- If no errors occur, the result is 0.
- 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 not 0, 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 0. 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 the console issues an error message.

If an error occurs during loading, the contents of the memory being loaded are unpredictable.

Echo is suppressed during the receiving of the data string and checksums.

To avoid treating flow control characters from the terminal as valid command line checksums, all flow control is terminated at the reception of the carriage return terminating the command line.

It is possible to control the console serial line through the use of the control characters (Ctrl C, Ctrl S, and so on) during a binary unload. It is not possible to control the console serial line during a binary load, because all received characters are valid binary data.

The console must recieve the data being loaded with a binary load command at a rate of at least 1 byte every 60 seconds. The console must receive the command checksum that precedes the data 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's specified character rate. The console is able to receive at least 4 kilobytes of data in a single X command.

Examples

None.

! Comment

Format

Qualifiers

None.

Arguments

None.

Description

The comment command is used to document command sequences. The ! comment character can appear anywhere on the command line. All characters following the comment character are ignored.

Examples

```
>>>! The console ignores this line.
>>>
```

!

9.8.1 Command Summary

Table 9–6 and Table 9–7 summarize the console commands.

Conventions for Table 9–6 and Table 9–7

UPPERCASE denotes the command or qualifier keyword.
{} enclose a mandatory item that must be syntactically correct.
[ ] enclose an optional item.
! separates optional items.

Parameters

boot_flags, count, size, address, and parameters are hexadecimal longword values.
boot_device is a legal boot device name.
csr_address is a Q22-bus I/O page CSR address.
controller_number is a controller number from 0 to 255.
halt_action is the value of the user-defined halt action, from 0 to 4.
language_type is the language code, from 1 to 15.
command is any console command other than REPEAT.
data, pattern, and mask are hexadecimal values of the current size.
test_number is a hexadecimal byte test number.

Table 9–6 Console Command Summary

Command	Qualifiers	Argument	Other(s)
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	—	{*bitmap*}	—
SET BOOT	—	{*device_string*}	—
SET CONTROLP	—	{0/1}	—

Table 9–6 (Cont.) Console Command Summary

Command	Qualifiers	Argument	Other(s)
SET HALT	—	{*halt_action*}	—
SET HOST	/DUP /DSSI /BUS:{0/1}	{*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	—	{0/1}	—
SHOW BFL(A)G	—	—	—
SHOW BOOT	—	—	—
SHOW CONTROLP	—	—	—
SHOW DEVICE	—	—	—
SHOW DSSI	—	—	—
SHOW ETHERNET	—	—	—
SHOW HALT	—	—	—
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*}
XDELTA	/CONTINUE	—	—

Table 9–7 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.
Address Space Control	
/G	General-purpose registers
/I	Internal processor registers
/V	Virtual memory
/P	Physical memory, both VAX memory and I/O spaces
/U	Protected memory (ROMs, SSC RAM, PFN bitmap, 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 bitmap to pass to VMB through R5. See Figure 9–7 for a bit description of R5. Either form of the command is acceptable.
/RPB, /MEM	FIND command only. Search for valid RPB or good block of memory.
/DUP, /DSSI, /UQSSP, /DISK, /TAPE, /MAINTENANCE, /SERVICE	SET HOST command only. Refer to command description for usage.
/CONTINUE	XDELTA command only. Enter XDELTA step mode at current PC.

9.9 Diagnostics

The ROM-based diagnostics constitute the bulk of the firmware on the KA670. These diagnostics run automatically on power-up. You can run one or all of the tests interactively using the TEST command. This section summarizes the functions of the diagnositics.

The ROM-based diagnostics have several functions:

1. During power-up, they determine if enough of the KA670 is working to allow the console to run.
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 they 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 meet these requirements, the diagnostics have been 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 the board is on the manufacturing line. Sophisticated users may also create their own scripts interactively. The script also contain the following 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 be run only from the EPROM. Other tests are position-independant code (PIC) that may be run from EPROM or main memory, in the interests of execution speed.

The diagnostic executive interprets scripts to determine what tests are to be run. There are several built-in scripts on the KA670 that are used for manufacturing, power-up, and field service functions. The diagnostic executive automatically invokes the correct script based on the current environment of the KA670. Any script can be explicitly run with the TEST command from the console terminal.

The diagnostic executive is responsible for controlling the tests, so errors can be caught and reported to the user. The executive also ensures the machine is left in a consistent and well-defined state when the tests are run.

9.9.1 Error Reporting

Before a console is established, the only error reporting is on the KA670 diagnostic LEDs and any LEDs on other boards. After a console is established, it reports all errors detected by the diagnostics. When possible, the diagnostics issue an error summary on the console.

For example, Example 9–1 shows a typical error display.

```
❶    ?9A 2 02 FF 0000 0000 01    ; SUBTEST_9A_02, DE_INTERACTION.LIS

❷    P1=00000002  P2=00000000  P3=00004000  P4=00008000  P5=0000C000
❸    P6=00000000  P7=00000002  P8=00000002  P9=84004000 P10=00001FFF
❹    r0=00000054  r1=00000040  r2=00000000  r3=0000C524  r4=00000014
❺    r5=30002800  r6=0000C4E0  r7=20008000  r8=00004000 EPC=20057BBD

     Normal operation not possible.
```

Example 9–1 Diagnostic Register Dump

❶ The first line is a test summary, containing six hexadecimal fields.

- ?9A identifies the diagnostic **test**.
- 2 is the **severity** level of a test failure, as dictated by the script. Severity level 2 failures display this five-line error printout and halt an autoboot to console I/O mode. Severity level 1 errors display the first line of the error printout, but do not interrupt an autoboot. Most tests have a severity level of 2.
- 02 is a **subtestlog** number that, in conjunction with listing files, isolates to within a few instructions where the diagnostic detected the error.
- FF is a **de_error** code used by the diagnostic executive to signal the diagnostic's state and any illegal behavior. This field indicates a condition that the diagnostic expects on detecting a failure. Possible codes:

  FF–Normal error exit from diagnostic
  FE–Unanticipated exception/interrupt in diagnostic.
  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 is the SCB **vector** (if nonzero) through which an unexpected exception or interrupt trapped, when the **de_error** field indicates an unexpected exception or interrupt (FE or EF).
- 0000 is the **count** of previous errors.
- 01 is the **loop_subtest**, 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.
- SUBTEST_9A_02 is a **subtest_symbol** that identifies the most recent subtestlog entry in the listing file.
- DE_INTERACTION.LIS is the name of the **listing_file** that contains the failed diagnostic.

❷ P1...P5 are the first five longwords of the diagnostic state. This is internal information that is used by repair personnel.

❸ P6...P10 are the last five longwords of the diagnostic state.

❹ R0...R4 are the first five GPRs at the moment the error was detected.

❺ R5...R8 are additional GPRs and ERF is a diagnostic summary longword.

9.9.2 Diagnostic Interdependencies

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

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:

- 16 multiplexed data/address lines —BDAL<15:00>
- 2 multiplexed address/parity lines —BDAL<17:16>
- 4 extended address lines —BDAL<21:18>
- 6 data transfer control lines —BBS7, BDIN, BDOUT, BRPLY, BSYNC, BWTBT
- 6 system control lines —BHALT, BREF, BEVNT, BINIT, BDCOK, BPOK
- 10 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 KN210 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 called the *bus master*, or *arbiter*. The master device controls the bus when communicating with another device on the bus, called 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 master-slave 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 completion of the bus cycle 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. Table A–5 lists the data transfer bus cycles.

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 the 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 9 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 (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 between 0 ns (minimum) and 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.

MA-1074-87

Figure A–1 DATI Bus Cycle

When the bus master receives BRPLY L, it does the following:

- Waits at least 200 ns deskew time, 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) before removing 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

When BSYNC L is continuously asserted, the bus master retains control of the bus 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.

MA-1084-87

Figure A–2 DATI Bus Cycle Timing

DATOB

DATOB (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.

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, the bus master outputs the data on BDAL<15:00> L at least 100 ns after BSYNC L assertion. BWTBT L remains negated for the length of the bus cycle. If the transfer is a byte transfer, BWTBT L remains asserted. If it is the output of a DATIOB, BWTBT L becomes asserted and lasts the duration of the bus cycle.

During a byte transfer, BDAL<00> L selects the high or low byte. This occurs in the addressing part of the cycle. If asserted, the high byte (BDAL<15:08> L) is selected; otherwise, the low byte (BDAL<07:00> L) is selected. An asserted BDAL 16 L at this time forces a parity error to be written into memory if the memory is a parity-type memory. BDAL 17 L is not used for write operations. The bus master asserts BDOUT L at least 100 ns after BDAL and BDWTBT L bus drivers are stable. The slave device responds by asserting BRPLY L within 10 µs to avoid bus timeout. This completes the data setup and deskew time.

During the data hold and deskew time, the bus master receives BRPLY L and negates BDOUT L, which must remain asserted for at least 150 ns from the receipt of BRPLY L before being negated by the bus master. BDAL<17:00> L bus drivers remain asserted for at least 100 ns after BDOUT L negation. The bus master then negates BDAL inputs.

During this time, the slave device senses BDOUT L negation. The data is accepted and the slave device negates BRPLY L. The bus master responds by negating BSYNC L. However, the processor does not negate BSYNC L for at least 175 ns after negating BDOUT L. This completes the 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.

MA-1080-87

Figure A–4 DATO or DATOB Bus Cycle Timing

DATIOB

The protocol for a DATIOB bus cycle (Figure A–5) is identical to the addressing and data transfer part of the DATI and DATOB bus cycles. After addressing the device, a DATI cycle is performed as explained in the DATI section; 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 shows the DATIOB bus cycle timing.

Figure A–5 DATIO or DATIOB Bus Cycle

MA-1060-87

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.

BDMGO L/BDMGI L is driven out of the processor on the BDMGO L pin, enters each module on the BDMGI L pin, then 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.

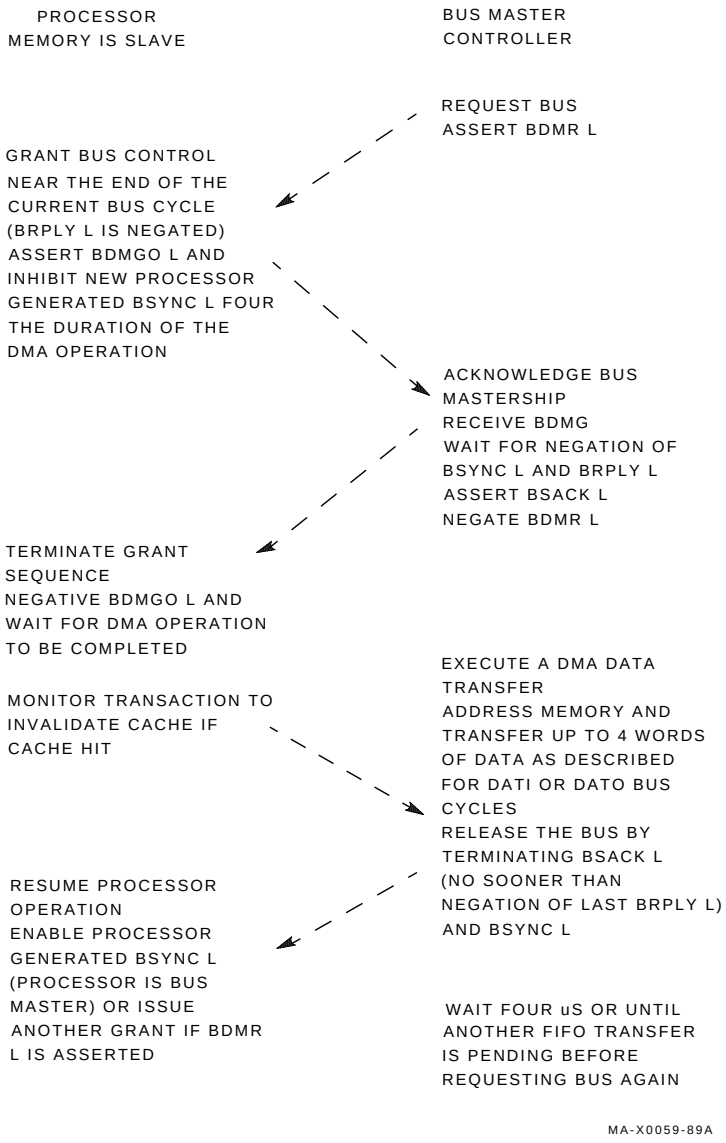


Figure A–7 DMA Protocol

MA-1078-87

Figure A–8 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).

- Section A.4.2.1 describes the DATBI bus cycle (Figure A–9).
- Section A.4.2.2 describes the DATBO bus cycle (Figure A–10).

MA-1088-87

Figure A–9 DATBI Bus Cycle Timing

MA-1087-87

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 the address.** The appropriate memory device recognizes that it must respond to the address on the bus.
- **Request the 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 the data.** The bus slave asserts TRPLY between 0 ns (minimum) and 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 the 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 the bus master monitors RDMR, it may perform up to 16 transfers as long as RDMR is not asserted at the end of the seventh transfer.

- **Terminate the bus cycle.** 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 the output transfer.** The bus master negates TDOUT 150 ns (minimum) after receiving RRPLY.
- **Operation completed.** The bus slave negates TRPLY 0 ns (minimum) after receiving the negation of RDOUT. If RREF was asserted when TDOUT negated and if the bus master wants to transfer another word, the bus master gates the new data on TDATA<15:00> 100 ns (minimum) after negating TDOUT. RREF is stable from 75 ns (maximum) after RRPLY asserts until 20 ns (minimum) after RDOUT negates. (The 20 ns minimum represents minimum receiver delays for RDOUT at the slave and RREF at the master). The bus master asserts TDOUT 100 ns (minimum) after gating new data on TDATA<15:00> and 150 ns (minimum) after receiving the negation of RRPLY. The cycle continues with the timing relationship in receive data above.

  NOTE
  The bus master must limit itself to not more than eight transfers unless it monitors RDMR. If the bus master monitors RDMR, it may perform up to 16 transfers as long as RDMR is not asserted at the end of the seventh transfer.

- **Terminate the 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.

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 interrrupt 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.

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).

MA-1076-87

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.

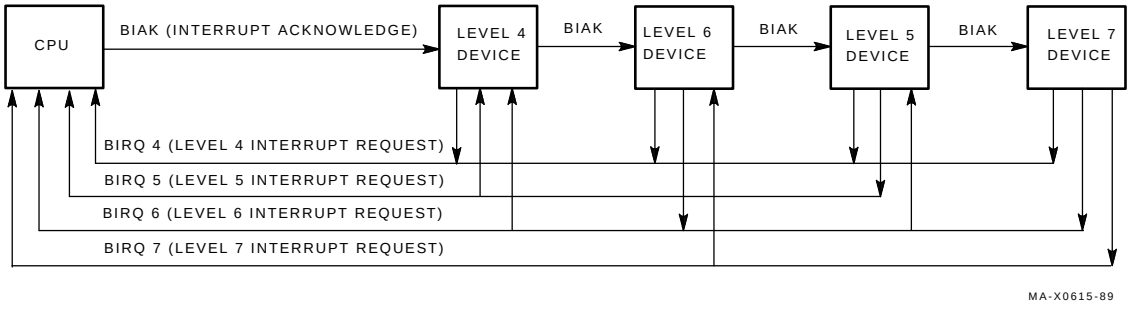


Figure A–13 Position-Independent Configuration

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.

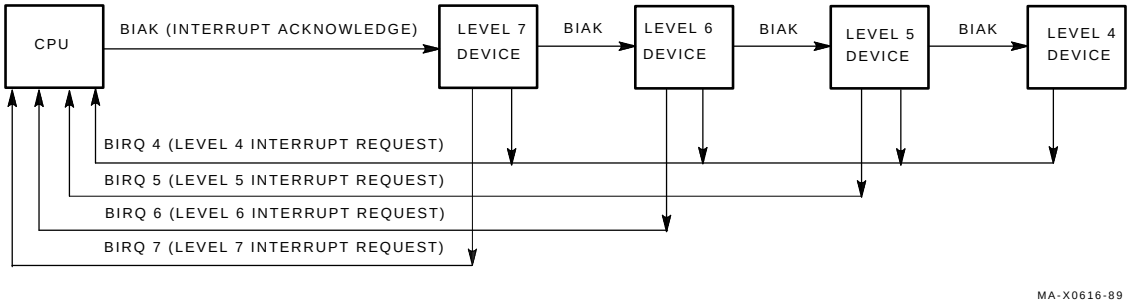


Figure A–14 Position-Dependent Configuration

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

Section A.7.1 lists the input and output logic levels for Q22-bus signals.

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	0.4 Vdc (maximum)
TTL logical high	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 is 1.3 V (maximum).
- Input high voltage is 1.7 V (minimum).
- 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.

MA-1071-87

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 ohms +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 percent. 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 percent, with a maximum ripple of 100 mV pp. +12 Vdc must be regulated to 3 percent, 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. 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.

MA-1072-87

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.

MA-1073-87

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.

MA-1054-87

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.

MA-1079-87

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.

Figure A–20 represents the dimensions for a typical Q22-bus module.

MA-1091-87

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 +5 B)	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.
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.

Table A–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
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.
AV1	+5 B	+5 V battery power —Secondary +5 V power connection. Battery power can be used with certain devices.

Table A–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
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 KA670 and MS670 are quad-height modules with the following dimensions:

Height	26.56 cm (+ 0.038 / –0.051 cm) 10.457 inches (+0.015 / –0.020 inches)
Length	21.412 cm (±0.025 cm) 8.430 inches (±0.010 inches)
Width	
Nonconductive	0.953 cm (maximum) 0.375 inches (maximum)
Conductive	0.871 cm (maximum) 0.343 inches (maximum)

NOTE
Width, as defined for Digital Equipment Corporation modules, is the height of components above the surface of the module.

B.1.1 KA670 Console Connector (J2)

The 100-pin console connector provides the connection between the KA670 CPU module and the H3604 console module. Table B–1 lists the J2 pinouts.

Table B–1 KA670 Console Connector (J2) Pinout

Pin	Signal Name	Usage	Meaning
1	GND	Ground	Signal ground
2 to 3	SH1_DATA<0> L	DSSI	DSSI 1 data bus bit 0.
4	GND	Ground	Signal ground.
5 to 6	SH1_DATA<1> L	DSSI	DSSI 1 data bus bit 1.
7	GND	Ground	Signal ground.
8 to 9	SH1_DATA<2> L	DSSI	DSSI 1 data bus bit 2.
10	GND	Ground	Signal ground.
11 to 12	SH1_DATA<3> L	DSSI	DSSI 1 data bus bit 3.

Table B–1 (Cont.) KA670 Console Connector (J2) Pinout

Pin	Signal Name	Usage	Meaning
13	GND	Ground	Signal ground.
14 to 15	SH1_DATA<4> L	DSSI	DSSI 1 data bus bit 4.
16	GND	Ground	Signal ground.
17 to 18	SH1_DATA<5> L	DSSI	DSSI 1 data bus bit 5.
19	GND	Ground	Signal ground.
20 to 21	SH1_DATA<6> L	DSSI	DSSI 1 data bus bit 6.
22	GND	Ground	Signal ground.
23 to 24	SH1_DATA<7> L	DSSI	DSSI 1 data bus bit 7.
25	GND	Ground	Signal ground.
26 to 27	SH1_DP L	DSSI	DSSI 1 data bus parity line.
28	GND	Ground	Signal ground.
29 to 30	SH1_ACK L	DSSI	This signal is driven by an initiator to indicate an acknowledgment for a REQ/ACK data transfer handshake.
31	GND	Ground	Signal ground.
32 to 33	SH1_RST L	DSSI	DSSI pin reset.
34	GND	Ground	Signal ground.
35 to 36	SH1_SEL L	DSSI	DSSI pin select A signal. Used by the initiator to select a target.
37	GND	Ground	Signal ground.
38 to 39	SH1_C/D L	DSSI	Pin command/data signal. Driven by a target that indicates whether conrtol or data information is on the data bus. When asserted (low), indicates control.
40	GND	Ground	Signal ground.
41 to 42	SH1_REQ L	DSSI	Request signal. Driven by a target to indicate a request for a REQ/ACK data transfer handshake.
43	GND	Ground	Signal ground.
44 to 45	SH1_I/O L	DSSI	Input/output A signal. Driven by a target that controls the direction of data movment on the data bus with respect to the initiator. When asserted (low), indicates input.
46	GND	Ground	Signal ground.
47 to 48	SH1_BSY L	DSSI	Busy signal. This is an OR-tied signal, indicating the bus is being used.
49	GND	Ground	Signal ground.
50	GND	Ground	Signal ground.
51	GND	Ground	Signal ground.
52	GND	Ground	Signal ground.

Table B–1 (Cont.) KA670 Console Connector (J2) Pinout

Pin	Signal Name	Usage	Meaning
53	TXD H	Ethernet	Console terminal data out signal. This signal outputs serial character data from the console terminal transmitter.
54	GND	Ground	Signal ground.
55	RXD H	Ethernet	Console terminal data in signal. This signal inputs serial character data to the console terminal receiver.
56	GND	Ground	Signal ground.
57	TB25K H	Console	This is the 25.6 kHz oscillator from the H3604 console module, which supplies the timebase for the time-of-year (TOY) clock.
58	GND	Ground	Signal ground.
59	TDATA H	Ethernet	Transmit data signal.
60	GND	Ground	Signal ground.
61	XMTEN H	Ethernet	Transmit enable signal.
62	GND	Ground	Signal ground.
63	RCAR H	Ethernet	Receive enable signal.
64	GND	Ground	Signal ground.
65	COL H	Ethernet	Collision detect signal.
66	GND	Ground	Signal ground.
67	RDATA H	Ethernet	Receive data signal.
68	GND	Ground	Signal ground.
69	TCLK H	Ethernet	Transmit clock signal.
70	GND	Ground	Signal ground.
71	RCLK H	Ethernet	Receive clock signal.
72	GND	Ground	Signal ground.
73	BITRATE <2> L	Console	Bit rate feld bit 2.
74	BITRATE <1> L	Console	Bit rate feld bit 1.
75	BITRATE <0> L	Console	Bit rate feld bit 0.
76	LEDCODE <3> L	Console	Bit 3 (MSB) of the LED code going to the hexadecimal display on the console module.
77	LEDCODE <2> L	Console	Bit 2 of the LED code going to the hexadecimal display on the console module.
78	LEDCODE <1> L	Console	Bit 1 of the LED code going to the hexadecimal display on the console module.
79	LEDCODE <0> L	Console	Bit 0 (LSB) of the LED code going to the hexadecimal display on the console module.

Table B–1 (Cont.) KA670 Console Connector (J2) Pinout

Pin	Signal Name	Usage	Meaning
80	VDDI H	Console	Battery backup supply for the SSC TOY clock.
81	DSSI1_UID <2> L	DSSI	DSSI 1 node identifcation (ID) number bit 2 (MSB).
82	DSSI1_UID <1> L	DSSI	DSSI 1 node identifcation (ID) number bit 1.
83	DSSI1_UID <0> L	DSSI	DSSI 1 node identifcation (ID) number bit 0 (LSB)
84	DSSI2_UID <2> L	DSSI	DSSI 2 node identifcation (ID) number bit 2 (MSB).
85	DSSI2_UID <1> L	DSSI	DSSI 2 node identifcation (ID) number bit 1.
86	DSSI2_UID <0> L	DSSI	DSSI 2 node identifcation (ID) number bit 0 (LSB).
87	BOOTDIAG<1> L	Console	Boot and diagnostic code bit 1.
88	BOOTDIAG<0> L	Console	Boot and diagnostic code bit 0.
89	ENBHALT L	Console	The halt enable bit.
90	BTRYBAD H	Console	The battery bad signal that comes from the console module and goes to the battery sense circuitry.
91	NC		
92	NC		
93	NC		
94	NC		
95	NC		
96	NC		
97	NC		
98	NC		
99	NC		
100	CABLE_OK_IN L	Console	This pin indicates if the cable is properly installed.

B.2 DC Power Consumption

The KA670-AA/BA CPU module power requirements are as follows:

KA670-AA/BA

7.40 amps maximum, at +5.00 Vdc
0.27 amps maximum, at +3.30 Vdc

MS670-BA

1.16 amps maximum at +5.00 Vdc
2.52 amps maximum at +5.00 Vdc during fast diagnostic mode

H3604

1.50 A maximum at +5.00 Vdc
500 mA maximum at +12.0 Vdc
62 mA maximum at −12.0 Vdc

Fast diagnostic mode (FDM) is run only during power-up ROM diagnostics.

Typical currents are 10 percent less than the specified maximum.

B.3 Bus Loads

The KA670 CPU bus loads are as follows:

DC Loading

The KA670-AA/BA module presents a value of less than 1 dc load to the Q22-bus. The actual maximum value is specified to be 0.57 dc loads.

AC Loading

The KA670-AA/BA presents a maximum of 4 ac loads to the Q22-bus.

B.4 Battery Backup Specifications

When dc power is supplied to the KA670 module, it charges the external batteries from +5 volts through a 240-ohm resistor.

When dc power is removed from the KA670 module, it drains the external batteries at a rate of 1.0 milliamps/hour.

NOTE

These batteries supply power to the KA670 time-of-year clock and SSC RAM only. There is no battery backup for the memory system.

B.5 Operating Conditions

Temperature	+5° to +60° C (−40° to +140° F), with a rate of change no greater than 20 ±2° C/hour (36 ±4° F/hour) at sea level. The maximum temperature must be derated by 1.8° C/1000 meters (1° F/1000 feet) above sea level.
Humidity	10 to 95% noncondensing, with a maximum wet bulb temperature of 32° C (90° F) and a minimum dew point temperature of 2° C (36° F).
Altitude	Up to 2,400 meters (8,000 feet), with a rate of change no greater than 300 meters/minute (1000 feet/minute).
Airflow	The airflow required to meet these specifications is 200 lfm.

B.6 Nonoperating Conditions (Less Than 60 Days)

Temperature	−40° to +66° C (−40° to +151° F), with a rate of change no greater than 11 ±2° C/hour (20 ±4° F/hour) at sea level. The maximum temperature must be derated by 1.8° C/1000 meters (1° F/1000 feet) above sea level.
Humidity	Up to 95% noncondensing.
Altitude	Up to 4,900 meters (16,000 feet), with a rate of change no greater than 600 meters/minute (2000 feet/minute).

B.7 Nonoperating Conditions (Greater than 60 Days)

Temperature	+5° to +60° C (–40 to +140° F), with a rate of change no greater than 20 ±2° C (36 ±4° F) per hour at sea level. The maximum temperature must be derated by 1.8° C/1000 meters (1° F/1000 feet) above sea level.
Humidity	10 to 95% noncondensing, with a maximum wet bulb temperature of 32° C (90° F) and a minimum dew point temperature of 2° C (36° F).
Altitude	Up to 2,400 meters (8,000 feet), with a rate of change no greater than 300 meters/minute (1000 feet/minute).

C
Address Assignments

C.1 KA670 General Local Address Space Map

Address Range	Contents
VAX Memory Space	
0000 0000 to 1FFF FFFF	Local memory space (512 Mbytes)
VAX I/O Space	
2000 0000 to 2000 1FFF	Local Q22-bus I/O space (8 Kbytes)
2000 2000 to 2003 FFFF	Reserved local I/O space (248 Kbytes)
2004 0000 to 2007 FFFF	Local UVROM space
2008 0000 to 201F FFFF	Local register I/O space (1.5 Mbytes)
2020 0000 to 23FF FFFF	Reserved local I/O space (62.5 Mbytes)
2400 0000 to 27FF FFFF	Reserved local I/O space (64 Mbytes)
2008 0000 to 2BFF FFFF	Reserved local I/O space (64 Mbytes)
2C08 0000 to 2FFF FFFF	Reserved local I/O space (64 Mbytes)
3000 0000 to 303F FFFF	Local Q22-bus memory space (4 Mbytes)
3040 0000 to 33FF FFFF	Reserved local I/O space (60 Mbytes)
3400 0000 to 37FF FFFF	Reserved local I/O space (64 Mbytes)
3800 0000 to 3BFF FFFF	Reserved local I/O space (64 Mbytes)
3C00 0000 to 3FFF FFFF	Reserved local I/O space (64 Mbytes)

C.2 KA670 Detailed Local Address Space Map

Contents	Address
Local memory space (up to 512 Mbytes)	0000 0000 to 1FFF FFFF
Q22-bus map—top 32 Kbytes of main memory	

VAX I/O Space

Contents	Address
Local Q22-bus I/O Space	**2000 0000 to 2000 1FFF**
Reserved Q22-bus I/O space	2000 0000 to 2000 0007
Q22-bus floating address space	2000 0008 to 2000 07FF
User-reserved Q22-bus I/O space	2000 0800 to 2000 0FFF
Reserved Q22-bus I/O space	2000 1000 to 2000 1F3F
Interprocessor communication register	2000 1F40
Reserved Q22-bus I/O space	2000 1F44 to 2000 1FFF
Local Register I/O Space	**2000 2000 to 2003 FFFF**
Reserved local register I/O space	2000 4000 to 2000 402F
SHAC1 SSWCR	2000 4030
Reserved local register I/O space	2000 4034 to 2000 4043
SHAC1 SSHMA	2000 4044
SHAC1 PQBBR	2000 4048
SHAC1 PSR	2000 404C
SHAC1 PESR	2000 4050
SHAC1 PFAR	2000 4054
SHAC1 PPR	2000 4058
SHAC1 PMCSR	2000 405C
Reserved local register I/O space	2000 4060 to 2000 407F
SHAC1 PCQ0CR	2000 4080
SHAC1 PCQ1CR	2000 4084
SHAC1 PCQ2CR	2000 4088
SHAC1 PCQ3CR	2000 408C
SHAC1 PDFQCR	2000 4090
SHAC1 PMFQCR	2000 4094
SHAC1 PSRCR	2000 4098
SHAC1 PECR	2000 409C
SHAC1 PDCR	2000 40A0
SHAC1 PICR	2000 40A4
SHAC1 PMTCR	2000 40A8
SHAC1 PMTECR	2000 40AC
Reserved local register I/O space	2000 40B0 to 2000 422F
SHAC2 SSWCR	2000 4230
Reserved local register I/O space	2000 4234 to 2000 4243
SHAC2 SSHMA	2000 4244
SHAC2 PQBBR	2000 4248

VAX I/O Space Contents	Address
Local Register I/O Space	**2000 2000 to 2003 FFFF**
SHAC2 PSR	2000 424C
SHAC2 PESR	2000 4250
SHAC2 PFAR	2000 4254
SHAC2 PPR	2000 4258
SHAC2 PMCSR	2000 425C
Reserved local register I/O space	2000 4260 to 2000 427F
SHAC2 PCQ0CR	2000 4280
SHAC2 PCQ1CR	2000 4284
SHAC2 PCQ2CR	2000 4288
SHAC2 PCQ3CR	2000 428C
SHAC2 PDFQCR	2000 4290
SHAC2 PMFQCR	2000 4294
SHAC2 PSRCR	2000 4298
SHAC2 PECR	2000 429C
SHAC2 PDCR	2000 42A0
SHAC2 PICR	2000 42A4
SHAC2 PMTCR	2000 42A8
SHAC2 PMTECR	2000 42AC
Reserved local register I/O space	2000 42B0 to 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 Revision number 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 to 2003 FFFF
UVROM Space	**2004 0000 to 2007 FFFF**
MicroVAX system type register (in UVROM)	2004 0004
Local UVROM (halt-protected)	2004 0000 to 2007 FFFF
Local register I/O space	**2008 0000 to 201F FFFF**

*These registers are not fully implemented. Accesses yield unpredictable results.

VAX I/O Space

Contents	Address
Local register I/O space	**2008 0000 to 201F FFFF**
DMA system configuration register	2008 0000
DMA system error register	2008 0004
DMA master error address register	2008 0008
DMA slave error address register	2008 000C
Q22-bus map base register	2008 0010
Reserved local register I/O space	2008 0014 to 2008 00FF
Error status register (Reg. 32)	2008 0180
Memory error address (Reg. 33)	2008 0184
I/O Error address (Reg. 34)	2008 0188
DMA memory error address (Reg. 35)	2008 018C
DMA Mode control and diagnostic status register (Reg. 36)	2008 0190
Reserved local register I/O space	2008 0194 to 2008 3FFF
Boot and diagnostic register (32 copies)	2008 4000 to 2008 407C
Reserved local register I/O space	2008 4080 to 2008 7FFF
Q22-bus map registers	2008 8000 to 2008 FFFF
Reserved local register I/O space	2009 0000 to 2013 FFFF
SSC base address register	2014 0000
SSC configuration register	2014 0010
CP bus timeout control register	2014 0020
Diagnostic LED register	2014 0030
Reserved local register I/O space	2014 0034 to 2014 006B
The following addresses allow those KA670 internal processor registers that are implemented in the SSC chip (external, internal processor registers) to be accessed using the local I/O page. These addresses are documented for diagnostic purposes only and should not be used by nondiagnostic 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 to 2014 00DB
I/O bus reset register	2014 00DC
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 to 2014 00FF

*These registers are not fully implemented. Accesses yield unpredictable results.

†These registers are internal SSC registers used for SSC chip test purposes only. They should not be accessed by the CPU.

VAX I/O Space Contents	Address
Local register I/O space	
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 to 2014 012F
BDR address decode match register	2014 0130
BDR address decode mask register	2014 0134
Reserved local register I/O space	2014 0138 to 2014 03FF
Battery backed-up RAM	2014 0400 to 2014 07FF
Reserved local register I/O space	2014 0800 to 201F FFFF
Reserved local I/O space	2020 0000 to 2FFF FFFF
Local Q22-bus memory space	3000 0000 to 303F FFFF
Reserved Local register I/O space	3040 0000 to 3FFF FFFF

C.3 External, Internal Processor Registers

Several of the internal processor registers (IPRs) on the KA670 are implemented in the C-chip or SSC chip rather than the CPU chip. These registers are referred to as external, internal processor registers and are listed here.

IPR	Register Name	Abbreviation
27	Time-of-year register	TOY
28	Console storage receiver status	CSRS*
29	Console storage receiver data	CSRD*
30	Console storage transmitter status	CSTS*
31	Console storage transmitter data	CSDB*
32	Console receiver control/status	RXCS
33	Console receiver data buffer	RXDB
34	Console transmitter control/status	TXCS
35	Console transmitter data buffer	TXDB
55	I/O system reset register	IORESET
112	Backup cache reserved register	BC112*
113	Backup cache tag store	BCBTS
114	Backup cache P1 tag store	BCP1TS
115	Backup cache P2 tag store	BCP2TS
116	Backup cache refresh register	BCRFR

IPR	Register Name	Abbreviation
117	Backup cache index register	BCIDX
118	Backup cache status register	BCSTS
119	Backup cache control register	BCCTL
120	Backup cache error register	BCERR
121	Backup cache flush backup cache tag store	BCFBTS
122	Backup cache flush primary cache tag store	BCPBTS
123	Backup cache reserved register	BC123*

C.4 Global Q22-bus Address Space Map

Q22-bus Memory Space	
Q22-bus memory space (octal)	0000 0000 to 1777 7777
Q22-bus I/O Space (BBS7 Asserted)	
Q22-bus I/O Space (octal)	1776 0000 to 1777 7777
Reserved Q22-bus I/O space	1776 0000 to 1776 0007
Q22-bus floating address space	1776 0010 to 1776 3777
User-reserved Q22-bus I/O space	1776 4000 to 1776 7777
Reserved Q22-bus I/O space	1777 0000 to 1777 7477
Interprocessor communication register	1777 7500
Reserved Q22-bus I/O space	1777 7502 to 1777 7777

D
VAX Instruction Set

The information in this appendix is for reference only.

D.1 Syntax

The standard notation for operand specifiers is

```
<NAME>.<access type><data type>
```

Name
is a suggestive name for the operand in the context of the instruction. It is the capitalized name of a register or block for implied operands.

Access type
is 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 *Rn*, same as a, otherwise R[n+1]'R[n].

w = write-only operand.

Data type
is 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
are locations accessed by the instruction, but not specified in an operand. They appear in 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.

Table D–1 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 -	
88	BISB2 mask.rb, dst.mb	* * 0 -	
C8	BISL2 mask.rl, dst.ml	* * 0 -	
A8	BISW2 mask.rw, dst.mw	* * 0 -	

Table D–1 (Cont.) Integer Arithmetic and Logical Instructions

Opcode	Instruction	N Z V C	Exceptions
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
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 -	

Table D–1 (Cont.) Integer Arithmetic and Logical Instructions

Opcode	Instruction	N Z V C	Exceptions
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
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 -	

Table D–2 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 -	
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 -	

Table D–3 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}, fndpos.wl	0 * 0 0	rsv
EA	FFS startpos.rl, size.rb, base.vb, {field.rv}, fndpos.wl	0 * 0 0	rsv

Table D–4 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	- - - -	
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

Table D–4 (Cont.) Control Instructions

Opcode	Instruction	N Z V C	Exceptions
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 *	
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

Table D–5 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

Table D–6 Miscellaneous Instructions

Opcode	Instruction	N Z V C	Exceptions
B9	BICPSW mask.rw	* * * *	rsv

Table D–6 (Cont.) Miscellaneous Instructions

Opcode	Instruction	N Z V C	Exceptions
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, sub indexout.wl	* * 0 0	
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	

Table D–7 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
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	* * * *	

Table D–8 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*} Where y=MINU(x, PSL<CURRENT_MODE>)	0 0 0 0	
06	LDPCTX {PCB.r*, -(KSP).w*}	- - - -	rsv, prv

Table D–8 (Cont.) Operating System Support Instructions

Opcode	Instruction	N Z V C	Exceptions
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

Table D–9 Floating Point Instructions

Opcode	Instruction	N Z V C	Exceptions
These instructions are implemented by the KA670 floating point accelerator.			
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
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	

Table D–9 (Cont.) Floating Point Instructions

Opcode	Instruction	N Z V C	Exceptions
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
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

Table D–10 Microcode-Assisted Emulated Instructions

Opcode	Instruction	N Z V C	Exceptions
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.			
20	ADDP4 addlen.rw, addaddr.ab, sumlen.rw, sumaddr.ab	* * * 0	rsv, dov
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, fll.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

Table D–10 (Cont.) Microcode-Assisted Emulated Instructions

Opcode	Instruction	N Z V C	Exceptions
22	SUBP4 sublen.rw, subaddr.ab, difflen.rw, difaddr.ab	* * * 0	rsv, dov
23	SUBP6 sublen.rw, subaddr.ab, minlen.rw, minaddr.ab, difflen.rw, difaddr.ab	* * * 0	rsv, dov

E
Machine State on Power-Up

This appendix describes the state of the KA670 after a power-up halt.

The descriptions in this appendix assume

- The machine has no errors.
- The machine has just been turned on.
- Only the power-up diagnostics have been run.

The state of the machine is undefined after 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 KA670 firmware. The placement and/or existence 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 shows how main memory is partioned after diagnostics.

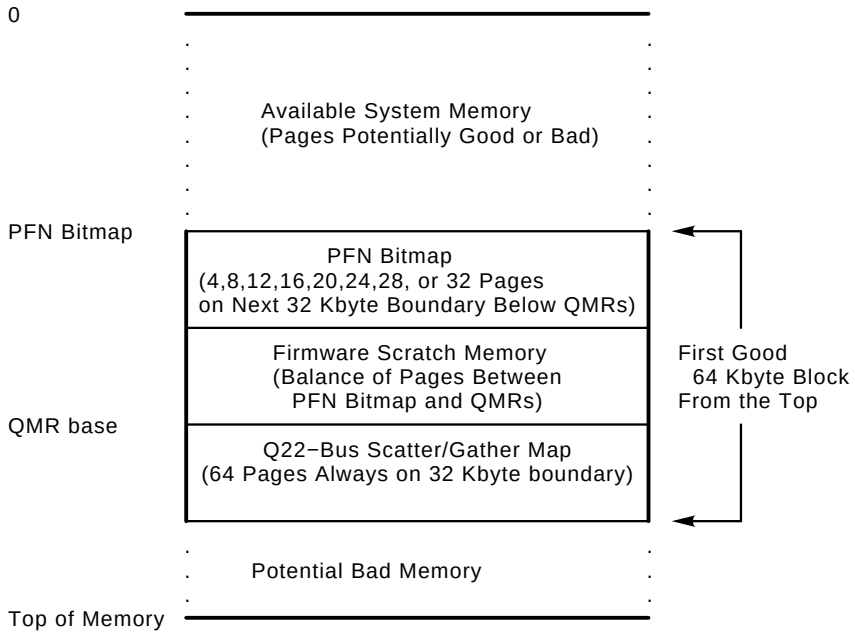


Figure E–1 Memory Layout After Power-Up Diagnostics

E.1.1 Reserved Main Memory

In order to build the scatter/gather map and the bitmap, the firmware tries to find a physically contiguous, page-aligned, 176-kilobyte block of memory at the highest possible address that has no multiple-bit errors. Single-bit errors are tolerated in this section.

Of the 176 kilobytes, the upper 32 kilobytes is dedicated to the Q22-bus scatter/gather map, as shown in Figure E–1. Of the lower portion, up to 128 kilobytes at the bottom of the block is allocated to the page frame number (PFN) bitmap. The size of the PFN bitmap depends 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 scatter/gather map (16 kilobytes minimum) is allocated for the firmware.

E.1.1.1 Page Frame Number (PFN) Bitmap

The page frame number bitmap is a data structure that indicates which pages in memory are deemed usable 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 kilobyte for every 4 megabytes of main memory:

System Size	Main Memory Required for the Bitmap
8 Mbytes	2 Kbytes
16 Mbytes	4 Kbytes
32 Mbytes	6 Kbytes
64 Mbytes	8 Kbytes

The bitmap does not map itself or anything above it. There may be memory above the bitmap that 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 1 in the bitmap indicates that the page is good and can be used. A 0 indicates that the page is bad and should not be used. By default, a page is flagged as 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 as bad.

The PFN bitmap is protected by a checksum stored in the battery backed-up RAM (BBU RAM). The checksum is a simple byte-wide, two's complement checksum. The sum of all bytes in the bitmap and the bitmap checksum should 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 megabytes of main memory. Main memory pages are not mapped if there is a corresponding page in Q22-bus memory, or if the pages are marked bad by the PFN bitmap.

On a processor halt other than power-up, the contents of the scatter/gather map is undefined and depends on operating system usage.

Operating systems should not move the location of the scatter/gather map. They 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

Scratch memory is reserved for the firmware. However, scratch memory is used only after successful execution of the memory diagnostics and initialization of the PFN bitmap and scatter/gather map. This memory is primarily for diagnostic purposes.

E.1.2 Contents of Main Memory

The contents of main memory are undefined after the diagnostics have run. Typically, nonzero test patterns are left in memory.

The diagnostics will "scrub" all of main memory, so that no power-up induced errors remain in the memory system. On the KA670 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 being written, because the ECC bits are not in agreement with their correspsonding data bits. An aligned longword write to every location (done by diagnostics) eliminates all power-up induced errors.

E.2 Memory Controller Registers

The KA670 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 0. 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 registers MEMCSR32, MEMCSR33, MEMCSR34, MEMCSR35, and MEMCSR36, as needed.

E.2.1 Primary (On-Chip) Cache

The CPU primary (on-chip) cache is tested during the power-up diagnostics, flushed, and turned off. The cache is again turned on by the BOOT and INIT commands. 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 since it runs in physical mode. The translation buffer can be invalidated by using PR$_TBIA, IPR 57.

E.2.3 Halt-Protected Space

The KA670 firmware runs mostly in halt-unprotected space. Only the first 8 kilobytes of the total 256 kilobytes are protected.

F
Maintenance Operation Protocol (MOP) Support

F.1 Network Listening

While the KA670 is waiting for a load volunteer during bootstrap, it listens on the network for other maintenance messages directed to the node. The KA670 identifies itself periodically at the end of each 8 to 12 minute interval before a bootstrap retry operation. This listening function supplements the MOP functions of the VMB load requester typically found in bootstrap firmware. The listening function 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
  Solicited SYSTEM_ID messages in response to REQUEST_ID messages
  Recognition of BOOT messages.

- A loopback server that responds to Ethernet LOOPBACK messages by echoing the message to the requester.
- An IEEE 802.2 responder that replies to both XID and TEST messages.

During network operation, the firmware listens only to MOP load/dump, MOP remote console, and Ethernet loopback assistance message protocols (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. IEEE 802.3 messages are also processed by the network listener.

Tables F–1 to F–4 summarize the MOP functions and message types supported by the KA670.

Table F–1 KA670 Network Maintenance Operations Summary

Function	Role	Transmit		Receive
MOP Ethernet and IEEE 802.3 Messages [1]				
Dump	Requester	—		—
	Server	—		—
Load	Requester	REQ_ PROGRAM[2]	to solicit	VOLUNTEER
		REQ_MEM_ LOAD	to solicit and acknowldege	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_ XFER	Code 00	Load # nn	Load addr aa-aa-aa-aa		Image data None		Xfer addr aa-aa-aa-aa
MEM_LOAD	Code 02	Load # nn	Load addr aa-aa-aa-aa		Image data dd-...		
REQ_ PROGRAM	Code 08	Device 05 QNA 25 LQA 3D KA640 49 KA670	Format 01 V3 04 V4	Program 02 Sys	SW ID [3] C-17 [1] C-128 [2] If C[1] >00 Len 00 No ID FF OS FE Maint	Procesr 00 Sys	Info (See SYSTEM_ID.)
REQ_MEM_ LOAD	Code 0A	Load # nn	Error ee				
PARM_LOAD_ w_XFER	Code 14	Load # nn	Prm typ 01 02 03 04 05 06 00 End	Prm len I-16 I-06 I-16 I-06 0A 08	Prm val Target name [1] Target addr [1] Host name [1] Host addr [1] Host time [1] Host time [2]		Xfer addr aa-aa-aa-aa
VOLUNTEER	Code 03						
Remote Console							
REQUEST_ID	Code 05	Rsrvd xx	Recpt # nn-nn				

[1]MOP V3.0 only.

[2]MOP x4.0 only.

[3]Software ID field is loaded from the string stored in the 40-byte RPB$T_FILE field of the RPB on a solicited boot.

Table F–2 (Cont.) Supported MOP Messages

Message Type	Message Fields							
Remote Console								
SYSTEM_ID	Code 07	Rsrvd xx	Recpt # nn-nn or 00-00	Info type 01-00 Version 02-00 Functions 07-00 HW addr 64-00 Device 90-01 Datalink 91-01 Bufr size		Info len 03 02 06 01 01 02	Info value 04-00-00 00-59 ee-ee-ee-ee-ee-ee 05, 25, 3D, or 49 01 06-04	
REQ_ COUNTERS	Code 09	Recpt # nn-nn						
COUNTERS	Code 0B	Recpt # nn-nn	Counter block					
BOOT [4]	Code 06	Verification vv-vv-vv-vv-vv-vv-vv-vv		Procesr 00 Sys	Control xx	Dev ID C-17	SW ID [3] (see REQ_ PROGRAM)	Script ID [2] C-128
Loopback								
LOOP_DATA	Skpcnt nn-nn	Skipped bytes bb-...		Function 00-02 Forward data		Forward addr ee-ee-ee-ee-ee-ee	Data dd-...	
LOOPED_DATA	Skpcnt nn-nn	Skipped bytes bb-...		Function 00-01 Reply		Recpt # nn-nn	Data dd-...	
IEEE 802.2								
XID_CMD/RSP	Form 81	Class 01	Rx window size (K) 00					
TEST_CMD/RSP	Optional data.							

[2]MOP x4.0 only.

[3]Software ID field is loaded from the string stored in the 40-byte RPB$T_FILE field of the RPB on a solicited boot.

[4]A BOOT message is not verified, since in this context, a boot is already in progress. However, a received BOOT message will cause the boot backoff timer to be reset to it's minimum value.

Table F–3 Ethernet & 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 following counters are kept for the Ethernet boot channel. All counters are unsigned integers. V4 counters rollover 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 MOP Versions 3.0 and 4.0.

Table F–5 MOP Counter Block

	Byte Length		
Name	**V3**	**V4**	**Description**
TIME_SINCE_CREATION	2	16	Time since last zeroed
Rx_BYTES	4	8	Bytes received
Tx_BYTES	4	8	Bytes sent
Rx_FRAMES	4	8	Frames received
Tx_FRAMES	4	8	Frames sent
Rx_MCAST_BYTES	4	8	Multicast bytes received
Rx_MCAST_FRAMES	4	8	Multicast frames received
Tx_INIT_DEFFERED	4	8	Frames sent, initially deferred[1]
Tx_ONE_COLLISION	4	8	Frames sent, single collision [1]
Tx_MULTI_COLLISION	4	8	Frames sent, multiple collisions[1]
	2		Send failure [2]
	2		Send failure bitmap [2]
TxFAIL_EXCESS_COLLS		8	Send failure - 0 Excessive collisions
TxFAIL_CARIER_CHECK		8	Send failure - 1 Carrier check failed
TxFAIL_SHRT_CIRCUIT		8	Send failure - 2 Short ciruit [3]
TxFAIL_OPEN_CIRCUIT		8	Send failure - 3 Open Circuit [3]
TxFAIL_LONG_FRAME		8	Send failure - 4 Frame too long[3]
TxFAIL_REMOTE_DEFER		8	Send failure - 5 Remote failure to defer[3]
	2		Receive failure [2]
	2		Receive failure bitmap [2]
RxFAIL_BLOCK_CHECK		8	Receive failure - Block check failure
RxFAIL_FRAMING_ERR		8	Receive failure - Framing error
RxFAIL_LONG_FRAME		8	Receive failure - Frame too long[3]
UNKNOWN_ DESTINATION	2	8	Unrecognized frame destination
DATA_OVERRUN	2	8	Data overrun

[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 bitmap indicating which failures occurred.

[3]Always 0.

Table F–5 (Cont.) MOP Counter Block

	Byte Length		
Name	**V3**	**V4**	**Description**
NO_SYSTEM_BUFFER	2	8	System buffer unavailable[3]
NO_USER_BUFFER	2	8	User buffer unavailable [3]
FAIL_COLLIS_DETECT		8	Collision detect check failure

[3]Always 0.

The following list describes each of the counters in more detail.

- **Time since last zeroed.** The time which has elapsed, since the counters were last zeroed. Provides a frame of reference for the other counters by indicating the amount of time they cover. For MOP Version 3, this time is the number of seconds. MOP Version 4 uses the UTC binary relative time format.
- **Bytes received.** 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.
- **Bytes sent.** 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.
- **Frames received.** The total number of frames successfully received. These are frames that passed hardware filtering. Provides a gross measurement of incoming Ethernet usage by the local system. Provides information used to determine the ratio of the error counters to successful transmits.
- **Frames sent.** The total number of frames successfully transmitted. This does not include data link generated retransmissions. Provides a gross measurement of outgoing Ethernet usage by the local system. Provides information used to determine the ratio of the error counters to successful transmits.
- **Multicast bytes received.** 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, 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.
- **Multicast frames received.** The total number of multicast frames successfully received. In conjunction with total frames received, provides a gross percentage of the Ethernet usage for multicast frames addressed to this system.
- **Frames sent, initially deferred.** The total number of times that a frame transmission was deferred on its first transmission attempt. In conjunction with total frames sent, measures Ethernet contention with no collisions.

- **Frames sent, single collision.** 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 total frames sent, measures Ethernet contention at a level where there are collisions but the backoff algorithm still operates efficiently.
- **Frames sent, multiple collisions.** 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 total frames sent, measures Ethernet contention at a level where there are collisions and the backoff algorithm no longer operates efficiently.

  NOTE
  No single frame is counted in more than one of the above three counters.

- **Send failures.** The total number of times a transmit attempt failed. Each time the counter is incremented, a type of failure is recorded. When a read counter function reads the counter, the list of failures is also read. When the counter is set to 0, the list of failures is cleared. In conjunction with total frames sent, this provides a measure of significant transmit problems. All of the problems reflected in this counter are also captured as events. Following are the possible failures. More information on their meanings and use can be found in the section on events.
  - **Excessive collisions.** Exceeded the maximum number of retransmissions due to collisions. Indicates an overload condition on the Ethernet.
  - **Carrier check failed.** The data link did not sense the receive signal that is required to accompany the transmission of a frame. Indicates a failure in either the transmitting or receiving hardware. Could be caused by either transceiver, transceiver cable, or a babbling controller that has been cut off.
  - **Short circuit.** 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.
  - **Open circuit.** 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.
  - **Frame too long.** 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 cutoff transmission too soon.
  - **Remote failure to defer.** 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.
- **Receive failures.** 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 total frames received, this provides a measure of data related receive problems. All of the problems reflected in this counter are also captured as events. Following are the possible reasons. More information on their meaning and use can be found in the section on events.
  - **Block check error.** A frame failed the CRC check. This indicates several possible failures, such as, EMI, late collisions, or improperly set hardware parameters.

- **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.
- **Frame too long.** 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.
- **Unrecognized frame destination.** 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.
- **Data overrun.** 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 total frames received, provides a measure of hardware resource failures. The problem reflected in this counter is also captured as an event.
- **System buffer unavailable.** The total number of times no system buffer was available for an incoming frame. In conjunction with total frames received, 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.
- **User buffer unavailable.** 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 total frames received, provides a measure of user buffer related receive problems. The problem reflected in this counter is also captured as an event.
- **Collision detect check failure.** 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.

G
ROM Partitioning

This appendix describes public ROM partitioning and subroutine entry points that are guaranteed to be compatible over future versions of the KA670 firmware.

G.1 Firmware EPROM Layout

The KA670 uses two 128-kilobyte EPROMs for a total of 256 kilobytes. 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 (Figure G–1) is halt-protected.

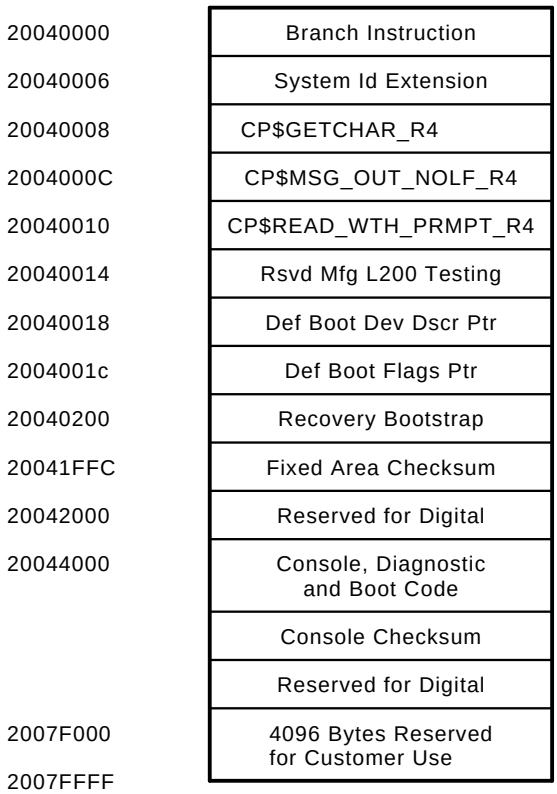


Figure G–1 KA670 EPROM Layout

The first instruction executed on halts is a branch around the system ID extension (SIE) and the callback entry points. This allows these public data structures to reside in fixed locations in the EPROM.

The callback area entry points provide a simple interface to the currently defined console for VMB and secondary bootstraps. 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.

The console, diagnostic and boot code constitute the bulk of the KA670 firmware. This code is field-upgradeable. 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 EPROMs is reserved for Digital, for future expansion of the KA670 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 Section 9.5.3.2 and Figure 9–9 for a description of the boot block mechanism.

G.1.1 Call-Back Entry Points

The KA670 firmware provides several entry points that facilitate I/O to the designated console device. Users of these entry points do not need to be aware of the console device type, be it a video terminal or workstation.

The primary intent of these routines is to provide a simple console device to VMB and secondary bootstraps, before operating systems load their own terminal drivers.

These are JSB (subroutine, as opposed to procedure) entry points located in fixed locations in the firwmare. 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 physical mode. Virtual mode is not supported. Due to internal firmware architectural restrictions, users are encouraged to only call into the halt-protected entry points. These entry points are listed below.

CP$GET_CHAR_R4	20040008
CP$MSG_OUT_NOLF_R4	2004000C
CP$READ_WTH_PRMPT_R4	20040010

G.1.1.1 CP$GETCHAR_R4

This routine returns the next character entered by the operator in R0. A timeout interval can be specified. If the timeout interval is zero, no timeout is generated. If a timeout is specified and if timeout occurs, a value of 18 (CAN) is returned instead of normal input.

Registers R0,R1,R2,R3 and R4 are modified by this routine, all others are preserved.

```
;---------------------------------------------------------------------
; Usage with timeout:

movl    #timeout_in_tenths_of_second,r0 ; Specify timeout.
jsb     @#CP$GET_CHAR_R4                ; Call routine.
cmpb    r0,#^x18                        ; Check for timeout.
beql    timeout_handler                 ; Branch if timeout.
; Input is in R0.

;---------------------------------------------------------------------
; Usage without timeout:

clrl    r0                              ; Specify no timeout.
jsb     @#CP$GET_CHAR_R4                ; Call routine.
; Input is in R0.

;---------------------------------------------------------------------
```

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 and therefore the caller need not provide an input buffer. Successive calls however destroy the previous contents of the input buffer.

Registers R0, R1, R2, R3, and R4 are modified by this routine. All other registers are preserved.

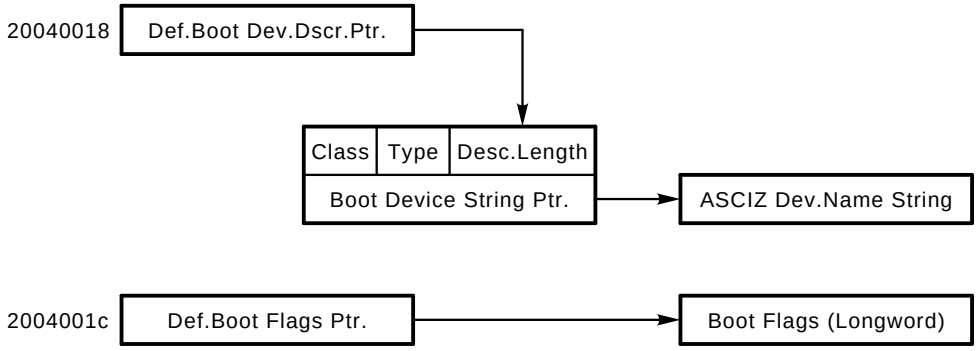
```
;-----------------------------------------------------------------------
; Usage with a message descriptor (position independent).
pushab  10$                             ; Generate prompt desc.
pushl   #10$-5$                         ; on stack.
movl    sp,r0                           ; Pass desc. 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, since the actual location of this data may change in successive versions of the firmware. Any software that uses these pointers should reference them at the addresses in halt-protected space.



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
RAM Partitioning

This appendix describes how the KA670 firmware partitions the 1 kilobyte of battery backed-up (BBU) RAM on the SSC chip.

H.1 SSC RAM Layout

The KA670 firmware uses the 1 kilobyte of battery backed-up RAM on the SSC to store 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 (Figure H–1). The BBU RAM should not be used by the operating systems except as documented here. The BBU RAM is not reflected in the bitmap built by the firmware.

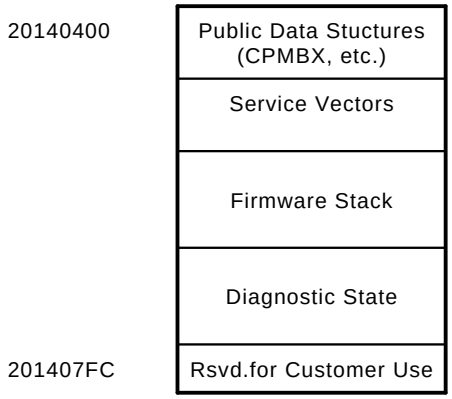


Figure H–1 KA670 SSC BBU RAM Layout

H.1.1 Public Data Structures

This section describes the public data structures in BBU RAM used by the console.

Fields that are desginated as reserved and/or internal use should not be written, since 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 KA670 firmware and diagnostics, VMB, or an operating system. It consists of three bytes referred to here as NVR0, NVR1, and NVR2 (Figures H–2 to H–4).

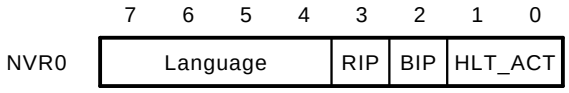


Figure H–2 NVR0 (20140400) : Console Program Mailbox (CPMBX)

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 9–1 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.

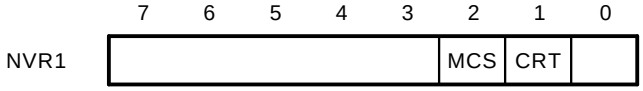


Figure H–3 NVR1 (20140401)

Field	Name	Description
2	MCS	If set, indicates that the attached terminal supports DEC Multinational Character Set. If clear, MCS is not supported.
1	CRT	If set, indicates that the attached terminal is a CRT. If clear, indicates that the terminal is hardcopy.

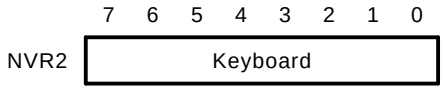


Figure H–4 NVR2 (20140402)

Field	Name	Description
7:0	Keyboard	This field indicates the national keyboard variant in use.

H.1.3 Firmware Stack

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

This area is used by the firmware-resident diagnostics. This section is not documented here.

H.1.5 User Area

The KA670 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 used by the KA670 firmware.

I.1 Halt Dispatch State Machine

The KA670 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 bitmap 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,** used for resolving external halts. 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,** 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,** 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,** specified with the SET HALT console command.

  000 : default
  001 : restart, halt
  010 : boot, halt
  011 : halt
  100 : restart, boot, halt

- **HEN,** BREAK (halt) enable switch, BDR<7>

- **ERR,** error status

- **TIP,** trace in progress
- **DIP,** diagnostics in progress
- **BIP,** bootstrap in progress CPMBX<2>
- **RIP,** restart in progress CPMBX<3>

Table I–1 Firmware State Transition Table

Current State	Next State	Halt Type	Halt Code	Mailbx 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]				
INIT	–>TRACE	xxx	10	xx	xxx	x - x - 1 - x - x - x
TRACE	–>EXIT	xxx	10	xx	xxx	x - 0 - 1 - x - x - x

[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, 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.

x = don't care field.

Table I–1 (Cont.) Firmware State Transition Table

Current State	Next State	Halt Type	Halt Code	Mailbx Action	User Action	HEN-ERR-TIP-DIP-BIP-RIP
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
REST	–>HALT	xxx	xx	xx	xxx	x - x - x - x - x - x

[5] Bootstrap,

if powerup and halts are disabled.
if powerup and halts are enabled and user action is 2 or 4.
if not powerup and mailbox is 2.
if not powerup and mailbox is 0 and user action is 2.
if not powerup and restart failed and mailbox is 0 and user action is 0 or 4.

[6] Restart the operating system if not power-up and,

if mailbox is 1.
if mailbox is 0 and user action is 1 or 4.
if mailbox is 0 and 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.

x = don't care field.

Table I–1 (Cont.) Firmware State Transition Table

Current State	Next State	Halt Type	Halt Code	Mailbx Action	User Action	HEN-ERR-TIP-DIP-BIP-RIP
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 field.

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.

I.2 Restart Parameter Block(RPB)

VMB typically utilizes the low portion of memory unless there are bad pages in the first 128 kilobytes. The first page in its block is used for the restart parameter block (RPB), which the VMB uses to communicates with the operating system. Usually, this is page 0.

VMB will initialize the restart parameter block as follows (Table I–2):

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.

Table I–2 (Cont.) Restart Parameter Block Fields

(R11)+	Field Name	Description
24:	RPB$L_ BOOTR2	R2 on entry to VMB.
28:	RPB$L_ BOOTR3	R3 on entry to VMB.
2C:	RPB$L_ BOOTR4	R4 on entry to VMB. **NOTE** **The 48 bit booting node address is stored in RPB$L_BOOTR3 and RPB$L_BOOTR4 for compatibility with ELN V1.1 (this field is only initialized this way when performing a network boot).**
30:	RPB$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$Q_ PFNMAP	The PFN bitmap is a array of bits, where each bit has the value 1 if the corresponding page of memory is valid, or has the value 0 if the corresponding page of memory contains a memory error. Through use of the PFNMAP, the operating system can avoid memory errors by avoiding known bad pages altogether. The memory bitmap is always page-aligned, and describes all the pages of memory from physical page #0 to the high end of memory, but excluding the PFN bitmap itself and the Q-bus map registers. If the high byte of the bitmap spans some pages available to the operating system and some pages of the PFN bitmap itself, the pages corresponding to the bitmap itself will be marked as bad pages. The first longword of the PFNMAP descriptor contains the number of bytes in the PFNMAP. The second longword contains the physical address of the bitmap.
4C:	RPB$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 bitmap 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.

Table I–2 (Cont.) Restart Parameter Block Fields

(R11)+	Field Name	Description
66:	RPB$B_ DEVTYP	Device type code of boot device.
67:	RPB$B_SLAVE	Slave number of boot device.
68:	RPB$T_FILE	Name of secondary bootstrap image (defaults to [SYS0.SYSEXE]SYSBOOT.EXE). This field (up to 40 bytes) is over-written with the input string on a 'solicit' boot. **NOTE** **1 : For VAX/VMS, the RPB$T_FILE must contain the root directory string SYSn. on a non-network boot-strap. This string is parsed by SYSBOOT (ie SYSBOOT does not use the high nibble of BOOTR5). 2 : The RPB$T_FILE is over-written to contain the boot node name for compatibility with ELN V1.1 (this field is only initialized this way when performing a network boot).**
90:	RPB$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$L_ MEMDSC	Count of pages in physical memory including both good and bad pages. The high 8 bits of this longword contain the TR #, which is always zero for KA670.
C0:	RPB$L_ MEMDSC+4	PFN of the first page of memory. This field is always zero for KA670, 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 VAX/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, else if initialized, A=1, B=2, etc. (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 (Table I–3):

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 bitmap. First longword contains count of bytes in bitmap. Second longword contains physical address of bitmap. (Same rules as for RPB$Q_PFNMAP listed above.)
1C:	VMB$Q_ UCODE	Quadword.
24:	VMB$B_ SYSTEMID	A 48-bit (actually a quadword is allocated) booting node address which is initialized when performing a network boot. This field is copied from the target system address parameter of the parameters message. (The DECnet HIORD value is added if the field was 2 bytes.)
2C:	VMB$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 KA670 frmware fall into three catgories:

- Halt code messages
- VMB error messages
- Console messages

In general, error messages are in cryptic, to avoid the space requirements of translating a large number of messages.

J.1 Halt Code Messages

The messages in Table J–1 are issued by the frmware whenever the processor halts (except on power-up, which is not treated as an error condition).

For example:

```
?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 written on any CVAX processor restart operation.

Table J–1 HALT Messages

Code	Message	Description
?02	EXT HLT	External halt, caused by either console BREAK condition, Q22-bus BHALT_L, or 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*	PSL<26:24> = 5 on interrupt or exception.
?1A	PSL EXC6*	PSL<26:24> = 6 on interrupt of exception.
?1B	PSL EXC7*	PSL<26:24> = 7 on interrupt or exception.
?1D	PSL REI5*	PSL<26:24> = 5 on an rei instruction
?1E	PSL REI6*	PSL<26:24> = 6 on an rei instruction.
?1F	PSL REI7*	PSL<26:24> = 7 on an rei instruction.
?3F	MICROVERIFY FAILURE	Microcode power-up self-test failed.

*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.2 VMB Error Messages

The following errors are issued by VMB (Table J–2):

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.3 Console Error Messages

The following error messages are issued in response to a console command that has error(s) (Table J–3):

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.
?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	Qualifier conflict, for example, 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 rather than being 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 of 80_{16} or 81_{16}, indicating a read or write memory error.
?70	UNIMPLEMENTED	Unimplemented function.
?71	NO VALUE QUALIFIER	Qualifier does not take a value.
?72	AMBIGUOUS QUALIFIER	There were not enough unique characters to determine the qualifier.
?73	VALUE QUALIFIER	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.

Table J–3 (Cont.) Console Error Messages

Code	Message	Description
?76	AMBIGUOUS COMMAND	There were not enough unique characters to determine the command.
?77	TOO FEW ARGUMENTS	Insufficient arguments 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.
?7C	HARD ERROR	A hard error occurred.
?7D	MACHINE CHECK	A machine check occurred.

Glossary

BBU RAM
Battery backed-up RAM. On the KA670, this is 1 Kbyte of battery backed-up RAM on the SSC.

BFLAG
Boot flags is the longword supplied in the SET BFLAG and BOOT /R5: commands that qualify the bootstrap operation. SHOW BFLAG displays the current value.

BHALT
Q22-bus HALT signal, usually associated with the halt switch on the front panel.

BIP
Boot in progress flag in CPMBX<2>.

CPMBX
Console program mailbox is used to pass information between operating systems and the firmware.

CQBIC
Q22-bus interface chip.

DCOK
Q22-bus signal indicating dc power is stable. This signal is usually associated with the restart switch on the front panel.

DNA
Digital network architecture.

EPROM
Erasable programable read only memory. Used on some products to store firmware. Commonly used synonyms are PROM or ROM. Erasable by using ultraviolet light.

DE
Diagnostic executive. 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 256 kilobytes of VAX instruction code residing at physical address 20040000 on the KA670. Functionally, it consists of diagnostics, bootstraps, console, and halt entry/exit code.

GPR
General-purpose registers. On the KA670, these are the 16 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).

IPL
Interrupt priority level. Ranges from 0 to 31 (0 to $1F_{16}$).

IPR
Internal processor registers. On the KA670, these are implemented by the CPU 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.

KA670
Q22-bus CPU processor module, DSSI port, and Ethernet adapter.

LED
Light emitting diode.

MSCP
Mass storage control protocol. Used in Digital disks and tapes.

MOP
Maintenance operations protocol. Specifies message protocol for network loopback assistance, network bootstrap, and remote console functions.

ms
Millisecond (10e-3 seconds).

PC
Program counter (R15).

PCB
Process control block. A data structure pointed to by the PR$_PCBB register. Contains the hardware context for the current process.

PFN
Page frame number. 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
R(X)eceive console status (IPR 32).

PR$_RXDB
R(X)eceive 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). Commonly referred to as the time-of-year register or TOY clock.

PR$_TXCS
T(X)ransmit console status (IPR 34).

PR$_TXDB
T(X)ransmit data buffer (IPR 35).

PSL, PSW
Processor status longword. The VAX extension of the PSW (processor status word). The PSW (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 acess memory.

RIP
Restart in progress flag in CPMBX<3>.

RPB
Restart parameter block. 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. A data structure pointed to by PR$_SCBB. THESCB contains a list of longword exception and interrupt vectors.

SGEC
Second-generation Ethernet chip.

SHAC
Single host adapter chip.

SP
Stack pointer (R14).

SRM
Standard reference manual, as in VAX SRM.

SSC
System support chip.

µs
Microsecond (10e-6 seconds).

VMB
Virtual memory boot. The portion of the firmware dedicated to booting the operating system.

Index

D

E

F

N

O

P

Q

R

S

T

V